From: pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Tuesday, July 25, 2023 5:56 PM
To: pqc-forum
Subject: [pqc-forum] Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

Hi All,

PoC code discussed below can be found at: https://github.com/mjosaarinen/dme-py

Inverting

The multivariate candidate DME-Sign has three submitted parameter sets, q in {2^32,2^48,2^48} for NIST security targets I, III, and V. In the following discussion, I will concentrate on the "32-bit" level I version. We will describe a 2^-96 complexity forgery attack on it.

DME-Sign is built on a "trapdoor function" (in the style of RSA); there is a secret mapping from 256 bits to 256 bits (used for creating signatures) and a matching public mapping which is its inverse (for verifying signatures).

The file invert_demo.py demonstrates how to invert half of this mapping quickly; given a public key and a target for the first 128 bits of the "secret" side of the permutation, it selects 128 bits in the signature side that matches it.

```
$ python3 invert_demo.py
=== count = 0
m1 = 060a1e26 6cb1fe61 0ba18e4c 1b9a410b 8e1c10e0 a3417574 140bcf0a 159b1899
m2 = 060a1e26 6cb1fe61 0ba18e4c 1b9a410b 8e1c10e0 a3417574 140bcf0a 159b1899 [OK] simplified mapping match.
m3 = 53579029 d9eb70e6 08090a0b 0c0d0e0f 7cb8b25e 4de5ac5c 142325cb 7b5bda96
[OK] linear mapping to m[2:4]
sv = fe9c5602 108eb7c6 00000000 00000000 0b0d9ad1 be13a58c 01234567 89abcdef
m4 = 00010203 04050607 08090a0b 0c0d0e0f 7cb8b25e 4de5ac5c 142325cb 7b5bda96
[OK] Half of function inverted!
```

The same public keys are generated as in the KAT test vectors. The first comment, "simplified mapping match," indicates that the simplified algebraic description (below) is working fine -- the final comment indicates that the first 128 bits of the result of "public key verification" are set to target value 000102..0e0f in the trapdoor function. This has also been verified against the reference C implementation.

Observation on invertibility

The DME trapdoor function is based on computations in binary field Fq. Public mapping boils down to the evaluation of multivariate polynomials whose coefficients are defined by the public key. The input variables come from the signature.

The input and output for the public key mapping is a vector of eight 32-bit field elements. I prefer zero-based indexing, so I write the signature variables as (s[0], s[1], .. s[7]) and verification (message) variables as (m[0], m[1], .. m[7]). Apologies -- the technical specification document Implementation of DME-3rnds-8vars-32bits-sign.pdf indexes signature variables from x1 to x8.

We observe that setting signature words s[2]=0 and s[3]=0 (x3=x4=0 in the equations of the paper) causes a vast majority of the monomials in the public mapping to vanish, massively simplifying the mapping. There are other options with similar effects.

Let t(i) denote some power-of-2 exponentiation of signature word i -- s[i]^(2^f) for some power f defined the public key. Since this is a binary field, we have (x+y)^2 = x^2 + y^2, and squaring is bitwise linear (DME is "bitwise multilinear."). Hence t(i) is a constant linear combination of bits in s[i], defined by the public key.

With the setting t(2)=t(3)=0, the dependencies in the public mapping can be expressed as a function of a subset of multivariate polynomial coefficients a[], b[], c[], d[] in the public key and linear combinations of signature bits as:

*m[0..1] = a[4]*t(6)*t(4)*t(0) + a[9]*t(7)*t(4)*t(0) + a[14]*t(6)*t(5)*t(0) + a[19]*t(7)*t(5)*t(0) + a[24]*t(0) + a[27]*t(6)*t(4)*t(1) + a[30]*t(7)*t(4)*t(1) + a[33]*t(6)*t(5)*t(1) + a[36]*t(7)*t(5)*t(1) + a[39]*t(1) + a[44]*t(6)*t(4) + a[49]*t(7)*t(4) + a[54]*t(6)*t(5) + a[59]*t(7)*t(5) + a[64]*

