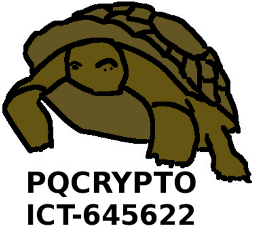
Cryptanalysis of NISTPQC submissions

Daniel J. Bernstein, Tanja Lange, Lorenz Panny

University of Illinois at Chicago, Technische Universiteit Eindhoven



18 August 2018

Workshops on Attacks in Cryptography

NSA announcements

August 11, 2015

> *IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.*

NSA announcements

August 11, 2015

> *IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.*

August 19, 2015

> *IAD will initiate a transition to quantum resistant algorithms in the not too distant future.*

Post-quantum cryptography

- 2015 Finally even NSA admits that the world needs post-quantum crypto.
- 2016 Every agency posts something (NCSC UK, NCSC NL, NSA (broken certificate!)).
- 2016 NIST announces call for submissions to post-quantum project, solicits submissions on signatures, encryption, and key exchange.

Post-quantum cryptography

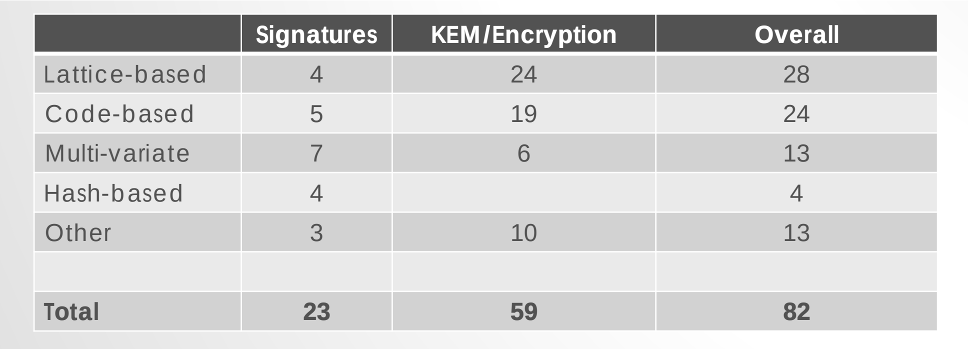
- 10 years of motivating people to work on post-quantum crypto.
- 2015 Finally even NSA admits that the world needs post-quantum crypto.
- 2016 Every agency posts something (NCSC UK, NCSC NL, NSA (broken certificate!)).
- 2016 NIST announces call for submissions to post-quantum project, solicits submissions on signatures, encryption, and key exchange.

NIST Post-Quantum “Competition”

December 2016, after public feedback: NIST calls for submissions of post-quantum cryptosystems to standardize.

30 November 2017: NIST receives 82 submissions.

Overview from Dustin Moody’s (NIST) talk at Asiacrypt:



	Signatures	KEM/Encryption	Overall
Lattice-based	4	24	28
Code-based	5	19	24
Multi-variate	7	6	13
Hash-based	4		4
Other	3	10	13
Total	**23**	**59**	**82**

“Complete and proper” submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. BIKE. CFPKM. Classic McEliece. Compact LWE. CRYSTALS-DILITHIUM. CRYSTALS-KYBER. DAGS. Ding Key Exchange. DME. DRS. DualModeMS. Edon-K. EMBLEM and R.EMBLEM. FALCON. FrodoKEM. GeMSS. Giophantus. Gravity-SPHINCS. Guess Again. Gui. HILA5. HiMQ-3. HK17. HQC. KINDI. LAC. LAKE. LEDAkem. LEDApkc. Lepton. LIMA. Lizard. LOCKER. LOTUS. LUOV. McNie. Mersenne-756839. MQDSS. NewHope. NTRUEncrypt. NTRU-HRSS-KEM. NTRU Prime. NTS-KEM. Odd Manhattan. OKCN/AKCN/CNKE. Ouroboros-R. Picnic. pqNTRUSign. pqRSA encryption. pqRSA signature. pqsigRM. QC-MDPC KEM. qTESLA. RaCoSS. Rainbow. Ramstake. RankSign. RLCE-KEM. Round2. RQC. RVB. SABER. SIKE. SPHINCS+. SRTPI. Three Bears. Titanium. WalnutDSA.

