



MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Some lessons learned from NIST PQC

Peter Schwabe

April 23, 2025

This talk is biased – after all, NIST PQC *is* a competition.

This talk is biased – after all, NIST PQC *is* a competition.

- Based on invited talk at PQCRYPTO 2022
- Not much on signature on-ramp

This talk is biased – after all, NIST PQC *is* a competition.

- Based on invited talk at PQCRYPTO 2022
- Not much on signature on-ramp
- Cryptographic engineering point of view

This talk is biased – after all, NIST PQC *is* a competition.

- Based on invited talk at PQCRYPTO 2022
- Not much on signature on-ramp
- Cryptographic engineering point of view
- Looking back after 9 years of NIST PQC:
 - What went well
 - What went not so well

This talk is biased – after all, NIST PQC *is* a competition.

- Based on invited talk at PQCRYPTO 2022
- Not much on signature on-ramp
- Cryptographic engineering point of view
- Looking back after 9 years of NIST PQC:
 - What went well
 - What went not so well (in “our submissions”)

Our submissions

CRYSTALS-Kyber

Roberto Avanzi
Joppe Bos
Jintai Ding
Léo Ducas
Eike Kiltz
Tancrede Lepoint
Vadim Lyubashevsky
John M. Schanck
Peter Schwabe
Gregor Seiler
Damien Stehle

CRYSTALS-Dilithium

Léo Ducas
Eike Kiltz
Tancrede Lepoint
Vadim Lyubashevsky
Peter Schwabe
Gregor Seiler
Damien Stehlé
Shi Bai

SPHINCS⁺

Jean-Philippe Aumasson
Daniel J. Bernstein
Ward Beullens
Christoph Dobraunig
Maria Eichlseder
Scott Fluhrer
Stefan-Lukas Gazdag
Andreas Hülsing
Panos Kampanakis
Stefan Kölbl
Tanja Lange
Martin M. Lauridsen
Florian Mendel
Ruben Niederhagen
Christian Rechberger
Joost Rijneveld
Peter Schwabe
Bas Westerbaan

1. Designing

“A complete written specification of the algorithms shall be included, consisting of all necessary mathematical operations, equations, tables, diagrams, and parameters that are needed to implement the algorithms. The document shall include design rationale and an explanation for all the important design decisions that are made”

—Dustin Moody, February 24, 2016 (PQCRYPTO 2016)

1. Designing – so many decisions!

1. Designing – so many decisions!

“Oh, you mean numbers?!”

—Giulio Malavolta, September 2022

1. Designing – so many decisions!

“Oh, you mean numbers?!”

—Giulio Malavolta, September 2022

- Choose concrete parameters for different security levels
- All randomness from `randombytes` or just a seed?
- Fix sampling algorithms (e.g., constant-time sorting)
- Select symmetric primitives
- Concretize domain separation

1. Designing – so many decisions!

“Oh, you mean numbers?!”

—Giulio Malavolta, September 2022

- Choose concrete parameters for different security levels
- All randomness from `randombytes` or just a seed?
- Fix sampling algorithms (e.g., constant-time sorting)
- Select symmetric primitives
- Concretize domain separation
- Tradeoffs, tradeoffs, tradeoffs. . .

1. Designing – Exhibit A

Dilithium – v3.0 vs. v3.1

Sign(sk, M)

```
09  $\mathbf{A} \in R_q^{k \times \ell} := \text{ExpandA}(\rho)$   
10  $\mu \in \{0, 1\}^{384} := \text{CRH}(tr \parallel M)$   
11  $\kappa := 0, (\mathbf{z}, \mathbf{h}) := \perp$ 
```

Sign(sk, M)

```
09  $\mathbf{A} \in R_q^{k \times \ell} := \text{ExpandA}(\rho)$   
10  $\mu \in \{0, 1\}^{512} := \text{H}(tr \parallel M)$   
11  $\kappa := 0, (\mathbf{z}, \mathbf{h}) := \perp$ 
```

- Message hash only 384 bits
- 192 bits of sec. against collisions
- Not sufficient for NIST level 5