*m[2..3] = b[20]*t(6)*t(4) + b[21]*t(7)*t(4) + b[22]*t(6)*t(5) + b[23]*t(7)*t(5) + b[24]*

*m[4..5] = c[20]*t(6)*t(4) + c[21]*t(7)*t(4) + c[22]*t(6)*t(5) + c[23]*t(7)*t(5) + c[24]*

*m[6..7] = d[52]*t(6)*t(4) + d[53]*t(7)*t(4) + d[54]*t(6)*t(5)*t(4) + d[55]*t(7)*t(5)*t(4) + d[56]*t(4) + d[57]*t(6)*t(5) + d[58]*t(7)*t(5) + d[59]*t(5) + d[60]*t(6)*t(4) + d[61]*t(7)*t(4) + d[62]*t(6)*t(5) + d[63]*t(7)*t(5) + d[64]*

Here m[0..1] means that the mapping for m[0] and m[1] is of the same form with the same input variables; the public key coefficients a[] for m[0] and m[1] are different. Similarly, for the other three word pairs m[2..3], m[4..5], m[6..7].

Simple 2^96 Forgery

Each randomized trial for forgery of some "msg" under a given public key first proceeds just like the signing function:

1. msg = {0,1)^* input message
2. r[0:8] = pick a random 64-bit salt
3. w[0:16] = SHA3(msg || r), with 128-bit w result (yep)
4. g[0:16] = SHA3(w[0:16]) g[0:8] ^= r[0:8] # we XOR r on the lower half of g
5. We turn 256-bit (w || g} into eight 32-bit target words m[i]

Forgery steps:

Or forged signatures are of form [s0, s1, 0, 0, s4, s5, s6, s7], with s[2] and s[3] set to zeros.

1. We first select s[4..7] so that m[2..5] will have the desired value. The demo forces only m[2..3] and treats s[6..7] as constants -- thereby turning a bilinear equation into a linear one and allowing an efficient solution. For this attack, we assume that with at most 2^96 offline effort (e.g., a table), we succeed in the 128-bit inversion with probability 2^-32.
2. We then treat m[6..7] as constants. Now the equations for m[0..1] are linearized as a function of s[0..1] (just like in the demo) and can be solved easily. Changing s[s..1] does not affect the already solved m[2..5] target values.
3. We have forced 192 bits to the target -- as much as one can hope with s[2] and s[3] set to zeros. Now we check for a match in m[6..7], which will occur with probability 2^-64. This gives the attack a total success probability of 2^-96, violating the Level-1 claims. There may be much better attacks (by solving the bilinear forms in step 1 algebraically rather than by brute force)

Note that the description of verification ("dme-open") in the document Implementation of DME-3rnds-8vars-32bits-sign.pdf only checks 128 bits of the w value (step 4); the attack would be more efficient in this case. The reference implementation further performs a consistency check of 8 bytes of g.

Best Regards,

- markku

--

From: 'Maxime Bros' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Wednesday, July 26, 2023 11:27 AM
To: pqc-forum
Cc: Markku-Juhani O. Saarinen
Subject: [pqc-forum] Re: Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

Dear Markku,

According to the spec, the polynomial map (F_2^32)^8 -> (F_2^32)^8 is bijective “almost everywhere”. More precisely, the paragraph right before Lemma 1.2 gives the domain D of the map D, thus one gets a bijection between D and the image of D.
Lemma 1.2 gives estimation of the probability that a randomly chosen “message” is outside of this domain.

If I understood correctly, the goal of the salt (or padding) r is not only to randomize the signature algorithm but also to make sure that one does not end with a signature outside of D.

While it is pretty clear throughout the specs that there are values that make it impossible to get a bijection from (F_2^32)^8 -> (F_2^32)^8, the actual verification algorithm page 7 makes it look like all signature of length 256 could be inputs, not raising any error.

