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 15/10/2016 to 21/10/2016. 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: “Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1”

by DANIEL BLEichenBacher

Author’s Note: I am covering this paper because it was something that was presented in school this week, and it was relatively interesting. However, please note that this paper is quite old (1998), and I’m sorry if many of you have probably seen this before.

Premise: In this article, the author presents a Chosen Ciphertext Attack on RSA encryption with PKCS #1. PKCS #1 is a padding scheme used to pad block cipher messages, and introduce randomness to schemes. In this, the author requires an oracle that will return whether an encrypted message will decrypt to something that is PKCS conforming or not.

It is important first to note what PKCS #1 padding is. Consider the message $M$, strictly smaller (by at least 11 bytes) than the total block size, which we denote $k$ (in bytes). We randomly pick $PS$, a string of $k - 11 - |M|$ bytes, where no byte is 00. Here, each double digit figure denotes a byte.

$$PKCS(M) = 00||02||PS||00||M$$

. In other words, a block is PKCS-conforming if the first byte is 00, the second is 02, the next 8 are non-zero, and at least one of the remaining bytes is 00.

Finally, we recall what a chosen ciphertext attack on RSA is: In RSA, a user has public key $e$, $n$, and private key $d$. These are such that $M^{de} \equiv M \pmod{n}$ for all messages $M$. The ciphertext of $M$ is then $M^e \pmod{n}$, and ciphertext $C$ can be decrypted by taking $C^d \pmod{n}$. In RSA with PKCS #1, the message $M$ must by PKCS-confirming. Adversary $A$ has access to an oracle that decrypts any provided ciphertext $C$ with secret-key $d$ and returns whether or not this value is PKCS conforming. The goal here is for the adversary to compute $C^d \pmod{n}$ without using the decryption oracle on ciphertext $C$.

Basics of Method: I will go into the details of the method in a later section, but for those who do not want to get lost in the math, the method makes use of the fact that if $C$ decrypts to a $M$, a PKCS-conforming value, then $2B \leq M < 3B$, where $B = 2^{8k-8}$. Given enough queries to the oracle, there is an adversary $A$ that can compute $C^d \pmod{n}$, without ever querying the oracle with $C$. This attack expects to need

$$\frac{3}{Pr(P)} + \frac{16k}{Pr(P|A)}$$

queries to the oracle, where $P$ is the event that a random block decrypts to something PKCS-conforming, and $A$ is the event that the first two bytes of the message are 00 and 02 respectively. They also show that

$$0.18(2^{-16}) < Pr(P) < 0.97$$

and

$$2^{-16} < Pr(A) < 2^{-8}$$

. For example, for a 1024-bit modulus, we need $2^{20}$ chosen ciphertexts.
Observations/ Discussion: The authors make a few interesting notes:

- The authors assume the padding PS to be chosen randomly and independently of each other. If there is some relation between how they are chosen, this could be exploited for better results.

- Next, the authors note that the choice to make an RSA modulus a multiple of 8 is a good one, making $\Pr(P)$ small.

- Finally, they present three ways an “oracle” can be obtained by Eve in the real world:
  1. By impersonating Alice and sending messages to Bob, Eve can send messages and check if Bob succeeds in reading them, which will happen only if they are conforming to PKCS. Even if authenticity checks are done at a later stage, Eve will still succeed if she can get the information she needs before she needs to provide authenticity checks.
  2. If detailed error messages are given by the system in question, Eve can exploit this to find out when her message was decrypted correctly or not.
  3. If the channel leaks side-channel information, it can be exploited. For example if the system checks PKCS conformance, then only performs integrity checks on PKCS conforming instances, the timing information can be used to see whether the PKCS check passed.

Experimental Results: The authors also did reference a small number of experiments. First, they succeeded in implementing the attack on 1024 and 512-bit keys, using between 300 thousand and 2 million chosen ciphertexts. In this, the oracle was their own implementation, instead of any existing implementation (e.g. SSL). Second, they point to Finney’s work on SSL servers, to see how amenable they are to providing an “oracle”. One of the servers did not verify authenticity, and therefore any rejection of a message can be associated with non-PKCS conformance. The second checked PKCS conformance, message length and version number, providing different error messages for each, a problem as the authors note above. The final one did not leak information through its error messages, and performed all aspects correctly (but this does not cover potential side-channel attacks).

