From: Alperin-Sheriff, Jacob (Fed)
Sent: Friday, January 12, 2018 12:51 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: LAC

LAC Team:

Your submission seems to incorrectly claim a connection to worst-case hardness problems for ideal lattices. The connection only applies to error distributions (statistically close enough to) the distributions for alpha described in Section 3 of LPR10 (https://eprint.iacr.org/2012/230.pdf), and that requires alpha * q >= omega(\sqrt{log n).

The {-1,0,1} distributions for error you used are not statistically close to any such distribution.

—Jacob Alperin-Sheriff

From: Jacob Alperin-Sheriff <jacobmas@gmail.com>
Sent: Sunday, January 14, 2018 11:38 AM
To: pqc-comments
Cc: pqc-forum
Subject: OFFICIAL COMMENT: LAC

LAC Team:

I believe there is an attack that reduces the security of your scheme by (close to) a factor of 2 in the exponent from the estimates given in your submission.

The key point is that for n=2^k, k>=2

x^{n}+1 = (x^(n/2) + 91*x^(n/4) - 1)(x^(n/2) - 91*x^(n/4) - 1) mod 251
(note that x^n+1 is reducible modulo any prime p when k>=2, and for some choices it reduces to even smaller polynomials that may be more devastating).

The aggressive choice of error distribution {-1,0,1} used in LAC appears to then yield an attack that essentially combines the observations/framework in Section 3.2 of "How (Not) to Instantiate Ring-LWE" by
Peikert https://eprint.iacr.org/2016/351.pdf with algorithms for solving SVP.

Let g=x^(n/2)+91*x^(n/4)-1, h=x^(n/2)-91*x^(n/4)-1, and let Id_g, Id_h be the associated ideals generated by g and 251 (respectively, h and 251)

Then for e a degree at most n polynomial with coefficients in (-1,0,1), the coefficients of e reduced modulo the ideal generated by g and 251

(i.e. e mod Id_g)

will all be of the form

+-{0,1,2} +- {0,91}

(and similarly for h)

Let (a,b=s*a+e (mod R_q) be the public key for LAC.

Now, let a_g = a mod Id_g, b_g = b mod Id_g
a_h = a mod Id_h, b_h = b mod Id_h

Then I believe (I haven't checked yet with easily feasible n (like n=64), will hopefully do next week) that the shortest vector x=(x_1,x_2,x_3,x_4)

solution to the (ring)-I-SIS problem

[a_g, 91*a_g, 1, 91]x = b_g

should be such that x_1+91*x_2 = s (mod Id_g), x_3+91*x_4=e (mod Id_g), and similarly for the case of h.

Assuming this is true, recovering s and e reduces to solving SVP for 2 instances of (n/2) dimensional lattices and then Chinese remainder theorem-ing to recover s and e.

I will try to implement this some time this week in Sage for an easily feasible n to see if I'm correct about the form of the shortest vector.

--
-Jacob Alperin-Sheriff

From: Jacob Alperin-Sheriff <jacobmas@gmail.com>
Sent: Sunday, January 14, 2018 2:40 PM
To: pqc-comments
Cc: pqc-forum
Subject: Re: OFFICIAL COMMENT: LAC

Hi everybody,

Vadim Lyubashevsky pointed out that I made a stupid mistake here about the dimensions of the resulting lattice so disregard everything above except (if it helps) the form of the error modulo the ideals.

On Jan 14, 2018 11:38 AM, "Jacob Alperin-Sheriff" <jacobmas@gmail.com> wrote:

> LAC Team:
>
> I believe there is an attack that reduces the security of your scheme by (close to) a factor of 2 in the exponent from the estimates given in your submission.
>
> The key point is that for n=2^k, k>=2
>
> x^{n}+1 = (x^(n/2) + 91*x^(n/4) - 1)(x^(n/2) - 91*x^(n/4) - 1) mod 251
> (note that x^n+1 is reducible modulo any prime p when k>=2, and for some choices it reduces to even smaller polynomials that may be more devastating).
>
> The aggressive choice of error distribution {-1,0,1} used in LAC appears to then yield an attack that essentially combines the observations/framework in Section 3.2 of "How (Not) to Instantiate Ring-LWE" by
> Peikert https://eprint.iacr.org/2016/351.pdf with algorithms for solving SVP.
>
> Let g=x^(n/2)+91*x^(n/4)-1, h=x^(n/2)-91*x^(n/4)-1, and let Id_g, Id_h be the associated ideals generated by g and 251 (respectively, h and 251)
>
> Then for e a degree at most n polynomial with coefficients in (-1,0,1), the coefficients of e reduced modulo the ideal generated by g and 251
>
> (i.e. e mod Id_g)
>
> will all be of the form
>
> +-{0,1,2} +- {0,91}
>
> (and similarly for h)
>
> Let (a,b=s*a+e (mod R_q) be the public key for LAC.
>
> Now, let a_g = a mod Id_g, b_g = b mod Id_g
> a_h = a mod Id_h, b_h = b mod Id_h
>
> Then I believe (I haven't checked yet with easily feasible n (like n=64), will hopefully do next week) that the shortest vector x=(x_1,x_2,x_3,x_4)

From: Alperin-Sheriff, Jacob (Fed)
Sent: Thursday, January 18, 2018 10:59 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: LAC

Hi all,

New idea on LAC.

Remember, the ring is x^512+1, q=251, error distribution is +- with probability 1/4, 0 with probability 1/2 in LAC.

I noticed (well I had the general idea and then used Sage to check all the possible multipliers) that if you multiply s and e by 11 (so the error distribution becomes \pm 11 each with probability 1/4, 0 with probability 1/2) then each of the error coefficients e_i will be at most +- 25 after reducing modulo g=x^(n/2)+91*x^(n/4)-1, specifically in the distribution

[0, 3, 8, 11, 14, 19, 22, 25, 226, 229, 232, 237, 240, 243, 248], and similarly for s
Moreover, the smaller values are more likely, for each individual coefficient they are distributed with probability {0: 0.111, 3: 0.111, 8: 0.074, 11: 0.074, 14: 0.074, 19: 0.037, 22: 0.037, 25: 0.037, 226: 0.037, 229: 0.037, 232: 0.037, 237: 0.074, 240: 0.074, 243: 0.074, 248: 0.111}

I haven't attempted to prove anything yet for the values of n that are of interest, but exhaustively computing for w=8 reveals that with probability at least .5, the Euclidean norm of the coefficients of e mod g [e being the error], will be at most 21,

For 5000 random samples in the w=512 case, the Euclidean norm
of the coefficient vector of ((11*e) mod g) was 207 or less with probability .5.

So, assuming I didn't make another stupid mistake again, the question is then whether this is going to be short enough that z=(11*s (mod g) ,11*e (mod g),-1)
will be the shortest solution of

Az=0

where A= [(a)mod g | 1 | (11*b) mod g] is a 256 x 513

By the above bound on the coefficient vector (11*e), we can get the (11*s (mod g), 11*e (mod g), -1) should be at most 292.75 with probability at least .25.

Lemma 5.2 in Micciancio-Regev's "Worst-case to Average-case Reductions based on Gaussian Measures"
https://pdfs.semanticscholar.org/9935/ac933507b63a2d6eac41c87aa4ab4835be52.pdf
gives that a solution **has** to exist of norm at least 356.9 in this kind of a SIS instance, but I'm not so up on how tight that bound is and if we can hope that (11*s (mod g), 11 * e(mod g), -1) will be the shortest vector.

I did try something in Sage for smaller values of n, but the results were inconclusive (BKZ didn't find the desired vector, but the desired vector was also of smaller norm than anything in the BKZ-reduced basis, suggesting that I'm doing something wrong). Code is included below; help would be appreciated.

```
#!/usr/bin/env sage

import sys
from sage.all import *

def sample_single_error():
    t=randint(0,3)
    if t==0:
        return -1
    elif t<3:
        return 0
    return 1

def sample_noise(S):
    n=S.degree()
    ret_lst=[sample_single_error() for i in range(n)]
    return S(ret_lst)

def my_gen_lattice2(n=4, q=11, seed=None,
                quotient=None, dual=False, ntl=False, lattice=False,t=5):
    """
    This is a modification of the code for the gen_lattice function from Sage

    Randomness can be set either with ``seed``, or by using
    :func:`sage.misc.randstate.set_random_seed`.

    INPUT:

    - ``type`` -- one of the following strings
        - ``'cyclotomic'`` -- Special case of ideal. Allows for
          efficient processing proposed by [LM2006]_.
    - ``n`` -- Determinant size, primal:`det(L) = q^n`, dual:`det(L) = q^{m-n}`.
      For ideal lattices this is also the degree of the quotient polynomial.
    - ``m`` -- Lattice dimension, `L \subseteq Z^m`.
    - ``q`` -- Coefficient size, `q-Z^m \subseteq L`.
    - ``t`` -- BKZ Block Size
    - ``seed`` -- Randomness seed.
    - ``quotient`` -- For the type ideal, this determines the quotient
      polynomial. Ignored for all other types.
    - ``dual`` -- Set this flag if you want a basis for `q-dual(L)`, for example
      for Regev's LWE bases [Reg2005]_.
    - ``ntl`` -- Set this flag if you want the lattice basis in NTL readable
      format.
    - ``lattice`` -- Set this flag if you want a
```

```
  :class:`FreeModule_submodule_with_basis_integer` object instead
  of an integer matrix representing the basis.

OUTPUT: ``B`` a unique size-reduced triangular (primal: lower_left,
  dual: lower_right) basis of row vectors for the lattice in question.

EXAMPLES:



Cyclotomic bases with n=2^k are SWIFFT bases::

    sage: sage.crypto.gen_lattice(type='cyclotomic', seed=42)
    [11  0  0  0  0  0  0  0]
    [ 0 11  0  0  0  0  0  0]
    [ 0  0 11  0  0  0  0  0]
    [ 0  0  0 11  0  0  0  0]
    [ 4 -2 -3 -3  1  0  0  0]
    [ 3  4 -2 -3  0  1  0  0]
    [ 3  3  4 -2  0  0  1  0]
    [ 2  3  3  4  0  0  0  1]

Dual modular bases are related to Regev's famous public-key
encryption [Reg2005]_::

    sage: sage.crypto.gen_lattice(type='modular', m=10, seed=42, dual=True)
    [ 0  0  0  0  0  0  0  0  0 11]
    [ 0  0  0  0  0  0  0  0 11  0]
    [ 0  0  0  0  0  0  0 11  0  0]
    [ 0  0  0  0  0  0 11  0  0  0]
    [ 0  0  0  0  0 11  0  0  0  0]
    [ 0  0  0  0 11  0  0  0  0  0]
    [ 0  0  0  1 -5 -2 -1  1 -3  5]
    [ 0  0  1  0 -3  4  1  4 -3 -2]
    [ 0  1  0  0 -4  5 -3  3  5  3]
    [ 1  0  0  0 -2 -1  4  2  5  4]


"""
from sage.rings.finite_rings.integer_mod_ring import IntegerModRing
from sage.matrix.constructor import identity_matrix, block_matrix
from sage.matrix.matrix_space import MatrixSpace
from sage.rings.integer_ring import IntegerRing
if seed is not None:
    from sage.misc.randstate import set_random_seed
    set_random_seed(seed)



ZZ = IntegerRing()
ZZ_q = IntegerModRing(q)

A = identity_matrix(ZZ_q, n/2)


from sage.arith.all import euler_phi
from sage.misc.functional import cyclotomic_polynomial

# we assume that n+1 <= min( euler_phi^{-1}(n) ) <= 2*n
found = False
for k in range(2*n,n,-1):
    if euler_phi(k) == n:
```

```
            found = True
            break
        if not found:
            raise ValueError("cyclotomic bases require that n "
                             "is an image of Euler's totient function")
    R = ZZ_q['x'].quotient(cyclotomic_polynomial(k, 'x'), 'x')
    g=x**(n/2)+91*x**(n/4)-1
    T=ZZ_q['x'].quotient(x**(n/2)+91*x**(n/4)-1)


    a_pol=R.random_element()

#    for i in range(m//n):
#        print("i={0}".format(i))
#        A = A.stack(R.random_element().matrix())
    s_pol=sample_noise(R)
    e_pol=sample_noise(R)

    s_pol2=T((11*s_pol).list())
    e_pol2=T((11*e_pol).list())

    Z_mat=s_pol2.matrix().augment(e_pol2.matrix())
    Z_mattop=Z_mat[0:1].augment(matrix(1,1,ZZ.one()*-1))


    b_pol=(a_pol*s_pol+e_pol)*11
    print("s_pol={0}\ne_pol={1}".format((s_pol*11).list(),(e_pol*11).list()))

    a_pol2 = T(a_pol.list())# % S(g.list())
    b_pol2 = T(b_pol.list())# % S(g.list())
#    print("a={0}\nb={1}".format(a_pol,b_pol))

    A=a_pol2.matrix().stack(A)
    A=A.stack(b_pol2.matrix()[0:1])

    print("{0}\n".format(A))
    # switch from representatives 0,...,(q-1) to (1-q)/2,....,(q-1)/2
    def minrepnegative(a):
        if abs(a-q) < abs(a): return (a-q)*-1
        else: return a*-1
    def minrep(a):
        if abs(a-q) < abs(a): return (a-q)
        else: return a
    A_neg = A[0:(n/2)].lift().apply_map(minrepnegative)
    b_neg= A[(n):(n+1)].lift().apply_map(minrepnegative)
    Z_fixed=Z_mattop.lift().apply_map(minrep)
    print("Z_fixed={0}\n||Z_fixed||={1}".format(Z_fixed,float(Z_fixed[0].norm())))
    print('Z_fixed*A={0}\n\n'.format(Z_fixed*A))

    #    print("{0}\n".format(A_neg))
    B=block_matrix([[ZZ(q), ZZ.zero(),ZZ.zero()],[ZZ.one(),A_neg,ZZ.zero()
],[ZZ.zero(),b_neg,ZZ.one()]],
                       subdivide=False)
    #print("B=\n{0}".format(B))
    #print("B*A=\n{0}\n\n".format(B*A))

    B_BKZ=B.BKZ(block_size=t,proof=True)

    print("B_BKZ[0]=\n{0}\n".format(B_BKZ[0]))

    for i in range(0,n+1):
        print('{0}: {1}'.format(i,1.*float(B_BKZ[i].norm())))
```

```
if ntl and lattice:
    raise ValueError("Cannot specify ntl=True and lattice=True ")
if ntl:
    return B._ntl_()
elif lattice:
    from sage.modules.free_module_integer import IntegerLattice
    return IntegerLattice(B)
else:
    return B
```

From:	sd lattice
To:	pqc-comments
Cc:	pqc-forum@list.nist.gov
Subject:	OFFICIAL COMMENT: LAC
Date:	Tuesday, January 23, 2018 10:32:58 AM
Attachments:	norms.pdf testpolynomialring.py

Dear Alperin-Sheriff, Jacob

Thank you very much for your comments on LAC.