However, if we look carefully at the definition of dme-enc and at the bottom of page 5 and the beginning of page 6 of the specs, it is said “It is easy to verify that E […] do not have a zero entry”.

All this to say that if their reference/optimized implementations do not raise error in that case, it is an issue and your attack is perfectly valid.
But an easy tweak to their implementations, to make them follow their specs, would make the signature resist to your attack ; in addition to this, performance-wise, I do not think that these verifications are costly at all.

I do not mean that the scheme is secure, just that your attack seems to exploit what looks like an implementation mistake to me, not a flaw in the signature scheme.

Sincerely,

Maxime Bros

On Tuesday, July 25, 2023 at 5:56:01 PM UTC-4 Markku-Juhani O. Saarinen wrote:

> Hi All,
>
> PoC code discussed below can be found at: https://github.com/mjosaarinen/dme-py
>
> **Inverting**
>
> The multivariate candidate DME-Sign has three submitted parameter sets, q in {2^32,2^48,2^48} for NIST security targets I, III, and V. In the following discussion, I will concentrate on the "32-bit" level I version. We will describe a 2^-96 complexity forgery attack on it.
>
> DME-Sign is built on a "trapdoor function" (in the style of RSA); there is a secret mapping from 256 bits to 256 bits (used for creating signatures) and a matching public mapping which is its inverse (for verifying signatures).

From: pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Wednesday, July 26, 2023 12:36 PM
To: Bros, Maxime P. (IntlAssoc)
Cc: pqc-forum
Subject: [pqc-forum] Re: Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

On Wed, Jul 26, 2023 at 4:26 PM Maxime Bros <maxime.bros@nist.gov> wrote:
(..)

> While it is pretty clear throughout the specs that there are values that make it impossible to get a bijection from (F_2^32)^8 -> (F_2^32)^8, the actual verification algorithm page 7 makes it look like all signature of length 256 could be inputs, not raising any error.

Hi Maxime,

You're right in that the mathematical descriptions discuss non-invertibility. However, the reference implementation does not perform any checks for this condition, nor are any included in the algorithm pseudocode. So clearly DME-Sign does not have these checks. Furthermore, the authors do not discuss the implications of non-invertibility -- no security analysis related to this condition is provided.

In many ways, this is similar to the issues we've had with ALTEQ, MEDS, and LESS -- it's a cryptanalytically exploitable input validation problem. They may seem trivial in hindsight, but -- on the other hand -- the design teams themselves have made these omissions with both code and specification. I don't mean to make these issues seem more important than they are, but recall that attackers will generally use the easiest way to break a security system, rather than the expected one.

> However, if we look carefully at the definition of dme-enc and at the bottom of page 5 and the beginning of page 6 of the specs, it is said “It is easy to verify that E […] do not have a zero entry”.

My first reading of this passage was that the authors are actually discussing why and when dme-dec is the inverse of dme-enc, rather than suggesting any kind of concrete algorithmic check for the signature verification function. But yes, they mention the mathematical properties that cause the mapping to not work.

If the team is to fix this, they should clarify what are the "entries" of the multiplicative subgroup of the extension field (F*_q^2)^4 that need to be checked. Perhaps entries of F_q rather than F_q^2 (32-bit rather than 64-bit)? One could also check for multiplicative identity, which is unchanged under the "squaring" f mapping, and effectively reduces the keyspace.

> All this to say that if their reference/optimized implementations do not raise error in that case, it is an issue and your attack is perfectly valid.
> But an easy tweak to their implementations, to make them follow their specs, would make the signature resist to your attack ; in addition to this, performance-wise, I do not think that these verifications are costly at all.

I've double-checked against the reference implementation; there are no checks. Indeed, the algorithmic descriptions explicitly state that full range is allowed --- line 1 of dme-open in the "Implementation\ of\ DME-3rnds-8vars-32bits-sign.pdf" document explicitly states that s is in {0,1}^256.