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5

2017.12.21 NIST posts 69 submissions

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange: attack script breaking CCA for HILA5

2017.12.21 NIST posts 69 submissions

2017.12.21 Panny: attack script breaking Guess Again

2017.12.23 Hülsing–Bernstein–Panny–Lange: attack scripts breaking RaCoSS

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again
2017.12.23 Hülsing–Bernstein–Panny–Lange:
attack scripts breaking RaCoSS
2017.12.25 Panny: attack script breaking RVB;
RVB withdrawn

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again
2017.12.23 Hülsing–Bernstein–Panny–Lange:
attack scripts breaking RaCoSS
2017.12.25 Panny: attack script breaking RVB;
RVB withdrawn
2017.12.25 Bernstein–Lange: attack script breaking HK17

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again
2017.12.23 Hülsing–Bernstein–Panny–Lange:
attack scripts breaking RaCoSS
2017.12.25 Panny: attack script breaking RVB;
RVB withdrawn
2017.12.25 Bernstein–Lange: attack script breaking HK17
2017.12.26 Gaborit: attack reducing McNie security level

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange:
attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again
2017.12.23 Hülsing–Bernstein–Panny–Lange:
attack scripts breaking RaCoSS
2017.12.25 Panny: attack script breaking RVB;
RVB withdrawn
2017.12.25 Bernstein–Lange: attack script breaking HK17
2017.12.26 Gaborit: attack reducing McNie security level
2017.12.29 Gaborit: attack reducing Lepton security level

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange: attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again
2017.12.23 Hülsing–Bernstein–Panny–Lange: attack scripts breaking RaCoSS
2017.12.25 Panny: attack script breaking RVB; RVB withdrawn
2017.12.25 Bernstein–Lange: attack script breaking HK17
2017.12.26 Gaborit: attack reducing McNie security level
2017.12.29 Gaborit: attack reducing Lepton security level
2017.12.29 Beullens: attack reducing DME☢ security level

Attack timeline: month 0

2017.12.18 Bernstein–Groot Bruinderink–Panny–Lange: attack script breaking CCA for HILA5
2017.12.21 NIST posts 69 submissions
2017.12.21 Panny: attack script breaking Guess Again
2017.12.23 Hülsing–Bernstein–Panny–Lange: attack scripts breaking RaCoSS
2017.12.25 Panny: attack script breaking RVB; RVB withdrawn
2017.12.25 Bernstein–Lange: attack script breaking HK17
2017.12.26 Gaborit: attack reducing McNie security level
2017.12.29 Gaborit: attack reducing Lepton security level
2017.12.29 Beullens: attack reducing DME☢ security level

☢: submitter has claimed patent on submission.
Warning: Other people could also claim patents.

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI; SRTPI withdrawn

Attack timeline: month 1

- 2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn
- 2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM
- 2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM
- 2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI; SRTPI withdrawn
- 2018.01.05 Lequesne–Sendrier–Tillich: attack breaking Edon-K; script posted 2018.02.20; Edon-K withdrawn

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI; SRTPI withdrawn

2018.01.05 Lequesne–Sendrier–Tillich: attack breaking Edon-K; script posted 2018.02.20; Edon-K withdrawn

2018.01.05 Beullens: attack script breaking DME☢

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI; SRTPI withdrawn

2018.01.05 Lequesne–Sendrier–Tillich: attack breaking Edon-K; script posted 2018.02.20; Edon-K withdrawn

2018.01.05 Beullens: attack script breaking DME☢

2018.01.05 Li–Liu–Pan–Xie, independently Bootle–Tibouchi–Xagawa: attack breaking Compact LWE☢; script from 2nd team

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI; SRTPI withdrawn

2018.01.05 Lequesne–Sendrier–Tillich: attack breaking Edon-K; script posted 2018.02.20; Edon-K withdrawn

2018.01.05 Beullens: attack script breaking DME☢

2018.01.05 Li–Liu–Pan–Xie, independently Bootle–Tibouchi–Xagawa: attack breaking Compact LWE☢; script from 2nd team

2018.01.11 Castryck–Vercauteren: attack breaking Giophantus

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi, Li–Liu–Pan–Xie: faster attack script breaking HK17; HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia: attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI; SRTPI withdrawn

2018.01.05 Lequesne–Sendrier–Tillich: attack breaking Edon-K; script posted 2018.02.20; Edon-K withdrawn

2018.01.05 Beullens: attack script breaking DME☢

2018.01.05 Li–Liu–Pan–Xie, independently Bootle–Tibouchi–Xagawa: attack breaking Compact LWE☢; script from 2nd team

2018.01.11 Castryck–Vercauteren: attack breaking Giophantus

2018.01.22 Blackburn: attack reducing WalnutDSA☢ security level

Attack timeline: month 1

2018.01.01 Bernstein, building on Bernstein–Lange, Wang–Malluhi,
Li–Liu–Pan–Xie: faster attack script breaking HK17;
HK17 withdrawn

2018.01.02 Steinfeld, independently Albrecht–Postlethwaite–Virdia:
attack script breaking CFPKM

2018.01.02 Alperin-Sheriff–Perlner: attack breaking pqsigRM

2018.01.04 Yang–Bernstein–Lange: attack script breaking SRTPI;
SRTPI withdrawn

2018.01.05 Lequesne–Sendrier–Tillich: attack breaking Edon-K;
script posted 2018.02.20; Edon-K withdrawn

2018.01.05 Beullens: attack script breaking DME☢

2018.01.05 Li–Liu–Pan–Xie, independently Bootle–Tibouchi–Xagawa:
attack breaking Compact LWE☢; script from 2nd team

2018.01.11 Castryck–Vercauteren: attack breaking Giophantus

2018.01.22 Blackburn: attack reducing WalnutDSA☢ security level

2018.01.23 Beullens: another attack reducing WalnutDSA☢ security level

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level
2018.04.03 Debris-Alazard–Tillich: attack breaking RankSign; RankSign withdrawn