1: About the connection to worst-case hardness problems: In the design of LAC we considered the requirement that alpha*q>=omega(\sqrt{log n}). The parameters of LAC do not satisfy this requirement directly. However, we notice that, according to the modulus reduction result proposed in [1], varying the dimension n and the modulus q individually while keeping nlogq fixed essentially preserves the hardness of LWE. We estimate the concrete hardness of the RLWE problem by using the method given in [2]. The result shows that the modulus reduction result also works for the hardness of RLWE. Concretely, for the parameter of LAC-128, where q=251,n=512, sigma=1/sqrt(2), the hardness is:

primal attack:
classical cost= 148
quantum cost= 135
dual attack:
classical cost= 147
quantum cost= 133

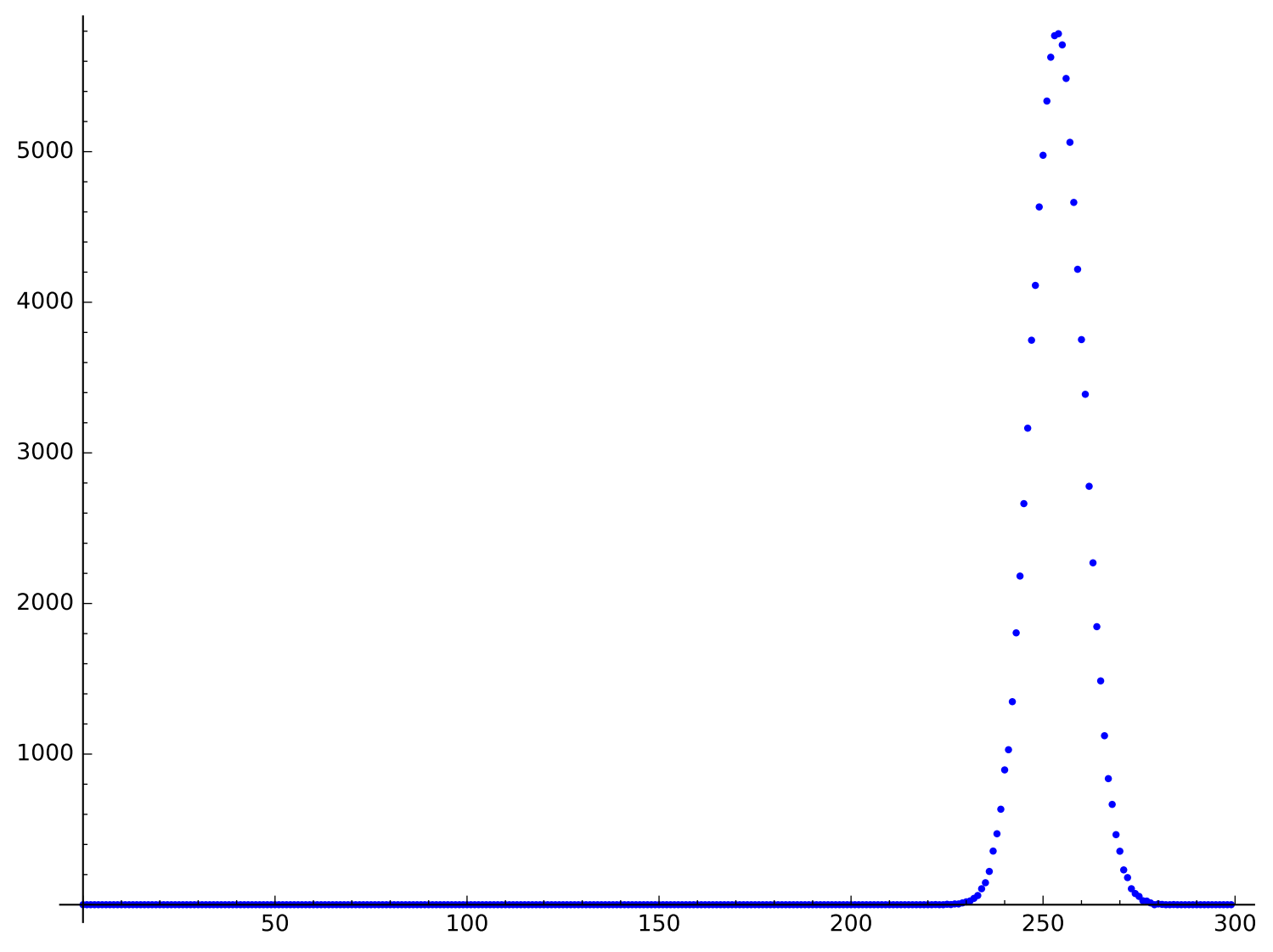
when we set q=251*251, n=256, sigma= 1/sqrt(2) * 251, the hardness is:
primal attack:
classical cost= 156
quantum cost= 141
dual attack:
classical cost= 154
quantum cost= 139
Note that, during the varying of n and q, we need to keep alpha=sigma*sqrt(2*pi)/q fixed, so we set sigma= 1/sqrt(2) * 251. In this case, alpha*q=1/sqrt(2) * 251>omega(\sqrt{log 256}).

2: About the modulo attack. This is really an interesting observation. We used Sage to check this idea, for 100000 random samples, our result shows that the length of z=[11*s mod g, 11*e mod g, -1] is an Gaussian distribution with mean 253.59 and standard deviation 6.9. The sage code and the figure of the distribution can be found in the attachment. According to Micciancio and Regev's "Lattice Based Cryptography",:
lambda_1(\Lambda_q^{\perp})=sqrt(513/(2*pi*e))*251^{256/513}=86.36. It is clear that, the length of vector z is much longer than lambda_1, and the BKZ algorithm cannot be used to find z.

Note that, the bound of Lemma 5.2 in Micciancio-Regev's "Worst-case to Average-case Reductions based on Gaussian Measures" is not tight enough. Concretely, the bound in this lemma is sqrt(513)*251^{256/513}=356.9, which is much larger than 86.36.

[1]Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, Damien Stehlé: **Classical hardness of learning with errors.** STOC 2013: 575-584
[2]Erdem Alkim, Léo Ducas, Thomas Pöppelmann, Peter Schwabe: **Post-quantum Key Exchange - A New Hope.** USENIX Security Symposium 2016: 327-343
[3] https://cims.nyu.edu/~regev/papers/pqc.pdf

Best Wishes.
LAC Team


5000
4000
3000
2000
1000
50
100
150
200
250
300

From: Alperin-Sheriff, Jacob (Fed)
To: sd lattice; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: OFFICIAL COMMENT: LAC
Date: Tuesday, January 23, 2018 10:38:34 AM

---

1. [1] doesn't work for ring-LWE.

2. You're right that my idea definitely doesn't work as written, I've since confirmed that the shortest vectors modulo the ideal are several times shorter than the induced (z,e,-1) vector (it may simply never be helpful in any ring, I'm still investigating that)

Thanks for the response.

From: sd lattice <luxianhui@outlook.com>
Date: Tuesday, January 23, 2018 at 10:33 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>
Subject: OFFICIAL COMMENT: LAC

Dear Alperin-Sheriff, Jacob

Thank you very much for your comments on LAC.

1: About the connection to worst-case hardness problems: In the design of LAC we considered the requirement that alpha*q>=omega(\sqrt{log n}). The parameters of LAC do not satisfy this requirement directly. However, we notice that, according to the modulus reduction result proposed in [1], varying the dimension n and the modulus q individually while keeping nlogq fixed essentially preserves the hardness of LWE. We estimate the concrete hardness of the RLWE problem by using the method given in [2]. The result shows that the modulus reduction result also works for the hardness of RLWE. Concretely, for the parameter of LAC-128, where q=251,n=512, sigma=1/sqrt(2), the hardness is:

primal attack:
classical cost= 148
quantum cost= 135
dual attack:
classical cost= 147
quantum cost= 133

when we set q=251*251, n=256, sigma= 1/sqrt(2) * 251, the hardness is:
primal attack:
classical cost= 156
quantum cost= 141
dual attack:
classical cost= 154
quantum cost= 139
Note that, during the varying of n and q, we need to keep alpha=sigma*sqrt(2*pi)/q fixed, so we set sigma= 1/sqrt(2) * 251. In this case, alpha*q=1/sqrt(2) * 251>omega(\sqrt{log 256}).

2: About the modulo attack. This is really an interesting observation. We used Sage to check this idea, for 100000 random samples, our result shows that the length of z=[11*s mod g, 11*e mod g, -1] is an Gaussian distribution with mean 253.59 and standard deviation 6.9. The sage code and the figure of the distribution can be found in the attachment. According to Micciancio and Regev's "Lattice Based Cryptography",:
lambda_1(\Lambda_q^{\perp})=sqrt(513/(2*pi*e))*251^{256/513}=86.36. It is clear that, the length

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 23, 2018 11:03 AM
To: Alperin-Sheriff, Jacob (Fed)
Cc: pqc-comments; pqc-forum@list.nist.gov; sd lattice
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Tue, Jan 23, 2018 at 10:38 AM Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov> wrote:

> 1. [1] doesn't work for ring-LWE.

To be clear, modulus switching does work for both LWE and Ring-LWE (among other problems), as claimed below. But the results from [1] do not say anything about (Ring-)LWE with *binary errors*, which is apparently what LAC uses. (Gaussians are the only error distributions considered in [1].) It does not appear that LAC is supported asymptotically by any known worst-case hardness theorem.

Sincerely yours in cryptography, Chris

From: sd lattice <luxianhui@outlook.com>
Date: Tuesday, January 23, 2018 at 10:33 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>
Subject: OFFICIAL COMMENT: LAC

Dear Alperin-Sheriff, Jacob

Thank you very much for your comments on LAC.

1: About the connection to worst-case hardness problems: In the design of LAC we considered the requirement that alpha*q>=omega(\sqrt{log n}). The parameters of LAC do not satisfy this requirement directly. However, we notice that, according to the modulus reduction result proposed in [1], varying the dimension n and the modulus q individually while keeping nlogq fixed essentially preserves the hardness of LWE. We estimate the concrete hardness of the RLWE problem by using the method given in [2]. The result shows that the modulus reduction result also works for the hardness of RLWE. Concretely, for the parameter of LAC-128, where q=251,n=512, sigma=1/sqrt(2), the hardness is:

primal attack:

classical cost= 148

From: Alperin-Sheriff, Jacob (Fed)
Sent: Tuesday, January 23, 2018 11:06 AM
To: Christopher J Peikert
Cc: pqc-comments; pqc-forum@list.nist.gov; sd lattice
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

LAC uses +-1 with probability 1/4 each, 0 with probability 1/2 so technically not binary. You're right that a better answer would have been things like "Hardness of SIS and LWE with Small Parameters
" https://eprint.iacr.org/2013/069.pdf don't apply to ring-LWE (although it could easily be adapted for the LAC error distribution for the general LWE case).

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Date: Tuesday, January 23, 2018 at 11:02 AM
To: "Alperin-Sheriff, Jacob (Fed)" <jacob.alperin-sheriff@nist.gov>
Cc: pqc-comments <pqc-comments@nist.gov>, "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>, sd lattice <luxianhui@outlook.com>
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Tue, Jan 23, 2018 at 10:38 AM Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov> wrote:

> 1. [1] doesn't work for ring-LWE.

To be clear, modulus switching does work for both LWE and Ring-LWE (among other problems), as claimed below. But the results from [1] do not say anything about (Ring-)LWE with *binary errors*, which is apparently what LAC uses. (Gaussians are the only error distributions considered in [1].) It does not appear that LAC is supported asymptotically by any known worst-case hardness theorem.

Sincerely yours in cryptography, Chris

From: sd lattice <luxianhui@outlook.com>
Date: Tuesday, January 23, 2018 at 10:33 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>
Subject: OFFICIAL COMMENT: LAC

Dear Alperin-Sheriff, Jacob

> Thank you very much for your comments on LAC.

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 23, 2018 11:16 AM
To: Alperin-Sheriff, Jacob (Fed)
Cc: pqc-comments; pqc-forum@list.nist.gov; sd lattice
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Tue, Jan 23, 2018 at 11:05 AM Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov> wrote:

> LAC uses +-1 with probability 1/4 each, 0 with probability 1/2 so technically not binary. You're right that a better answer would have been things like "Hardness of SIS and LWE with Small Parameters
>
> " https://eprint.iacr.org/2013/069.pdf don't apply to ring-LWE (although it could easily be adapted for the LAC error distribution for the general LWE case).

Yes, that paper gets closer to the mark for plain LWE with binary (or ternary) errors. But to prevent any confusion: it would not support the plain-LWE analogue of LAC either, because the number of samples is limited to smaller than what LAC reveals to the attacker.

Sincerely yours in cryptography, Chris

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Date: Tuesday, January 23, 2018 at 11:02 AM
To: "Alperin-Sheriff, Jacob (Fed)" <jacob.alperin-sheriff@nist.gov>
Cc: pqc-comments <pqc-comments@nist.gov>, "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>, sd lattice <luxianhui@outlook.com>
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Tue, Jan 23, 2018 at 10:38 AM Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov> wrote:

> 1. [1] doesn't work for ring-LWE.

To be clear, modulus switching does work for both LWE and Ring-LWE (among other problems), as claimed below. But the results from [1] do not say anything about (Ring-)LWE with *binary errors*, which is apparently what LAC uses. (Gaussians are the only error distributions considered in [1].) It does not appear that LAC is supported asymptotically by any known worst-case hardness theorem.

Sincerely yours in cryptography, Chris

From: Alperin-Sheriff, Jacob (Fed)
Sent: Thursday, February 15, 2018 2:23 PM
To: Christopher J Peikert
Cc: pqc-comments; pqc-forum@list.nist.gov; sd lattice
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

Coming back to this point "Yes, [Hardness of SIS and LWE with Small Parameters] gets closer to the mark for plain LWE with binary (or ternary) errors. But to prevent any confusion: it would not support the plain-LWE analogue of LAC either, because the number of samples is limited to smaller than what LAC reveals to the attacker."

If I'm understanding everything properly, the paper doesn't say anything at all about the plain LWE analogue of LAC and more generally, the public key in any ring-LWE (or plain LWE) scheme using the Lindner-Peikert formulation of primal Regev, because that paper requires m >= n + omega(log n) in order to use the Micciancio-Mol SIS equivalence, and we have m=n for LAC. Is this correct?

