Hello there!

Thank you for your interest in Ruth’s musings in security. I aim to read one academic paper in cryptography (theory) and computer security (application) each week. Here are the musings from the week 10/9/2016 to 16/9/2016. I am very far from being an expert on these topics, therefore, if you need to contact me, to report errors in my stuff, my email address is thisemailisnotruthless@gmail.com. Also, for clarification, this is not research that was done this week. This is research that was read by Ruth this week =P

This Week in Crypto: “Cache-timing Attacks on AES”

by Daniel J. Bernstein

Summary of Result and Implications: In this work, the authors do a side-channel attack on AES, and manage to retrieve the key through timing attacks on OpenSSL (on a Pentium III). The author states also that this attack could work on other chips, such as an AMD Athlon, Intel Pentium III, Intel Pentium M, IBM PowerPC RS64 IV and a Sun UltraSPARC III, as they leak the same timing information. The problem lies in how it is difficult to write constant time high-speed AES software for the common general-purpose CPU. While AES lends itself nicely to constant time low-speed encryption, it is when the SBoxes are implemented in “T-tables”, to speed it up, that cache hits and misses become a source of side-channel information.

The author considers this a call for encryption that uses constant time primitives. Examples of such are the TEA, SHA-256, Helix and Salsa20. These all allow high-speed performance that exhibits timing that is independent of input.

Brief Overview of the Attack: The idea behind the attack is simple, to collect timing data when \( n_i \oplus k_i \) is known, then using this to infer \( k_i \) when only \( n_i \) is known. Here, \( n_i \) are the bytes of PT and \( k_i \) are the bytes of key. The attack solves for one byte of key (\( k_i \)) at a time in the following way, for a particular \( i \):

1. Attacker observes the timing data on a setup with a known key, \( k_i \), under random \( n_i \), and notes that at \( n_i \), the timing is at a maximum.
2. Attacker observes the timing data on the setup with unknown key, under random \( n_i \), and notes that at \( n_i \), the timing is at a maximum.
3. Attacker then guesses that \( n_i \oplus k_i' = n_i' \oplus k_i \), and then derives \( k_i \).

More broadly speaking, the attacker can go further to calculate the correlation between the timing data with respect to the various \( n_i \) against the known timing, subject to varying linear shifts. Note also that the attacker is simply doing a known PT attack here, since he assumes random packets being sent, and does not influence the packets being sent. The attack will succeed so long as he is able to collect data from a large enough number of different \( n_i \). The author also notes that the introduction of random noise will make the attack have higher data complexity, but the attack will still succeed since the effects of the noise can be normalized with enough data.

In Practice: In carrying out this attack, the author noted that for some bytes, he noted that he was able to ascertain some but not all bits in the byte (e.g. when \( n_i \), with the same prefix all perform similarly, and significantly differently from the rest). To circumvent this, he ran the attack a few times, varying the packet size, and was able to attain more bits. Finally, for the bits that could not be found, the author did a brute force search. For each byte, he did not require more than \( 2^{30} \) data complexity, and the attack was as fast as the connection was able to transmit that number of packets.

Recommendations: The success of such a simple timing attack against one of the top encryption standards is very disturbing. The author notes that erroneous thinking
in the AES standardization process led to this being overlooked. In particular, NIST failed to consider that table lookups do not take constant time when the tables are large. Specifically, the speeding up of AES with T-tables generated this problem.

The authors’ recommendations reiterate that constant time AES implementations with current technology is going to result in a huge performance hit. However, he also warns against continuing with high-speed AES as it is, in light of these findings. In addition, he notes the following additional challenges to constant time AES implementations:

- If one collects timings and tries to evaluate if AES implementations have constant time, it is not guaranteed that any and all correlations would be found, as it is difficult to separate random noise from a correlation without prior knowledge of what kind of correlations one is looking for. This is not unlike the problems faced by statistical testing.

- Without information from CPU designers on the expected timings of each operation, it is impossible to “theoretically” generate constant time encryption. CPU designers are notorious for not documenting this information about their chips.

- In order to speed up encryption, algorithms tend to do input dependent branches, which result in drastically different timings. These are optimizations of certain steps of high-speed AES on general purpose CPUs

- Cached memory is the cause of a large amount of discrepancy between timings, since input can determine if a cached or uncached line is being read, which differ significantly in timing. Even worse is the fact that modern CPUs have L1 and L2 caches, creating even more differentiation in timing. The only way to get around this is to ensure that all the SBox lines that could possibly be needed are in the same level of cache before computation, which is difficult.