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level
2018.04.03 Debris-Alazard–Tillich: attack breaking RankSign; RankSign withdrawn
2018.04.04 Beullens–Blackburn: attack script breaking WalnutDSA☢

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level
2018.04.03 Debris-Alazard–Tillich: attack breaking RankSign;
RankSign withdrawn
2018.04.04 Beullens–Blackburn:
attack script breaking WalnutDSA☢
2018.05.09 Kotov–Menshov–Ushakov:
another attack breaking WalnutDSA☢

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level
2018.04.03 Debris-Alazard–Tillich: attack breaking RankSign; RankSign withdrawn
2018.04.04 Beullens–Blackburn: attack script breaking WalnutDSA☢
2018.05.09 Kotov–Menshov–Ushakov: another attack breaking WalnutDSA☢
2018.05.16 Barelli–Couvreur: attack reducing DAGS security level

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level
2018.04.03 Debris-Alazard–Tillich: attack breaking RankSign; RankSign withdrawn
2018.04.04 Beullens–Blackburn: attack script breaking WalnutDSA☢
2018.05.09 Kotov–Menshov–Ushakov: another attack breaking WalnutDSA☢
2018.05.16 Barelli–Couvreur: attack reducing DAGS security level
2018.05.30 Couvreur–Lequesne–Tillich: attack breaking "short" parameters for RLCE☢

Attack timeline: subsequent events

2018.02.01 Beullens: attack breaking WalnutDSA☢
2018.02.07 Fabsic–Hromada–Zajac: attack breaking CCA for LEDA
2018.03.27 Yu–Ducas: attack reducing DRS security level
2018.04.03 Debris-Alazard–Tillich: attack breaking RankSign; RankSign withdrawn
2018.04.04 Beullens–Blackburn: attack script breaking WalnutDSA☢
2018.05.09 Kotov–Menshov–Ushakov: another attack breaking WalnutDSA☢
2018.05.16 Barelli–Couvreur: attack reducing DAGS security level
2018.05.30 Couvreur–Lequesne–Tillich: attack breaking "short" parameters for RLCE☢
2018.06.11 Beullens–Castryck–Vercauteren: attack script breaking Giophantus

“Complete and proper” submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. BIKE. CFPKM. Classic McEliece. Compact LWE. CRYSTALS-DILITHIUM. CRYSTALS-KYBER. DAGS. Ding Key Exchange. DME. DRS. DualModeMS. Edon-K. EMBLEM and R.EMBLEM. FALCON. FrodoKEM. GeMSS. Giophantus. Gravity-SPHINCS. Guess Again. Gui. HILA5. HiMQ-3. HK17. HQC. KINDI. LAC. LAKE. LEDAkem. LEDApkc. Lepton. LIMA. Lizard. LOCKER. LOTUS. LUOV. McNie. Mersenne-756839. MQDSS. NewHope. NTRUEncrypt. NTRU-HRSS-KEM. NTRU Prime. NTS-KEM. Odd Manhattan. OKCN/AKCN/CNKE. Ouroboros-R. Picnic. pqNTRUSign. pqRSA encryption. pqRSA signature. pqsigRM. QC-MDPC KEM. qTESLA. RaCoSS. Rainbow. Ramstake. RankSign. RLCE-KEM. Round2. RQC. RVB. SABER. SIKE. SPHINCS+. SRTPI. Three Bears. Titanium. WalnutDSA.

“Complete and proper” submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. BIKE. CFPKM. Classic McEliece. Compact LWE. CRYSTALS-DILITHIUM. CRYSTALS-KYBER. DAGS. Ding Key Exchange. DME. DRS. DualModeMS. Edon-K. EMBLEM and R.EMBLEM. FALCON. FrodoKEM. GeMSS. Giophantus. Gravity-SPHINCS. Guess Again. Gui. HILA5. HiMQ-3. HK17. HQC. KINDI. LAC. LAKE. LEDAkem. LEDApkc. Lepton. LIMA. Lizard. LOCKER. LOTUS. LUOV. McNie. Mersenne-756839. MQDSS. NewHope. NTRUEncrypt. NTRU-HRSS-KEM. NTRU Prime. NTS-KEM. Odd Manhattan. OKCN/AKCN/CNKE. Ouroboros-R. Picnic. pqNTRUSign. pqRSA encryption. pqRSA signature. pqsigRM. QC-MDPC KEM. qTESLA. RaCoSS. Rainbow. Ramstake. RankSign. RLCE-KEM. Round2. RQC. RVB. SABER. SIKE. SPHINCS+. SRTPI. Three Bears. Titanium. WalnutDSA.