1. Designing – Exhibit B

SPHINCS⁺, round 3

- Security relies on DM-SPR (Distinct-function multi-target second preimage resistance) of underlying hash function
- Three different choices of hash function: SHA-256, SHAKE-256, Haraka

1. Designing – Exhibit B

SPHINCS⁺, round 3

- Security relies on DM-SPR (Distinct-function multi-target second preimage resistance) of underlying hash function
- Three different choices of hash function: SHA-256, SHAKE-256, Haraka
- Sydney Antonov, April 20, 2022: attack against DM-SPR of SHA-256
- Attack cost higher than NIST level 1, but lower than level 3 and 5

1. Designing – Exhibit B

SPHINCS⁺, round 3

- Security relies on DM-SPR (Distinct-function multi-target second preimage resistance) of underlying hash function
- Three different choices of hash function: SHA-256, SHAKE-256, Haraka
- Sydney Antonov, April 20, 2022: attack against DM-SPR of SHA-256
- Attack cost higher than NIST level 1, but lower than level 3 and 5

“This is an interesting attack that does demonstrate that our real hash functions do not perfectly behave like random oracles”

—Andreas Hülsing, April 21, 2022

1. Designing – Two questions

1. Will the schemes selected now be widely used?
2. Will those schemes survive in the long run?

1. Designing – Two questions

Deployment of ML-KEM

- Integrated into Firefox, Chrome, Brave. . .
- Server-side supported by, e.g., Google and Cloudflare
- More than 100 billion connections per day at Cloudflare alone

1. Designing – Two questions

Deployment of ML-KEM

- Integrated into Firefox, Chrome, Brave. . .
- Server-side supported by, e.g., Google and Cloudflare
- More than 100 billion connections per day at Cloudflare alone
- Also deployed in Signal, iMessage, by AWS. . .
- Certified smartcard by Infineon

1. Designing – Two questions

Deployment of ML-KEM

- Integrated into Firefox, Chrome, Brave. . .
- Server-side supported by, e.g., Google and Cloudflare
- More than 100 billion connections per day at Cloudflare alone
- Also deployed in Signal, iMessage, by AWS. . .
- Certified smartcard by Infineon

Deployment of ML-DSA and SLH-DSA

- SLH-DSA for secure boot on OpenTitan
- First cars with ML-DSA-secured software updates soon

2. Proving

"Submitters are not required to provide a proof of security, although such proofs will be considered if they are available."

—NIST PQC, Call for Proposals

2. Proving – Exhibit A

Kyber round 1

- LPR scheme's public key is $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$
- This is an (R/M)LWE sample and assumed to be uniform in the proof

2. Proving – Exhibit A

Kyber round 1

- LPR scheme's public key is $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$
- This is an (R/M)LWE sample and assumed to be uniform in the proof
- Kyber in round 1 compressed this (round off low bits)
- $\mathbf{t}' = \text{Decompress}(\text{Compress}(\mathbf{t}))$ is **not** uniform

2. Proving – Exhibit A

Kyber round 1

- LPR scheme's public key is $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$
- This is an (R/M)LWE sample and assumed to be uniform in the proof
- Kyber in round 1 compressed this (round off low bits)
- $\mathbf{t}' = \text{Decompress}(\text{Compress}(\mathbf{t}))$ is **not** uniform
- Reduction from MLWE in round-1 Kyber was invalid:

"We note that a potential issue is that the security proof does not directly apply to Kyber itself, but rather to a modified version of the scheme which does not compress the public key."

—NIST IR 8240

2. Proving – Exhibit B

SPHINCS⁺ – original proof

- Reduce from second-preimage resistance
- Place challenge $x = H(y)$ inside hash chains
- Forgery produces preimage of x with certain prob.
- Reduction hopes to obtain second preimage $y' \neq y$ with $x = H(y')$

2. Proving – Exhibit B

SPHINCS⁺ – original proof

- Reduce from second-preimage resistance
- Place challenge $x = H(y)$ inside hash chains
- Forgery produces preimage of x with certain prob.
- Reduction hopes to obtain second preimage $y' \neq y$ with $x = H(y')$
- Problem: $\text{len}(x) = \text{len}(y)$
- Second preimage does not exist with high probability

