From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, July 17, 2023 1:09 PM
To: pqc-comments
Cc: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: KAZ-SIGN
Attachments: signature.asc

Running the Sage script below in the
KAZ-SIGN/Reference_Implementation/kaz458 directory rapidly forges a signature on any desired message under essentially any desired public key, and checks that the signature passes verification with the reference crypto_sign_open() software.

The script uses a particular message and the first public key in *.rsp as an example, but I've also tested it with another random public key and with various further messages. The reason I'm saying "essentially"
is that the KAZ-SIGN integer encoding looks like it will fail for 1/256 of all possible inputs; the reference software doesn't seem to handle this, and this Sage script also doesn't handle this.

---D. J. Bernstein

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

import os
import subprocess
import ctypes
from ctypes import c_int,c_char_p,c_ulonglong,POINTER,byref,create_string_buffer
import hashlib
import random
def hash(seed): h = hashlib.sha256(); h.update(seed); return h.digest()

proof.all(False)

# ----- copied from kaz_api.h

N =
37470874733837919416563211326754079989324849463818175868172713496859968436633910633680216649416805806774541289433279788468718778634973256 5
PHIN =
71467427390759841729059757466289459181369050713019533645557376916391119997637068708795979336636964345506399166398464000000000000000000000000
G =
37260025342153877976366022431631274000323014010699999970111761875964418126632549841893154834592996350134421573562483983305651325914860373 3
ORDERG = 144070022526464542998162540305862391968000
PHIORDERG = 17966317053413597259085197820821504000000
R =
64411872697932469627298024328629501373966883091743836804603337113399879222101827567038858750269057628226431053470498775743682787912336229
```

```
ORDERR =
8815508518609347572770628774410485749927476308817571946515551877469252673303656529920000000000000
00000000

ALPHABYTES = 18
VBYTES = 59
S1BYTES = 19
S2BYTES = 58
SALTBYTES = 4

# ----- public information copied from *.rsp

mlen = 32
msg = 'D81C4D8D734FCBFBEADE3D3F8A039FAA2A2C9957E835AD55B22E75BF57BB556A'
pk =
'2020C105E4CE23ABB476713D7805654CF78802EF11CA4B6903B0407FE23897F33CD4B41A4A15AF68E7BCD6B486920E
5D6E42E5C3E86ECF6FF57F49'
sm =
'20019D011CE55E5F96EACC650084407061DC0520085CB9DACF314194F1F254D8EAF6D815D5D7B9D82FDD0D0AE1C63
F4B9C0FA19DE06D640FFF775FA8DAB052D8576CB53AB7DEF64C26E038B6C2D81C4D8D734FCBFBEADE3D3F8A039FAA
2A2C9957E835AD55B22E75BF57BB556A7C9935A0'

# ----- miscellaneous tests

assert PHIN == euler_phi(N)
assert ORDERG == Mod(G,N).multiplicative_order() assert ORDERR == Mod(R,PHIN).multiplicative_order()

msg = bytes.fromhex(msg)
pk = bytes.fromhex(pk)
sm = bytes.fromhex(sm)

def decode(b):
  b = bytearray(b)
  while b[:1] == b' ': b = b[1:]
  return sum(c<<(8*i) for i,c in enumerate(reversed(b)))

def encode(i,targetbytes):
  result = bytearray()
  while i > 0:
    result = bytearray([i%256])+result
    i >>= 8
  while len(result) < targetbytes:
    result = bytearray([32])+result
  assert len(result) == targetbytes
  return bytes(result)

assert decode(encode(31415,5)) == 31415

def open(sm,pk):
  s1,sm = sm[:S1BYTES],sm[S1BYTES:]
  s2,sm = sm[:S2BYTES],sm[S2BYTES:]
  m,salt = sm[:-SALTBYTES],sm[-SALTBYTES:]
```

