If I understand correctly, the current version of ac-main.pdf was completed on 2018. And I assume there is no update on your draft since 2019. Could you confirm this?

>>>> The draft has not been updated since 2018. Just a brief explanation of the attack before answering the specific questions.

However, the fault attack idea is pretty generic and the main idea is to change the structure of the NTT transform (Refer Fig.1 in the paper). If you refer to to the non-faulted NTT transform, you can see that every output coefficient is dependent on all input coefficients (Fig 1(a)). The main idea of the fault attack is to fault the NTT transform such that every output coefficient is only dependent on very few number of input coefficients (Fig 1(b)).

There are two ways to create an LWE instance.

(1) B = INTT(NTT(A) o NTT(S) + NTT(E)) -- In this case, we need to fault both the NTT(S) and NTT(E).

(2) B = INTT(NTT(A) o NTT(S)) + E – In this case, we need to fault both NTT(S) and INTT(S).

By doing the same, we can ensure that every coefficient in output B in both cases is only dependent on a very few coefficients of the input S and E. If the number of dependent coefficients are very low, we can simply look at each coefficient of B and brute-force the possible values for both S and E.

In the paper, we only looked at faulting from a high level abstraction (i.e) faulting a C variable). In particular, we considered faulting an internal variable (which we denoted as the distance variable). This C variable would either be stored in a memory location in the RAM or will be just kept in the registers (This will depend on the type of implementation we are looking at). But, irrespective of the implementation, the high level idea of the attack remains the same, irrespective of the implementation. Just how we are going to achieve the faulted behaviour, depends on the implementation that we target.

But, we can also explore other avenues such as reducing the number of computed stages in the NTT transform. One single stage combines two coefficients. So, if we fault such that only one stages is computed, then we ensure that each output coefficient is only dependent on two input coefficients. This also can lead to a brute force attack. So, the lesser number of stages we execute, the easier the brute force attack is. So, we can also try to look if we can directly fault the stage counter of the NTT operation.

[NewHope]

After the announcement of round 1 winners, there are two updates for NewHope: 2019-04-10: Version 1.02 and 2019-07-10: Version 1.03. Did you already aware the difference from the version change that will cause the previous attack in ac-main.pdf to fail? I have not taken a close look into it yet. If you know, could you specify that difference for me?

>> I have not looked at the exact implementation of the NTT in NewHope. As said earlier, the implementation of the NTT will only probably make us attack a different operation within the NTT. Since the attack is described from the algorithmic level, implementation changes might not necessarily impact it. But, we need to indeed analyze the implementation to find the vulnerable spot, it if exists.

I already checked the latest specification in 2019-07-10: Version 1.03. From the errata, I found that there is no significant update on the NTT of NewHope. Do this mean there is a high probability that your previous fault analysis can work well on the latest version?

>>> It could definitely be the case. Actually, in our paper, we did not analyze the assembly implementation to see which operations have to be exactly targeted. So, we again need to look at the assembly level implementation of the NTT to figure out. Moreover, nowadays most schemes use handwritten assembly code for NTT and also perform complex optimizations such as combining multiple stages etc. So, we need to see how the attack will work on these optimized implementations.

[KYBER]

After the announcement of round 1 winners, I think there is no major revision on the NTT part of KYBER. The only difference I found from the change log is listed as below. In my opinion, it just updated the expression and did not change the implementation of NTT. I assume your previous fault analysis can directly work on the latest version of KYBER, am I right?

>> Here again, the NTT implementation is handwritten using assembly. So, analysis of the assembly implementation of the NTT is required.

[Dilithium]

The round 2 version of Dilithium had a significant change. As the website says: “**As an update for round 2 of the NIST project we propose a variant of Dilithium, called Dilithium-AES, that uses AES-256 in counter mode instead of SHAKE to expand the matrix and the masking vectors, and to sample the secret polynomials.**” Due to this change, I assume your previous attack method may not work or it may need to adjust accordingly? Shall we start to work this target first?

>>> The Dilithium-AES and Dilithium-SHA only change the way, the sample generation is done. To the best of my knowledge, it does not have an impact on the way the NTT is performed. So, I think the attack might still work, subjected to the way the NTT has been implemented.

The specification also highlight such change, which can also be found in the public website: “**The other changes were in the implementation. We made various optimizations in the signing algorithm. The most important optimization is how the rejection condition based on the low part of the vector w and the hint vector is computed. Our AVX2 optimized implementation now makes more use of vectorization and includes a simpler assembler NTT implementation using macros.**”

As for me, I think the implementation of the NTT does matter the fault analysis.

>>> Yes, definitely the underlying assembly level implementation does make a huge difference in the way the fault attack works. Moreover, the optimization levels (O0 to O3) also significantly changes the assembly instructions generated by the compiler. When I was analyzing the assembly level implementations, I could see that O0 implementations (with no optimizations) are more easier to attack than O1-O3 optimizations.