2. Proving – Exhibit B

SPHINCS⁺ – original proof

- Reduce from second-preimage resistance
- Place challenge $x = H(y)$ inside hash chains
- Forgery produces preimage of x with certain prob.
- Reduction hopes to obtain second preimage $y' \neq y$ with $x = H(y')$
- Problem: $\text{len}(x) = \text{len}(y)$
- Second preimage does not exist with high probability
- Forger **can refuse to forge if there is a second preimage**

2. Proving – so many failure modes

- Proof is wrong

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct
 - Theorem is also wrong
 - Scheme is still (possibly) secure
 - Scheme is efficiently broken

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct
 - Theorem is also wrong
 - Scheme is still (possibly) secure
 - Scheme is efficiently broken
- Proof doesn't apply to the scheme

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct
 - Theorem is also wrong
 - Scheme is still (possibly) secure
 - Scheme is efficiently broken
- Proof doesn't apply to the scheme
- Proof correct, but theorem “insufficient”

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct
 - Theorem is also wrong
 - Scheme is still (possibly) secure
 - Scheme is efficiently broken
- Proof doesn't apply to the scheme
- Proof correct, but theorem “insufficient”
 - Example: attack hides in non-tightness

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct
 - Theorem is also wrong
 - Scheme is still (possibly) secure
 - Scheme is efficiently broken
- Proof doesn't apply to the scheme
- Proof correct, but theorem “insufficient”
 - Example: attack hides in non-tightness
- Proof (and possibly theorem) too vague

2. Proving – so many failure modes

- Proof is wrong
 - Theorem is correct
 - Theorem is also wrong
 - Scheme is still (possibly) secure
 - Scheme is efficiently broken
- Proof doesn't apply to the scheme
- Proof correct, but theorem “insufficient”
 - Example: attack hides in non-tightness
- Proof (and possibly theorem) too vague
- Theorem and proof correct, but not very useful

“A is secure if A is secure”

3. Implementing

*“NISTPQC, despite being an important and timely project, has produced the largest regression **ever** in the quality of cryptographic software. This will not be easy to fix.”*

—Daniel J. Bernstein, October 5, 2018

3. Implementing – Exhibit A

Dilithium commit on Dec. 28, 2017

212	-	t = buf[pos];
213	-	t = (uint32_t)buf[pos + 1] << 8;
214	-	t = (uint32_t)buf[pos + 2] << 16;
215	-	t &= 0xFFFF;
337	+	t0 = buf[pos];
338	+	t0 = (uint32_t)buf[pos + 1] << 8;
339	+	t0 = (uint32_t)buf[pos + 2] << 16;
340	+	t0 &= 0xFFFF;
216	341	
217	-	t = buf[pos + 2] >> 4;
218	-	t = (uint32_t)buf[pos + 3] << 4;
219	-	t = (uint32_t)buf[pos + 4] << 12;
342	+	t1 = buf[pos + 2] >> 4;
343	+	t1 = (uint32_t)buf[pos + 3] << 4;
344	+	t1 = (uint32_t)buf[pos + 4] << 12;

- Bug in Dilithium sampler
- Two consecutive coefficients are equal
- Allows key recovery
- Reported by Peter Pessl on Dec. 27, 2017

3. Implementing – Exhibit B

PQClean

- Joint work with Matthias Kannwischer, Joost Rijneveld, John Schanck, Douglas Stebila, Goutam Tamvada, Thom Wiggers
- Test harness for PQC implementations
- Integrate reference implementations
 - Run through test harness
 - “clean up”

3. Implementing – Exhibit B

Flaw	KEMs	Sigs	Flaw	KEMs	Sigs
Memory safety	3	4	Endianness assumptions	7	2
Signed integer overflow	3	1	Platform-specific behavior	4	0
Alignment assumptions	4	4	Variable-Length Arrays	4	1
Other Undefined Behavior	1	1	Compiler extensions	5	2
Dead code	3	4	Integer sizes	6	3
Global state	2	1	Non-constant time	4	0
Licensing unclear	3	1			

3. Implementing – Exhibit B

“In almost every scheme we identified “unclean” code, ranging from missing casts to memory safety problems and other forms of undefined behavior.”