Color coding: **total break**; **partial break**

PQCRYPTO
ICT-645622

HILA5

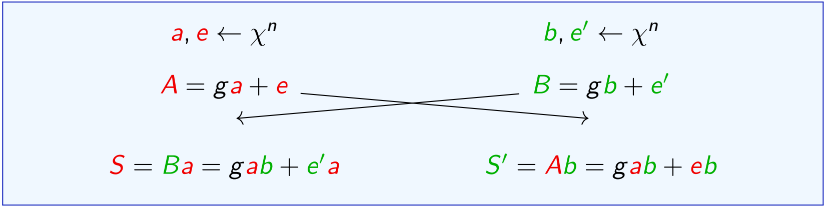
- HILA5 is a RLWE-based KEM submitted to NISTPQC.

  > *This design also provides* ***IND-CCA secure*** *KEM-DEM public key encryption if used in conjunction with an appropriate AEAD such as NIST approved AES256-GCM.*
  >
  > — HILA5 NIST submission document (v1.0)

- Decapsulation much faster than encapsulation (and faster than any other scheme).
- No mention of a CCA transform (e.g. Fujisaki–Okamoto).

Noisy Diffie–Hellman

- Have a ring $R = \mathbf{Z}[x]/(q, \varphi)$ where $q \in \mathbf{Z}$ and $\overset{\text{degree } n}{\varphi} \in \mathbf{Z}[x]$.
- Let $\underline{\chi}$ be a narrow distribution around $0 \in R$.
- Fix some "random" element $g \in R$.

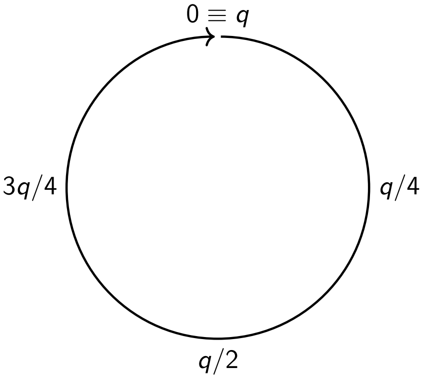


$$\begin{tikzcd}
a, e \leftarrow \chi^n & & b, e' \leftarrow \chi^n \\
A = ga + e \arrow[drr] & & B = gb + e' \arrow[dll] \\
S = Ba = gab + e'a & & S' = Ab = gab + eb
\end{tikzcd}$$

$$\Longrightarrow S - S' = e'a - eb \underset{\substack{\uparrow \\ \chi\text{ small}}}{\approx} 0$$

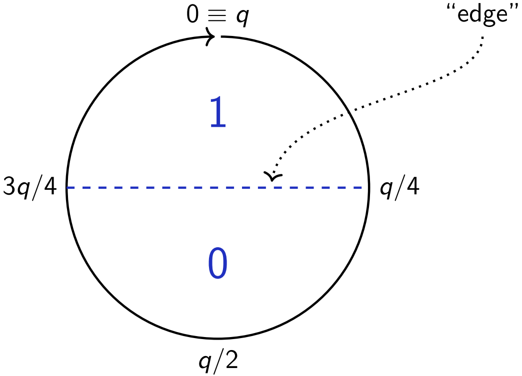
Reconciliation

Alice and Bob obtain close secret vectors $S, S' \in (\mathbf{Z}/q)^n$.
How to map coefficients to bits?



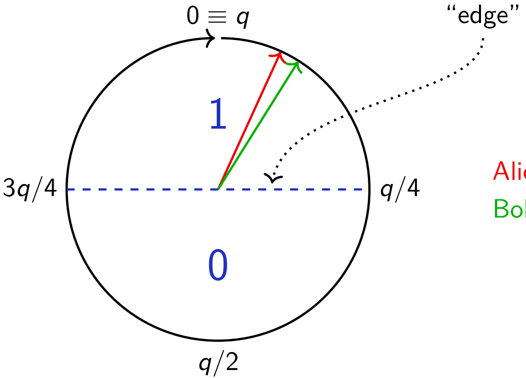
Reconciliation

Alice and Bob obtain close secret vectors $S, S' \in (\mathbf{Z}/q)^n$.
How to map coefficients to bits?



Reconciliation

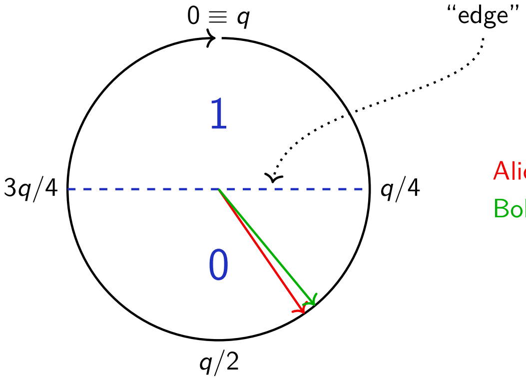
Alice and Bob obtain close secret vectors $S, S' \in (\mathbf{Z}/q)^n$.
How to map coefficients to bits?



