While going through the codes for Dilithium, I found a non-portable assumption made about the C language.

Namely, it’s assumed when right-shifting signed integer types from the <stdint.h> header, higher-order bits will be filled with the sign of the original operand. AFAIK, nowhere in ISO 9899 was it mentioned such behavior is mandated on implementations.

Would the Crystals team clarify that what had been intended in the reference implementation?
(I am not part of the Dilithium team and I speak only for myself here.)

In ISO C, right-shifting a signed negative integer yields an "implementation-defined result". Crucially, this is not an "undefined behavior"; you will get a value and nothing else will break. It is true that the C standard does not mandate a specific result, but it does mandate the "implementation" (i.e. the C compiler) to do something and also to document that something.

In practice, most if not all compilers do an arithmetic shift, because that's what is most useful in such a case: the underlying CPU knows how to do an arithmetic shift and there is no other C operator that would produce it. This is not a risky bet. I would personally be OK with an implementation documenting that it relies on arithmetic shift for signed integers. In an ideal world, there would be a mechanism to detect proper support at compilation time and abort if right-shifting a signed negative value is not (always) an arithmetic shift; but in an ideal world, we would not be writing C code either.

If you want to write code which would work even if the compiler did not do an arithmetic shift, you can use something like this:

```c
static inline int32_t
arsh(int32_t x, int n)
{
    #if ROBUST_ARITHMETIC_RIGHT_SHIFT
        uint32_t y = (uint32_t)x >> n;
        y |= -(y & (0x80000000u >> n));
        return *(int32_t *)&y;
    #else
        return x >> n;
    #endif
}
```

Note that the cast from uint32_t to int32_t uses pointer aliasing, and this is guaranteed correct by the C standard because exact-width types like int32_t use two's complement and have no padding bits and no trap representation. That would not work with other types such as plain 'int', though.

--Thomas Pornin

Le dimanche 16 juin 2019 16:53:56 UTC-4, Danny Niu a écrit :

Hi, all.

I've posted this as an official comment on the website, but didn't get a reply obviously because it didn't reach the forum.

The question is simple, did the Crystals team intend arithmetic shift when shifting signed integer types from <stdint.h>?

Because that's not mandated as a standard behavior in ISO C right now.

Thanks.

--

You received this message because you are subscribed to the Google Groups "pqc-forum" group.
'Thomas Pornin' via pqc-forum <pqc-forum@list.nist.gov> wrote:

Dear Thomas, dear all,

> (I am not part of the Dilithium team and I speak only for myself
> here.)

The Dilithium team wouldn't have been able to explain it any better.
Yes, our code assumes that right shifting a signed integer is an arithmetic right shift.

All the best,

Peter

--
You received this message because you are subscribed to the Google Groups "pqc-forum" group.
To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.
To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/20190618053540.cpmyp7y6tptg3ago%40localhost.
Dear Dilithium team:

In the presentation of Dilithium at the first PQC standardization conference, whether better trade-offs on the already remarkable performance of Dilithium can be made is left as an interesting open question.

In cryptology eprint archive report 2018/1180, which is available from https://eprint.iacr.org/2018/1180 (with recent update by providing new parameters), we provide new insights in interpreting the design of Dilithium, in terms of key consensus previously proposed in the literature for key encapsulation mechanisms (KEM) and key exchange (KEX). Based on the deterministic version of the optimal key consensus with noise (OKCN) mechanism, originally developed in [JZ16] for KEM/KEX, we present \textit{signature from key consensus with noise} (SKCN), which could be viewed as generalization of Dilithium. The construction of SKCN is generic, modular and flexible.

We made much efforts to search new parameters for SKCN as well as Dilithium. We focus on parameters for about 128-bit pq-security, which we believe is the most important set of parameters for practice. On the recommended parameters, compared with Dilithium SKCN is more efficient both in computation and in bandwidth, while preserving the same level of post-quantum security. In addition, using the same routine of OKCN for both KEM/KEX and digital signature eases implementation and deployment in practice, and is useful to simplify the system complexity of lattice-based cryptography in general.

For new parameters of future version of Dilithium, we may suggest to use $\eta=2$ (i.e., the secret and the noise are from $[-2, 2]$). In this case, $q=1952257$ is for hardness of LWE of 128 pq-security. The new parameter set for SKCN (that is also applicable to Dilithium) is summarized as follows.

<table>
<thead>
<tr>
<th>SKCN</th>
<th>Dilithium</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>1952257</td>
</tr>
<tr>
<td>$n$</td>
<td>256</td>
</tr>
<tr>
<td>$(h,l)$</td>
<td>(5,4)</td>
</tr>
<tr>
<td>$\eta$</td>
<td>2</td>
</tr>
<tr>
<td>$</td>
<td>pk</td>
</tr>
<tr>
<td>$</td>
<td>sk</td>
</tr>
<tr>
<td>$</td>
<td>sig</td>
</tr>
<tr>
<td>Repetition</td>
<td>5.67</td>
</tr>
<tr>
<td>MLWE</td>
<td>128</td>
</tr>
<tr>
<td>MSIS</td>
<td>125</td>
</tr>
</tbody>
</table>

Best regards

Yunlei