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Date: Tuesday, January 23, 2018 at 11:16 AM
To: "Alperin-Sheriff, Jacob (Fed)" <jacob.alperin-sheriff@nist.gov>
Cc: pqc-comments <pqc-comments@nist.gov>, "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>, sd lattice <luxianhui@outlook.com>
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Tue, Jan 23, 2018 at 11:05 AM Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov> wrote:

> LAC uses +-1 with probability 1/4 each, 0 with probability 1/2 so technically not binary. You're right that a better answer would have been things like "Hardness of SIS and LWE with Small Parameters
>
> " https://eprint.iacr.org/2013/069.pdf don't apply to ring-LWE (although it could easily be adapted for the LAC error distribution for the general LWE case).

Yes, that paper gets closer to the mark for plain LWE with binary (or ternary) errors. But to prevent any confusion: it would not support the plain-LWE analogue of LAC either, because the number of samples is limited to smaller than what LAC reveals to the attacker.

Sincerely yours in cryptography, Chris

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Date: Tuesday, January 23, 2018 at 11:02 AM
To: "Alperin-Sheriff, Jacob (Fed)" <jacob.alperin-sheriff@nist.gov>
Cc: pqc-comments <pqc-comments@nist.gov>, "pqc-forum@list.nist.gov" <pqc-forum@list.nist.gov>, sd lattice <luxianhui@outlook.com>
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

From: Mike Hamburg <mike@shiftleft.org>
Sent: Thursday, February 15, 2018 8:39 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: LAC

Hello LAC authors,

I noticed that the LAC reference and optimized code submissions take variable time, at least for the BCH decoding step.

How difficult is it to fix this to run in constant time? Do you plan to implement a constant-time decoding algorithm so that we can test its performance?

Thanks,
— Mike Hamburg

From: Alperin-Sheriff, Jacob (Fed)
Sent: Friday, February 16, 2018 8:54 AM
To: Mike Hamburg; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: OFFICIAL COMMENT: LAC

Mike,

I had also noticed that the BCH decoding algorithms are written in a variable time manner, but I ended up forgetting to/putting off mention it here after I did some testing of the times/cycles taken by the BCH decoding step and it didn't seem to vary depending on the number of errors it had to decode.

This doesn't mean I or NIST consider the code or the algorithm "okay" (meaning of no potential concern), just that I wasn't able to find a way that it leaked any useful information and so it slipped my mind.

Did you find something different?

On 2/15/18, 8:39 PM, "Mike Hamburg" <mike@shiftleft.org> wrote:

> Hello LAC authors,
>
> I noticed that the LAC reference and optimized code submissions take variable time, at least for the BCH decoding step.
>
> How difficult is it to fix this to run in constant time? Do you plan to implement a constant-time decoding algorithm so that we can test its performance?
>
> Thanks,
> — Mike Hamburg

From: Alperin-Sheriff, Jacob (Fed)
Sent: Friday, February 16, 2018 10:07 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: LAC

LAC authors:

I believe LAC256 falls significantly short of 256 bits of security under the CCA attack model (that allows up to 2^64 chosen ciphertext queries [see pg 15 of the CFP https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf]).

A simple calculation of binom(2048,1024+310)/(2**2048) gives that (modeling the hash function as a random oracle) for a message chosen uniformly at random, the vector (r,e_1) in LAC.CPA.Enc (which in the LAC.CCA.Enc gets its "randomness" from the message) will have Hamming weight of at least 1024+310 with probability at least 2^{-143.3}, so with high probability, 2^{-210} or so hash function evaluations should yield at least 2^{64} messages giving such high Hamming weights for (r,e_1) in LAC.CPA.Enc.

After doing this 2^{210}*cost(hash) precomputation (which presumably costs signficantly less than 256 bits), we should be able to recover almost any key with 2^{64} chosen ciphertext queries.

For messages that yield Hamming weights of (r,e_1) of 1024+310, testing gives that each bit fails to be correct with probability 2^{-6.0496}, so a simple Python script that enough bits (56 or more) to cause failure on the entire message will occur with probability ~ 2^{-50.51}. As a result, over 2^{64} message queries we should expect with high probability to get far more than the necessary number of failures to recover s based on a rough application of the analysis in https://eprint.iacr.org/2016/085.pdf

—Jacob Alperin-Sheriff

From: Martin Tomlinson <mt@post-quantum.com>
Sent: Friday, February 16, 2018 10:40 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: LAC
Attachments: signature.asc

Dear LAC authors,

Although you decided to use BCH codes as the error correcting code in LAC, a better choice is to use binary Goppa codes since the maximum code length of these codes is 2^m and not 2^m-1. There is an additional information bit that can be corrected and no messy padding. A BCH decoder can be used straightforwardly to implement the error correction decoder for Goppa codes as first pointed out by Charles Retter in his 1975 paper: "Decoding Goppa codes with a BCH decoder.", IEEE Transactions on Information Theory, 21(1):112–112.

One significant advantage is that you can benefit from the extensive research in the last decade that has been applied to the McEliece system to avoid side channel information leakage in the syndrome calculation, Berlekamp-Massey and root finding algorithms. For example the reference implementations of the NTS-KEM submission or the Classic McEliece submission could be used in LAC with only minor modifications to integrate within the procedures the variability of the parameter t.

Best regards

Martin

From: cpeikert@gmail.com on behalf of Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Friday, February 16, 2018 10:43 PM
To: Alperin-Sheriff, Jacob (Fed)
Cc: pqc-comments; pqc-forum@list.nist.gov; sd lattice
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Thu, Feb 15, 2018 at 2:22 PM, Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov> wrote:
> Coming back to this point "Yes, [Hardness of SIS and LWE with Small
> Parameters] gets closer to the mark for plain LWE with binary (or
> ternary) errors. But to prevent any confusion: it would not support
> the plain-LWE analogue of LAC either, because the number of samples is
> limited to smaller than what LAC reveals to the attacker."
>
> If I'm understanding everything properly, the paper doesn't say
> anything at all about the plain LWE analogue of LAC and more
> generally, the public key in any ring-LWE (or plain LWE) scheme using
> the Lindner-Peikert formulation of primal Regev, because that paper
> requires m >= n + omega(log n) in order to use the Micciancio-Mol SIS
> equivalence, and we have m=n for LAC. Is this correct?

Well, those m parameters don't appear to mean the same thing.

I think the main issue is this: for plain LWE with binary (or ternary) errors, [Hardness ... with Small Parameters] proves hardness when the number of LWE samples with *uniform secret over Zq* is

m < n(1 + 1/log n). (See discussion following Thm 4.6.)

However, Lindner-Peikert encryption uses *normal form* LWE, where the entries of the secret are chosen from the error distribution (not uniformly).

Transforming from uniform-secret to normal-form LWE costs at least n samples, which for the above m leaves < n/log n normal-form samples.
But the Lindner-Peikert system reveals at least n normal-form samples.
(So does the original Regev cryptosystem.)

Sincerely yours in cryptography,
Chris

From: sd lattice
To: Alperin-Sheriff, Jacob (Fed); luxianhui@iie.ac.cn
Cc: pqc-comments; pqc-forum@list.nist.gov; Christopher J Peikert
Subject: Re: OFFICIAL COMMENT: LAC
Date: Friday, February 23, 2018 5:44:32 AM

Dear Alperin-Sheriff, Jacob

I agree with you and Peikert that, existing result cannot guarantee the worst-case hardness of the Ring-LWE problem with ternary errors used in LAC. I think it is an interesting question to study the worst-case hardness of Ring-LWE problem or plain LWE problem with ternary errors and 2n samples. I will try to solve this question.

I checked your analysis of the CCA security of LAC256. Although the attacker can find enough ciphertexts with Hamming weights of (r, e_1) of 1024+310, I do not understand how to recover s based on the analysis in https://eprint.iacr.org/2016/085.pdf.

The main difficulty is that, this analysis needs to change the error-reconciliation vector. However, in the decryption algorithm of CCA LAC256, there is a re-encryption process to check whether the ciphertext is valid. This means that, if the attacker changes the error-reconciliation vector, the cipherrtext will be rejected.

Another difficulty is that, in this analysis, the attacker needs to set e_1 to be special vectors with 1 in one coefficient and 0 elsewhere. However, (r, e_1) are generated as the output of the hash function, and the attacker cannot control the value of e_1.

It seems that, when the attacker finds one failure decryption it can only get the information that there are more than 55 bits errors.

Best Wishes.
LAC Team

发件人: Christopher J Peikert
发送时间: 2018年2月17日 11:43
收件人: Alperin-Sheriff, Jacob (Fed)
抄送: pqc-comments; pqc-forum@list.nist.gov; sd lattice
主题: Re: [pqc-forum] Re: OFFICIAL COMMENT: LAC

On Thu, Feb 15, 2018 at 2:22 PM, Alperin-Sheriff, Jacob (Fed)
<jacob.alperin-sheriff@nist.gov> wrote:
> Coming back to this point "Yes, [Hardness of SIS and LWE with Small
> Parameters] gets closer to the mark for plain LWE with binary (or ternary)
> errors. But to prevent any confusion: it would not support the plain-LWE
> analogue of LAC either, because the number of samples is limited to smaller
> than what LAC reveals to the attacker."
>
> If I'm understanding everything properly, the paper doesn't say anything at
> all about the plain LWE analogue of LAC and more generally, the public key
> in any ring-LWE (or plain LWE) scheme using the Lindner-Peikert formulation
> of primal Regev, because that paper requires m >= n + omega(log n) in order
> to use the Micciancio-Mol SIS equivalence, and we have m=n for LAC. Is this
> correct?

Well, those m parameters don't appear to mean the same thing.

I think the main issue is this: for plain LWE with binary (or ternary)
errors, [Hardness ... with Small Parameters] proves hardness when the
number of LWE samples with *uniform secret over Zq* is

m < n(1 + 1/log n). (See discussion following Thm 4.6.)

From: luxianhui@iie.ac.cn
Sent: Friday, February 23, 2018 6:01 AM
To: mt@post-quantum.com; pqc-comments
Cc: pqc-forum; Alperin-Sheriff, Jacob (Fed)
Subject: Re:OFFICIAL COMMENT: LAC

Hi Mike and Martin,

I will try to implement a constant-time decoding algorithm according to the suggestion of Martin. Just as Martin pointed out, there are extensive research results about McEliece system in the last decade. Maybe I can use the result in [1].

I think it is a good idea to use Goppa codes in LAC and I will try it.

[1] Daniel J. Bernstein, Tung Chou, Peter Schwabe: **McBits: Fast Constant-Time Code-Based Cryptography.** CHES 2013: 250-272.

Best Wishes.

LAC Team

From: Mike Hamburg <mike@shiftleft.org>
Sent: Thursday, April 12, 2018 2:52 PM
To: Alperin-Sheriff, Jacob (Fed)
Cc: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

By the way, this analysis comes from several productive discussions at PQC, with the likes of Hart, Sauvik, Aemon, and others.

— Mike

> On Apr 12, 2018, at 2:50 PM, Mike Hamburg <mike@shiftleft.org> wrote:
>
> Hello Jacob and other PQCers,
>
> Further analysis suggests that this failure mode of LAC is worse than previously expected.
>
> I assume that an attack constitutes a "certificational weakness" if it requires < 2^255 classical work (vs Class V), 2^64 total decryption queries, and 2^64 different target public keys, and breaks at least one public key with probability > 1/2. Note that this excludes attacks that eg send a single message and pray that it causes a failure; such attacks would constitute certificational weaknesses in many systems but don't translate well to the real world.
>
> The following analysis is for LAC256, but the attacks it proposes work are likely to take < 2^128 work.
>
> LAC's failure probability is influenced by both the Hamming weight of the key and of the ciphertext. An adversary can spend about 2^62 work to find a ciphertext with Hamming weight 9 standard deviations above the mean, which would be 0.77 per position (instead of 0.5). If my estimates are correct, such a ciphertext fails with probability about 2^-62 for a random key. This would mean an attack using about 2^124 work and 2^62 queries to find the first failure. But this might not work against a single key, because a significant fraction of the failure probability comes from the possibility that the key may have high Hamming weight.
>
> However, LAC doesn't hash the public key when expanding an encryption seed. Therefore, the attack actually only takes 2^62 work to find an evil seed, and then that seed can be turned into a ciphertext for each of 2^62 public keys. (We are assuming here that each bacterium inside the gut of each human on Earth has a public key.) Send one evil query to each key, and probably one of them will fail to decrypt.
>
> Afterward, the adversary can be confident that the victim's key has a high Hamming weight (median ~2.8 standard deviations above 0.5), and that it has a large correlation with the (secret,noise) vectors produced by the evil ciphertext. Finding enough further ciphertexts that have a high correlation with the initial one might be feasible, though I haven't calculated it. Just going on Hamming weight would give 2^108 work per additional failure (with < 2^60 additional decryption queries per failure), but I expect that considering the correlation would greatly reduce the required work.
>
> The adversary doesn't know which 56 of the 1024 positions failed, nor whether the failures were positive or negative. But he can align the failing ciphertexts with each other by computing their mutual correlations. Then averaging the failing ciphertexts gives an approximation of the (reverse of the) key. Some non-rigorous experiments with single-failure systems suggests that a few dozen failing ciphertexts may be enough to recover the key entirely, or at least bring it within range of a practical hybrid attack.
>
> Overall, these back-of-the-envelope calculations suggest that CCA failure attacks on LAC could succeed with high probability for less than 2^128 work. If many targets are available, the effort might even approach a feasible level.
>

