

Forum:[Hack Forums](#)

Thread: [Undoing an XOR function](#)

opua - January 4, 2024, 1:31 pm

Brand new here so be kind...

This is a decryption question that I have been trying to figure out for over a month now, but can't, so looking for help.

An encryption function returns the XOR sum of 1 to 40 binary constants. The constants are fixed and known and there are no repeats.

The function is passed a subset of these constants - anywhere from 1 to 40 of them- and returns the XOR sum.

Question: Given the XOR sum and the set of constants (but without knowing how many were combined), is it possible to tell which of the 40 constants are in the XOR sum?

I'm aware there are ways to brute force it by trying all 2^{40} combinations (or 2^{20} with a meet-in-the-middle attack) however, these take far more time and memory than the simple, low speed microcontroller (256 byte of RAM!), that can do it in less than a second. That's the part that's driving me crazy. If you look at the table of 40 numbers in binary format there are some patterns which makes me think the values were especially chosen to enable some magic "shortcut". If anyone thinks they can help I can post the table of binary numbers. Thanks.

webdev77 - January 4, 2024, 1:38 pm

Typically you would just XOR the key with the output to recover the original message. There is no highly efficient way to recover the key by design.

Tentco - January 4, 2024, 8:34 pm

(Jan 4, 2024 02:31 PM)opua Wrote: [\[CITE\]](#) Brand new here so be kind...

This is a decryption question that I have been trying to figure out for over a month now, but can't, so looking for help.

An encryption function returns the XOR sum of 1 to 40 binary constants. The constants are fixed and known and there are no repeats.

The function is passed a subset of these constants - anywhere from 1 to 40 of them- and returns the XOR sum.

Question: Given the XOR sum and the set of constants (but without knowing how many were combined), is it possible to tell which of the 40 constants are in the XOR sum?

I'm aware there are ways to brute force it by trying all 2^{40} combinations (or 2^{20} with a meet-in-the-middle attack) however, these take far more time and memory than the simple, low speed microcontroller (256 byte of RAM!), that can do it in less than a second. That's the part that's driving me crazy. If you look at the table of 40 numbers in binary format there are some patterns which makes me think the values were especially chosen to enable some magic "shortcut". If anyone thinks they can help I can post the table of binary numbers. Thanks.

I think that the information given and the fact that we do not know how to calculate the numbers

involved make it impossible to know which of the 40 constants are in the XOR sum. The problem of determining the exact combination of a given set is computationally complex, like solving a cryptographic problem.

Regarding the example you are describing, the numbers 1 through 40 in binary form have some interesting properties. Here are the two references:

0000 0000 0000 0001 (1) 0000 0000 0000 0010 (2) ... 0000 0000 0010 1000 (40) .

If you look closely, in a binary representation of 40 (101000), the first binary digit to the right is always 1, no matter how many we choose. Therefore, the XOR does not give us any additional information to reduce the numbers involved in the combination.

However, the 40 position has more patterns than the other binary digits, and that is why using a brute force or precomputed table (with all possible subsets) may not be possible with a low-speed microcontroller with Only 256 bytes of RAM.

opua - January 5, 2024, 6:43 am

(Jan 4, 2024 02:38 PM)webdev77 Wrote: [\[CITE\]](#) Typically you would just XOR the key with the output to recover the original message. There is no highly efficient way to recover the key by design. I have all 40 keys. I just don't know which were combined to produce the XOR sum - could be 2 could be 40, could be anything in between.

Your'e right - if you knew in advance only 2 numbers were combined, it's trivial (and fast) to XOR each of the 40 keys to the sum, one at a time and compare the result to the table of numbers. If the result equals one of the other numbers, you've found both. I just can't see how to extend that to the general case where you have no knowledge of the number of inputs.

webdev77 - January 5, 2024, 6:55 am

(Jan 5, 2024 07:43 AM)opua Wrote: [\[CITE\]](#) I have all 40 keys. I just don't know which were combined to produce the XOR sum - could be 2 could be 40, could be anything in between.

Your'e right - if you knew in advance only 2 numbers were combined, it's trivial (and fast) to XOR each of the 40 keys to the sum, one at a time and compare the result to the table of numbers. If the result equals one of the other numbers, you've found both. I just can't see how to extend that to the general case where you have no knowledge of the number of inputs.

Maybe I should have said if you don't already know which key, there is no simple way to recover. That's the point of using XOR. It is designed to be difficult to recover the key and message without knowing both up front.

opua - January 5, 2024, 7:44 am

(Jan 4, 2024 09:34 PM)Tentco Wrote: [\[CITE\]](#) I think that the information given and the fact that we do not know how to calculate the numbers involved make it impossible to know which of the 40 constants are in the XOR sum. The problem of determining the exact combination of a given set is computationally complex, like solving a cryptographic problem.

Except that it is being done - in under a second, with a low speed, 8-bit microcontroller with 256 bytes of RAM. I've confirmed this. If I hadn't seen it with my own eyes, I wouldn't believe it could be done as quickly as I'm seeing it done. Just mind boggling.

Also let me clarify - the table values aren't 1-40. There are 40 numbers - each (coincidentally?) 40 bits long. What I found interesting is the patterns that pop up in several places in the table where adjacent values are 40-bit rotations of the one before/after it. Elsewhere, they look random(?). This led me to suspect, as mentioned in my first post, that the 40 values are specifically chosen from the set of 2^{40} numbers to create a secret shortcut to decrypting. Why the patterns? Wouldn't numbers with a random distribution of 1's and 0's be more secure?

A little more information...the XOR function is used to encrypt a 40 bit message: each bit position in the message is associated with one of the table values. If a message bit is 1, it's table value is XOR'd to the sum. If it's a 0, it's value is not XOR'd.

It took a lot of experimentation to determine these 40 values. With them I can encrypt messages that are compatible with, and decipherable by, the receive system. This was my original goal and all I really NEED to do. How the receive system is unpacking the message so quickly has become a personal challenge.

The table of values might be useful to see but I notice that I can't just paste a jpg in. Any other way to do this without typing in 40, 40 bit numbers?

Suspect. - *January 31, 2024, 2:40 pm*

I highly doubt you are actually going to get the original input back infact i'd almost guarantee it. Xor was designed that way for that exact purpose.

When I've used this in projects of my own it was never for the intent of retrieving the original var/input.