```
  assert len(s1) == S1BYTES
  assert len(s2) == S2BYTES
  assert len(salt) == SALTBYTES
  h = hash(m+salt+m+salt)
  pk,s1,s2,h = map(decode,(pk,s1,s2,h))
  assert Mod(G,N)^(Mod(s1,PHIN)^s2) == Mod(pk,N)^(Mod(R,PHIN)^h)
  return m

subprocess.run('gcc -shared -o libkaz.so kaz_api.c sign.c rng.c sha256.c -fPIC -lcrypto -lgmp',shell=True)

libkaz = ctypes.CDLL(f'{os.getcwd()}/libkaz.so')
libkaz_open = libkaz.crypto_sign_open
libkaz_open.argtypes = c_char_p,POINTER(c_ulonglong),c_char_p,c_ulonglong,c_char_p
libkaz_open.restype = c_int

def reference_open(sm,pk):
  smlen = c_ulonglong(len(sm))
  m = create_string_buffer(len(sm))
  mlen = c_ulonglong(0)
  pk = create_string_buffer(pk)
  assert libkaz_open(m,byref(mlen),sm,smlen,pk) == 0
  return m.raw[:mlen.value]

assert open(sm,pk) == msg
assert reference_open(sm,pk) == msg
assert reference_open(sm,pk) == msg

phiphin = euler_phi(PHIN)
realRorder = Mod(R,ORDERG).multiplicative_order()

def forge(m,pk):
  salt = os.urandom(SALTBYTES)
  h = hash(m+salt+m+salt)
  pk = decode(pk)
  h = decode(h)
  r = random.randrange(2**256)
  while not Mod(r,ORDERG).is_unit(): r += 1
  s1 = ZZ(Mod(R,ORDERG)^r)
  loggV = Mod(pk,N).log(Mod(G,N))
  assert Mod(G,N)^loggV == Mod(pk,N)
  alpha = Mod(loggV,ORDERG).log(Mod(R,ORDERG))
  alpha += realRorder*random.randrange(2**256)
  assert Mod(R,ORDERG)^alpha == Mod(loggV,ORDERG)
  s2 = ZZ(Mod(alpha+h,ORDERG)/r)
  s2 += ORDERG*random.randrange(2**256)
  s2 %= phiphin
  assert Mod(G,N)^(Mod(s1,PHIN)^s2) == Mod(pk,N)^(Mod(R,PHIN)^h)
  return encode(s1,S1BYTES)+encode(s2,S2BYTES)+m+salt

newmsg = b'forged message'
while True:
  sm = forge(newmsg,pk)
```

```
 try:
   assert newmsg == open(sm,pk)
   assert newmsg == reference_open(sm,pk)
   break
 except:
   pass