> These calculations do not account for the possibility that the key and/or ciphertext may also have a high autocorrelation. This might increase the correlation between failures, and thus further increase the probability that > 56 failures occur.
>
> I suggest that LAC should retune in the following ways:
> * Smaller noise variance, though this increases the risk of a hybrid attack.
> * Fixed Hamming weight noise instead of IID weighted ternary noise. This increases complexity and risk of side-channel attacks, but LAC's large error correction already induces similar risk of side-channel attacks. Also, whether this works depends on how much autocorrelation affect probability of decryption failure, because fixed Hamming weight doesn't mean fixed autocorrelation.
> * Hash the public key into the FO mode's seed, to prevent multi-target attacks.
>
> Hashing the public key into the seed probably makes the attack infeasible, but it would probably still be present as a certificational weakness, so at least one of the other two countermeasures is probably also required.
>
> Finally, note that this concern applies also to ThreeBears and to HILA5, and neither of these has a completely rigorous failure analysis yet. However, all the threats described in this comment have already been accounted for in the ThreeBears failure analysis, including:
>
> * Correlation due to larger-than-expected secret key.
> * Correlation due to larger-than-expected ciphertext.
> * (After publication) Correlated failures due to autocorrelation of key/ciphertext.
> * Multi-target attacks (because ThreeBears hashes part of the pubkey into the seed).
>
> These attacks are not accounted for in the HILA5 failure analysis, but HILA5's failure analysis is very conservative (Saarinen estimates 2^-135, but I think the real answer is more like 2^-165), and 5-error-correction is much less than 55-error-correction. So I believe both ThreeBears and HILA5 to be safe from this attack, but further investigation is warranted.
>
> Cheers,
> — Mike Hamburg
>
>> On Feb 16, 2018, at 3:02 PM, Mike Hamburg <mike@shiftleft.org> wrote:
>>
>> Hi Jacob,
>>
>> I think Fluhrer's attack https://eprint.iacr.org/2016/085.pdf doesn't apply here, because it assumes unconstrained chosen messages and not random ones. Furthermore, while "HILA5 Pindakaas" discussed attacking a 5-ECC with unconstrained chosen messages, attacking a 55-ECC with random high-weight (but otherwise not maliciously chosen) messages would be much more complicated.
>>
>> On the negative side, LAC256 private keys have somewhat low entropy of only 1.5kibit (1.5 bits per position). So having 2^14 known failures and 2^64 non-failures might open up information-theoretic approaches. For example with 2^14 failures you may be able to reduce the entropy quite a bit by constraining that it has a large with the known failure cases. I think more analysis is warranted here.
>>
>>
>>
>> You also asked whether I had measured the BCH code. I haven't, I was just guessing by reading it, especially since it uses very different numbers of iterations depending on the number of errors.
>>
>> If the BCH code's behavior is visible to a software attacker, then LAC might be attackable with practical complexity. If there are enough errors for Chien search to be used, an adversary would know with reasonable confidence exactly how many errors are present and where they are, or at the very least where the last one is. Furthermore, the ciphertexts

From: Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be>
Sent: Thursday, April 12, 2018 6:23 PM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Mike, Hi all,

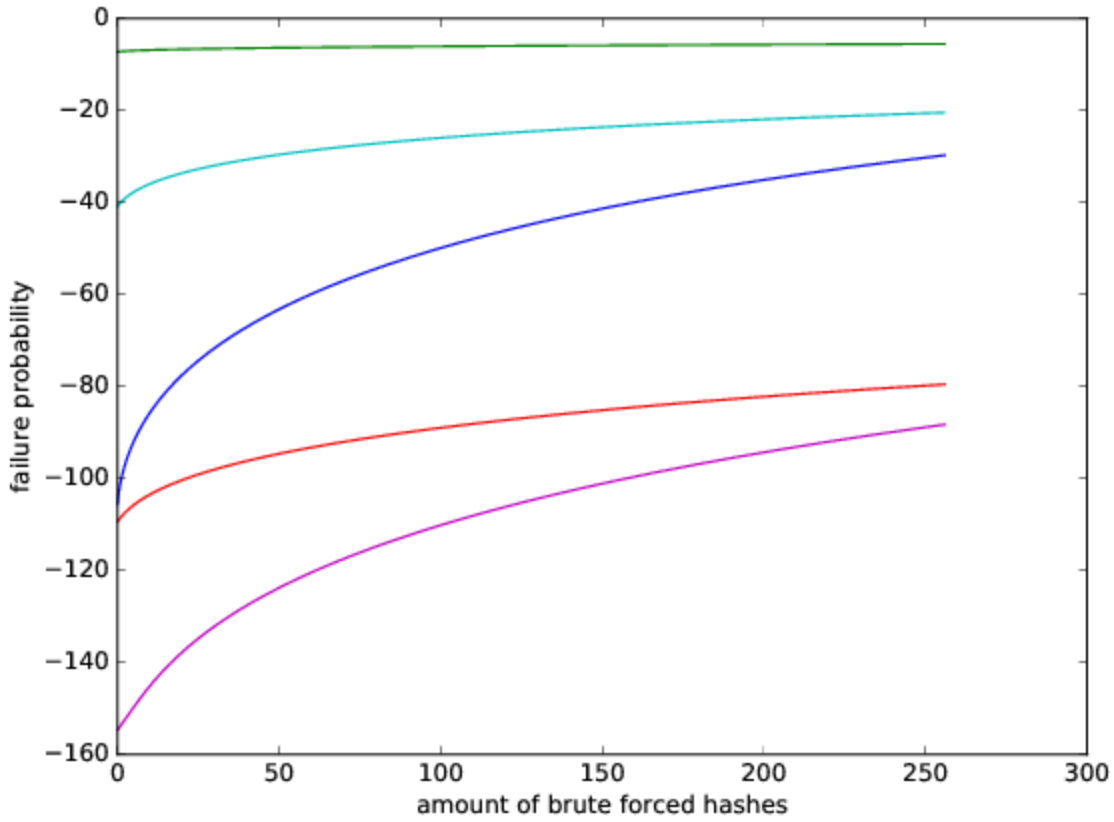
Short version

- the error correction of LAC makes it more prone to attacks where the attacker precomputed weak values of (e', s') (Bob's secret and noise) with high hamming weight
- however, because of the error correction and the relatively low error rate of the individual coefficients, it is seems hard to determine which coefficients are failing and whether they are positive or negative failure coefficients. This makes it difficult to get any information about the secret (e, s) from the failures (or: low failure rate is not necessarily a bad thing)
- Using enough failures (2^10? maybe lower), we could construct classifiers that we can use to roughly estimate the secret (e, s)

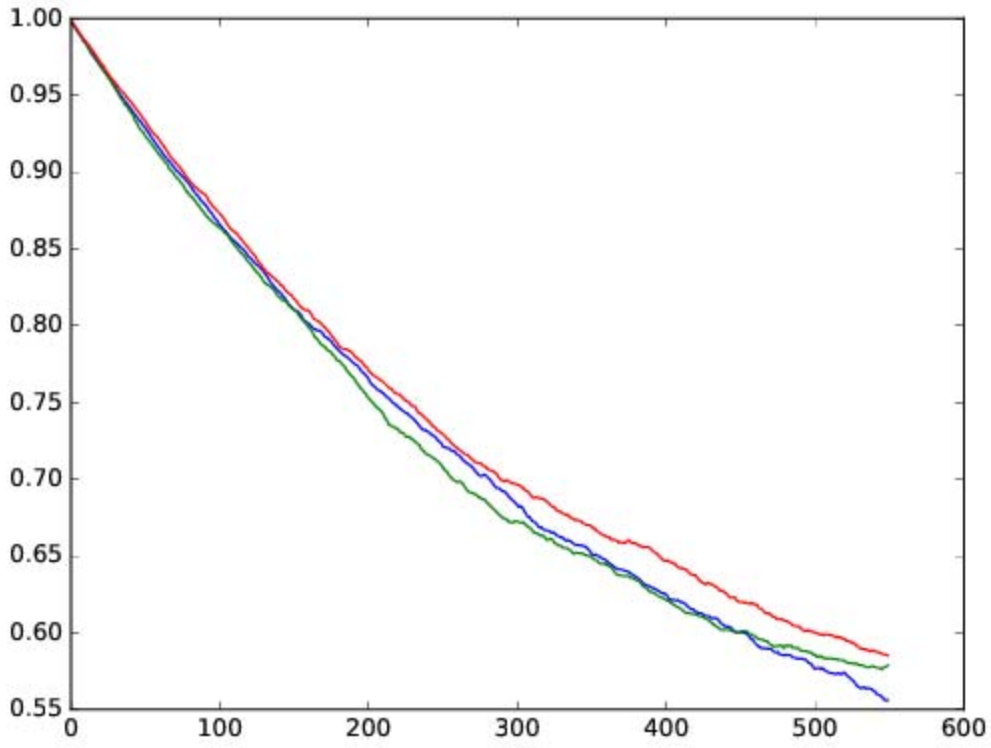
Long version

I have been working on an analysis of the attack angle where the adversary influences the hamming weight of his secrets (e', s') to obtain information about (e, s). In the graph beneath you can see how searching through the hashes for weak high hamming weight (e', s') influences the failure probability. The different schemes depicted are LAC256 (dark blue), LAC256 without error correction (green), a LAC256-alike scheme without error correction but adjusted to have similar error so that you can see the influence of the error correction (red), Frodo (light blue), FrodoKEM (purple). Here you can clearly see that the error correction of LAC heavily decreases the failure probability, but also that it makes is way more susceptible for searching for high (e', s').

Note that this search can be sped up with Grover, and that as you noted, the search has to be done only once if no multitarget protection is in place. Fixed hamming weight secrets (e', s') would also defeat this approach, but they might of course have other security implications.



If the failure rate would be around 2^-128, a failure would have very high correlation with the secrets, so that you could run an autocorrelation between the samples and recover the positions of the failure coefficients and determine whether they are positive or negative. Then it is relatively easy to get a good estimate of the key using any statistical technique. In the picture a simple non optimized Maximum Likelihood estimation is used to plot the entropy reduction vs the amount of failure vectors that are used. Note that one failure sample in case of LAC would amount to 56 failure vectors that give information.



Once a decent estimate of the key has been made, we can compute a new RLWE problem by subtracting b'=As_est+e_est from b, which would result in the RLWE problem A(s-s_est)+(e-e_est), with smaller noise and error terms, and therefore more susceptible to attacks on the RLWE problem.

However, this all assumes that it is easy to distinguish positive, negative and no failure coefficients. This is not necessarily the case! Since the failure rate of one single coefficient is rather low (+/-2^-8), the correlation of the secret (e, s) with the input (e', s') is not high. From preliminary experiments, it seems not trivial to distinguish the autocorrelations leading to a failure coefficients from these leading to a no failure coefficient, and therefore it seems not trivial to get information about the secret (e, s) from a failure sample!

If this information could be obtained from side channels, then the adversary can use the statistical methods from above to estimate the key and thus recover (e, s). If not, the adversary needs to find another way to gain information about (e, s). We now that in case of a failure (e, s) has a slightly higher correlation with the absolute value of 56 coefficients of (e', s') than usual. Using this information we can construct a (very noisy) classifier that can (with very low probability of success) distinguish between a sample (e_est, s_est) that is close to the secret (e, s) and one that is not close to (e, s) by summing the absolute value of the 56 biggest coefficients. As said before, this classifier will not be very informative, but by combining (averaging) several different classifier from different failure samples, we can improve the quality of our classifier. Early calculations indicate that we need to have around 2^10 failures to construct a such a decent classifier to get a very rough estimate of (e, s), but I'm not sure yet. I'm currently working on a more exact estimate of the needed amount of failures.

Once we have a decent classifier, we can do a search for a rough estimate of (e, s). which in turn will enable us to 1) or directly make a better estimate of (e, s) 2) or use (e, s) to classify our coefficients into positive, negative and no failure coefficients, so that the statistical techniques from above can be used to get a better estimate of (e, s).

As I see it now, two factors determine the success probability of the attack:
1) the probability of finding failures though searching for weak (e, s) keys as depicted in the first figure
2) the amount of failures we need to make a rough estimate of the secret (e, s) which can be used to directly make a good estimate of (e, s) or to distinguish positive/negative/no failure coefficients.

For most schemes (1) will be the bottleneck and (2) will not pose any problem because of the low failure rate and thus high correlation between the failure (e', s') and the secret (e, s). For LAC (2) might also make the attack more difficult since multiple failures might be needed.

I will keep you posted with updates.

Regards,

Jan-Pieter

> By the way, this analysis comes from several productive discussions at PQC, with the likes of Hart, Sauvik, Aemon, and others.
>
> — Mike
>
> > On Apr 12, 2018, at 2:50 PM, Mike Hamburg <mike@shiftleft.org> wrote:
> >
> > Hello Jacob and other PQCers,
> >
> > Further analysis suggests that this failure mode of LAC is worse than previously expected.
> >
> > I assume that an attack constitutes a “certificational weakness” if it requires < 2^255 classical work (vs Class V), 2^64 total decryption queries, and 2^64 different target public keys, and breaks at least one public key with probability > 1/2. Note that this excludes attacks that eg send a single message and pray that it causes a failure; such attacks would constitute certificational weaknesses in many systems but don’t translate well to the real world.
> >
> > The following analysis is for LAC256, but the attacks it proposes work are likely to take < 2^128 work.
> >
> > LAC’s failure probability is influenced by both the Hamming weight of the key and of the ciphertext. An adversary can spend about 2^62 work to find a ciphertext with Hamming weight 9 standard deviations above the mean, which would be 0.77 per position (instead of 0.5). If my estimates are correct, such a ciphertext fails with probability about 2^-62 for a random key. This would mean an attack using about 2^124 work and 2^62 queries to find the first failure. But this might not work against a single key, because a significant fraction of the failure probability comes from the possibility that the key may have high Hamming weight.
> >
> > However, LAC doesn’t hash the public key when expanding an encryption seed. Therefore, the attack actually only takes 2^62 work to find an evil seed, and then that seed can be turned into a ciphertext for each of 2^62 public keys. (We are assuming here that each bacterium inside the gut of each human on Earth has a public key.) Send one evil query to each key, and probably one of them will fail to decrypt.
> >
> > Afterward, the adversary can be confident that the victim’s key has a high Hamming weight (median ~2.8 standard deviations above 0.5), and that it has a large correlation with the (secret,noise) vectors produced by the evil ciphertext. Finding enough further ciphertexts that have a high correlation with the initial one

From: LU XIANHUI <luxianhui@outlook.com>
Sent: Wednesday, October 24, 2018 3:31 AM
To: pqc-comments; pqc-forum@list.nist.gov
Subject: Formal Analysis of LAC

Dear All,

We give a formal analysis of subfield attack and high hamming weight attack on LAC, detailed result can be found at https://eprint.iacr.org/2018/1009 .

Briefly, we reduce the session key length to 256bits for all the three security levels (just like Kyber). As a result we have enough space to promote the error correction ability, especially for the case of LAC256. Finally, we show that decryption error rate is small enough to resist the High hamming weight attack.

We remark that, when the session key length or the message space is reduced to 256, we can use D2 and BCH together in LAC256. With the assist of D2, the BCH error correction code only need to correct 20bist errors at most, which is much smaller than the current version of 55bits. This makes the LAC256 more efficient. For the sake of simplicity, we can use the same parameter set for all of the three security levels.

LAC Team
Best Regards.