> I do not mean that the scheme is secure, just that your attack seems to exploit what looks like an implementation mistake to me, not a flaw in the signature scheme.

This is an error in both the algorithm specification and its implementation. Exploiting something like a buffer overflow or similar would be more clearly an implementation error, but the implementation matches with the spec here. The implementation actually adds to it -- there's an additional "g" hash check in the verification function that the spec doesn't mention, as discussed at the end of my message.

ps. I'd like to add that there's a typo in step 2 of the linearization attack description below, "s[4..7]" are treated as constants after step 1, not m[6..7]. Please refer to the description and PoC at https://github.com/mjosaarinen/dme-py rather than the email below for a technical description.

Cheers,
- markku

> Sincerely,
>
> Maxime Bros
>
> On Tuesday, July 25, 2023 at 5:56:01 PM UTC-4 Markku-Juhani O. Saarinen wrote:
> > Hi All,
> >
> > PoC code discussed below can be found at: https://github.com/mjosaarinen/dme-py
> >
> > **Inverting**
> >
> > The multivariate candidate DME-Sign has three submitted parameter sets, q in {2^32,2^48,2^48} for NIST security targets I, III, and V. In the following discussion, I will concentrate on the "32-bit" level I version. We will describe a 2^-96 complexity forgery attack on it.
> >
> > DME-Sign is built on a "trapdoor function" (in the style of RSA); there is a secret mapping from 256 bits to 256 bits (used for creating signatures) and a matching public mapping which is its inverse (for verifying signatures).
> >
> > The file invert_demo.py demonstrates how to invert half of this mapping quickly; given a public key and a target for the first 128 bits of the "secret" side of the permutation, it selects 128 bits in the signature side that matches it.
> >
> > *$ python3 invert_demo.py*
> > *=== count = 0*
> > *m1 = 060a1e26 6cb1fe61 0ba18e4c 1b9a410b 8e1c10e0 a3417574 140bcf0a 159b1899*
> > *m2 = 060a1e26 6cb1fe61 0ba18e4c 1b9a410b 8e1c10e0 a3417574 140bcf0a 159b1899 [OK] simplified mapping match.*
> > *m3 = 53579029 d9eb70e6 08090a0b 0c0d0e0f 7cb8b25e 4de5ac5c 142325cb 7b5bda96*
> > *[OK] linear mapping to m[2:4]*
> > *sv = fe9c5602 108eb7c6 00000000 00000000 0b0d9ad1 be13a58c 01234567 89abcdef*
> > *m4 = 00010203 04050607 08090a0b 0c0d0e0f 7cb8b25e 4de5ac5c 142325cb 7b5bda96*
> > *[OK] Half of function inverted!*
> >
> > The same public keys are generated as in the KAT test vectors. The first comment, "simplified mapping match," indicates that the simplified algebraic description (below) is working fine -- the final comment indicates that the first

From: pqc-forum@list.nist.gov on behalf of ilu...@ucm.es <iluengo@ucm.es>
Sent: Friday, July 28, 2023 1:17 AM
To: pqc-forum
Cc: Markku-Juhani O. Saarinen; pqc-forum; Bros, Maxime P. (IntlAssoc)
Subject: [pqc-forum] Re: Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

Dears Markku and Maxime,

We would like to thank you for pointing out a missing check In our implementation of
the crypto_sign_open() function, and several unintentional omissions in the documentation of the algorithms.

In the description of dme-sign it is implicit that a signature s is not valid if interpreted
as a vector in (F_q2)^4 has a zero entry and we acknowledge that it has to be stated more explicitly.

As specified in the second last paragraph of sect 5 of NIST document (dme-sig.pdf) and in remark 44 of [1],
at each step of the computation of F^{-1}(pad(m)) we check if there is a zero component , in case affirmative
we start with a new padding pad(m) and in particular final signature dme-dec(w||g) has no zeroes as a vector in (F_q2
)^4.
We have this flag in the different implementations on our web but we forgot to include it in the
implementation sent to NIST due to last minute rush.