assert newmsg == open(sm,pk)
assert newmsg == reference_open(sm,pk)
print(f'newmsg: {newmsg}')
print(f'sm: {sm.hex()}')
```

From: 'Scott Fluhrer (sfluhrer)' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Monday, July 17, 2023 1:46 PM
To: Scott Fluhrer (sfluhrer); pqc-forum
Subject: [pqc-forum] RE: KAZ-SIGN

Immediately after hitting send, I noticed that they specify that N is a product of a number of smallish primes; this makes the discrete log problem easy – however it makes all the steps I outlined below in solving the hard problem also easy for a classical computer…

From: 'Scott Fluhrer (sfluhrer)' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Monday, July 17, 2023 1:28 PM
To: pqc-forum@list.nist.gov
Subject: [pqc-forum] KAZ-SIGN

I have examined KAZ-SIGN (https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/kaz-sign-spec-web.pdf), and it would appear that the hard problem it relies on is not Quantum Resistant (and hence the signature algorithm it is based on is not).

The problem they state is, given A, g, N (composite) and Q, find x such that:

g^{Q^x} = A \bmod N

However, it would appear that three applications of Shor's algorithm would be sufficient to recover x:

1) Find z with g^z = A \pmod N
2) Factor N to be able to compute \phi(N) (actually, with KAZ, we don't need to do this step, as it has \phi(N) in the system parameters)
3) Find x with Q^x = z \pmod \phi(N)

Being able to solve this allows us to recover the private key from the public key

Section 4 of the submission (which covers Quantum Resistance) only addresses Grover's algorithm.

In addition, this being hard is also in conflict with the key generation, signing and verification algorithms they give (Algorithms 1-3) – those have several steps where you are expected to solve a discrete logarithm (either to base N or to base \phi(N)).

Hence, unless either I completely misunderstood this submission, I think we can drop this one.

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, July 18, 2023 5:36 AM
To: pqc-forum
Subject: Re: [pqc-forum] KAZ-SIGN
Attachments: signature.asc

KAZ-SIGN, like SIKE, doesn't claim that discrete logs are hard. It says that they're easy for the selected groups. It uses them in the stated algorithms and software. It then argues that the ways that the attacker can use discrete logs are blocked by the details of the verification process. See the documentation, especially Sections 8 and 9.

'Scott Fluhrer (sfluhrer)' via pqc-forum writes:
> Being able to solve this allows us to recover the private key from the
> public key

No. There are many solutions to this equation, and whichever solution you pick (even if you get past the existence question in your step 3) has negligible chance of being the private key.

Perhaps you meant "a private key", but then you have to define this concept and, more to the point, argue that signatures produced by this replacement key will pass verification.

It's not hard to see that they won't _always_ pass. A full analysis is trickier than one might expect. In any case, for verifiability, claims of fast attacks should be consistently backed up by scripts that demonstrate those attacks against the official software.

(In cases where quantum computers are claimed to be essential for a fast attack, the risk of error is even higher, and there should be even more attention to applying known error-detection techniques.)

I wrote an attack script in a way that should be reasonably robust against variations in the verification details. I don't see how the verification procedure could be tweaked to block this attack while still accepting the signatures generated by the legitimate signer.

---D. J. Bernstein

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Thursday, August 3, 2023 3:20 AM
To: pqc-forum
Subject: [pqc-forum] [KAZ-SIGN OFFICIAL]

Dear all,

First and foremost, KAZ-Team would like to extend our thanks to Prof. Bernstein for his comment/concern. It was really an eye opener. It has been approximately 2 weeks since Prof. Bernstein hinted about tweaking KAZ-SIGN to identify the forged signature equation. As Prof. Bernstein correctly pointed out in his response to an earlier comment, KAZ-SIGN aspires not to be dependent on the Discrete Logarithm Problem. Rather we coin our hard problem as the Second Order Discrete Logarithm Problem (2-DLP) as defined in our Algorithm Specifications and Supporting Documentation write-up.

The design of KAZ-SIGN also aspires to be flexible, that is when a forgery mechanism has been obtained, KAZ-SIGN can identify such forgery during verification.