From: Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be>
Sent: Tuesday, November 20, 2018 8:11 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: LACattack.zip

Dear all,

Multiple submissions to the NIST Post-Quantum Standardization Process rely heavily on Error Correcting Codes (ECC) to decrease decryption failures. Their particular implementation behavior however adds another layer of complexity to the scheme, and it also potentially makes the scheme more vulnerable to side-channel attacks, as shown below on LAC. Especially advanced ECC's, such as BCH, are more susceptible for timing attacks, where an adversary can gain knowledge on the number of errors or their location from timing information. In general, one should take utmost caution in using ECC's that are not side-channel resistant.

In the following attack, timing variations in the ECC of LAC are used to break its IND-CCA security and obtain the secret key in a matter of hours. The attack works for any security level of LAC. I will first present a basic but very powerful attack that recovers the full secret.
Then I will show variations to argue the generality of the attack.

First we look at three phases of the decapsulation: decryption, decoding and re-encryption. The decryption takes two polynomial inputs b' and v'
in \mathbb{Z}_q[X]/(X^n+1), and calculates \Delta v = v'-b's, where s is the secret. \Delta v is used to recover (an approximation of) the decoded message c, following this formula for a scheme dependent threshold q_t:

c_i = 0 if -q_t < \Delta v_i <= q_t
1 otherwise

c is then corrected using the ECC, resulting in the message m. This steps leaks timing information that allows us to easily distinguish between a c that contains an error and a correct c.
What happens if we choose as input the polynomials b'=1 and v' = q_t + c_v, with c_v a valid codeword? Without loss of generality, we choose the zero polynomial as codeword, so that we can write \Delta v = q_t - s. The only possible error location is \Delta v_0, as the perturbation on other coefficients of \Delta v is very small (-1, 0, +1). The value of c_0 is calculated as follows:

c_i = 0 if q_t - s_0 <= q_t
1 otherwise

or:

c_i = 0 if s_0 = (0, 1)
1 if s_0 = -1

We can therefore use the timing information to derive whether an error occurred, and thus test if s_0 equals -1. Similarly, we can test if s_0 equals 1 by choosing b' to be -1. Other coefficients of s can be tested with the appropriate b' = +/- X^k.

This attack is implemented on both the reference and optimized implementation of LAC and can be found in attachment. To run, just type "make" and run "./lac".

Given the specific structure of b' and v', which is easy to detect, one could think of a countermeasure where these type of inputs are rejected.
However, an easy generalization of the attack also works on inputs b'
and v' that look totally genuine:
We start by trying honest inputs b' and v', until we find a pair that doesn't have any errors. This can be detected using the timing information. Once we find such a pair, we make small adjustments a to one of the coefficients of v' to generate b', v'_a = v' + a X^k and we input these as new ciphertexts. Starting at 1, we increase a until we find the point where we obtain a failure. At this point, we know that:
\Delta v_k = (v'_a - b's)_k = q_t + 1 or (b's)_k = v'_a - (q_t + 1), which gives us a linear equation in s. Repeating this procedure we obtain enough equations to solve for s.

To conclude, although ECCs are useful to lower decryption failures, their correct implementation and overall behaviour is a huge cause for concern. At the very least, constant time implementations will be necessary, but more general, they should not leak any side-channel information. As such, it is very much up for debate if the added complexity of correctly implementing these ECCs is worth it, compared to the efficiency gain they provide by lowering decryption failures.

Kind regards,

Jan-Pieter

From: 'Alperin-Sheriff, Jacob (Fed)' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Tuesday, November 20, 2018 9:05 AM
To: Jan-Pieter D'Anvers
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Jan,
Thank you for this work (we haven't fully tested it yet, but it makes sense at first read). Any idea how much of a penalty schemes would pay in speed for the use of constant-time ECCs (or any works to point to on the complexity of making them constant time?)

On 11/20/18, 8:12 AM, "Jan-Pieter D'Anvers" <janpieter.danvers@esat.kuleuven.be> wrote:

Dear all,

Multiple submissions to the NIST Post-Quantum Standardization Process rely heavily on Error Correcting Codes (ECC) to decrease decryption failures. Their particular implementation behavior however adds another layer of complexity to the scheme, and it also potentially makes the scheme more vulnerable to side-channel attacks, as shown below on LAC. Especially advanced ECC's, such as BCH, are more susceptible for timing attacks, where an adversary can gain knowledge on the number of errors or their location from timing information. In general, one should take utmost caution in using ECC's that are not side-channel resistant.

In the following attack, timing variations in the ECC of LAC are used to break its IND-CCA security and obtain the secret key in a matter of hours. The attack works for any security level of LAC. I will first present a basic but very powerful attack that recovers the full secret. Then I will show variations to argue the generality of the attack.

First we look at three phases of the decapsulation: decryption, decoding and re-encryption. The decryption takes two polynomial inputs b' and v' in \mathbb{Z}_q[X]/(X^n+1), and calculates \Delta v = v'-b's, where s is the secret. \Delta v is used to recover (an approximation of) the decoded message c, following this formula for a scheme dependent threshold q_t:

c_i = 0 if -q_t < \Delta v_i <= q_t
1 otherwise

c is then corrected using the ECC, resulting in the message m. This steps leaks timing information that allows us to easily distinguish between a c that contains an error and a correct c.
What happens if we choose as input the polynomials b'=1 and v' = q_t + c_v, with c_v a valid codeword? Without loss of generality, we choose the zero polynomial as codeword, so that we can write \Delta v = q_t - s. The only possible error location is \Delta v_0, as the perturbation on other coefficients of \Delta v is very small (-1, 0, +1). The value of c_0 is calculated as follows:

From: Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be>
Sent: Thursday, November 22, 2018 4:15 AM
To: Alperin-Sheriff, Jacob (Fed)
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Jacob,

At this moment, I'm not aware of any work on constant-time ECCs like BCH.

There are simple ECCs that can easily be implemented in constant time, for example repetition codes, but they do not attain the same level of error correction.

Regards,

Jan-Pieter

On 20-11-18 15:04, Alperin-Sheriff, Jacob (Fed) wrote:
> Jan,
> Thank you for this work (we haven't fully tested it yet, but it makes
> sense at first read). Any idea how much of a penalty schemes would pay
> in speed for the use of constant-time ECCs (or any works to point to
> on the complexity of making them constant time?)
>
> On 11/20/18, 8:12 AM, "Jan-Pieter D'Anvers" <janpieter.danvers@esat.kuleuven.be> wrote:
>
> Dear all,
>
> Multiple submissions to the NIST Post-Quantum Standardization Process
> rely heavily on Error Correcting Codes (ECC) to decrease decryption
> failures. Their particular implementation behavior however adds another
> layer of complexity to the scheme, and it also potentially makes the
> scheme more vulnerable to side-channel attacks, as shown below on LAC.
> Especially advanced ECC's, such as BCH, are more susceptible for timing
> attacks, where an adversary can gain knowledge on the number of errors
> or their location from timing information. In general, one should take
> utmost caution in using ECC's that are not side-channel resistant.
>
> In the following attack, timing variations in the ECC of LAC are used to
> break its IND-CCA security and obtain the secret key in a matter of
> hours. The attack works for any security level of LAC. I will first
> present a basic but very powerful attack that recovers the full secret.
> Then I will show variations to argue the generality of the attack.
>
> First we look at three phases of the decapsulation: decryption, decoding
> and re-encryption. The decryption takes two polynomial inputs b' and v'
> in \mathbb{Z}_q[X]/(X^n+1), and calculates \Delta v = v'-b's, where s is
> the secret. \Delta v is used to recover (an approximation of) the
> decoded message c, following this formula for a scheme dependent
> threshold q_t:

From: Mike Hamburg <mike@shiftleft.org>
Sent: Thursday, November 22, 2018 4:32 AM
To: Jan-Pieter D'Anvers
Cc: Alperin-Sheriff, Jacob (Fed); pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Jacob and Jan-Pieter,

It's probably very tedious, but not outright prohibitive, to make a constant-time BCH decoder for many errors. Really the only hard part is the Berlekamp-Massey step, since it's easy to make Chien search constant-time. I looked into this for ThreeBears, but it seemed painful enough that I settled for 2-error-correction.

Also, there is literature in the code-based crypto community on constant-time Goppa codes, though they might not correct as well as BCH. But I'm not up to date on code-based crypto that resists more powerful side-channels like power or EM… can anyone on the codes side comment?

Regards,
— Mike

> On Nov 22, 2018, at 1:14 AM, Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be> wrote:
>
> Hi Jacob,
>
> At this moment, I'm not aware of any work on constant-time ECCs like BCH.
>
> There are simple ECCs that can easily be implemented in constant time, for example repetition codes, but they do not attain the same level of error correction.
>
> Regards,
>
> Jan-Pieter
>
>
> On 20-11-18 15:04, Alperin-Sheriff, Jacob (Fed) wrote:
>> Jan,
>> Thank you for this work (we haven't fully tested it yet, but it makes sense at first read). Any idea how much of a penalty schemes would pay in speed for the use of constant-time ECCs (or any works to point to on the complexity of making them constant time?)
>>
>> On 11/20/18, 8:12 AM, "Jan-Pieter D'Anvers" <janpieter.danvers@esat.kuleuven.be> wrote:
>>
>> Dear all,
>> Multiple submissions to the NIST Post-Quantum Standardization Process
>> rely heavily on Error Correcting Codes (ECC) to decrease decryption
>> failures. Their particular implementation behavior however adds another
>> layer of complexity to the scheme, and it also potentially makes the
>> scheme more vulnerable to side-channel attacks, as shown below on LAC.
>> Especially advanced ECC's, such as BCH, are more susceptible for timing
>> attacks, where an adversary can gain knowledge on the number of errors
>> or their location from timing information. In general, one should take
>> utmost caution in using ECC's that are not side-channel resistant.

From: Martin Tomlinson <mt@post-quantum.com>
Sent: Sunday, November 25, 2018 5:04 AM
To: Mike Hamburg
Cc: Jan-Pieter D'Anvers; Alperin-Sheriff, Jacob (Fed); pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: signature.asc

Hi Mike,

I can comment from the codes side…

Goppa codes are closely related to BCH codes and it turns out that BCH codes are a sub class of Goppa codes.
Any Goppa code decoder can be used to decode BCH codes simply by using a trivial one bit padding to extend the code length

For most code parameters Goppa codes are superior to BCH codes as the code length is one bit longer and conveniently the code length is a binary number.
indeed for most applications Goppa codes are the preferred choice over BCH codes.

Probably the reason that Goppa codes are not more widely known and used is that BCH codes were discovered first and were the subject of significant R & D in the West before Goppa published his paper in Russian.

Regards

Martin

> On 22 Nov 2018, at 09:32, Mike Hamburg <mike@shiftleft.org> wrote:
>
> Hi Jacob and Jan-Pieter,
>
> It's probably very tedious, but not outright prohibitive, to make a constant-time BCH decoder for many errors. Really the only hard part is the Berlekamp-Massey step, since it's easy to make Chien search constant-time. I looked into this for ThreeBears, but it seemed painful enough that I settled for 2-error-correction.
>
> Also, there is literature in the code-based crypto community on constant-time Goppa codes, though they might not correct as well as BCH. But I'm not up to date on code-based crypto that resists more powerful side-channels like power or EM… can anyone on the codes side comment?
>
> Regards,
> — Mike
>
>> On Nov 22, 2018, at 1:14 AM, Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be> wrote:
>>
>> Hi Jacob,
>>
>> At this moment, I'm not aware of any work on constant-time ECCs like BCH.
>>

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, November 26, 2018 11:05 AM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: signature.asc

Martin Tomlinson writes:
> For most code parameters Goppa codes are superior to BCH codes as the
> code length is one bit longer and conveniently the code length is a
> binary number.

Actually, BCH codes have noticeably better parameters than most (classical binary) Goppa codes. The reason is that BCH codes have symmetries that most Goppa codes don't have, and these symmetries create overlaps in the standard list of parity checks. For example:

```
sage: C = codes.BCHCode(GF(2),1023,107)
sage: C.base_field()
Finite Field of size 2
sage: C.length()
1023
sage: C.designed_distance()
107
sage: C.dimension()
543
sage:
```

Mindlessly adding a 0 bit to each codeword produces length 1024 (if desired), designed distance still 107, dimension still 543. Compare this to McEliece using Goppa codes of length 1024, designed distance only 101, dimension usually only 524. This BCH code transmits 3.6% more data in the same length while correcting 6% more errors with the same decoding algorithms. (There are other codes that are even better.)

Traditional code-based cryptography uses random Goppa codes rather than BCH codes, but the reasons have nothing to do with Goppa codes being better. On the contrary: BCH codes are better, but this was and is overridden by other concerns, as I'll now explain.

McEliece's security analysis relied on the number of Goppa codes available for the user to select as part of the secret key: "The first attack seems hopeless if n and t are large enough because there are so many possibilities for G, not to mention the possibilities for S and P."
McEliece's emphasis on counting the number of G was later justified (for almost all codes---I haven't checked whether BCH codes are among the
exceptions) by Sendrier's support-splitting algorithm, which, given G and the public key, finds S and P.

McEliece didn't consider the possibility of obtaining even more codes by randomly puncturing---dropping some code positions, enough for n to be considerably smaller than the field size. This can't produce a big security loss (if removing some positions from G produces G' then removing the same positions from Gm+e produces G'm+e', and then an attack against G' reveals e', after which the rest of e is relatively easy to find) while it could produce a big security gain. Concretely,

https://cr.yp.to/talks.html#2012.09.24

notes that there are no known attacks against a randomly permuted randomly punctured BCH code. (Of course this is not the conservative thing to do.)

I'm not saying that puncturing is always necessary. McEliece's original system is holding up just fine without it. But puncturing does create an extra attack obstacle. The Classic McEliece submission provides both punctured and unpunctured parameters, and comments that the extra difficulty of finding the secret puncturing "can be viewed as a reason to keep n somewhat smaller" than the field size. Furthermore, puncturing doesn't make parameters worse: on the contrary, as page 20 of

https://classic.mceliece.org/nist/vsntskem-20180629.pdf

illustrates, puncturing produces the best tradeoffs known between key size and security against known attacks.

---Dan

From: 'Alperin-Sheriff, Jacob (Fed)' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Monday, November 26, 2018 12:48 PM
To: D. J. Bernstein; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

This argument is all very interesting, but is rather tangential to the constant-time (and more generally side channel resistant) implementability question I had about ECCs; the specific ECC shouldn't matter except for the purpose of optimizing error correction (i.e. it doesn't have to have any cryptographic difficulties stemming from it), as they are used in this context (and in several other submissions) in a manner independent of the underlying cryptographic hardness.

