

Re: ANNOUNCE: "jscryptor: client-side web page encryption" using JavaScript

Re: ANNOUNCE: "jscryptor: client-side web page encryption" using JavaScript

Source: <http://www.derkeiler.com/Newsgroups/sci.crypt/2006-10/msg00195.html>

- *From:* "d" <d@xxxxxxxxxx>
 - *Date:* 4 Oct 2006 11:36:05 -0700
-

David Wagner wrote:

d wrote:

Does this have any obvious problems? ('+' means concatenation)

```
r = Math.random(); // 0 <= r < 1.0; or say, millisecs since epoch
```

```
iv = h(k + p + r)
```

I would imagine that Javascript's `Math.random()` is not cryptographically secure. This makes me a little concerned. If it ever repeats, then you may have an information leakage (where sending the same plaintext twice is recognizable because it leads to re-using the same IV twice).

`Math.random()` is certainly NOT cryptographically secure: js uses 64bit floats (with 32bit ints for bitwise stuff), eg. `Math.random() = 0.44596719776789584` – that's still a lot of messages though. This is why I suggest the much less random but never(??) repeating timestamp – or even both.

Can you replace `Math.random()` with some cryptographically secure pseudorandom number generator? That would mean you don't have to worry about this.

Not really. User mouse event coordinates and timings at best. 'Quasi' random is the best we can do. This is why avoiding RNGs altogether has such appeal.

When using a hash function, always make sure that it's input is uniquely decodable. What does `r` become here? A variable-length string? If so, I suggest changing it to something like

Re: ANNOUNCE: "jscryptor: client-side web page encryption" using JavaScript

$$iv = h(k + p + ";" + r)$$

where ";" is guaranteed not to appear in the string-ization of r.

I don't understand 'uniquely decodable'. Surely any old 'non-repeatable/unpredictable' rubbish can be passed to H(), regardless of internal structure?

Also, if you use h() for any other purpose, I recommend ensuring that the inputs to h at any pair of call sites are from disjoint message spaces. One simple way to do that is to prepend a fixed string that is different for each call site or each purpose with which you use h(). In this case, that might become something like

$$iv = h("generate iv:" + k + p + ";" + r)$$

Is this just defensive coding, in case we 'accidentally' call h(k + p + r) somewhere else?

or even – as we don't really want to use an RNG (– though getting different ciphertexts (c) with the same plaintext (p) and key (k) seems an appealing idea...)

$$iv = h(k + p)$$
$$nonce = h(k) \wedge iv$$
$$c = nonce + e(k, p, iv) + hmac(k, p + iv)$$

Using h(k) to hide the iv seems to be the main problem, but I can't see how: the iv seems sufficiently hidden.

This doesn't make any sense to me. Why do you make a distinction between iv vs nonce? Why not just use "nonce = iv"?

On reflection, my terminology is unclear. The idea is that the nonce is sent/stored in the clear; the iv is encrypted – BECAUSE we don't use an RNG – being extracted from the nonce by XORing it with h(k). Hence the (sloppy) use of different terms.

Maybe this is unnecessary – especially if we decide to use a random iv instead – and maybe also provides another way to attack the key. I'm not sure.

Re: ANNOUNCE: "jscryptor: client-side web page encryption" using JavaScript

Re: ANNOUNCE: "jscryptor: client-side web page encryption" using JavaScript

Note that this has the weakness where sending the same plaintext twice is recognizable because it leads to re-using the same IV twice. Therefore, I wouldn't recommend this, if it is possible that plaintexts may repeat.

The nonce WILL repeat with the same p and k – so it's not a nonce – as with the iv, but is this a big deal? Some might see generating the same ciphertext from the same key and plaintext as a feature, not a bug...

A similar comment about making sure the inputs to h() are disjoint applies here.

I don't see any way to get around some kind of randomness. You need an IV that you can be sure won't repeat, if you want to avoid the weakness that sending the same plaintext twice leads to recognizably related ciphertexts.

Maybe it IS better to generate different ciphertexts (because we can) from the same key and plaintext on different calls to encrypt. So we definitely need some randomness in the iv...

So let's say we use:

```
iv = h(k + p + quasi_random_value + timestamp)
```

```
c = iv + e(k, p, iv) + hmac(k, p + iv) // iv in hmac to make it unique
```

This will offer SOME assurance (quite a lot in practice) of not accidentally re-using the iv.

This presumably gets round the need to encrypt the iv with said 'nonce'?

I was just trying to avoid RNGs!

.