Alice: 1
Bob: 1

Reconciliation

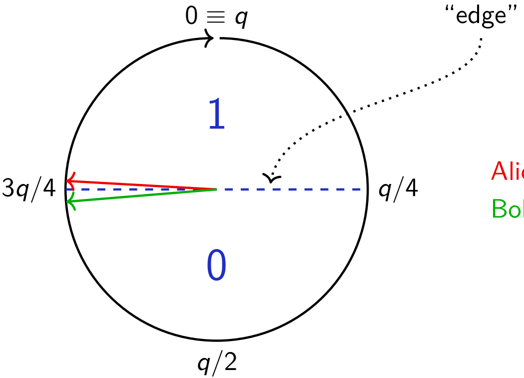
Alice and Bob obtain close secret vectors $S, S' \in (\mathbf{Z}/q)^n$.
How to map coefficients to bits?



Alice: 0
Bob: 0

Reconciliation

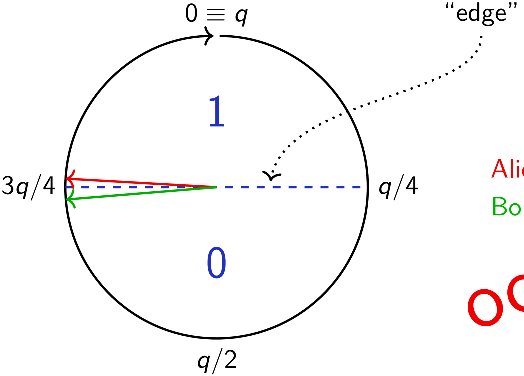
Alice and Bob obtain close secret vectors $S, S' \in (\mathbf{Z}/q)^n$.
How to map coefficients to bits?



Alice: 1
Bob: 0

Reconciliation

Alice and Bob obtain close secret vectors $S, S' \in (\mathbf{Z}/q)^n$.
How to map coefficients to bits?



Alice: 1
Bob: 0

oops!

Reconciliation

Mapping coefficients to bits using <u>fixed intervals</u> is <u>bad</u>.

Reconciliation

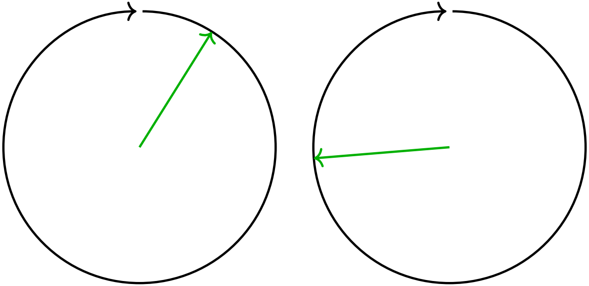
Mapping coefficients to bits using fixed intervals is bad.

Better: Bob chooses a mapping based on his coefficient and tells Alice which mapping he used.

Reconciliation

Mapping coefficients to bits using fixed intervals is bad.

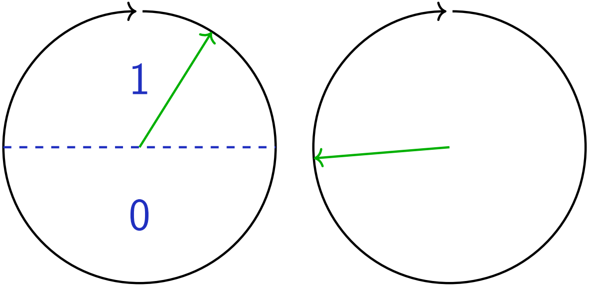
Better: Bob chooses a mapping based on his coefficient and tells Alice which mapping he used.



Reconciliation

Mapping coefficients to bits using fixed intervals is bad.

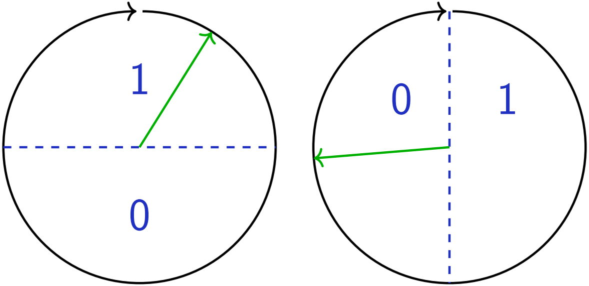
Better: Bob chooses a mapping based on his coefficient and tells Alice which mapping he used.



Reconciliation

Mapping coefficients to bits using fixed intervals is bad.

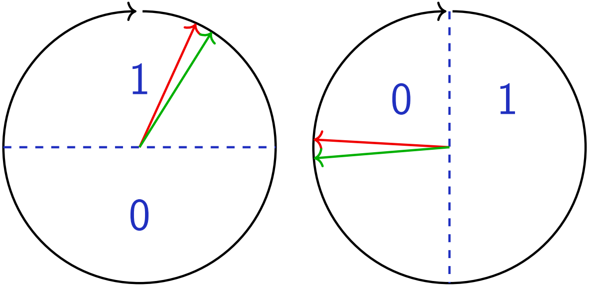
Better: Bob chooses a mapping based on his coefficient and tells Alice which mapping he used.