—<https://eprint.iacr.org/2022/337>

3. Implementing – Exhibit C

Questions about the range analysis of iNTT for "Faster Kyber and Dilithium on the Cortex-M4" #226



JunhaoHuang opened this issue on Mar 3 · 4 comments



JunhaoHuang commented on Mar 3 · edited

Hi team, I am reading the Kyber code regarding the recent paper "Faster Kyber and Dilithium on the Cortex-M4", and I have a question about the matrix-vector product and Better Accumulation part regarding the `f_stack` version code.

I see that using the better accumulation technique in the `f_speed` version code, we can reduce each element of the output vector of matrix-vector product down to $(-q, q)$. Since `poly_invntt` is normally used after the matrix-vector product, the range of the input vector of `poly_invntt` lies in $(-q, q)$ in the `f_speed` version code. The `invntt` function works in this situation.

What I wonder is that in the `f_stack` version code, the `matacc` function actually uses the previous double basemul accumulation function, and it should produce the result vector with element in $(-kq, kq)$, k is the security parameter of Kyber. For Kyber1024, the range of each polynomial element that `invntt` takes should be $(-4q, 4q)$. However, the `invntt` function is the same as the `f_speed` version code. The first four layers of the light butterflies in `invntt` involve some additions and subtractions without multiplication. Therefore, For Kyber1024 in the `f_stack` version code, two layers of addition/subtraction might overflow the `int16_t`. I wonder how you deal with this problem in the `f_stack` code and why does it still work?

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3. Implementing – Exhibit C

“...two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?”

3. Implementing – Exhibit C

“... two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?”

“... On your question on why it still works, I believe that this is an edge case that does not get triggered by the testing scripts.”

3. Implementing – Exhibit C



vincentvbh commented on Mar 6, 2021

Contributor

Author



There is a bug in the inverse of NTT in Saber. But the bug is triggered with a very low probability that it is not triggered on testing.

4. Benchmarking

Two main benchmarking frameworks

- eBACS/SUPERCOP (<https://bench.cr.yp.to>)
 - Run by Bernstein and Lange
 - Overlap with submissions: **Classic McEliece**, **NTRU Prime**, **PQ-RSA**, **SPHINCS⁺**
 - Benchmark on variety of “large” platforms
 - Need to run Linux/UNIX OS

4. Benchmarking

Two main benchmarking frameworks

- eBACS/SUPERCOP (<https://bench.cr.yp.to>)
 - Run by Bernstein and Lange
 - Overlap with submissions: **Classic McEliece**, **NTRU Prime**, **PQ-RSA**, **SPHINCS⁺**
 - Benchmark on variety of “large” platforms
 - Need to run Linux/UNIX OS
- pqm4 (<https://github.com/mupq/pqm4>)
 - Run by Kannwischer, Petri, Rijneveld, Schwabe, Stoffelen
 - Overlap with submissions: **Classic McEliece**, **Kyber**, **Dilithium**, **NTRU**, **MQDSS**, **NewHope**, **SPHINCS⁺**, **MAYO**, **UOV**
 - Benchmark on Arm Cortex M4

4. Benchmarking – some thoughts

Lack of *independent* benchmarking

- Benchmarking in the hands of submitters
- **Much more desirable:** Independent benchmarking by competition “hosts”

4. Benchmarking – some thoughts

APIs matter

- NIST-API: serialization part of **keypair**, **encaps**, **decaps**, **sign**, **sign_open**
- *Deployments* use API with separate serialization
 - Ephemeral deployments never serialize secret keys
 - Static deployments amortize de-serialization
- Massive impact on performance and design choices!

