

Re: Is this Possible?

Source: <http://www.derkeiler.com/Newsgroups/sci.crypt/2003-07/2033.html>

From: Myrddin Emrys (*myrddin_at_iosys.net*)

Date: 07/29/03

Date: 29 Jul 2003 10:03:40 -0700

clem <clem@numeral.com> wrote in message news:<54csfv0pt4kmd82saq9gq2g1ebct7u0llk@4ax.com>...

> On 26 Jun 2003 04:09:14 -0700, myrddin@iosys.net (Myrddin Emrys)

> wrote:

>

> <snip>

> >However, I believe I came up with a solution that does not require
> >greatly increasing the security without having to add 8 bytes of
> >garbage onto every single encrypted packet. I realized that rather
> >than encrypting the current IP:PORT only, I could encrypt that + the
> >list of previously encrypted IP:PORTS passed on. On the very first
> >hop, I could add 10 bytes, making the first IP:PORT encrypted cypher
> >secure. Then, every additional hop would need to add only 6 bytes...
> >so on hop 2, it would encrypt the first 16 (already encrypted) bytes +
> >6 more. Hop 3 would be $16 + 6 + 6 = 28$ bytes long, and so on.

> >

> >By building upon the entropy introduced from the very first packet, I
> >can maintain the randomness for all the packets. Not only that, but it
> >maintains my design goal of making it impossible to modify or reroute
> >a packet, without requiring local storage that 'remembers' every
> >packet that has passed through (like Gnutella does).

> >

> >First step: IP:PORTplustenbytesofentropy -> encryptedstream
> >Second step: IP:PORTencryptedstream -> longerencryptedstream
> >Third step: IP:PORTlongerencryptedstream ->
> >muchmuchlongerencryptedstream
> >etc.

>

> <snip>

> >Now that I think I've discovered a secure method of encoding the data,
> >what would be the most appropriate algorithm to use to encrypt an
> >arbitrary length byte array with a secret key?

<snip>

> From your example above you can XOR your 1st 6-byte value with the
> first 48-bits, XOR the 2nd 48-bits, and XOR 32-bits of the 3rd 48-bit
> value, before you will need to increment the counter to get another
> 128-bit XOR pad, of which you will use 16-bits to complete your 3rd

sci.crypt: Re: Is this Possible?

- > 48-bit IP:PORT, at which time you'll have 112-bits of pad left to use
- > or throw away, whichever you need.
- >
- > It gets even better because in CTR mode you are basically generating a
- > pad with which to XOR either the cleartext or ciphertext and as soon
- > as you have your 10-byte nonce you can start generating the pad for as
- > much as you want, using CPU cycles while you are waiting for data to
- > arrive. It is a very useful mode, ideally suited for your needs.

Unfortunately, not suited to my needs. Specifically, there are two issues in which this method fails. First, the entropy in the result is not shuffled... the 10 bytes of entropy that I introduce is not mixed in with the 6 bytes of IPPORT.

Second, it is not reversible without knowing WHERE in the CTR sequence the packet was generated. The return packets will be coming back in arbitrary order... I could get a packet I made 10 minutes ago or one I made 10 seconds ago at the same time. Without adding MORE data onto each packet, how am I to tell which value I need to XOR the two packets with and get the result back?

I need an encryption algorithm that uses a consistent secret key, takes arbitrary length input to be encrypted, and outputs a jumbled (not just XORed) result. In other words, if one bit of input is different, the same secret key needs to produce an entirely different output.

Now, it can be a bit simpler. For example, the input is not really arbitrary... it ranges from 16 bytes on up, in 6 byte increments. I can cap hops at an arbitrary 9 maximum, giving me 54 bytes + 10 of entropy = 64bytes maximum (likely, the practical limit will be lower, more like 5-7). I have no problem with coding 9 different encryption algorithms with 9 different secret keys, each designed to take a different length input... 16, 22, 28, etc.

But the important thing to note is that every time I take an input of a particular length (such as 28 bytes), I need to be able to encrypt it with the same key, over and over, and get extremely different results each time. This is because I will be getting these packet headers back at unknown intervals, and I need to be able to decrypt them in arbitrary order. Like I said in my original post, I do NOT want to maintain a table of 'this key matches this packet', so that means (as far as my limited cryptography knowledge extends) I am limited to a constant encryption key that cannot change from packet to packet.

I do greatly appreciate the information you all have posted, it has helped greatly.