Details of Method: For those that are interested, I provide a transcribed copy of the attack below, with slightly more description to (hopefully) aid understanding. Here, we assume that the message that the adversary is trying to decrypt is $c$ to $m$, the plaintext, and for brevity when we say that a ciphertext is PKCS conforming, we mean that the ciphertext decrypts to a message that is PKCS conforming:

1. Set $M_0 \leftarrow \{[2B,3B-1]\}$ ($B$ is defined above), $i \leftarrow 1$.
   
   (a) If $c$ is already PKCS conforming, set $s_0 \leftarrow 1$, $c_0 \leftarrow c$.
   
   (b) Else, choose random $s_0$ until $c(s_0)^e \mod n$ is PKCS conforming, set $c_0 \leftarrow c(s_0)^e \mod n$.

2. If $i = 0$, go to 2a, else if $|M_{i-1}| > 1$ go to 2b, else go to 2c.
   
   (a) Search for smallest possible $s_1 \geq \frac{n}{2N}$ such that $c_0(s_1)^e \mod n$ is PKCS conforming.
   
   (b) Search for the smallest $s_i > s_{i+1}$ such that $c_0(s_i)^e \mod n$ is PKCS conforming.
   
   (c) Suppose $M_i = [a, b]$. Choose small integer values $r_i, s_i$ such that $c_0(s_i)^e \mod n$ is PKCS conforming, satisfying
   
   $$r_i \geq \frac{2bs_{i-1} - 2B}{n}$$
   
   and
   
   $$\frac{2B + r_in}{b} \leq s_i \leq \frac{3B + r_in}{a}$$

3. Compute $M_i$ as follows, where the union is taken over all $a, b, r$ such that $[a, b] \in M_i$ and $\frac{as_i - 3B + 1}{n} \leq r \leq \frac{bs_i - 2B}{n}$:

   $$M_i \leftarrow \bigcup_{(a,b,r)} \left[ \max\left(a, \frac{2B+rn}{s_i}\right), \min\left(b, \frac{3B-1+rn}{s_i}\right) \right]$$

4. If $M_i = \{[a, a]\}$ for some $a$, set $m \leftarrow a(s_0)^{-1} \mod n$ and return $m$, else, set $i \leftarrow i + 1$ and go to step 2.

The author also proves this method to be correct by induction, showing that with each step, the union of the intervals in $M_i$ strictly decreases, and the $a$ such that $m \equiv a(s_0)^{-1} \mod n$ is always in $M_i$. Therefore, the algorithm terminates, and always with the correct solution.

This Week in Security: "Hidden Voice Commands"

by Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, Wenchao Zhou

Premise: Many voice-controlled devices have adopted an “always on” model in which they continuously listen for voice input. However, voice is a broadcast channel open to any attacker that is able to create sound within the vicinity of a device. The severity of a hidden voice command depends upon what commands the targeted device will accept. Depending on the device, attacks could lead to information leakage (e.g. posting your location on Twitter), cause denial-of-service (e.g.
activate airplane mode) or as a stepping stone to further attacks (e.g. opening a webpage with malware). Hidden voice commands could be communicated through anything from loud speaker broadcasts to viral videos.

**Background:** The authors begin by noting how the process of voice recognition usually works:

1. **Pre-processing:** Here speech is separated from non-speech in a rudimentary way to cut out background noise.

2. **Feature extraction:** Here speech the filtered audio signal is split into short frames and we extracts features from each frame. This almost always is done with a Mel-frequency central (MFC) transform.

3. **Model-based prediction:** The features are compared to a precomputed model. This can be done with a Hidden Markov Model, or other machine learning techniques.

4. **Post-Processing:** This ranks the predicted text with regards to additional information, such as grammar and locality of words.

The authors also note related work in the field:

- **Diao et al, Jang et al:** Demonstrated that malicious apps can be used to inject audio commands to control smartphones, but these are detectable by humans in earshot

- **Esteves:** Injected voice commands that were hidden by transmitting FM signals to devices with FM antenna

- **Schlegel et al:** Malicious apps that eavesdrop and record phone calls to extract information.