4. Benchmarking – some thoughts

Consider HW acceleration for symmetric crypto

- Many PQC schemes bottlenecked by Keccak
- Example: ML-KEM-768 encapsulation on OpenTitan¹
 - Software Keccak takes 66% of total cycles
 - With hardware acceleration: 3% of total cycles

¹A. Abdulrahman, F. Oberhansl, H. Pham, J. Philipoom, P. Schwabe, T. Stelzer, A. Zankl. *Towards ML-KEM & ML-DSA on OpenTitan*, IEEE S&P 2025, to appear

4. Benchmarking – some thoughts

Consider HW acceleration for symmetric crypto

- Many PQC schemes bottlenecked by Keccak
- Example: ML-KEM-768 encapsulation on OpenTitan¹
 - Software Keccak takes 66% of total cycles
 - With hardware acceleration: 3% of total cycles
- **Recommendation:** benchmark also with HW Keccak, AES, SHA-2, SM3, SM4, ZUC
- Alternative: use very fast *fake* symmetric crypto for benchmarks

¹A. Abdulrahman, F. Oberhansl, H. Pham, J. Philipoom, P. Schwabe, T. Stelzer, A. Zankl. *Towards ML-KEM & ML-DSA on OpenTitan*, IEEE S&P 2025, to appear

5. Attacking

"The idea is that participants put their algorithms into the ring, and then we all spend a few years beating on each other's submissions."

—Bruce Schneier, August 8, 2022

5. Attacking – Guessed Once

```
def recover_bit(ct, bit):  
    assert bit < len(ct) // 4000  
    ts = [struct.unpack('BB', ct[i:i+2]) for i in range(4000*bit, 4000*(bit+1), 2)]  
    xs, ys = [a for a, b in ts if b == 1], [a for a, b in ts if b == 2]  
    return sum(xs) / len(xs) >= sum(ys) / len(ys)  
  
def decrypt(ct):  
    res = sum(recover_bit(ct, b) << b for b in range(len(ct) // 4000))  
    return int.to_bytes(res, len(ct) // 4000 // 8, 'little')
```

—Lorenz Panny, December 21, 2017

5. Attacking in 2022

`ia.cr/2022/975`

5. Attacking in 2022

Castryck, Decru: *An efficient key recovery attack on SIDH*

5. Attacking in 2022

Castryck, Decru: *An efficient key recovery attack on SIDH*

- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)

Castryck, Decru: *An efficient key recovery attack on SIDH*

- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)
- SIKE **lowered** parameters during NIST PQC
(following Jaques, Schanck: *Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE* (ePrint 2019/103))

5. Attacking in 2022

Castryck, Decru: *An efficient key recovery attack on SIDH*

- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)
- SIKE **lowered** parameters during NIST PQC
(following Jaques, Schanck: *Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE* (ePrint 2019/103))
- Competent, smart people tried to break it
(e.g., Martindale, Panny: *How to not break SIDH* (ePrint 2019/558))

5. Attacking in 2022

Castryck, Decru: *An efficient key recovery attack on SIDH*

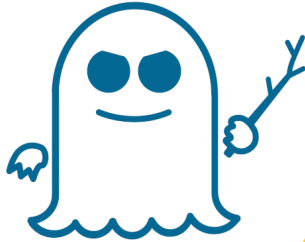
- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)
- SIKE **lowered** parameters during NIST PQC
(following Jaques, Schanck: *Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE* (ePrint 2019/103))
- Competent, smart people tried to break it
(e.g., Martindale, Panny: *How to not break SIDH* (ePrint 2019/558))

Yet, **full break** without any “warning”

5. Attacking – even more attacks!



MELTDOWN



Hertzbleed



CACHE OUT

6. Communicating

"I don't know if you're familiar with this website, twitter.com? If you like crypto drama, this is where you go. Except if you go to the pqc-forum, which is also... generally... it's even better".

—Bor de Kock, August 17, 2022

6. Communicating

"I don't know if you're familiar with this website, twitter.com? If you like crypto drama, this is where you go. Except if you go to the pqc-forum, which is also... generally... it's even better".

—Bor de Kock, August 17, 2022

<https://www.youtube.com/watch?v=kXcYy8L0I9s>, starting at 20:22.

6. Communicating – pqc-forum

6. Communicating – pqc-forum

“Follow the “Rule of 1” and the “Rule of n”: When you speak, make 1 point and then let others speak, and when in a group of “n” people, speak “1/nth” of the time.”

