LSB Algorithm

Welcome back. In this video, I'm going to introduce a specific steganography algorithm called the least significant bit algorithm. This algorithm makes use of digital images specifically, but can be used with other cover objects. If we first off consider a digital image which is uncompressed--

and we'll look specifically at 24-bit images.

In this instance, we have three different colour channels. And we've got one byte. So that's eight bits representing a value for the intensity of light related to red, green, and blue. So that's what an individual pixel is. It's created from that byte of each of those different colour channels.

And the least significant bit within a given byte is the right most bit. Now, this is because the change in the rightmost bit going to cause a minimal change to the naked eye. This can be seen if we look at, for example, taking a decimal representation of a byte. So that is getting the number value.

If we change the final bit from a 1 to a 0, then it's only going to change the actual value by 1, because 2 to power of 0 is 1. However, if we were to move to the left and take the left-most bit, that's the most significant bit, it's obviously going to change much more substantially.

If you then take that example and apply that to how the different colour intensity for each of the red, green, and blue changes, you can see that if you move more to the left, you can have more of an impact on the colour. You can see that within an example that I've shown here. So as you might imagine, the least significant bit algorithm takes the benefit of that very minor adjustment.

What we do is we go through the payload bit by bit. We take the next bit of the payload, and we have to then, compare that to the next we significant back within our cover image. If those two match, then we can leave it at is. If they don't match, we will have to flip that bit. So that involves taking the covered image, least significant bit--

if it doesn't match the payload that we're trying to hide, then you want to change it from a 0 to a 1, or 1 to a 0, depending on what your payload bit is.

And you start to iterate through the rest of the cover image, looking at the least significant bit of each byte, which is where you see the pixels values within that, and changing that to match the payload. So you're not necessarily always changing those least significant bit values. It's just that it has to match what your payload is.

So you iterate through, until effectively, you have completed. So it's a very straightforward type of algorithm to try and wrap your head around. There are a few little bits and pieces that can sometimes trip people up when they are trying to implement it. In particular, you need to know when to stop extracting.

So if you were trying to send a message to someone using a steganography program that you create, you need to know at what point does the message stop. And where does just the rest of the cover image start? You also need to know what's the extension of the file is that you're hiding. Obviously, if you're just working with strings, then that's fine. But if you are trying to hide digital files, then you would need to know the extension.

So you need to come up with a way of ensuring that is consistent and that you are able to determine those different values before being able to complete your program. There are ways of completing the bit twiddling or bit flipping within different programming languages, and shown on the screen here, are some examples of how you can complete this in Java.

So these are generally called bit manipulation operations. And Python has very similar operations to what's shown here. So we can see, for example, if we take an integer, which is our least significant bit, we can do and ampersand 0x1 to get the least significant bit. What that's doing is it's getting rid of everything up until the least significant bit.

We can also change the least significant bit to a 0 by using the complement function. So what we do is we take our byte, then we perform an and operation with the complement of 0x1. So this is effectively changing it to a whole bunch of 1, seven 1s, and then a 0 at the end. And then you're doing an and operation of that with your byte. So if you perform this and operation with the complement of 0x1, that means on your far right--

so your least significant bit is going to be if it's a 1, then you're performing an and operation with a 0.

Because remember it's the complement is 0x1. Then you're going to get a 0. And if it's a 0, then you're doing an and operation with a 0. You're still going to get a 0. And then obviously, you've got changing your least significant bit to a 1 value. And again, we're looking at bit

manipulation, this time, using the OR function. And you're doing that with 0x1. So again, it's looking at each of the bits in turn, within a byte, performing the OR operation with a whole bunch of 0s seven 0s, followed by 1 at the end.

Other operations which can be helpful in implementing this in Java includes shifting bits. So we've got shifting to the right. So you can move along by an interval value, so an integer. And we've got the same thing moving on to the left. We've also got an exclusive OR operation, which can be helpful. And that's the little, kind of hat symbol that we've got there. However, there are some problems with least significant bit.

In particular, it's very sequential by nature. So if someone was particularly determined, there are only a certain number of ways that you could implement the foundational, straightforward, least significant algorithm. So what can be done to try and remove that issue is to incorporate some form of randomness.

So using random number generators within Java and then having some form of seed. So it's pseudo-random, which means that we can replicate it if somebody has the seed value, and then using that to determine the next position, which is used to hide some information in. Obviously, that makes things a little bit more complicated than just doing it in a straightforward way.

So that's a very quick overview of the least significant bit algorithm I hope you've enjoyed this video, and I'll see you next time.