Reconciliation

Mapping coefficients to bits using fixed intervals is bad.

Better: Bob chooses a mapping based on his coefficient and tells Alice which mapping he used.

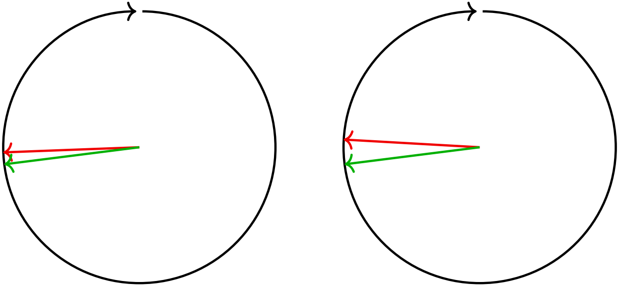


Fluhrer's attack https://ia.cr/2016/085

Problem: Evil Bob can trick Alice into leaking information by deliberately using the wrong mapping for one coefficient.

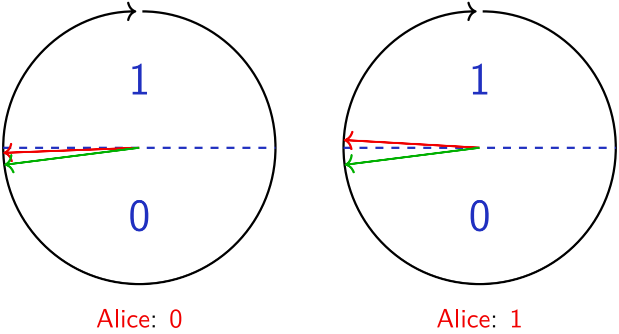
Fluhrer's attack https://ia.cr/2016/085

Problem: Evil Bob can trick Alice into leaking information by deliberately using the wrong mapping for one coefficient.



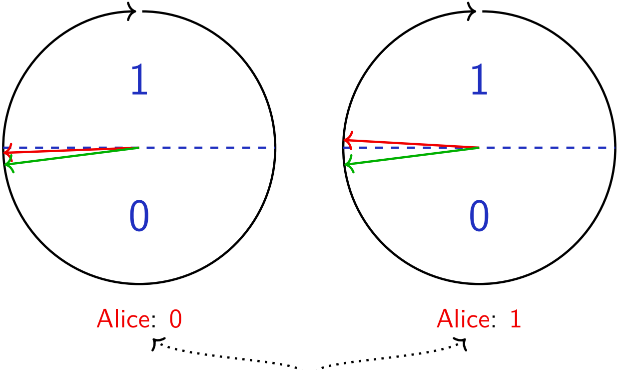
Fluhrer's attack https://ia.cr/2016/085

Problem: Evil Bob can trick Alice into leaking information by deliberately using the wrong mapping for one coefficient.



Fluhrer's attack https://ia.cr/2016/085

Problem: Evil Bob can trick Alice into leaking information by deliberately using the wrong mapping for one coefficient.



Alice: 0

Alice: 1

Evil Bob can distinguish these cases!

(He knows all the other key bits.)

Chosen-ciphertext information leaks

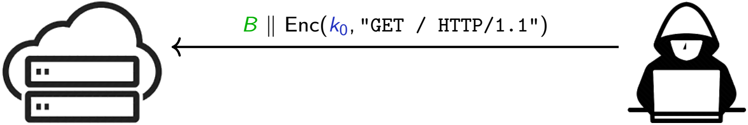
Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.

Alice

Evil Bob

Chosen-ciphertext information leaks

Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.



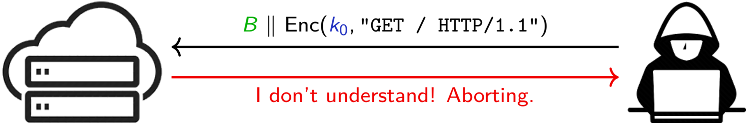
$B \parallel \mathrm{Enc}(k_0, \texttt{"GET / HTTP/1.1"})$

Alice

Evil Bob

Chosen-ciphertext information leaks

Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.



$B \parallel \mathrm{Enc}(k_0, \texttt{"GET / HTTP/1.1"})$

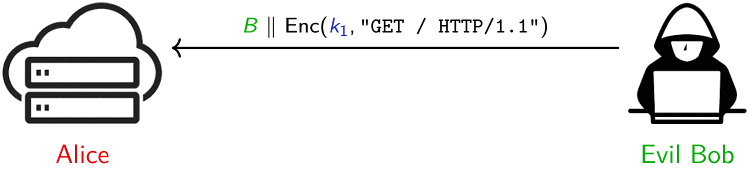
I don't understand! Aborting.