We note here that, in our endeavor after Prof. Bernstein comments, the aim is to tweak the signature scheme to be able to identify the forged signature without amending the foundations and increasing the key and digital signature lengths significantly. As an overview, minor changes have been made and we have managed to omit time consuming procedures and also the need to use salt.

Let us recall; let Gg be the order of g in N. That is, g^{Gg}=1 mod N. Let GRg be the order of R in Gg. That is, R^{GRg}=1 mod Gg.

To this end, we have realized that the 2-DLP is an "expensive" way of describing the underlying mathematical hard problem of KAZ-SIGN. In reality, the private signing key α is kept unknown to the adversary via the relation $\alpha = \alpha F \bmod GRg$. This situation can be viewed via $t=(\alpha - \alpha F)/GRg$. When αF and GRg are known, the length of t, will determine the difficulty level of retrieving back the private signing key α. For KAZ-SIGN documentation formality we coin this as the Modular Reduction Problem (MRP). Details are in the write-up.

As for the extra overhead, the private signing key and public verification key have increased. It has increased approximately 400, 700 and 1,000 bits for the public verification keys for security levels 1, 3 and 5. As for the private signing key, it sees an increase of approximately 128, 192 and 256 for security levels 1, 3 and 5. However, the signature size remains approximately the same.

Even though the size has increased, KAZ-SIGN procedures has been strip down from its heavy computations and hence a speed up of more than 500% (to say the least). One can make comparison from both write-ups for a more accurate figure.

In brief we defined the public verification key-1 as $V1=\alpha \bmod GRg$, the public verification key-2 as a

random k-bit prime (where k is the security level value 128 or 192 or 256) and V3= α mod V2. Observe that to obtain α from V1 or V3 is the MRP.

Then we sign as S1=R^(r mod GRg) mod Gg, S2=(α ^(r mod V2) +h)/r mod GRgV2 and S3=r mod V2.

The verification procedure is as follows:

Compute w0=S2S3-h mod V2 and w1=V3^(S3) mod V2. If w0 ≠ w1 reject signature. Else continue to verify (the same procedures in our NIST submission, see write-up).

Note, the suggested Prof. Bernstein forgery mechanism (which can be utilized for this new setup) is of the form S2F=(V1^(r mod V2)+h+GRgx)/r mod GRgV2 for some random x in Z_{GRg} (Prof. Bernstein's random structure is x=x1+x2r for some random x1 and x2). S2F will still pass our verification procedures. But it will not pass our filtering procedures via computing w0 and w1 and making comparison.

Its clear that w0=S2FS3-h mod V2 and w1=V3^(S3) mod V2 would result in w0 ≠ w1. This is because S2FS3 - h ≠ V3^(S3) mod V2.

Thus, the adversary will have to forge S3 also (i.e. denoted as S3F). To do this, the adversary will need to resolve

V3^(S3F)-S2FS3F+h = 0 mod V2. After substitutions, this would mean the adversary needs to solve V3^(S3F)-V1^(S3F)-GRgx = 0 mod V2 ---(eq A). And after obtaining S3F, the adversary can set S1=R^(S3F) mod Gg. All these forged parameters will pass the filtering and verification procedures.

Observe that, to solve (eq A), the complexity is O(V2). Furthermore, V2 is a prime number of k-bits and the adversary will not be able to execute the Chinese Remainder Theorem to reduce this complexity.

KAZ-Team has setup a website (https://www.antrapol.com/KAZ-SIGN) that lists our C codes and Algorithm Specifications and Supporting Documentation write-up according to version. The version sent to NIST prior to Prof. Bernstein comment will be known as Version 1.0 (v1.0). The version with the new procedures as mentioned above will be known as Version 1.1 (v1.1).

Thank you for the precious comment. KAZ-Team hopes that this minor tweak will not be a hindrance for KAZ-SIGN to be further evaluated.

Best regards

KAZ-Team

--

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, August 3, 2023 5:00 AM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]
Attachments: kaz11-forge.sage; signature.asc

Here's a Sage script forging signatures for the new version of KAZ-SIGN and checking that the forgeries are accepted by the reference software.
For convenience, this script automatically extracts parameters and public keys from the KAZ-SIGN code in the current directory, and automatically tries all public keys from the KATs. I've checked that the script works in the kaz980, kaz1703, and kaz2311 directories.