We were absolutely aware of the possibility of the attacks with zeros (thus the reason we say that dme-enc is a bijection when restricted to a map from D to E), but we forgot to perform "input validation" in the dme-open() function code. The omission can be easily fixed (as indicated by one of the reviewers), and we hope it does not preclude DME from being considered as a signature scheme candidate.

To avoid the proposed attack, only the following line has to be added to the pseudo-code of the dme-open() function.

2. if the interpretation of s as a vector in (Fq2)4 has a zero entry, return error,

The website has been updated with the revised code and a matching
specification.

Thanks again for your comments and interest,

Best regards,
Ignacio Luengo

[1] "DME: A Full Encryption, Signature and KEM Multivariate Public Key Cryptosystem", to appear in
T. Johansson and D. Smith-Tone (Eds.): PQCrypto 2023, LNCS 14154, pp. 1–24, 2023.

El miércoles, 26 de julio de 2023 a las 18:36:30 UTC+2, Markku-Juhani O. Saarinen escribió:

> On Wed, Jul 26, 2023 at 4:26 PM Maxime Bros <maxim...@nist.gov> wrote:
> (..)
>
> While it is pretty clear throughout the specs that there are values that make it impossible to get a bijection from (F_2^32)^8 -> (F_2^32)^8, the actual verification algorithm page 7 makes it look like all signature of length 256 could be inputs, not raising any error.

From: pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Friday, July 28, 2023 5:27 AM
To: pqc-forum
Cc: ilu...@ucm.es; Markku-Juhani O. Saarinen; pqc-forum; Bros, Maxime P. (IntlAssoc)
Subject: [pqc-forum] Re: Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

On Friday, July 28, 2023 at 6:16:38 AM UTC+1 ilu...@ucm.es wrote:

> Dears Markku and Maxime,
>
> We would like to thank you for pointing out a missing check In our implementation of
> the crypto_sign_open() function, and several unintentional omissions in the documentation of the algorithms.

(..)

> To avoid the proposed attack, only the following line has to be added to the pseudo-code of the dme-open() function.
>
> 2. if the interpretation of s as a vector in (Fq2)4 has a zero entry, return error,
>
> The website has been updated with the revised code and a matching
> specification.
>
> Thanks again for your comments and interest,

Hello Ignacio,

If I understand correctly, the revised pseudocode of dme-open (for the Level I variant DME-3rnds-8vars-32bits-sign) with this newly added line becomes:

1. let (msg, s) in {0,1}^∗ × {0,1}^256 be the input message and its corresponding signature
2. (*new*) If the interpretation of s as a vector in (Fq2)4 has a zero entry, return error,
3. compute w in {0,1}^128 and g in {0,1}^128 as w || g = dme-enc(s),
4. compute r in {0,1}^64 as the first 64 bits of SHA3(w) xor g,
5. if w != SHA3(msg||r), return error,
6. otherwise, return the original message msg.

The simpler verification issue (briefly mentioned at the end of my message) has not been addressed with this single modification. I'll describe how that leads to a 2^96 forgery attack as well.

We observe that the only check leading to rejection is in step 5, which compares the 128-bit quantity w to truncated SHA3(msg || r), where r = SHA3(w) xor g truncated to 64 bits.

In other words, only low 192 bits of the output of dme-enc(s) are used in the verification; the high 64 bits of g can have any value.

For the sake of simplicity, let's fix r=0. We have a successful forgery for a given public key if we can find s and m such that the low 192 bits of these two functions match:

f1_192(m) = SHA3_128(m || 0^64) || SHA3_64(SHA3_128(m || 0^64))
f2_192(s) = dme_enc(s) truncated to 192 bits

We can use a sqrt(2^192) = 2^96 collision search to find a match between these two functions: f1(m) == f2(s) indicates that s is a valid signature for message m. (Cycle-based classical memoryless collision search algorithms can be used with slight additional cost by treating the f1/f2 "function selector" as an extra input bit.)