—Aspiration Participants:Guidelines

<https://facilitation.aspirationtech.org/index.php?title=Participants:Guidelines>

6. Communicating – pqc-forum

Download all mails, run statistics

6. Communicating – pqc-forum

Download almost all mails, run statistics

6. Communicating – pqc-forum

Download almost all mails, run statistics

- pqc-forum had 666 threads (“conversations”) on Sep. 14, 2022
- First mail by Dustin Moody from Aug. 1, 2016
- I have 2805 mails (first one from Nov. 2, 2016)

6. Communicating – pqc-forum

Download almost all mails, run statistics

- pqc-forum had 666 threads (“conversations”) on Sep. 14, 2022
- First mail by Dustin Moody from Aug. 1, 2016
- I have had 2805 mails (first one from Nov. 2, 2016)

6. Communicating – pqc-forum

```
for i in mails/*;do
  FROM=$(grep ^From: $i | head -n 1 | sed "s/From:\ //" | sed "s/.*<\[^\>\]*\>/\1/")
  if [ "$FROM" = "pqc-forum@list.nist.gov" ]; then
    FROM=$(grep ^X-Original-From: $i | head -n 1 | \
      sed "s/X-Original-From:\ //" | sed "s/.*<\[^\>\]*\>/\1/")
  fi
  echo $FROM
done | sort | uniq -c | sort -n
```

- 369 sender addresses
- Sometimes multiple addresses for one person
- 131 addresses sent just one mail
- 275 addresses sent at most 5 mails

6. Communicating – pqc-forum

The “Top 10”

1.	address1	407
2.	address2	146
3.	address3	113
4.	address4	106
5.	address5	100
6.	address6	81
7.	address7	69
8.	address8	68
9.	address9	50
10.	address10	47
10.	address11	47

6. Communicating – pqc-forum

The “Top 10”

1.	address1	407
2.	dustin.moody@nist.gov	146
3.	address3	113
4.	address4	106
5.	daniel.apon@nist.gov	100
6.	jacob.alperin-sheriff@nist.gov	81
7.	address7	69
8.	ray.perlner@nist.gov	68
9.	address9	50
10.	address10	47
10.	address11	47

6. Communicating – pqc-forum

The “Top 10”

1.	address1	407
2.	dustin.moody@nist.gov	146
3.	address3	113
4.	address4	106
5.	daniel.apon@nist.gov	100
6.	jacob.alperin-sheriff@nist.gov	81
7.	address7	69
8.	ray.perlner@nist.gov	68
9.	address9	50
10.	peter@cryptojedi.org	47
10.	address11	47

>50% of mails sent
by only 15 people.

6. Communicating – pqc-forum

The “Top 10”

1.	address1	407
2.	dustin.moody@nist.gov	146
3.	address3	113
4.	address4	106
5.	daniel.apon@nist.gov	100
6.	jacob.alperin-sheriff@nist.gov	81
7.	address7	69
8.	ray.perlner@nist.gov	68
9.	address9	50
10.	peter@cryptojedi.org	47
10.	address11	47

>50% of mails sent
by only 15 people.

>30% of all words by
non-NIST authors are
from one address.

So, what are the lessons learned?

1. Employ state-of-the-art software engineering techniques

So, what are the lessons learned?

1. Employ state-of-the-art software engineering techniques
2. Use mechanized proof techniques as early and extensively as possible

So, what are the lessons learned?

1. Employ state-of-the-art software engineering techniques
2. Use mechanized proof techniques as early and extensively as possible
3. Real-world test deployments as early as possible

So, what are the lessons learned?

1. Employ state-of-the-art software engineering techniques
2. Use mechanized proof techniques as early and extensively as possible
3. Real-world test deployments as early as possible
4. Use a different communication platform than Google Groups

So, what are the lessons learned?

1. Employ state-of-the-art software engineering techniques
2. Use mechanized proof techniques as early and extensively as possible
3. Real-world test deployments as early as possible
4. Use a different communication platform than Google Groups
5. **Enjoy the ride and be nice to each other!**