After I read your version, I think I understood where the fault is injected and what impact such fault will bring. But I haven’t fully understood how you conduct the fault analysis and how to recover the key. I may need more time to explore the attack.

At your convenience, could you comment what I have concluded and give me some guidance. We can leave the discussion of physical experiment later.

>>> I think we need to first understand clearly how the attack works. We can have a detailed discussion over Skype if required. Will be more than happy to discuss and improve the understanding of the attack). Once the high level attack idea is understood, then we can analyze the NTT implementation of any of the schemes (Kyber, NewHope or Dilithium) and then try to identify the vulnerabilites. For the paper, we actually did attack simulations and not practial fault attacks. We have some attack scripts here, which can perform the analysis on the faulted NTT outputs. Finally, I would like to say that the attack is indeed a bit complicated, but let us give it a try, because if it works, it would be a very nice work!!

Bolin

Mark on 2019.11.12

Thanks for your reply. From 2019.11.6 till now, I think I almost understand how the attack works. And I have some more questions about the attack.

• First I want to ensure that I understand the attack correctly. We inject the fault to make the *distance* in NTT to be constant like 0 or C, then the public key will only be related to and .Then because *s* and *e* are sampled from a very narrow distribution, we can test all the combination of s and e (just for j-th coefficient) to satisfy the equation b=a\*s+e. That’s the whole fault analysis, and do I understand right?

• >>> Yes, I think the understanding is correct. The distance variable means the distance between the coefficients that are combined in a single butterfly operation in each stage. But, this distance variable might not be directly used, but might be a bit more implicit in different implementations. So, we need to

• see analyze where to fault and how that affects the implicit distance variable in the NTT implementation.

•

• Another point I want to confirm is: This attack has a large probability that will lead to an invalid LWE instance and the cipher will break down. So the attack can be used just in the situation that the same secret key will be used again next time (or we don’t consider the case for session key)?

• >>>> The reuse of randomness assumption is actually a BIG assumption and the reviewers complained last time we submitted the paper. So, we need to do away with the randomness assumption. There is one way we can actually bypass this assumption. This is in the Man In The Middle setting.

• This kind of attack was actually done in one other paper that we worked on, titled “Number "Not Used" Once - Practical fault attack on pqm4 implementations of NIST candidates” available in the link https://eprint.iacr.org/2018/211.pdf. (You can refer to Section 3.3 titled “message recovery attacks” for the same). If we assume a man in the middle attacker, we can perform an attack without the need for reuse of randomness. We can discuss in detail about this if need more understanding.

•

• One more important point to note is that if S is large, then the LWE instance is still valid. But, if E is large, then there will be decryption failure and then the attack will not work. But, in the Man In the Middle Setting, I think you need not care about whether it is a valid or invalid LWE instance. Because lets say, you fault the LWE instance such that both S and E are big. Then, the LWE instance is anyways an invalid instance and during encryption/encapsulation, he hides a message within the faulty LWE instance. Since, we can recover S and E from the invalid LWE instance, we can also recover the message embedded within the LWE instance. Then, as a Man In The Middle attacker, you can generate another valid LWE instance with the same recovered message and pass it onto the decapsulator. He will decapsulate it and arrive at the recovered message. Now, both sides think that they have arrived at the message. But, now the attacker also knows what message was used. So, this is the core of the MITM attack. This was an idea we conceived after we submitted this work. It is not there in the current draft.

•

• So if I work on this, the next step for me is to check the implementation code and its assembly code to ensure that we can inject the fault in the variable i.e. *distance*. And the analysis method remain the same and can be directly applied.

>>> Yes, if we can somehow fault the NTT such that the combined coefficients are limited to a brute-force, then the attack and analysis remains the same and is generic. So yes, the assembly implementation needs to be checked.

• By the way, in the draft you wrote that you did the fault injection by laser and focus on instruction skip. But in your reply for my comment, you said you did not do the physical attack. What’s the reason for this? Do we need to conduct the laser attacks? We don’t have the laser equipment in our lab.

• >>>> We only performed fault characterisation to see if we were able to skip the add instruction and the shift instruction, but didn’t do it on the actual NTT implementation. So, the attack was only performed using simulated faults (in the high-level C language, by directly changing the distance variable).

POSSIBILITIES TO ANALYSE WITHIN ASSEMBLY IMPLEMENTATION:

1. Try to see if we can directly fault the variable that determines the number of stages within the NTT operation. If we can reduce the number of stages of NTT, we are indirectly reducing the number of dependent coefficients on the output, similar to faulting the distance variable.

2. If that is not possible to fault the stage counter variable, then we can try to see if we can reduce the number of butterfly operations within every stage,. In that way, we ensure that less number of coefficients are combining in every stage.

3. The next option would be to see which variable determines the distance between the coefficients used to perform the butterfly operation. And we can try to see if we can fault that.