This particular issue can be remediated by adding a check for the high part of g against the high part of SHA3_128(w). As noted, this was actually done by the reference code I saw but not in the specification (description of dme-open() in page 7.)

ps. I'm struggling to find this web site with the updated code and specification.. The link in the "cover sheet" submitted to NIST (gauss.mat.ucm.es/dme.html) doesn't work currently and has never been indexed by archive.org. I can only find some things that are 5+ years old and related to the previous NIST call.

Cheers,
- markku
--

From: pqc-forum@list.nist.gov on behalf of ilu...@ucm.es <iluengo@ucm.es>
Sent: Friday, July 28, 2023 12:05 PM
To: pqc-forum
Cc: Markku-Juhani O. Saarinen; ilu...@ucm.es; pqc-forum; Bros, Maxime P. (IntlAssoc)
Subject: [pqc-forum] Re: Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

Hi Markku,
the web page is gauss.mat.ucm.es/dme/ or gauss.mat.ucm.es/dme/index.html, there is an issue with the certificate but the page works.
In particular the revised code nist-pqc-2023-rev1.can be downloaded from the page gauss.mat.ucm.es/dme/code.html.

If I understand correctelly the check that you mention is made on all the bits not only in the low 192 bits as mentioned in Implementation of DME-3rnds-8vars-32bits-sign.pdf, I will check it and get back to you as soon as possible.
We will check this issue with Martin who made the implementation and is now in a vacation trip in Sud America with difficult access to Internet for a few days.

Thanks for your interest,
Ignacio

El viernes, 28 de julio de 2023 a las 11:27:19 UTC+2, Markku-Juhani O. Saarinen escribió:

> On Friday, July 28, 2023 at 6:16:38 AM UTC+1 ilu...@ucm.es wrote:
>
> > Dears Markku and Maxime,
> >
> > We would like to thank you for pointing out a missing check In our implementation of
> > the crypto_sign_open() function, and several unintentional omissions in the documentation of the algorithms.
>
> (..)
>
> > To avoid the proposed attack, only the following line has to be added to the pseudo-code of the dme-open() function.
> >
> > 2. if the interpretation of s as a vector in (Fq2)4 has a zero entry, return error,
> >
> > The website has been updated with the revised code and a matching
> > specification.
> >
> > Thanks again for your comments and interest,
>
> Hello Ignacio,
>
> If I understand correctly, the revised pseudocode of dme-open (for the Level I variant DME-3rnds-8vars-32bits-sign) with this newly added line becomes:
>
> 1. let (msg, s) in {0,1}^∗ × {0,1}^256 be the input message and its corresponding signature
> 2. (*new*) If the interpretation of s as a vector in (Fq2)4 has a zero entry, return error,
> 3. compute w in {0,1}^128 and g in {0,1}^128 as w || g = dme-enc(s),
> 4. compute r in {0,1}^64 as the first 64 bits of SHA3(w) xor g,
> 5. if w != SHA3(msg||r), return error,
> 6. otherwise, return the original message msg.
>
> The simpler verification issue (briefly mentioned at the end of my message) has not been addressed with this single

From: pqc-forum@list.nist.gov on behalf of ilu...@ucm.es <iluengo@ucm.es>
Sent: Wednesday, August 2, 2023 12:27 AM
To: pqc-forum
Cc: ilu...@ucm.es; Markku-Juhani O. Saarinen; pqc-forum; Bros, Maxime P. (IntlAssoc)
Subject: [pqc-forum] Re: Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

Dear Markku,
I check with Martin, and as you say in the reference implematation the check is for all the bits of g. It was a mistake in the implementation paper.
The corrected version is in the web I mention in my last message. gauss.mat.ucm.es/dme/.
We apreciate very much your interest and carefull checking ogf the DME-Sign,
Best regads, Ignacio

