From:	Mouha, Nicky W. (IntlAssoc)
To:	Kelsey, John M. (Fed)
Subject:	Re: comments on proposal
Date:	Monday, August 7, 2017 2:01:22 PM

Hi John,

I hope that you're enjoying your vacation!

After I had sent my e-mail to you, Dustin and Ray came by to discuss my concerns.

During the brief discussion that we had, I explained that it is not clear to me what a "seed expansion" actually is, and isn't clear to me either what a "seed" is. For example, do we assume that a seed is secret? I raised this question in my previous e-mail, but Dustin and Ray weren't sure about the answer.

I think that the requirements should be stated first, and then we can use a scientific approach to evaluate these requirements for a given solution. Here, it seems to be that the solution comes first, but nobody seems to be clear about the requirements.

Furthermore, I understand the desire to use a NIST-standardized DRBG, but it doesn't seem that what you're proposing follows any NIST standard. I also have some concerns about the "buffering" technique. In particular, I don't know how we can make sure that the buffer is not released early (e.g., because of an optimization that is made by the compiler).

Given the explanation above, the security model is not clear to me. Therefore, it is difficult for me to say whether your security analysis is correct.

However, I explained to Dustin and Ray which approach we'd like to take to provide random numbers for a side-channel resistant implementation of lightweight cryptography. The random numbers can be provided by an API, so that fair benchmarks can be performed on every platform (the most efficient way to generate random bits will be different for every platform). Additionally, we can let the implementers provide their own function to generate random numbers, but they will have to provide it at the level of the API, and it may be replaced by another function to perform fair benchmarks on a particular platform (e.g., some platforms have a built-in random number generator). Dustin and Ray seemed to be interested in this approach.

Maybe we can discuss this further when you return from your holiday?

Regards,
Nicky

From: Kelsey, John M. (Fed)
Sent: Friday, August 4, 2017 3:31 PM
To: Mouha, Nicky W. (IntlAssoc)
Subject: comments on proposal

Nicky,

Hi! I'm on vacation, but I'm trying to answer a question for the PQC team. A few months back, some people asked about some kind of generic seed expansion algorithm, and I proposed more-or-less what you saw—either AES in CTR mode or KMAC. This isn't intended as a generic object for all NIST things forever, just something we suggest using in PQC algorithms if they need this functionality and don't know what else to do. I'm trying to write this up more precisely for Larry to implement and also to get comments from the community on whether this is what they need.

Can you tell me what your concerns were with the stuff I sent, either the schemes or the security analysis? I'm not trying to nail things down to the point of writing a paper, but I'd like to not make some dumb mistake.

For AES-CTR, the ways I can imagine breaking it are:

a. Cryptanalysis of AES.
b. Some kind of key-guessing attack or time/memory/data tradeoff.
c. The distinguisher based on the fact that AES is a permutation, so we never get colliding blocks from the same key.

Is there some other class of attack I'm missing?

Here's my reasoning so far. Please tell me if you see something I'm getting wrong!

For the counter-mode distinguisher, if we replaced AES-CTR's outputs with a an ideal random sequence of 2^{32} bytes, and then broke the output sequence into 2^{28} 128-bit blocks, we'd expect

$2^{55} 2^{-128} = 2^{-73}$ colliding blocks.

So we'd have something close to a 2^{-73} probability of a collision. In terms of distinguishers, that means that a distinguisher limited to 2^{32} bytes of data would have only about a 2^{-73}

Trying for multiple nonces, we get better distinguishers. If we request 2^{32} nonces under the same key, we get $2^{32} 2^{28} = 2^{60}$ 128-bit blocks encrypted under the same key. That is, our distinguisher would just look for collisions in the 128-bit blocks from the sequence, return "ideal" if it found one, and "real" otherwise. The distinguisher would have a probability of distinguishing of

P[success] = P[ideal] P[collision|ideal] + P[real]

where P[collision|ideal] ~= 2^{-73}.

Then, we'd have

P[success] = (1/2 2^{-73} + 1/2) = 1/2 + 2^{-74}.

Does this look right?

The bias grows with the square of the number of 128-bit blocks we see from the same key. So if someone gives us 2^{32} nonces, each generating 2^{28} 128-bit blocks, we'd have

2^{56} blocks, thus a probability of collision of about 2^{111} 2^{-128} = 2^{-17}.

Then, the bias would be

1/2 + 2^{-18}.

So a distinguisher that could be given up to 2^{64} bytes of outputs would have a 2^{-18} bias in its favor.

Am I missing something?

Similarly, it's pretty easy to see how to find an input that gives a desired value for a single block, as I described in my note, as long as you're willing to lose control over other blocks.

Thanks,

--John