Alice

Evil Bob

Chosen-ciphertext information leaks

Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.



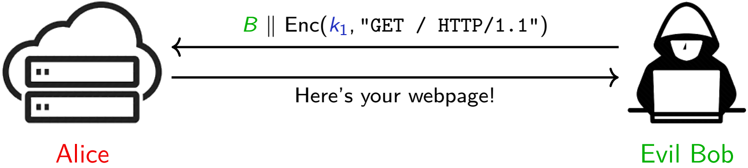
$B \parallel \mathrm{Enc}(k_1,$ `"GET / HTTP/1.1"`$)$

Alice

Evil Bob

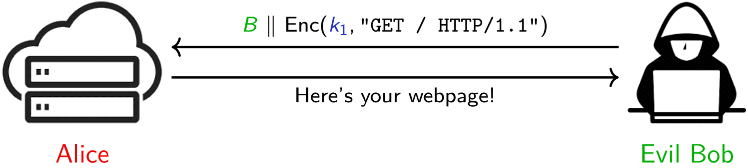
Chosen-ciphertext information leaks

Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.



Chosen-ciphertext information leaks

Evil Bob has $\underline{\text{two guesses } k_0, k_1}$ for what Alice's key k will be given his manipulated public key B.



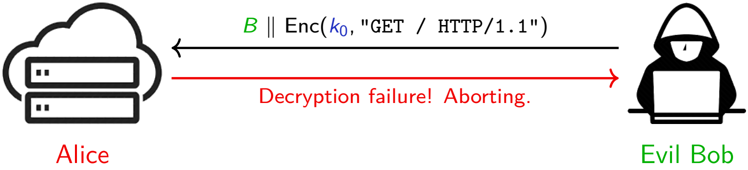
$B \parallel \mathrm{Enc}(k_1, \texttt{"GET / HTTP/1.1"})$

Here's your webpage!

$\Longrightarrow$ Bob learns that $k = k_1$.

Chosen-ciphertext information leaks

Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.



$B \parallel \mathrm{Enc}(k_0, \texttt{"GET / HTTP/1.1"})$

Decryption failure! Aborting.

Alice

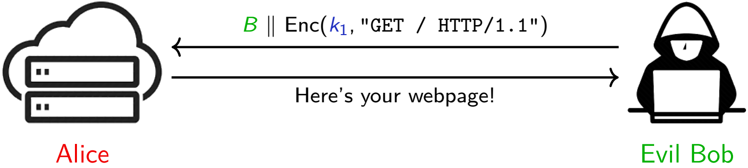
Evil Bob

$\Longrightarrow$ Bob learns that $k = k_1$.

This still works if Enc is an authenticated symmetric cipher!

Chosen-ciphertext information leaks

Evil Bob has two guesses k_0, k_1 for what Alice's key k will be given his manipulated public key B.



$B \parallel \mathrm{Enc}(k_1, \texttt{"GET / HTTP/1.1"})$

Here's your webpage!

$\Longrightarrow$ Bob learns that $k = k_1$.

This still works if Enc is an authenticated symmetric cipher!

Fluhrer's attack https://ia.cr/2016/085

Adaptive chosen-ciphertext attack against static keys.

Fluhrer's attack https://ia.cr/2016/085

Adaptive chosen-ciphertext attack against static keys.

Recall that Alice's "shared" secret is $gab + e'a$.

Fluhrer's attack https://ia.cr/2016/085

Adaptive chosen-ciphertext attack against static keys.

Recall that Alice's "shared" secret is $gab + e'a$.

Suppose Evil Bob knows b_δ such that $gab_\delta[0] = \overset{\text{edge}}{M} + \delta$.
$\Longrightarrow$ Querying Alice with $b = b_\delta$ leaks whether $-e'a[0] > \delta$.

Fluhrer's attack https://ia.cr/2016/085

Adaptive chosen-ciphertext attack against static keys.

Recall that Alice's "shared" secret is $g\,ab + e'a$.

Suppose Evil Bob knows b_δ such that $g\,ab_\delta[0] = \overset{\text{edge}}{M} + \delta$.

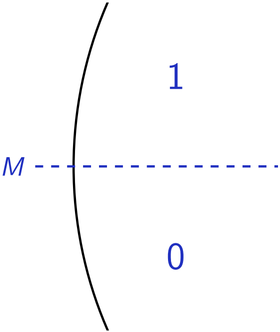
$\Longrightarrow$ Querying Alice with $b = b_\delta$ leaks whether $-e'a[0] > \delta$.

Structure of R

$\rightsquigarrow$ Can choose e' such that $e'a[0] = a[i]$ to recover all of a.

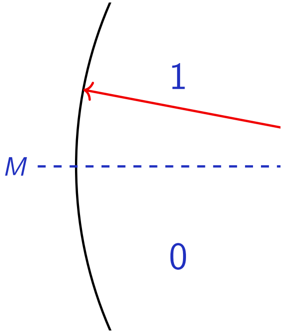
Fluhrer's attack https://ia.cr/2016/085

Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.