On 11/26/18, 11:05 AM, "D. J. Bernstein" <djb@cr.yp.to> wrote:

Martin Tomlinson writes:
> For most code parameters Goppa codes are superior to BCH codes as the
> code length is one bit longer and conveniently the code length is a
> binary number.

Actually, BCH codes have noticeably better parameters than most
(classical binary) Goppa codes. The reason is that BCH codes have
symmetries that most Goppa codes don't have, and these symmetries create
overlaps in the standard list of parity checks. For example:

```
sage: C = codes.BCHCode(GF(2),1023,107)
sage: C.base_field()
Finite Field of size 2
sage: C.length()
1023
sage: C.designed_distance()
107
sage: C.dimension()
543
sage:
```

Mindlessly adding a 0 bit to each codeword produces length 1024 (if
desired), designed distance still 107, dimension still 543. Compare this
to McEliece using Goppa codes of length 1024, designed distance only
101, dimension usually only 524. This BCH code transmits 3.6% more data
in the same length while correcting 6% more errors with the same
decoding algorithms. (There are other codes that are even better.)

Traditional code-based cryptography uses random Goppa codes rather than
BCH codes, but the reasons have nothing to do with Goppa codes being
better. On the contrary: BCH codes are better, but this was and is
overridden by other concerns, as I'll now explain.

McEliece's security analysis relied on the number of Goppa codes
available for the user to select as part of the secret key: "The first
attack seems hopeless if n and t are large enough because there are so
many possibilities for G, not to mention the possibilities for S and P."
McEliece's emphasis on counting the number of G was later justified (for
almost all codes---I haven't checked whether BCH codes are among the

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, November 27, 2018 4:11 AM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: signature.asc

Fast constant-time decoding algorithms for (classical binary) Goppa codes are explained in https://tungchou.github.io/papers/mcbits.pdf and https://tungchou.github.io/papers/mcbits_revisited.pdf. The algorithms rely on the algebraic structure of these codes.

An easy way to see the performance of these algorithms is to look at the performance of Classic McEliece decapsulation. The relevant points here are that

(1) the Classic McEliece software is constant-time,
(2) decapsulation in particular uses these decoding algorithms,
(3) there isn't much other work in decapsulation, and
(4) the decapsulation software includes serious optimizations for Intel CPUs.

Decapsulation on Haswell (titan0) takes 321580 cycles for mceliece6960119 and 342640 cycles for mceliece8192128.

Applications to lattice-based systems (more specifically, small systems with decryption failures) would typically use code lengths 1024 or below instead of 8192. Figuring out the exact speedup would need an implementation tuned for the specific size in question.

Regarding size and error-correcting capability, the obvious choice of Goppa code in the lattice context is a BCH code. For example, one can encode 256 bits as 512 bits while correcting 30 errors:

```
sage: C = codes.BCHCode(GF(2),511,61)
sage: C.dimension()
259
sage:
```

An easy variant of New Hope could encode 256 bits as 1024 bits while correcting 106 errors:

```
sage: C = codes.BCHCode(GF(2),1023,213)
sage: C.dimension()
258
sage:
```

For comparison, a random length-1024 Goppa code correcting just 77 errors would typically have dimension 254.

There's a misconception that BCH codes need a different decoder. It's important here that the better BCH parameters come from an overlap of parity checks (as stated in my previous message), not from a different decoding algorithm.

There's also a misconception that random Goppa codes are better than BCH. One source of this misconception is the fact that McEliece selected random Goppa codes rather than BCH codes. The actual reasons for this selection are different, and are explained in my previous message.

Research directions: In the same lattice context it should be possible to correct even more errors for these codes with a standard technique called "soft-decision decoding". Building constant-time software for this is an interesting research challenge, and certainly there will be a slowdown. Decoding even more errors is feasible with other codes, but again the speed of constant-time decoding is an open question. A few options (starting with BCH) are considered in

https://eprint.iacr.org/2018/150.pdf

but that paper says "constant-time implementations may be challenging".

> they are used in this context (and in several other submissions) in a
> manner independent of the underlying cryptographic hardness.

I'm skeptical of this claim. Why can't there be interactions between the underlying cryptographic structure and the error-correcting structure?
For example, could a different error-correcting code have avoided the
Round5 attack?

There are many claims, starting from many questionable independence assumptions, regarding

(1) decryption-failure probabilities for various submissions and
(2) the impact of these failures on security.

I don't see how anyone who reads https://eprint.iacr.org/2018/1089 (tricky calculations with the conclusion "an attacker can significantly reduce the security of several candidates") can think that these issues are well understood. I realize that not everyone agrees with my assessment that

* the (massive) auditing advantages of eliminating decryption failures

outweigh

* the (small) performance advantages of allowing decryption failures,

but surely everyone agrees that careful analyses are required--- including analyses of how error correction is affected by the structure of errors in the underlying cryptographic scheme.

---Dan

From: Martin Tomlinson <mt@post-quantum.com>
Sent: Tuesday, November 27, 2018 4:51 AM
To: pqc-forum@list.nist.gov; Alperin-Sheriff, Jacob (Fed); djb@cr.yp.to
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: signature.asc

Dan Bernstein writes:

> Actually, BCH codes have noticeably better parameters than most
> (classical binary) Goppa codes. The reason is that BCH codes have
> symmetries that most Goppa codes don't have, and these symmetries create
> overlaps in the standard list of parity checks.

This is simply not true. It is an invalid statement for most BCH codes.
What is true is that there are ***some*** examples of code parameters where BCH codes correct more errors than the corresponding, one bit longer, Goppa codes.
These examples are few and far between and arise because of fortuitous overlaps in the cyclotomic number sets. (In these cases the n-k binary parity check bits fortuitously satisfy additional useful parity check equations without the need for additional parity check bits.)

In applying error correction, sometimes the binary vector length and number of bits to be corrected can be chosen to correspond to one of these rare and special BCH codes. Otherwise Goppa codes are the preferred choice with the code length an integral number of bytes.

Anyway as Jacob has just pointed out this is rather besides the point.
What is needed is a constant time error correcting decoder.
The bottom line is that a constant time Goppa code decoder can be used to decode BCH codes in constant time.

Regards

Martin

> On 26 Nov 2018, at 17:47, 'Alperin-Sheriff, Jacob (Fed)' via pqc-forum <pqc-forum@list.nist.gov> wrote:
>
> This argument is all very interesting, but is rather tangential to the constant-time (and more generally side channel resistant) implementability question I had about ECCs; the specific ECC shouldn't matter except for the purpose of optimizing error correction (i.e. it doesn't have to have any cryptographic difficulties stemming from it), as they are used in this context (and in several other submissions) in a manner independent of the underlying cryptographic hardness.
>
> On 11/26/18, 11:05 AM, "D. J. Bernstein" <djb@cr.yp.to> wrote:
>
> Martin Tomlinson writes:

From: Pedro Maat Costa Massolino <P.Massolino@cs.ru.nl>
Sent: Tuesday, November 27, 2018 9:41 AM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: alternant_codes.png

Hi all,

I attached a figure from:
F.J. MacWilliams and N.J. A. Sloane. "The Theory of Error-Correcting Codes" North-Holland, vol 16, 1981.

The figure can help us understand the relation between BCH, Goppa codes and more important Alternant codes.

To sum up: Not all BCH codes are Goppa codes, and not all Goppa codes are BCH.
Also, not all BCH codes are Reed-Solomon codes, just as with Goppa codes, only part of it.
However, BCH belongs to Alternant codes set.

"What is this related to decoding matters?"

A Goppa code decoder that assumes Goppa codes structure would only work on that set of codes.
However, most decoders seem in literature work in a bigger set of codes.
For example, the Berlekamp-Massey pointed out before can work on Alternant codes, which compromises all the codes previously mentioned:

"H. Helgert". Decoding of alternant codes (Corresp.). IEEE Transactions on Information Theory (Volume: 23 , Issue: 4 , Jul 1977)

"How many decoders do we have?"

Several. If you want a good summary of decoders for Reed-Solomon codes, I suggest this paper:

Yongge Wang. "Decoding Generalized Reed-Solomon Codes and Its Application to RLCE Encryption Scheme".
https://eprint.iacr.org/2017/733

I can focus on 2 decoders:

- Berlekamp-Massey.
- Gao-Meter.

The first can decode alternant codes. The last one it was proposed only for Reed-Solomon codes.
All 3 decoders can be split into 3 parts:

1) Compute syndrome
2) Compute error locator polynomial
3) Find error positions.

In the first decoder the step 1 is a matrix multiplication and step 3 can be done through Chien search.
In the second decoder this is done through FFT in steps 1 and 3.

The second step, which is the hardest part, can be done though a iterative shift register (Berlekamp-Massey) or gcd euclidean algorithm (Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa)

Most of the papers when do constant time have to manually guarantee it.
Therefore it can easily be done wrong.
One example of how to do it, as mentioned before, can be seen in :
"Tung Chou". McBits Revisited. CHES-2017

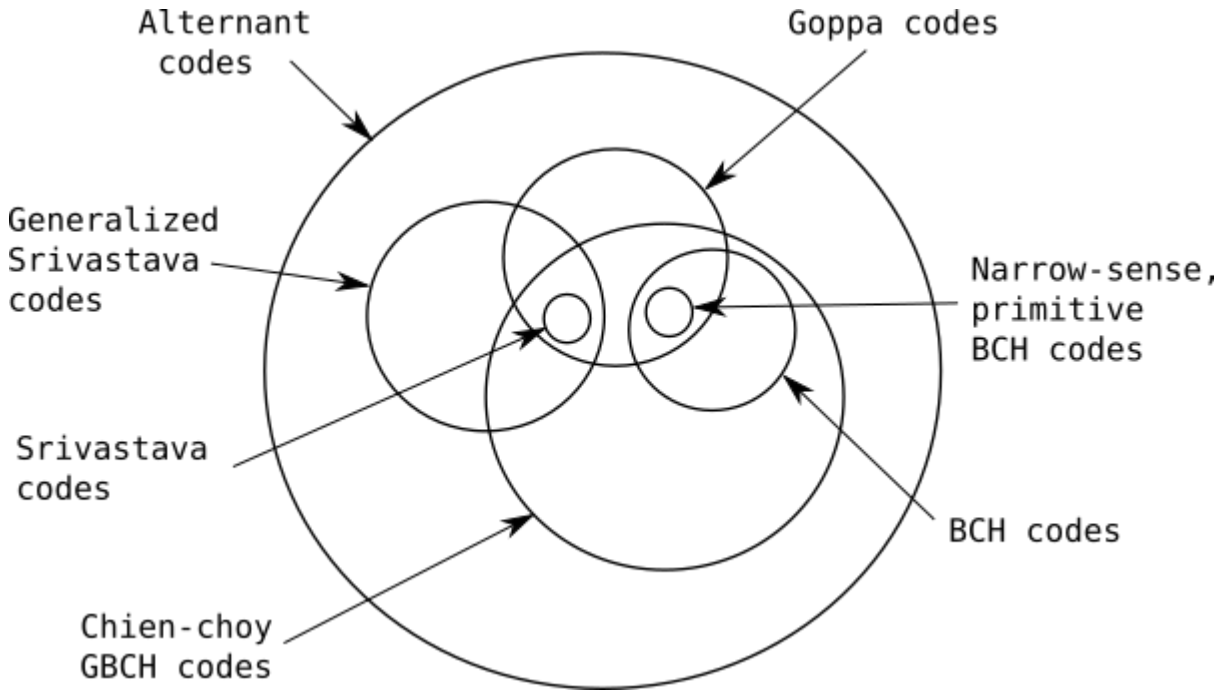
For the gcd case:

F. Arguello. "Binary GCD algorithm for computing error locator polynomials in Reed-Solomon decoding". Electronics Letters, Volume: 41,
Issue: 13, 2005.
Mariya Georgieva and Frédéric de Portzamparc. "Toward Secure Implementation of McEliece Decryption". COSADE 2015

"What about LACs code?"

The code used in LAC is a BCH one, thus one of the Alternant decoders could be used.
I tried to check if the BCH code is also Reed-Solomon, and I found a code with a bigger distance.
I found the following Reed Solomon code:
[511, 340, 172]
This code can use the Gao-Meter decoder and correct even more errors. I hope it can help the authors,

Best Regards,
Pedro



From: 'Edoardo Persichetti' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Tuesday, November 27, 2018 10:43 AM
To: Pedro Maat Costa Massolino
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Pedro

Thanks for citing the MacWilliams/Sloane book - I think all coding theorists are particularly fond of that book ;)

Indeed this is a very clear explanation.

The discussion had derailed a bit (as Jacob reminded us), but it should now be clear that the answer to the original question ("do we have constant-time implementations of ECC"), is yes. One such example is Classic McEliece's constant-time decoder - which builds on the McBits/McBits Revisited framework.

Best,
Edoardo

> On Nov 27, 2018, at 9:41 AM, Pedro Maat Costa Massolino <P.Massolino@cs.ru.nl> wrote:
>
> Hi all,
>
> I attached a figure from:
> F.J. MacWilliams and N.J. A. Sloane. "The Theory of Error-Correcting
> Codes" North-Holland, vol 16, 1981.
>
> The figure can help us understand the relation between BCH, Goppa codes
> and more important Alternant codes.
>
> To sum up: Not all BCH codes are Goppa codes, and not all Goppa codes
> are BCH.
> Also, not all BCH codes are Reed-Solomon codes, just as with Goppa
> codes, only part of it.
> However, BCH belongs to Alternant codes set.
>
> "What is this related to decoding matters?"
>
> A Goppa code decoder that assumes Goppa codes structure would only work
> on that set of codes.
> However, most decoders seem in literature work in a bigger set of codes.
> For example, the Berlekamp-Massey pointed out before can work on
> Alternant codes, which compromises all the codes previously mentioned:
>
> "H. Helgert". Decoding of alternant codes (Corresp.). IEEE Transactions
> on Information Theory (Volume: 23 , Issue: 4 , Jul 1977)
>
> "How many decoders do we have?"
>
> Several. If you want a good summary of decoders for Reed-Solomon codes,

From: Leo Ducas <leo.ducas1@gmail.com>
Sent: Tuesday, November 27, 2018 12:43 PM
To: pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi all,

> Research directions: In the same lattice context it should be possible to correct even more errors for these codes with a standard technique called "soft-decision decoding". Building constant-time software for this is an interesting research challenge, and certainly there will be a slowdown.

