Travis Dazell

I wanted a place to create random jabberings about software topics I'm working on. You'll find posts about software architecture, Java, web development, DSL development, language implementation, etc.

Friday, November 23, 2012

Many Time Pad Attack - Crib Drag

The one time pad (OTP) is a type of stream cipher that is a perfectly secure method of encryption. It's very simple to implement and is perfectly secure as long as the length of the key is greater than or equal to the length of the message. That's it's major downfall. However, it also requires that the key never be used more than once. This tutorial shows what happens when you re-use a key to encrypt more than one message. I also show how to uncover the plain-text of two messages that have been encrypted with the same key, without even knowing the key. I use a method called crib dragging.

Let's begin with a brief description of OTP and how it works. Let's take the following message and key:

message = "Hello World"
key = "supersecret"

If we convert both the message and key to hex strings, we get the following:

message = "48656c6c6f20576f726c64"
key = "7375706572736563726574"

If we do a simple XOR of the two hex strings we get the following cipher-text:

cipher-text = "3b101c091d53320c000910"

If we XOR the cipher-text with the key, we can recover the plain-text. That's how OTP works. Without the key, you have no way of uncovering the plain-text.

Let's consider what happens when you have two messages encrypted with the same key. Take the following two messages and key:

message1 = "Hello World"
message2 = "the program"
key = "supersecret"

If we convert each message and the key to hex strings, and then encrypt each message using a simple XOR with the key, we'll get the following cipher-texts:

cipher-text1: "3b101c091d53320c000910"
cipher-text2: "071d154502010a04000419"

Let's say that all we have is the two cipher-texts and the knowledge that they were encrypted with a supposed OTP; however, they were both encrypted with the same key. To attack this encryption and uncover the plain-text, follow the steps below.

1. Guess a word that might appear in one of the messages
2. Encode the word from step 1 to a hex string
3. XOR the two cipher-text messages
4. XOR the hex string from step 2 at each position of the XOR of the two cipher-texts (from step 3)
5. When the result from step 4 is readable text, we guess the English word and expand our crib search.
    1. If the result is not readable text, we try an XOR of the crib word at the next position.

Step 1 seems difficult (guessing a word that might appear in one of the messages), but when you think about it, the word "the" is the most commonly used English word. So, we'll start with assuming "the" is in one of the messages. After encoding "the" as a hex string, we'll get "746865". That takes care of steps 1 and 2. If we XOR the two cipher-texts, we'll get the following result:

cipher-text1 XOR cipher-text2 = "3c0d094c1f523808000d09"

The next step is to XOR our crib word "746865" at each position of the XOR of the cipher-texts. What we'll do is slide "746865" along each position of "3c0d094c1f523808000d09" and analyze the result. After the first XOR, we get the following result:

```
      3c0d094c1f523808000d09
XOR   746865
----------------------------------
      48656c
```

When we convert the hex string "48656c" to ASCII, we get the following text, "Hel". This takes us to step 5 from above. Because this looks like readable text, we can assume that the word "the" is in the first position of one message. If we didn't get readable text, we would slide "746865 (the)" one position to the right and try again (and keep repeating until the end of 3c0d094c1f523808000d09).

Note that we don't know which message contains the word "the". It could be in either message1 or message2. Next, we need to guess what the word "Hel" is when fully expanded. It could be "Help", "Hello", etc. If we guess "Hello", we can convert "Hello" to a hex string, we get "". We then XOR it with the XOR of the two cipher-texts (just like we did with "the"). Here's the result:

```
      3c0d094c1f523808000d09
XOR   48656c6c6f
----------------------------------
      7468652070
```

"7468652070" when converted to ASCII, is "the p". We then repeat the process, guessing what "the p" might be when expanded and then XOR that result with the XOR of the cipher-texts. Granted, guessing what "the p" might expand to is not super easy, but you get the idea. If we were to guess "the program", convert it to a hex string, and XOR it with the XOR of the cipher-texts, we'll get "Hello World".