---D. J. Bernstein

--

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Thursday, August 17, 2023 5:47 AM
To: pqc-forum
Subject: [pqc-forum] [KAZ-SIGN OFFICIAL]

Dear all,

First and foremost, KAZ-Team would like to extend our heartfelt thanks to Prof. Bernstein for his insights again. The usage of the Chinese Remainder Theorem (CRT) to construct a forgery of signature parameter S2 by utilizing given public parameters (V1,V3) upon modulars V2 and GRg does indeed bypass our existing filtering strategies.

In order to overcome this issue, it is necessary not to introduce more parameters that might open new avenues for forgery. A minor tweak was done on the definition of V2. The strategy is as follows:

The public verification key-1 as $V1 \equiv \alpha \bmod GRg$, the public verification key-2 is the product of a random k-bit prime, denoted as ρ (where k is the security level value 128 or 192 or 256) and the largest factor of GRg denoted as β. We denote the public verification key-2 as $V2=\beta\rho$. Finally the public verification key-3 is given by $V3 \equiv \alpha \bmod V2$. Observe that to obtain a from V1 or V3 is the MRP. To factor V2 is easy. We do not intend to deploy the integer factorization problem upon V2.

The signature procedure remains the same as v1.1. The signature parameters are $S1 \equiv R^{r \bmod GRg} \bmod Gg$, $S2 \equiv (\alpha^{r \bmod V2}+h)/r \bmod (GRgV2)$ and $S3 \equiv r \bmod V2$.

The verification procedure is similar to v1.1. At the same time, as a result of Prof Bernstein's 2nd input, we add the following procedures during verification:

Procedure 1: Compute $w0 \equiv S2S3-h \bmod V2$ and $w1 \equiv V3^{(S3)} \bmod V2$. If $w0 \neq w1$, reject the signature.

Procedure 2: From S1, solve the DLP to obtain rF, where $rF=DLog_{R} (S1 \bmod Gg)$. Compute $w2 \equiv (GRg/\beta)S3 \bmod GRg$ and $w3 \equiv (GRg/\beta)rF \bmod GRg$. If $w2 \neq w3$, reject the signature.

Procedure 3: Compute rF, where $rF=DLog_{R} (S1 \bmod Gg)$. Compute Chinese Remainder Theorem upon $w4 \equiv (V3^{S3}+h)S3^{-1} \bmod \rho$ and $w5 \equiv (V1^{S3}+h)rF^{-1} \bmod GRg$ to obtain $w6 \bmod \rho GRg$. Compute $w7=w6-S2$. If $w7=0$, reject the signature.

Further discussion on this matter can be seen in section 8 in v1.2 of our write-up on our website.

As for the extra overhead, the private signing key and public verification key have increased a few bits when compared to v1.1. That is approximate 35, 20 and 10 bits for the public verification keys for security levels 1, 3 and 5. As for the private signing key, it sees an increase of approximate 13 and 5 bits for security levels 1 and 3. There is no increase for security level 5. As for the signature, it sees an increase of

approximate 25, 20 and 10 bits for security levels 1, 3 and 5.

Even though the size has increased, KAZ-SIGN procedures still executes fast.

KAZ-Team has made available our C codes and Algorithm Specifications and Supporting Documentation on our website https://antrapol.com/KAZ-SIGN according to version.

Thank you for the precious comment. KAZ-Team hopes that this minor tweak will not be a hindrance for KAZ-SIGN to be further evaluated.

Best regards

KAZ-Team
Aug 17, 2023
--

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, August 17, 2023 9:14 AM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]
Attachments: kaz12-forge.sage; signature.asc

Here's a Sage script forging signatures for KAZ-SIGN 1.2 and checking that the forgeries are accepted by the reference software. I've checked that the script works in the kaz989, kaz1713, and kaz2311 directories.

---D. J. Bernstein

From: pqc-forum@list.nist.gov on behalf of Watson Ladd <watsonbladd@gmail.com>
Sent: Thursday, August 17, 2023 11:45 AM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]

Furthermore I propose the following to break all KAZ-SIGN 1.x

1: find the email announcing the tweak
2: wait four hours
3: read DJBs email breaking it

On Thu, Aug 17, 2023, 6:13 AM D. J. Bernstein <djb@cr.yp.to> wrote:

> Here's a Sage script forging signatures for KAZ-SIGN 1.2 and checking
> that the forgeries are accepted by the reference software. I've checked
> that the script works in the kaz989, kaz1713, and kaz2311 directories.
>
> ---D. J. Bernstein
>
> --
> You received this message because you are subscribed to the Google Groups "pqc-forum" group.
> To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.
> To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/20230817131334.2121348.qmail%40cr.yp.to.

--