A (small) step in this research direction exists, namely, a (tentative) constant time CVP algorithm for the Leech lattice. This is somewhat a form of soft-decoding, and gives maximum-likelihood decoding for gaussian errors.

https://eprint.iacr.org/2016/1050.pdf
https://github.com/avanpo/leech-decoding

A BDD variant could certainly be made faster.

the limitation to dimension 24 is certainly annoying, and one would certainly prefer larger codes/lattices with better rates. Though the algorithm should vectorize quite well, which could make it interesting to use several codes in parallel rather than a large one.

Best regards
-- Leo

From: luxianhui@gmail.com
Sent: Wednesday, November 28, 2018 11:03 AM
To: Jan-Pieter D'Anvers; pqc-comments; pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Jan-Pieter,

Thank you very much for the side channel analysis of LAC. I believe your attack idea are right. In fact, in February of this year, Alperin-Sheriff, Jacob, Mike Hamburg and Martin Tomlinson also noticed that our implementation is in a variable time manner, and may cause side channel attack.

We are already working on the constant time BCH and Goppa, and try to give a constant time implementation.

The main design goal of LAC is to decrease the size of pk and ciphertext as small as possible. I think, it is worth to add ECC to achieve this goal.

Best Regards,
Xianhui Lu
LAC Team.

luxianhui@gmail.com

From: Jan-Pieter D'Anvers
Date: 2018-11-20 21:11
To: pqc-comments
CC: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: LAC

Dear all,

Multiple submissions to the NIST Post-Quantum Standardization Process rely heavily on Error Correcting Codes (ECC) to decrease decryption failures. Their particular implementation behavior however adds another layer of complexity to the scheme, and it also potentially makes the scheme more vulnerable to side-channel attacks, as shown below on LAC. Especially advanced ECC's, such as BCH, are more susceptible for timing attacks, where an adversary can gain knowledge on the number of errors or their location from timing information. In general, one should take utmost caution in using ECC's that are not side-channel resistant.

From: Mike Hamburg <mike@shiftleft.org>
Sent: Wednesday, November 28, 2018 4:03 PM
To: 路献辉
Cc: Leo Ducas; pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Xianhui,

I've looked into this a little bit in the context of ThreeBears and Glowstick.

For ThreeBears, I investigated the possibility of soft-decoding a parity check. This is not difficult to do exactly, and it should perform almost as well as a 1-ECC but only requires one extra bit (or 4 bits for ThreeBears, because there are two candidates for the ring that have dimension 260 = 256+4). There is an approximation that's even easier: set a fixed threshold t, and track which symbols are >= t away from 0 or q/2. Then if the parity check fails, toggle all such symbols. This obviously has slightly worse error-correcting capability than a true soft-decoder, and it requires determining the optimal threshold upfront, but it doesn't require a constant-time implementation of max() so it's only a few lines of code. There is no error floor with such a design. At least for the exact version, it should be possible to determine the failure probability exactly, which is an improvement on the 2-ECC currently used, but that failure probability is almost certainly much higher.

Glowstick is an experimental, extremely aggressive RLWR KEM that I've been developing off-and-on (with input from Sauvik, Oscar, Ludo, Leo and others). I've tested polar codes and LDPC over GF(q) for q in {2,16,256}. I haven't been able to get polar codes to beat LDPC either computationally or in error-correction capability in this application, so let's talk LDPC.

LDPC codes have two easy decoding algorithms: belief propagation with likelihoods, and with log-likelihood ratios. The LLR version is faster and simpler for GF(2), slower for GF(16) and much slower GF(256) because you have more additions and can't use a Fourier transform.

In my code currently, there are several sources of non-constant-time behavior that are annoying or costly to remove, but nothing that's entirely prohibitive:

*** In both versions, you iterate belief propagation until you get a codeword. Replacing this with a fixed iteration count will hurt performance quite a bit, since usually 1-2 iterations suffice but you need tens of iterations to get a very low failure rate.

*** In the LLR version, the function x -> log(1+exp(x)) for fixed-point positive x is called many times. It uses a lookup table, but it could be converted to constant time at the cost of performance and/or precision. Possibly the easiest high-precision approximation here is either cubic or:

le1(x) - x = max(a-b*x,0)*(2*p - (c*x % p))>>(c*x // p + 2*lg(p) + 1)

where p is an amount of additional precision, and a,b,c are constants that depend on p. It's possible that 16-bit LLRs arithmetic is good enough.

*** In the likelihood version, float arithmetic including normalization routines. Probably tedious but not terribly hard to convert to fixed-point.

*** Initial LLR table lookups and final max() calculations are not difficult, but slightly tedious.

But in the end, I don't believe this matters much for LDPC codes, because the error floor precludes CCA security anyway. So Glowstick's actual strategy is just to achieve a failure rate which is low enough that I can't find any failures in a few days' compute on a quad-core desktop (eg 2^-35 or lower, which is borderline acceptable for CPA according to Adam Langley). Then you would deploy it in a hybrid mode with elliptic curves for ephemeral key exchange, using each ephemeral lattice key only once. To break this, an adversary would need both a high-bandwidth side channel and a quantum computer, which means it probably defeats mass surveillance attacks.

Maybe constant-time soft decoding would matter more for a code with a lower error floor, like LDPC+BCH.

— Mike

On Nov 28, 2018, at 8:41 AM, luxianhui@gmail.com wrote:

Hi Ducas,

I have tried to improve LAC by using soft-decision decoding for few months. In my opinion, D2, D4 and Leech lattice can all be seen as the soft-decision decoding versions of repetition codes.
However, for general codes such as LDPC or Turbo, soft-decision decoding can not guarantee a deterministic number of error correction bits with 100% probability. For example, it can correct up to x bits for most of the case, and may fail for some cases even if the number of error bits is smaller than x. The failure probability may be very small, such as 2^-10. It is ok for most of the application scenarios of error correction code. But, when used in lattice based PKE, this may cause CCA attack. The attacker may control the number of error bits, and find the failure case by using decryption oracle, and then the distribution of the secret key may be leaked.

Do you have any idea to solve this problem?

Best Regards
Xianhui Lu
LAC Team

---

luxianhui@gmail.com

From: Leo Ducas
Date: 2018-11-28 01:42
To: pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi all,

> Research directions: In the same lattice context it should be possible
> to correct even more errors for these codes with a standard technique

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Wednesday, November 28, 2018 5:20 PM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: signature.asc

I wrote, correctly:
> Actually, BCH codes have noticeably better parameters than most
> (classical binary) Goppa codes.

Martin Tomlinson wrote:
> This is simply not true. It is an invalid statement for most BCH codes.
> What is true is that there are some examples of code parameters where
> BCH codes correct more errors than the corresponding, one bit longer, Goppa codes.
> These examples are few and far between

To quantify the actual BCH advantage: The Sage script at the end of this message computes, for each degree t between 17 and 70,

* a Goppa code of length 512 using a random degree-t irreducible polynomial over GF(512), and

* a BCH code: a Goppa code of length 511 using the polynomial x^(2t) over GF(512).

This is a fair comparison since the aforementioned fast constant-time decoding algorithm (alternant decoding via Berlekamp's method plus many
optimizations) has a limit of exactly t errors for both of these codes.
The script checks that a comparable decoding method (also Berlekamp, no optimizations, not constant-time) successfully decodes a random word with t errors in both cases.

Every line of output shows the BCH code being superior to the random Goppa code. The output line

```
field 512 errors 30 goppa [512, 242] bch [511, 259]
```

illustrates one of the examples I gave earlier: BCH encodes 259 bits as
511 bits while correcting 30 errors. For comparison, this random
degree-30 irreducible Goppa code also corrects 30 errors but encodes only 242 bits. The output line

```
field 512 errors 28 goppa [512, 260] bch [511, 277]
```

shows a random degree-28 irreducible Goppa code encoding 260 bits, but this corrects only 28 errors, while the BCH code correcting 28 errors encodes 277 bits. The output line

```
field 512 errors 50 goppa [512, 62] bch [511, 157]
```

shows an even more dramatic gap. Here are all the output lines from one run of the script to show that these examples are the normal situation, not "few and far between", not "rare and special":

```
field 512 errors 17 goppa [512, 359] bch [511, 367]
field 512 errors 18 goppa [512, 350] bch [511, 358]
```

field 512 errors 19 goppa [512, 341] bch [511, 349]
field 512 errors 20 goppa [512, 332] bch [511, 340]
field 512 errors 21 goppa [512, 323] bch [511, 331]
field 512 errors 22 goppa [512, 314] bch [511, 322]
field 512 errors 23 goppa [512, 305] bch [511, 313]
field 512 errors 24 goppa [512, 296] bch [511, 304]
field 512 errors 25 goppa [512, 287] bch [511, 304]
field 512 errors 26 goppa [512, 278] bch [511, 295]
field 512 errors 27 goppa [512, 269] bch [511, 286]
field 512 errors 28 goppa [512, 260] bch [511, 277]
field 512 errors 29 goppa [512, 251] bch [511, 268]
field 512 errors 30 goppa [512, 242] bch [511, 259]
field 512 errors 31 goppa [512, 233] bch [511, 250]
field 512 errors 32 goppa [512, 224] bch [511, 241]
field 512 errors 33 goppa [512, 215] bch [511, 241]
field 512 errors 34 goppa [512, 206] bch [511, 241]
field 512 errors 35 goppa [512, 197] bch [511, 241]
field 512 errors 36 goppa [512, 188] bch [511, 241]
field 512 errors 37 goppa [512, 179] bch [511, 238]
field 512 errors 38 goppa [512, 170] bch [511, 229]
field 512 errors 39 goppa [512, 161] bch [511, 220]
field 512 errors 40 goppa [512, 152] bch [511, 211]
field 512 errors 41 goppa [512, 143] bch [511, 211]
field 512 errors 42 goppa [512, 134] bch [511, 202]
field 512 errors 43 goppa [512, 125] bch [511, 193]
field 512 errors 44 goppa [512, 116] bch [511, 184]
field 512 errors 45 goppa [512, 107] bch [511, 184]
field 512 errors 46 goppa [512, 98] bch [511, 175]
field 512 errors 47 goppa [512, 89] bch [511, 166]
field 512 errors 48 goppa [512, 80] bch [511, 157]
field 512 errors 49 goppa [512, 71] bch [511, 157]
field 512 errors 50 goppa [512, 62] bch [511, 157]
field 512 errors 51 goppa [512, 53] bch [511, 157]
field 512 errors 52 goppa [512, 44] bch [511, 148]
field 512 errors 53 goppa [512, 35] bch [511, 148]
field 512 errors 54 goppa [512, 26] bch [511, 139]
field 512 errors 55 goppa [512, 17] bch [511, 130]
field 512 errors 56 goppa [512, 8] bch [511, 121]
field 512 errors 57 goppa [512, 1] bch [511, 121]
field 512 errors 58 goppa [512, 0] bch [511, 121]
field 512 errors 59 goppa [512, 0] bch [511, 112]
field 512 errors 60 goppa [512, 0] bch [511, 103]
field 512 errors 61 goppa [512, 0] bch [511, 103]
field 512 errors 62 goppa [512, 0] bch [511, 94]
field 512 errors 63 goppa [512, 0] bch [511, 85]
field 512 errors 64 goppa [512, 0] bch [511, 76]
field 512 errors 65 goppa [512, 0] bch [511, 76]
field 512 errors 66 goppa [512, 0] bch [511, 76]
field 512 errors 67 goppa [512, 0] bch [511, 76]
field 512 errors 68 goppa [512, 0] bch [511, 76]
field 512 errors 69 goppa [512, 0] bch [511, 76]
field 512 errors 70 goppa [512, 0] bch [511, 76]

For t below 17, the BCH advantage disappears, but I don't see the cryptographic relevance of codes with such limited error-correcting capabilities. There are also some worse codes that are sometimes called "BCH" codes, but I have no idea why anyone would want to use those.

Pedro Maat Costa Massolino writes:
> [511, 340, 172]

That's over GF(512), whereas we're talking about lattice applications of codes over GF(2).

---Dan

```
q = 512
k = GF(q)
R.<x> = k[]

def Gamma(support,g):
  m = q.ord(2)
  n = len(support)
  t = g.degree()
  gs = [g(a) for a in support]
  M = matrix(GF(q),t,n,lambda i,j:support[j]^i/gs[j])
  M2 = matrix(GF(2),m*t,n,lambda i,j:M[i//m,j].polynomial()[i%m])
  return M2.right_kernel()

def randomerror(support,t):
  e = [0]*len(support)
  while sum(e) < t: e[randrange(len(support))] = 1
  return vector(GF(2),e)

def decode(support,g,r):
  n = len(support)
  t = g.degree()
  gs = [g(a) for a in support]
  M = matrix(GF(q),t,n,lambda i,j:support[j]^i/gs[j])
  synd = M * r
  errorpoly = berlekamp_massey(list(synd))
  e = [errorpoly(a) == 0 for a in support]
  r += vector(GF(2),e)
  if M * r != 0:
    if 0 in support:
      r += vector(GF(2),[a == 0 for a in support])
  assert M * r == 0
  return r

for t in range(17,71):
  # build random irreducible Goppa code:
  g = R.irreducible_element(t,'random')
  support = list(k)
  G = Gamma(support,g)
  assert G == Gamma(support,g^2)
  goppadim = G.dimension()
  goppalen = len(support)

  # try decoding random word with t errors:
```

```
c = G.random_element()
r = c + randomerror(support,t)
assert decode(support,g^2,r) == c

# build BCH code:
support = [a for a in k if a != 0]
g = x^(2*t)
G = Gamma(support,g)
bchdim = G.dimension()
bchlen = len(support)

# try decoding random word with t errors:
c = G.random_element()
r = c + randomerror(support,t)
assert decode(support,g,r) == c

print 'field',q,'errors',t,'goppa',[goppalen,goppadim],'bch',[bchlen,bchdim]
```

From: Martin Tomlinson <mt@post-quantum.com>
Sent: Thursday, November 29, 2018 3:50 AM
To: D. J. Bernstein
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: signature.asc

For most practical applications of error correcting codes we want bandwidth efficiency, a reasonably high throughput, For these applications binary Goppa codes are superior to the corresponding BCH codes because the special BCH codes are few and far between.
You are lucky if you find one whose code parameters are a reasonable fit to your application.

Your list of superior BCH codes of length 511 bits contains exactly no codes with an efficiency of 72% or more.

Besides this, what is the point of listing all those codes with a payload of less than 256 bits?

