Web-(developer)-oriented Crypto Algorithm

Web-(developer)-oriented Crypto Algorithm

Posted Aug 10, 2013 13:40 UTC (Sat) by **martin_vahi** (guest, #92302)
Parent article: Gräßlin: FLOSS after Prism: Privacy by Default

Dear lwn.net readers,

The purpose of this post is to advertise a one-time pad crypto algorithm derivative that has been specifically designed for web programming.

The advertised algorithm differs from the one-time pad by a fact that the one-time pad works with bits, but the derived algorithm uses single Unicode characters in stead of the single bits.

The specification resides at:
http://longterm.softf1.com/specifications/txor/index.html

Yours sincerely,
Martin.Vahi@softf1.com
Just another freelance software developer

P.S. Comments are also welcome. :-)

(Log in to post comments)

Web-(developer)-oriented Crypto Algorithm

Posted Aug 10, 2013 18:23 UTC (Sat) by **rmayr** (subscriber, #16880) [Link]

I am not sure if I see the applicability to web programming. How do you suggest web apps do the key management if no high-bandwidth out-of-band channel exists between the web server and the browser (or two browsers)?

Reply to this comment

An idea: If one does it, then one Should at Least try to do it Well

Posted Aug 12, 2013 9:46 UTC (Mon) by **martin_vahi** (guest, #92302) [Link]

The short answer is: TXOR and one-time pad are symmetric-key algorithms and therefore do not address the scenario that You described.

The motivation for rejecting public-key cryptography, regardless of crypto algorithm, is as follows:

Claim_1)

ciphertext=encryption(part_1_of_the_keypair,cleartext)=function_1(cleartext)
cleartext=decryption(part_2_of_the_keypair,ciphertext)=function_2(ciphertext)

That is to say:

cleartext=function_2(ciphertext) cleartext=function_2(function_1(cleartext))

id est

There is a bijection between cleartext and ciphertext and the function_2 is an inverse function of the function_1.

Claim_2)

In the case of all public key cryptography algorithms, one of the parts of the key pair is public. As the encryption/decryption algorithm is also public, one of the functions, function_1 or function_2 is always public.

It is only a matter of mathematical skill, how to construct function_1 or function_2, if it is firmly known that the publicly known function definitely has a reverse function.

Claim_3)

The fact that I'm dumb enough to fail the task, does not mean that others, professional mathematicians, who work on it as part of their day-job, may-be, but not only, in the NSA, can't solve it.

Reply to this comment

An idea: If one does it, then one Should at Least try to do it Well

Posted Aug 12, 2013 10:10 UTC (Mon) by **dlang** (guest, #313) [Link]

you seem to be missing the reason that public key algorithms have been used, namely the difficulty in distributing and managing symmetric-key algorithms (including one-time-pads)

public key algorithms work when function1 and function2 are both well known, but it is impractical to derive one key when you know the other. If you are not willing to trust that, it's trivial today to use symmetric keys (if you solve the key distribution problem)

Reply to this comment

An idea: If one does it, then one Should at Least try to do it Well

Posted Aug 12, 2013 10:23 UTC (Mon) by **rmayr** (subscriber, #16880) [Link]

I understand the difference between symmetric and asymmetric cryptography, and that is the reason for my question. OTR is the most difficult option in terms of key management. Not only do you need to keep the key secret during the exchange between involved parties (in contrast to asymmetric crypto), but is also going to have to be as long as the message itself, and must never be re-used again.

That is what confuses me: you are talking about web programming as the application area for your character-based OTR combination operator, but in web programming I see no way on how to realistically do the key management for anything remotely OTR-like (hence most/all claims of OTR for web applications are snake oil).

If you intend to reject asymmetric crypto, then I'd love to hear a better option for it (as we have known for quite a while that e.g. DH and RSA will be susceptible to quantum algorithms once we get a sufficient number of qbits in a stable configuration).

Btw, asymmetric crypto is not inherently less secure than symmetric crypto, as the decryption operation is always the inverse of encryption (we are talking about lossless encryption, I assume ;-)). It has just been studied a lot longer.

Rene

Reply to this comment

An idea: If one does it, then one Should at Least try to do it Well

Posted Aug 12, 2013 10:24 UTC (Mon) by **rmayr** (subscriber, #16880) [Link]

To clarify my btw, symmetric crypto has been studied for a lot longer than asymmetric. My original comment might be read the other way, sorry about that.

Reply to this comment

Web-(developer)-oriented Crypto Algorithm

Posted Aug 10, 2013 18:55 UTC (Sat) by **johill** (subscriber, #25196) [Link]

so unicode characters aren't stored as bits? ....

Reply to this comment

Web-(developer)-oriented Crypto Algorithm

Posted Aug 10, 2013 18:59 UTC (Sat) by **dps** (guest, #5725) [Link]

The proposal would seem to be about twice as expensive as xor for no increase in security given a genuinely random pad that is used only once. Generating, distributing and destroying pads could be a major problem.

c=m xor KDF(key), where KDF is a key derivation function that expands key the length of m, has already been proposed. One advantage of using xor is that the same code, or discrete logic, can be used for both encryption and decryption. The security of this construction is usually limited by the key size.