You received this message because you are subscribed to the Google Groups "pqc-forum" group.
To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.
To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/CACsn0c%3DxpSZ_fM6-LV4s3mBfGeWTLrMKmDjM4_dHLe8oOMYopA%40mail.gmail.com.

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Thursday, August 31, 2023 12:10 PM
To: pqc-forum
Subject: [pqc-forum] [KAZ-SIGN OFFICIAL]

Dear all,

Based on Prof Bernstein's comments, KAZ-SIGN v1.3 is now available for scrutiny via https://antrapol.com/KAZ-SIGN.

All comments are welcomed.

Best wishes

KAZ-Team

Aug 31, 2023

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, August 31, 2023 8:19 PM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]
Attachments: kaz13-forge.sage; signature.asc

With high probability, a public key and signed message for KAZ-SIGN v1.3 allow the following script to forge signatures on attacker-chosen messages under that public key. The script checks that the signatures pass verification with the reference software. The success probability is 93/100, 90/100, 90/100 for the KATs in the kaz1662, kaz2667, kaz3783 directories respectively.

---D. J. Bernstein

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Monday, September 25, 2023 5:42 AM
To: pqc-forum
Subject: [pqc-forum] [KAZ-SIGN OFFICIAL]

Dear all,

Based on Prof Bernstein's comments on KAZ-SIGN v1.3, in the course of scrutinizing the 10% cases where forgery was reported not able to be conducted, we have obtained interesting information. KAZ-SIGN v1.4 is now available for scrutiny on our website https://antrapol.com/KAZ-SIGN.

All comments are welcomed.

Best wishes

KAZ-Team

--

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, September 25, 2023 11:02 AM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]
Attachments: kaz14-forge.sage; signature.asc

Here's a Sage script forging signatures for KAZ-SIGN 1.4 and checking that the forgeries are accepted by the reference software. I've checked that the script works for all 100 KATs in the kaz1509, kaz2321, and kaz3241 directories.

---D. J. Bernstein

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Thursday, February 1, 2024 11:31 PM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]

Dear all,

KAZ-SIGN v1.5 is now available on https://antrapol.com/KAZ-SIGN. We thank Prof Bersntein for discussions leading up to v1.5.

All comments are welcomed.

Best wishes

KAZ-Team
Feb 2, 2024
--

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Friday, February 2, 2024 4:24 PM
To: pqc-forum
Subject: Re: [pqc-forum] [KAZ-SIGN OFFICIAL]
Attachments: signature.asc

To answer some off-list questions: I haven't been able to schedule the time to look at KAZ-SIGN v1.5, and I don't have comments on it.

---D. J. Bernstein

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Friday, March 15, 2024 7:17 AM
To: pqc-forum
Subject: [pqc-forum] [KAZ-SIGN OFFICIAL]

Dear all,

KAZ Team thanks anonymous input regarding some missing information in the specification document outlining KAZ-SIGN version 1.5, released on Feb 2, 2024.

Namely,

1. During key generation, there is missing information. The parameter \alpha is a prime.
2. Steps 3,4,5 during signing is not updated. It should be h=nextprime(H(m||salt)).

The reference implementation is correct. Both:

1. Choosing \alpha as a prime,
2. Computing h=nextprime(H(m||salt)) is conducted during signing

are executed in KAZ-SIGN v1.5 reference implementaiton. Hence, no changes are needed on the KAZ-SIGN v1.5 C codes released on Feb 2, 2024.

We thank the anonymous individual that scrutinized the reference implementation against the specification document.

We label the updated specification document with these changes as KAZ-SIGN v1.5.1.

The write-up can be accessed at the following link https://www.antrapol.com/KAZ-SIGN

All comments are welcomed.

Best wishes

KAZ-Team
March 15, 2024

From: pqc-forum@list.nist.gov on behalf of MUHAMMAD REZAL BIN KAMEL ARIFFIN / FS <rezal@upm.edu.my>
Sent: Sunday, April 7, 2024 6:03 AM
To: pqc-forum
Subject: [pqc-forum] Re: [KAZ-SIGN OFFICIAL]

Dear all,

We put forward KAZ-SIGN v1.6. We would like to thank discussion opportunities with Kai Chieh Chang (Jay) and the team at Phison Architecture Design Department which triggered discussions that lead towards version 1.6, that resulted in reduced number of steps for KAZ-SIGN key gen, sign and verify algorithms.

KAZ-SIGN v1.6 can be accessed at https://antrapol.com/KAZ-SIGN

All comments are welcomed.

Best wishes

KAZ-Team
April 7, 2024
--