This is called crib dragging. My suggestion is to first try " the " (note the spaces before and after). Most cipher-texts that you'll try cracking will contain that word somewhere in the text. If the result of your crib drag yields gibberish, then you can be sure " the " isn't in either of the plain-text messages. So, try another commonly used English word or phrase and keep trying until the result yields something that looks like readable text. Then you can just expand your guess and keep XORing until you uncover the plain-text messages.

In a future blog post, I'll demonstrate an implementation of a crib drag in Scala.

Posted by travisdazell at 8:50 PM

Labels: Cryptography

14 comments:

Unknown June 28, 2013 at 3:21 AM

Hi, I implemented it in scala.Take a look at my solution to traverse and match the ciphertexts. See my github account at https://github.com/vannicaruso/studying_crypto1

Reply

Anonymous December 17, 2013 at 2:27 AM

Nice article, but how would you do the third step if you had more than two messages ?

Reply

Replies

travisdazell December 17, 2013 at 9:23 PM

Thanks for the comment! For three messages (as an example), you'd XOR all three messages together. If you get readable text after Step 4, you know that the "guess word" appears in ONE of the three messages. You can then XOR just two of the three messages and repeat the process to deduce which encrypted message contains the guess word.

Reply

Anonymous December 19, 2013 at 7:03 AM

Nice article, but what if the two messages have different length. I tried with "Hello Simple World" and "the program" and couldn't get "Hel" later.

Reply

Replies

travisdazell February 11, 2014 at 10:28 PM

It will still work, but the key would of course need to be the same length as the longest message, since one time pad encryption assumes the key is the same length as the message. Therefore, the suffix of the key would be unused for the shorter message(s). Does that help? If not, I can put together an article to show this. Just let me know :)

Reply

Unknown May 5, 2014 at 7:32 AM

Nicely explained, thank you!

Reply

Unknown May 11, 2014 at 5:01 AM

great explanation thanks alot

Reply

Anonymous July 6, 2014 at 9:31 AM

Thank you very very much for the exlpanation!

Reply

Anonymous March 11, 2015 at 12:46 AM

Hi, this is really a good example, but I got one question, will this method going to be work when cracking 2 cipher-texts with the same hex ? e.g. Both of them have the same hex number "3c" at the first column.

Reply

Anonymous April 25, 2015 at 12:19 PM

Hello, I don't know if you're still around but I don't understand this part:
"Because this looks like readable text, we can assume that the word "the" is in the first position of one message."
Why?

Thanks a lot!

Reply

Replies

travisdazell May 3, 2015 at 8:26 PM

The first step is assuming that the word "the" is in one of the messages. I chose "the" because it's a very commonly used word. After you do the XOR routines described in the tutorial, if you get any readable text, you know that "the" is the clear text characters for one of the messages in that position. Long story short, that's how the approach works :)

I know it's in the first position, because I got clear text in that position.

The reality is, that there is some trial and error in guessing which word to try first. If you don't get readable text, you might try other commonly used words (e.g. "of", "there", "this", "with").

If you want more info, let me know.

Reply

Unknown May 22, 2015 at 8:28 AM

Is this method applied in real world scenario? like multimedia file?

Reply

Anonymous August 29, 2015 at 7:32 AM

"If we didn't get readable text, we would slide 48656c one position to the right and try again (and keep repeating until the end of 3c0d094c1f523808000d09)."

Isn't there a mistake? Shouldn't we slide 746865 (the) instead?

Reply

Replies

travisdazell April 21, 2016 at 7:01 PM

You are right. I've corrected the error. Thanks!

Scala Java Architecture DSLs Android Camel Cryptography Maven ANTLR Eclipse ActiveMQ Bouncy Castle Concurrency JAX-WS JMS JNI JavaOne Web Services XMPP

Blog Archive

- ► 2015 (7)
- ► 2014 (2)
- ► 2013 (27)
- ▼ 2012 (17)
    - ► December (1)
    - ▼ November (7)
        - Converting a Hex String to a Byte Array in Scala
        - Many Time Pad Attack - Crib Drag
        - Dependency Management - Avoid Cyclic Dependencies
        - Camel File Poller and JMS Consumer in Scala
        - XOR Hex-Encoded Strings in Scala
        - Exception Shielding Your Services
        - Scala Loops
    - ► October (9)

Reply

Enter Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)