Regards

Martin

> On 28 Nov 2018, at 22:19, D. J. Bernstein <djb@cr.yp.to> wrote:
>
> I wrote, correctly:
>> Actually, BCH codes have noticeably better parameters than most
>> (classical binary) Goppa codes.
>
> Martin Tomlinson wrote:
>> This is simply not true. It is an invalid statement for most BCH codes.
>> What is true is that there are some examples of code parameters where
>> BCH codes correct more errors than the corresponding, one bit longer, Goppa codes.
>> These examples are few and far between
>
> To quantify the actual BCH advantage: The Sage script at the end of
> this message computes, for each degree t between 17 and 70,
>
> * a Goppa code of length 512 using a random degree-t irreducible
> polynomial over GF(512), and
>
> * a BCH code: a Goppa code of length 511 using the polynomial x^(2t)
> over GF(512).
>
> This is a fair comparison since the aforementioned fast constant-time
> decoding algorithm (alternant decoding via Berlekamp's method plus
> many
> optimizations) has a limit of exactly t errors for both of these codes.
> The script checks that a comparable decoding method (also Berlekamp,
> no optimizations, not constant-time) successfully decodes a random
> word with t errors in both cases.
>
> Every line of output shows the BCH code being superior to the random

From: luxianhui@gmail.com
Sent: Sunday, December 02, 2018 8:26 AM
To: Mike Hamburg
Cc: pqc-forum
Subject: Re: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Mike,

Thank you very much for sharing the experience. Currently I have not tested LDPC yet.
I thought that LDPC is good for long code length beyond 1024 and not suitable for short code length such as 256 or 511.
So, currently, I am tring to figure out constant time BCH decoder.

As I understand, according to the requirement of NSIT, the error rate of 2^-35 is not small enough although it is hard to find a failure in few days.
So, for error correction code with large error correction ability, the error floor of soft-decision is really a problem.

Best Regards
Xianhui

luxianhui@gmail.com

From: Mike Hamburg
Date: 2018-11-29 05:02
To: 路献辉
CC: Leo Ducas; pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

Hi Xianhui,

I've looked into this a little bit in the context of ThreeBears and Glowstick.

For ThreeBears, I investigated the possibility of soft-decoding a parity check. This is not difficult to do exactly, and it should perform almost as well as a 1-ECC but only requires one extra bit (or 4 bits for ThreeBears, because there are two candidates for the ring that have dimension 260 = 256+4). There is an approximation that's even easier: set a fixed threshold t, and track which symbols are >= t away from 0 or q/2. Then if the parity check fails, toggle all such symbols. This obviously has slightly worse error-correcting capability than a true soft-decoder, and it requires determining the optimal threshold upfront, but it doesn't require a constant-time implementation of max() so it's only a few lines of code. There is no error floor with such a design. At least for the

From: Mike Hamburg <mike@shiftleft.org>
Sent: Sunday, December 02, 2018 3:45 PM
To: 路献辉
Cc: pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC

> On Dec 2, 2018, at 5:26 AM, luxianhui@gmail.com wrote:
>
> Hi Mike,
>
> Thank you very much for sharing the experience. Currently I have not tested LDPC yet.
> I thought that LDPC is good for long code length beyond 1024 and not suitable for short code length such as 256 or 511.

LDPC, and especially large-field LDPC, are competitive even for short blocks. See eg [1], which compares (128,64) codes.

> So, currently, I am tring to figure out constant time BCH decoder.

I agree that constant-time hard-decision BCH is the right first step for LAC.

> As I understand, according to the requirement of NSIT, the error rate of 2^-35 is not small enough although it is hard to find a failure in few days.
> So, for error correction code with large error correction ability, the error floor of soft-decision is really a problem.

The error floor probably makes LDPC alone impractical for encryption, but it may be practical for ephemeral key exchange, and at 2^-35 it's better than any hard decision code. IIRC there's no NIST requirement on the failure rate of key exchange — it's just a performance consideration. That said, Glowstick targets 2^-35, and uses LDPC, in large part because Glowstick is a proof of concept and not a NIST submission.

For cryptographically low failure rates, [2] suggests LDPC+BCH to avoid the error floor. You could probably also try large-field LDPC+RS. These would probably outperform BCH alone, but would be a pain to spec and implement in constant time. You could also just soft-decode BCH (eg using OSD as mentioned in [1]) which would perform even better, but would be a nightmare to decode in constant time.

> Best Regards
> Xianhui
>
> luxianhui@gmail.com

Cheers,
— Mike

[1] Gianluigi Liva, Lorenzo Gaudio, Tudor Ninacs and Thomas Jerkovits Code Design for Short Blocks: A Survey, https://arxiv.org/pdf/1610.00873.pdf

[2] Tim Fritzmann and Thomas Pöppelmann and Johanna Sepulveda, "Analysis of Error-Correcting Codes for Lattice-Based Key Exchange", https://eprint.iacr.org/2018/150

> From: Mike Hamburg

From: Martin Tomlinson <mt@post-quantum.com>
Sent: Monday, December 03, 2018 7:19 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: LAC, Potential attacks using a Dorsch decoder
Attachments: signature.asc

Dear All,

This is a general comment, not just on LAC, as this type of attack can be used on most schemes that use hard decision error correction to reduce decryption failures.

Taking LAC128 as an example:
LAC128 uses a BCH code with parameters (511, 264, 59) to correct up to 29 bit errors in the 511 bit vector and provides a payload of 264 bits.

Under the assumption that an adversary has an algorithm that can guess each of the 511 bits and can estimate the probability of each guess being correct then the adversary can use a Dorsch decoder to correct up to around 250 errors. More errors can be corrected, if greater resources are deployed by the adversary.

The Dorsch decoder works by selecting 264 bits from the 511 bits. The selected bits are those that have the highest probability estimates of being correct.
The non selected 247 bits are ignored by the Dorsch decoder. These bits can all be in error, it does not matter.
Knowing the BCH code, the adversary constructs the generator matrix for the 264 selected bits and generates the 511 bit corresponding codeword.
Thus the ignored 247 bits have now been regenerated and replaced with their correct versions, possibly.

This codeword is used to re-encrypt to check that it is correct or not.

If not correct, combinations of various bits in the selected 264 bits are bit flipped in turn and new trial codewords generated.
Up to 3 bit combinations are within reach for a modest Dorsch decoder implementation, explaining the figure of up to 250 bits corrected

From the correct codeword the adversary extracts the payload of 264 bits and breaks the scheme.

The Dorsch decoder is very amenable to parallel implementation. Also alternative algorithms for estimating bits can be run in parallel implementations

Unlike users, the adversary does not need to worry about constant time decoding or within reason, the decoding time. So it would seem that the adversary has a clear advantage in error correction.

Also it is worth noting that the Dorsch decoder is a general purpose decoder and can be used on different codes such as LDPC codes, Goppa codes, Hamming codes, etc.
As it is interpolation based, the Dorsch decoder principle can be used on any noisy vector containing redundancy.

Best regards

Martin

From: Mike Hamburg <mike@shiftleft.org>
Sent: Monday, December 03, 2018 1:22 PM
To: pqc-comments; Martin Tomlinson
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC, Potential attacks using a Dorsch decoder

... whoops, premature send.

I was going to ask, do the forum members know of techniques that would approximate computational LWE to enable this attack? Otherwise it's probably limited to side channels, but still worth considering in that context.

Cheers,
-- Mike

Sent from my phone

On Mon, Dec 3, 2018 at 10:17 AM -0800, "Mike Hamburg" <mike@shiftleft.org> wrote:

> Hi Martin,
>
> This seems pretty interesting as a step in a side-channel attack on decryption.
>
> As a front-channel attack though, as I understand it, finding guesses at LWE plaintext bits is tantamount to breaking the system anyway. In particular it breaks decision LWE.
>
> Sent from my phone

On Mon, Dec 3, 2018 at 4:18 AM -0800, "Martin Tomlinson" <mt@post-quantum.com> wrote:

> Dear All,
>
> This is a general comment, not just on LAC, as this type of attack can be used on most schemes that use hard decision error correction to reduce decryption failures.
>
> Taking LAC128 as an example:
> LAC128 uses a BCH code with parameters (511, 264, 59) to correct up to 29 bit errors in the 511 bit vector and provides a payload of 264 bits.
>
> Under the assumption that an adversary has an algorithm that can guess each of the 511 bits and can estimate the probability of each guess being correct then the adversary can use a Dorsch decoder to correct up to around 250 errors. More errors can be corrected, if greater resources are deployed by the adversary.
>
> The Dorsch decoder works by selecting 264 bits from the 511 bits. The selected bits are those that have the highest probability estimates of being correct.
> The non selected 247 bits are ignored by the Dorsch decoder. These bits can all be in error, it does not matter.
> Knowing the BCH code, the adversary constructs the generator matrix for the 264 selected bits and generates the 511 bit corresponding codeword.

From: Brent Kimberley <Brent.Kimberley@Durham.ca>
Sent: Monday, December 03, 2018 1:48 PM
To: 'Mike Hamburg'; pqc-comments; Martin Tomlinson
Cc: pqc-forum@list.nist.gov
Subject: RE: [pqc-forum] OFFICIAL COMMENT: LAC, Potential attacks using a Dorsch decoder

I presume you mean receive-only/passive/batch/historian/continuous learning mode - not transceiver/active/real-time mode.

From: Mike Hamburg [mailto:mike@shiftleft.org]
Sent: Monday, December 3, 2018 1:22 PM
To: pqc-comments@nist.gov; Martin Tomlinson <mt@post-quantum.com>
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC, Potential attacks using a Dorsch decoder

... whoops, premature send.

I was going to ask, do the forum members know of techniques that would approximate computational LWE to enable this attack? Otherwise it's probably limited to side channels, but still worth considering in that context.

Cheers,
-- Mike

Sent from my phone

On Mon, Dec 3, 2018 at 10:17 AM -0800, "Mike Hamburg" <mike@shiftleft.org> wrote:

> Hi Martin,
>
> This seems pretty interesting as a step in a side-channel attack on decryption.
>
> As a front-channel attack though, as I understand it, finding guesses at LWE plaintext bits is tantamount to breaking the system anyway. In particular it breaks decision LWE.
>
> Sent from my phone

On Mon, Dec 3, 2018 at 4:18 AM -0800, "Martin Tomlinson" <mt@post-quantum.com> wrote:

> ```
> Dear All,
>
> This is a general comment, not just on LAC, as this type of attack can be used on most
> schemes that use hard decision error correction to reduce decryption failures.
>
> Taking LAC128 as an example:
> LAC128 uses a BCH code with parameters (511, 264, 59) to correct up to 29 bit errors in
> the 511 bit vector and provides a payload of 264 bits.
>
> Under the assumption that an adversary has an algorithm that can guess each of the 511
> bits and can estimate the probability of each guess being correct then the adversary can
> ```

From: xianhui lu <luxianhui@gmail.com>
Sent: Monday, December 10, 2018 2:07 AM
To: Jan-Pieter D'Anvers
Cc: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LAC
Attachments: bch.c

Hi Jan-Pieter,

I modified the code of bch used in LAC.
Briefly, the main idea is to let looping statements such as "for" or "while" repeat constants times.
It seems that the attack can be prevented.

However, this modification decreases the performence of the decryption algorithm.

Now, I am trying to implement an efficient constant time goppa or bch code suitable for LAC, based on the implementation of Classic McEliece.

Best Regards
Xianhui Lu
LAC Team

Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be> 于2018年11月20日周二 下午9:11写道：

> Dear all,
>
> Multiple submissions to the NIST Post-Quantum Standardization Process
> rely heavily on Error Correcting Codes (ECC) to decrease decryption
> failures. Their particular implementation behavior however adds another
> layer of complexity to the scheme, and it also potentially makes the
> scheme more vulnerable to side-channel attacks, as shown below on LAC.
> Especially advanced ECC's, such as BCH, are more susceptible for timing
> attacks, where an adversary can gain knowledge on the number of errors
> or their location from timing information. In general, one should take
> utmost caution in using ECC's that are not side-channel resistant.
>
> In the following attack, timing variations in the ECC of LAC are used to
> break its IND-CCA security and obtain the secret key in a matter of
> hours. The attack works for any security level of LAC. I will first
> present a basic but very powerful attack that recovers the full secret.
> Then I will show variations to argue the generality of the attack.
>
> First we look at three phases of the decapsulation: decryption, decoding
> and re-encryption. The decryption takes two polynomial inputs b' and v'
> in \mathbb{Z}_q[X]/(X^n+1), and calculates \Delta v = v'-b's, where s is
> the secret. \Delta v is used to recover (an approximation of) the
> decoded message c, following this formula for a scheme dependent
> threshold q_t:
>
> c_i = 0 if -q_t < \Delta v_i <= q_t
> 1 otherwise
>
> c is then corrected using the ECC, resulting in the message m. This

From: Jan-Pieter D'Anvers <janpieter.danvers@esat.kuleuven.be>
Sent: Friday, December 21, 2018 2:11 PM
To: pqc-forum@list.nist.gov; pqc-comments
Subject: OFFICIAL COMMENT: LAC

Dear All,

In a recent eprint paper [1], we challenged the assumption that errors in the message of Ring/Mod-LWE/LWR based schemes are independent. We show theoretically and experimentally that there is a dependency relation between these errors. For schemes that use error correcting codes (ECC) this leads to an underestimation of the failure rate. The most serious case can be found in LAC-128, where the failure rate is underestimated with a factor 2^48. More specifically, for LAC-128, the failure rate increases from 2^-233 to 2^-185 and for LAC-256 from 2^-114 to 2^-92. These numbers do not take into account the failure boosting that is described in [2], so the failure probabilities under an attack will be even higher.

The results are not only applicable for LAC, but for any Ring/Mod-LWE/LWR based scheme that is using error correction. Schemes that do not perform error correction also have dependent errors, but in this case, the failure rate decreases slightly, leading to a small underestimation of their security. More information can be found in the paper [1], an implementation of the estimation technique for LAC can be found on my github: https://github.com/danversjp/LWEdependency.

Therefore, designers of Ring/Mod-LWE/LWR based schemes that are using error correction should look into the impact of these dependencies on the failure rate of their scheme.

Happy holidays,

Jan-Pieter

[1] J. P. D'Anvers, I. Verbauwhede, and F. Vercauteren, "The impact of error dependencies on Ring/Mod-LWE/LWR based schemes," IACR Cryptology ePrint Archive 2018(359), https://eprint.iacr.org/2018/1172
[2] J. P. D'Anvers, I. Verbauwhede, and F. Vercauteren, "On the impact of decryption failures on the security of LWE/LWR based schemes," IACR Cryptology ePrint Archive 2018(359), https://eprint.iacr.org/2018/1089