- Even if memory is cached correctly, other processes or parts of the AES computation can kick lines out of the cache. The authors’ suggestion in this case is to have a dedicated instruction to load all desired memory to the L1 cache and access it in constant time.

- Associativity of caches make for even more timing variation. For example, Athlon’s L1 cache is 2-way associative, meaning each line can only be loaded in two places. This gives even more credence to the suggestion above.

- Interrupts are hard to prevent, and these can stall cache loading and other parts of the AES implementation designed to achieve constant time encryption. This can be prevented by disabling hyper-threading.

- On some CPUs, it has been shown that a load from the L1 cache line (modulo 4096) that has recently seen a store is slower. Care has to be taken with the stack position to prevent this from happening.

- Finally, cache bank throughput is limited. For example, on the Athlon, each 64-byte cache line is divided into 8 banks, and if two loads are called from different banks, they can be processed at the same time, but if they are from the same bank, it will take longer. Therefore, care has to be taken to either, choose a bank size that prevents this problem, or arrange that each load occur in a separate cycle.

This Week in Security: “Security by Any Other Name: On the Effectiveness of Provider Based Email Security”

by Ian Foster, Jon Larson, Max Masich, Alex Snoeren, Stefan Savage, Kirill Levchenko

Premise: The authors review the real-world security of email. Specifically, this is the level of message integrity, authenticity and confidentiality offered to users by the top commercial email services. In this work, the authors assume a trusted provider, meaning that they do not tackle the case where a provider (e.g. Gmail) could be acting to subvert security. Instead, they answer the following two questions:

Are established protocols (e.g. TLS) sufficient to provide security to users?

Are these protocols being used in a way to achieve this level of security?

To do this, they establish a threat model, where they simplify the path of a mail message as in the below figure.

![Figure 1: Mail Message Path](image)

Here, we have a Mail User Agent (MUA) sending a message through a Message Submission Agent (MSA) to a Message Transfer Agent (MTA), both of which are provided by his chosen sending organization. The message is sent to the recipient’s MTA, which is passed to a Message Delivering Agent (MDA), then to the other MUA.

They consider three kinds of network attackers:
• **Active:** This is an attacker who can observe, modify, inject or remove packets as they are sent between users.

• **Peer:** This is an attacker who is an ordinary host connected to the Internet, that can send arbitrary packets, and received packages for which it is the destination.

• **Passive:** This attacker can observe but cannot modify the traffic between a target and the rest of the Internet.

As mentioned earlier, the authors concern themselves with evaluating security in three ways: authenticity, integrity and confidentiality.

**Study of Existing Protocols:** Without going into too much detail, the authors showed that with TLS, DKIM, SPF, DNSSEC used properly, all the security properties are guaranteed against the three types of attackers.

For confidentiality, it is sufficient to use TLS against a passive attacker. For an active attacker, doing a MITM attack, one can use DNS integrity with TLS, or checking the subjectAltName or CN are checked against the intended server domain name (and not just the IP address).

For authenticity, the active attacker or peer attacker may be seeking to forge messages. If so, certificate checking is required. SPF can protect against a peer attacker, while a combination of DKIM and DNSSEC can be used to protect against an active attacker.

For integrity, an active attacker may be hoping to modify a packet. If so, DKIM and DNSSEC can protect the messages from tampering.

The authors also did a survey into how well these are used/enforced. To this end, they realized that DKIM and SPF were introduced as methods of fighting spam, meaning that it is optional, not widely adopted, and not strictly enforced (if the verification fails, the message is flagged as spam, instead of being deleted/not delivered). In addition, due to the limited adoption of such signing methods, providers who want to enforce such usage must bilaterally agree and disseminate their signing policies with each other. For example, GMail has such an agreement with eBay and PayPal.

**Experiments:** The authors performed a few experiments to see how extensively and accurately the above policies were used. For automated tests, they made use of the top 1 million email address domains (“provider list”), and for manual tests, they made use of the top 22 email providers (“select provider list”). This was taken from the leaked Adobe user data set. Here is a brief description of the experiments ran:

1. **Incoming MTA Behavior:** Tried to connect to the domains in the provider list, begin a TLS session and send a message.

2. **Outgoing MTA Behavior:** Tried to connect to, establish a TLS session and receive a message via TLS with accounts in the select provider list.

3. **SMTP MSA Behavior:** Obtain the mail submission configuration information from the providers on the select provider list.

4. **POP/IMAP Behavior:** As in SMTP, the 22 select providers’ servers were probed for support for these, and whether they allowed TLS with them.

5. **Webmail Behavior:** Checked if the web interface of the select providers used HTTPS.