El viernes, 28 de julio de 2023 a las 11:05:02 UTC-5, ilu...@ucm.es escribió:

> Hi Markku,
> the web page is gauss.mat.ucm.es/dme/ or gauss.mat.ucm.es/dme/index.html, there is an issue with the certificate but the page works.
> In particular the revised code nist-pqc-2023-rev1.can be downloaded from the page gauss.mat.ucm.es/dme/code.html.
>
> If I understand correctelly the check that you mention is made on all the bits not only in the low 192 bits as mentioned in Implementation of DME-3rnds-8vars-32bits-sign.pdf, I will check it and get back to you as soon as possible.
> We will check this issue with Martin who made the implementation and is now in a vacation trip in Sud America with difficult access to Internet for a few days.
>
> Thanks for your interest,
> Ignacio

El viernes, 28 de julio de 2023 a las 11:27:19 UTC+2, Markku-Juhani O. Saarinen escribió:

> On Friday, July 28, 2023 at 6:16:38 AM UTC+1 ilu...@ucm.es wrote:
>
> > Dears Markku and Maxime,
> >
> > We would like to thank you for pointing out a missing check In our implementation of
> > the crypto_sign_open() function, and several unintentional omissions in the documentation of the algorithms.
> (..)
> > To avoid the proposed attack, only the following line has to be added to the pseudo-code of the dme-open()
> > function.
> >
> > 2. if the interpretation of s as a vector in (Fq2)4 has a zero entry, return error,
> >
> > The website has been updated with the revised code and a matching
> > specification.
> >
> > Thanks again for your comments and interest,
>
> Hello Ignacio,
>
> If I understand correctly, the revised pseudocode of dme-open (for the Level I variant DME-3rnds-8vars-32bits-sign)

From: pqc-forum@list.nist.gov on behalf of Smith-Tone, Daniel <daniel-c.smith@louisville.edu>
Sent: Tuesday, September 5, 2023 12:44 PM
To: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: DME Key Recovery Attack

Hello, Community,

We have discovered an efficient key recovery attack on DME. We have implemented this attack and it recovers an equivalent secret key for the 32-bit variant in less than half a second. Using the generated equivalent secret key we can successfully forge any signature as efficiently as the legitimate signer. We have communicated our results to the DME Team. They are still studying the attack, but acknowledge that our approach seems reasonable. The DME Team communicated to me at PQCrypto that they have another variant of DME that may resist this analysis, so we may soon have another interesting object to study.

The attack relies on a few properties of DME that are used in its design to achieve its efficiency. The key properties are the fact that the linear layers are block-wise diagonal with each linear map only mixing adjacent coordinates and the fact that the exponential matrices are limited to two nonzero coordinates per row.

The attack works by lifting the representation of the public key over F=GF(q) to E=GF(q^2), where we can consider the public key as acting on four variables over E (instead of 8 variables over F). Then the linear layers work merely coordinate-wise and have a near commutative property with power-of-two maps (the components of the exponential layers). Viewing the big field representation of the key allows us to find the structure of the exponential layers without guessing, and by raising coordinates to appropriate powers, we can solve for the unknown coefficients of the linear layer maps and inputs to the last exponential layer by solving a bilinear system at low degree. This process removes one of the layers of the DME construction, and the technique can be applied repeatedly until all of the layers are removed.

Cheers,
Daniel Smith-Tone
On behalf of
Pierre Briaud
Maxime Bros
Ray Perlner
and myself

From: pqc-forum@list.nist.gov on behalf of ilu...@ucm.es <iluengo@ucm.es>
Sent: Tuesday, September 5, 2023 5:26 PM
To: pqc-forum
Cc: Smith-Tone, Daniel
Subject: [pqc-forum] Re: OFFICIAL COMMENT: DME Key Recovery Attack