Reply to this comment

Web-(developer)-oriented Crypto Algorithm

Posted Aug 10, 2013 20:15 UTC (Sat) by **geofft** (subscriber, #59789) [Link]

You claim that the advantage of TXOR is that it's "computationally efficient to use within text processing oriented scripting languages". But this sort of processing goes back into arithmetic on the character values, and your algorithm has three arithmetic operations (add, subtract, and mod), whereas XOR only has one (XOR). So I expect it to be less computationally efficient than XOR.

Here's some quick anecdata in line with that, in Python 3:

```
>>> str1 = ''.join(chr(random.randint(0, 0x10ffff)) for i in range(4096))
>>> str2 = ''.join(chr(random.randint(0, 0x10ffff)) for i in range(4096))
>>> def xor(a, b):
...   return chr(ord(a) % ord(b))
...
>>> def txor(a, b):
...   return chr((ord(b) - ord(a) + 0x110000) % 0x110000)
...
>>> time.clock()
66.14
>>> for i in range(1000):
...   _ = ''.join(xor(c1, c2) for (c1, c2) in zip(str1, str2))
...
>>> time.clock()
69.41
>>> for i in range(1000):
...   _ = ''.join(txor(c1, c2) for (c1, c2) in zip(str1, str2))
...
>>> time.clock()
73.12
```

So 3.27 seconds for the XOR version, and 3.71 for the TXOR version. (I ran this a few times to avoid cold-cache effects and got the same values to within a few hundredths of a second.)

Besides, any serious use of encryption should be passing off the string as a binary string to an existing encryption library using an existing encryption mode, which should be doing all the XORs in C. There's generally no good reason to be implementing this yourself in a high-level language, and definitely no good reason to be using one-time pads directly.

Reply to this comment

What the TXOR is Good for

Posted Aug 12, 2013 9:19 UTC (Mon) by **martin_vahi** (guest, #92302) [Link]

Thank You for Your questions and Your answers!

The main problem that the TXOR solves is to allow a one-time pad like algorithm to be used without having to treat strings as bitstreams. Writing code that translates decrypted bitstreams back to strings takes considerable amount of labour and the only, or at least the main, benefit of the TXOR is to allow to skip the writing of the function decrypted_bitstream_2_programminglanguage_specific_string(..).

It seems to be a trend that the UTF-8 is the preferred format at server side, including databases, and UTF-16 is the default version in JavaScript. The UTF-8 and the UTF-16 are different binary formats that both use varying amounts of bytes for encoding Unicode characters. Part of the computational efficiency of the TXOR comes from the avoidance of the encoding format specific book-keeping that would be run within the decrypted_bitstream_2_programminglanguage_specific_string(..).

Given the task of writing the code that converts decrypted bitstreams back to strings that have the programming language specific/supported encoding, one might also say that the main benefit of the TXOR is to allow a one-time pad like algorithm to be used in situations, where the amount of available software development resources is very limited. Big corporations and NSA do not benefit from the TXOR that much, because they have loads of resources to write practically any extensive software that they please, but small software shops have to achieve the same result with considerably smaller amount of resources.

Thank You all again for
Your answers, questions and
for reading my comment. :-)

Reply to this comment

What the TXOR is Good for

Posted Aug 12, 2013 10:14 UTC (Mon) by **dlang** (guest, #313) [Link]

if you never have to interact with anything but your own software, you never have to convert strings from one representation to another (at which point TXOR doesn't save you anything)

But as soon as you need to deal with other software, you have to agree on a representation to use (at that point TXOR doesn't avoid needing to convert the data)

Java is stuck using UTF16 for legacy reasons, but everyone else has recognized that UTF8 is much more efficiant as a practical matter.

Reply to this comment

Why raw Bitstreams are Shaky

Posted Aug 12, 2013 16:09 UTC (Mon) by **martin_vahi** (guest, #92302) [Link]

Thank You for Your answer.

What about conversion between the UTF-8 (PHP, databases) and the UTF-16 (JavaScript)?

How to make sure that the decrypted_bitstream_2_string will not become jet-another-Y2K-UNIX-2038-problem after a Unicode standard update?

An attempt to write bitstream2string at one of the answers of the http://stackoverflow.com/questions/1240408/reading-bytes-from-a-javascript-string contains some Unicode specific constants, that probably have to be updated after a Unicode standard update.

Such library updates are not possible, either due to limited amount of development time or by mere lack of (legal) access to client's servers.

Reply to this comment

Why raw Bitstreams are Shaky

Posted Aug 12, 2013 21:17 UTC (Mon) by **dlang** (guest, #313) [Link]

if you have javascript talking directly to a database, something is wrong.

you should have javascript talking to a process on your server, and that process on your server will talk to a database.

That server process is going to have to convert the data from one format to another at some point, TXOR doesn't eliminate the need to do that.

If the UTF16 specifications change in a way that's incompatible with the exiting deployed data, the updated specifications are going to be ignored. The existing data is not going to be thrown away just because someone wants to change a spec. So your worries about how to handle such a problem are meaningless.

Reply to this comment

Why raw Bitstreams are Shaky

Posted Aug 23, 2013 2:03 UTC (Fri) by **elanthis** (guest, #6227) [Link]

> if you have javascript talking directly to a database, something is wrong.

One takes it you're not a NodeJS fan. :)

Reply to this comment