6. **Reported TLS Use:** Checked if the required Received header in SMTP reports support for TLS, in the list of select providers. This also showed whether TLS was used in the internal hops (MSA to MTA and MTA to MDA).

7. **Cross-Provider Validation:** Sent messages between each pair of providers in the select provider list, to look for anomalous pairwise behavior (with respect to the earlier experiments).

8. **Certificates:** Checked if server certificates provided by the select providers were revoked (via a CRL) or expired. Checked also if the subjectAltName matched the domain name, and if it was signed by a trusted (with the Mozilla list as root) CA.

9. **DKIM:** Checked if DKIM was used by the select providers. Then tested the results of sending a message with a DKIM signature, without a DKIM signature and with an invalid DKIM signature.

10. **SPF, ADSP, DMARC, DNSSEC:** To test support for SPF and DMARC, the DNS TXT record on the servers listed on the provider list was obtained. DNSSEC support was verified by querying the DNSKEY record of the providers on the providers list.

**Results:** For the full results, one should consult the original paper, as there is no way I can summarize the many details presented. I have, however, replicated the various diagrams that were used to represent the experimental results.
<table>
<thead>
<tr>
<th>Domain</th>
<th>SSL</th>
<th>SPF</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Commerce</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Entertainment</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Misc</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Banks</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Government</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Conferences</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>News</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Dating</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Table 4: TLS, SPF, and DMARC (DM) support among outgoing MTAs used by select Web services to send email. ✔️ indicates no support or protection, ✔️ indicates basic support, and ✔️ indicates that SPF or DMARC is configured in a strict manner.

<table>
<thead>
<tr>
<th>Status</th>
<th>Freq. 2014</th>
<th>Freq. 2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>75.86%</td>
<td>79.14%</td>
</tr>
<tr>
<td>Self Signed</td>
<td>20.47%</td>
<td>11.39%</td>
</tr>
<tr>
<td>Expired</td>
<td>3.41%</td>
<td>2.88%</td>
</tr>
<tr>
<td>Revoked</td>
<td>0.17%</td>
<td>0.04%</td>
</tr>
<tr>
<td>Non Matched</td>
<td>34.13%</td>
<td>37.26%</td>
</tr>
</tbody>
</table>

Table 5: Certificate status of the top 20 MTAs found in the Adobe data set.
Discussion: The authors summarized their work by considering how effective each of the three attackers considered would be in undermining email security.

For the passive attacker, his can be thwarted by use of SSL without certificate checking. In the best case scenario, where users are assumed to enable SSL in both submission and delivery wherever possible, TLS protected messages will comprise 74% of messages surveyed. Conversely, in the worst case, where TLS is toggled off whenever possible, TLS-protection only covers 24% of messages.

For the peer attacker, while some domains did honor a strict SPF policy, in the majority of domains it was possible to impersonate an email generator or provider to another provider. DKIM use is even less widespread. Of the top 5 domains, only GMail and Yahoo! used DKIM, and only Hotmail marked messages with invalid DKIM signatures as spam.

For the active eavesdropper, if he has full MITM capability, he can only be thwarted by proper certificate checking. The authors found that outside of the Web mail interface, there was little certificate checking, even on the MTA-MTA hop.

For the active attack who wishes to tamper with messages, the only defense at present is DKIM with DNSSEC and a strict DMARC policy. Only one provider of the top 22, Comcast, offers DNSSEC and of the 14 providers (out of 22) who provided some sort of verification, only 5 actually enforced it. Thus, active attacks on integrity will be unimpeded.

Recommendations: The authors summarize their recommendations as follows, to ensure the desired level of email security:

1. TLS support in SMTP, POP and IMAP are all mature and stable, so it should be used for mail submission
2. Certificates should have a name matching the DNS name, and this should be checked.
3. Certificates should be verified, including host name
4. TLS should be required at the MSA/MDA level, so that it can be used on the MTA-MTA level
5. The set of each peer’s allowed CAs should be fixed to prevent rogue CAs
6. DKIM and DMARC should be used to protect integrity
7. SPF and DKIM policy should be enforced strictly (mail failing the authentication should be rejected immediately)
8. DNSSEC needs to be adopted on a larger scale.

The final interesting contribution of the authors is to provide a game theoretic explanation for the current poor adoption of security measures. Stated plainly, the senders will not enforce validity until nearly all receivers have valid certificates, and the receivers will not use valid certificates until the senders refuse to accept mail that has invalid ones. They propose that the only way out of this deadlock is for the top providers to work together to enforce this policy for their users, so that smaller providers will be incentivized to follow.