Dear all,
I confirm the information given by Daniel. Our first impression is that the attack works and we are checking the details of the attack .
We are implementing a variant of the DME that may resist the attack but we have to verify it.
We thank Daniel and the NIST PQC team for his effort and good work.
Best regards,
Ignacio Luengo
El martes, 5 de septiembre de 2023 a las 18:43:47 UTC+2, Smith-Tone, Daniel escribió:

> Hello, Community,
>
> We have discovered an efficient key recovery attack on DME. We have implemented this attack and it recovers an equivalent secret key for the 32-bit variant in less than half a second. Using the generated equivalent secret key we can successfully forge any signature as efficiently as the legitimate signer. We have communicated our results to the DME Team. They are still studying the attack, but acknowledge that our approach seems reasonable. The DME Team communicated to me at PQCrypto that they have another variant of DME that may resist this analysis, so we may soon have another interesting object to study.
>
> The attack relies on a few properties of DME that are used in its design to achieve its efficiency. The key properties are the fact that the linear layers are block-wise diagonal with each linear map only mixing adjacent coordinates and the fact that the exponential matrices are limited to two nonzero coordinates per row.
>
> The attack works by lifting the representation of the public key over F=GF(q) to E=GF(q^2), where we can consider the public key as acting on four variables over E (instead of 8 variables over F). Then the linear layers work merely coordinate-wise and have a near commutative property with power-of-two maps (the components of the exponential layers). Viewing the big field representation of the key allows us to find the structure of the exponential layers without guessing, and by raising coordinates to appropriate powers, we can solve for the unknown coefficients of the linear layer maps and inputs to the last exponential layer by solving a bilinear system at low degree. This process removes one of the layers of the DME construction, and the technique can be applied repeatedly until all of the layers are removed.
>
> Cheers,
> Daniel Smith-Tone
> On behalf of
> Pierre Briaud
> Maxime Bros
> Ray Perlner
> and myself

From: 'Ignacio Luengo Velasco' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Thursday, February 8, 2024 7:02 AM
To: pqc-comments
Cc: pqc-forum
Subject: [pqc-forum] Round 1 (Additional Signatures) OFFICIAL COMMENT: DME-Sign

We are announcing an updated version of the DME signature scheme, in response to the attack [] that could recover an equivalent private key for a given public key. The new version, called DME^(-), has some extra security measures that make the vulnerability much harder to exploit.

The DME^(-) private key is the standard DME, which defines a public key polynomial map Fq^8 -> Fq^8, but four of those polynomials are removed from the public key, reducing it to a surjective map Fq^8 -> Fq^4. By doing so, we reduced the amount of coefficients in the public key by a factor of 2:1, and at the same time we speeded up the encryption (signature verification) time. The task of the attacker will be much harder here, since they have fewer equations and more unknowns (the coefficients of the polynomials that were deleted) to work with.

The size of the field Fq had to be increased to q=2^128 to keep the security level 5, and an extra exponential round was added and the polynomials have (85,85,109,109) monomials. By using the CLMUL instruction available in processors supporting Intel AVX2, the new version is only 4 times slower than the version for q=2^64. The timings for our CLMUL optimized implementation are shown in the following table for a 200 bytes plaintext

sch. parameters	keygen	sign	open	skey	pkey	Sign
DME^(−)(4,8,128)	2477	136	54	1619	6215	128
DME-Sign(4,8,128)	1530	108	38	1299	6535	128

The source code is available at www.mat.ucm.es/DME.

In the file dme-minus-2024.zip there is an implementation of the standard DME-Sign(4,8,128) with the same configuration matrices as above for testing and comparison with DME^(-). DME^(-)-Sign has 4 polynomials in the public key instead of 8 DME-Sign

but DME-Sign is faster because DME-Sign the number of monomials is (48,48,54,54) because it has not affine translation.

A further increase of the size of the field and the number of round would still be possible without any major problem, and still be competitive with other proposed signature schemes.For instance using 5 rounds the number of monomials of DME^(-)-Sign will pass from (85,85,109,109) to (120, 120, 180, 180) that will double the sizes and timings.

--