- **Vaidya et al:** Obfuscated voice commands that are accepted by voice interfaces, which the authors expand upon

**Black-box attacks:** Roughly speaking, this is when the attacker does not know the inner workings of the voice recognition system. Here, the attacker queries a voice recognition system, noting whether it recognized commands or not, and using this to build a model of the inner workings of the system. Their attack, in brief, goes as follows:

1. Attacker generates a candidate obfuscated audios, generated with audio manglers (which removes all features not being captured by the feature extractor).

2. Using this, the attacker figures out the parameters of the MFC transform, and settles on a particular obfuscated audio file that is accepted by the voice recognition system

3. Ascertain if the mangled audio is sufficiently unintelligible to human listeners.

The authors did an experiment to demonstrate this, using a Samsung Galaxy S4 and an Apple iPhone 6. They first studied the influence of background noise on the system, adjusting the signal-to-noise ratio that the systems tolerated, and the distance of the phones to the overhead speakers that were being used. They settled on speakers being 3.5 meters away, and the signal-to-noise ratio of less than 5dB. The phone correctly interpreted 60% of obfuscated commands played through the speakers, and 80% of normal voice commands. To test human understanding of the obfuscated command, they used human subjects on Amazon Mechanical Turk, who recognized 81% of normal commands, but only 41% of obfuscated commands.

**White-box attacks:** Now the authors studied how much their approach would be improved with knowledge of the underlying voice recognition system. They made use of CMU Sphinx for this, which makes use of a 13-dimensional MFC, as well as a 39-vector HMM, and a Gaussian Mixture Model.

The methodology of the black-box attack was used, but with improvements:

1. Using the exact MFC parameters, instead of guessing them, allowed the use of gradient descent methods, which not only seeks a solution accepted by the system, but a locally or globally optimal one.

2. With knowledge of the HMM being used, they could match their audio produced to the exact sequence of HMM states desired for recognition by the system.

3. Knowing that the HMM is rather insensitive to the number of times a state is repeated, but humans are, they could minimize the number of frames (of time) used to communicate the command, making it more unintelligible.

4. They could generate a model to understand the differences between the audio file generated by their methodology, and a recording of the audio being played back on the speaker. Then they could tune the gradient descent to penalize the generation of waveforms that the speaker could not replicate.

They did a similar experiment to the black-box instance and achieved much better results. Sphinx understood 82% of phrases generated with this method, and less than 1% (specifically, 1 out of 377) of phrases were “recognizable”. For comparison, 74% of phrases without obfuscation were recognizable.

**Defenses:** The authors briefly go over defenses, both those commonly suggested in practice, and novel ones:
• **Defenses that notify:** This is where the user is notified when the voice recognition system has received input, such as by having light that turns on, a beep or a buzz on a mobile phone. The authors note that humans have been shown to notoriously ignore such warnings.

• **Defenses that challenge:** This is where the device requires confirmation from the user (such as by a confirmation dialogue on a screen), or presents a challenge (such as an audio CAPTCHA) to detect if a human is generating the input. The authors do note that the former affects usability and the latter is not foolproof as an attacker can exploit the leeway given by the CAPTCHA to improve usability by humans.

• **Defenses that detect and prevent:** These come in a few forms:
  - A commonly proposed solution is speaker verification, so only the proper user of the device can issue voice commands. This reduces usability for items intended for multiple users (e.g. Amazon Echo). Also, research has shown that with minimal training, machine learning models can mimic the user's speech. For example, Aylett and Yamagishi mimicked George W. Bush's voice with only 10 minutes of his speech.
  
  – This method will slightly decrease audio quality before the command is processed. This affects normal voice commands minimally, but will affect obfuscated commands much more. The authors’ experiments were able to create a filter that drops the accuracy of obfuscated commands from 41.74% to 26.60%, but affects the accuracy of normal commands by 1.06%.

  – Finally, a machine learning classifier can be used to detect obfuscated audio. The authors were able to do this with 99.8% accuracy, after training the model with both obfuscated commands (from the black box experiment) and normal commands. An analogous experiment with the white box attack detected 70% of the obfuscated commands.

  The authors do concede that there are no proofs of security in this area, and that adversaries can incorporate the effects of these filters/ classifiers into their attack to generate obfuscated audio that cannot be detected this way.