Fluhrer's attack https://ia.cr/2016/085

Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.

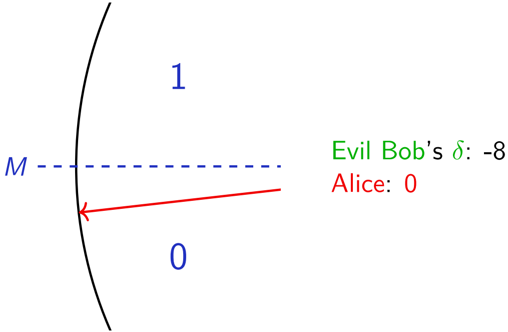


Evil Bob's δ: 0

Alice: 1

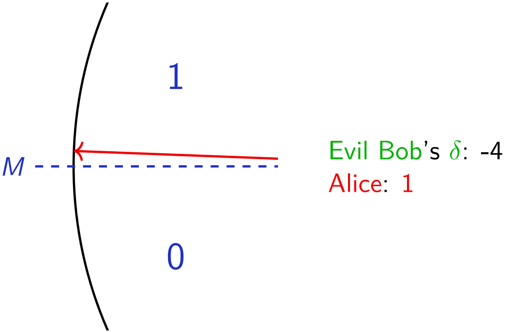
Fluhrer's attack https://ia.cr/2016/085

Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.



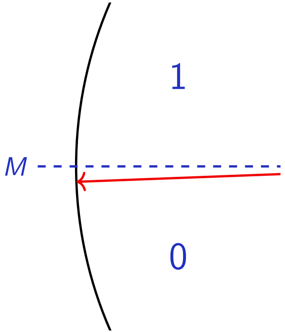
Fluhrer's attack https://ia.cr/2016/085

Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.



Fluhrer's attack https://ia.cr/2016/085

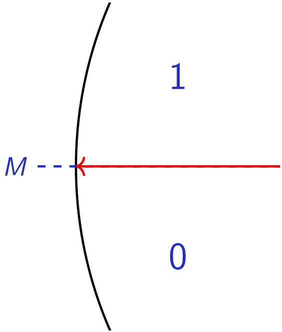
Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.



Evil Bob's δ: -6
Alice: 0

Fluhrer's attack https://ia.cr/2016/085

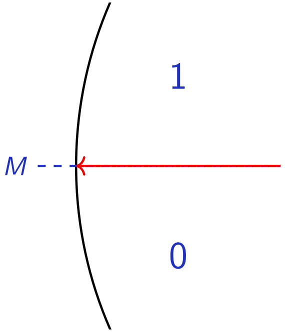
Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.



Evil Bob's δ: -5
Alice: 1

Fluhrer's attack https://ia.cr/2016/085

Querying Alice with $b = b_\delta$ and $e' = 1$ leaks whether $-a[0] > \delta$.



Evil Bob's δ: -5
Alice: 1

$\Longrightarrow$ Evil Bob learns that $a[0] = 5$.

Our work

Adaption of Fluhrer's attack to HILA5 and analysis

HILA5 https://ia.cr/2017/424 https://github.com/mjosaarinen/hila5

- Standard noisy Diffie–Hellman with new reconciliation.

HILA5 https://ia.cr/2017/424 https://github.com/mjosaarinen/hila5

- Standard noisy Diffie–Hellman with new reconciliation.
- Ring: $\mathbf{Z}[x]/(q, x^{1024}+1)$ where $q = 12289$.[1]
- Noise distribution χ: Ψ_{16}.[1]  on $\{-16, ..., 16\}$

---

[1] same as New Hope.

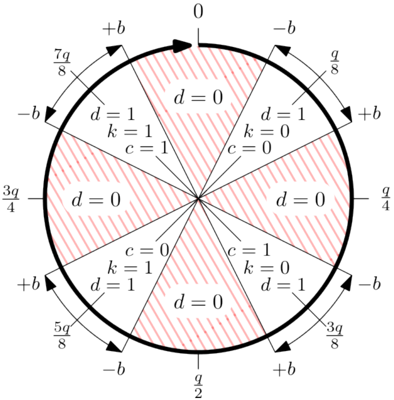
HILA5 https://ia.cr/2017/424 https://github.com/mjosaarinen/hila5

- Standard noisy Diffie–Hellman with new reconciliation.
- Ring: $\mathbf{Z}[x]/(q, x^{1024}+1)$ where $q = 12289$.[1]
- Noise distribution χ: Ψ_{16}.[1]  on $\{-16, \ldots, 16\}$
- New reconciliation mechanism:
  - Only use "safe bits" that are far from an edge.
  - Additionally apply an error-correcting code.

---

[1] same as New Hope.

HILA5's reconciliation



(picture: HILA5 documentation)

For each coefficient:

$d = 0$: Discard coefficient.

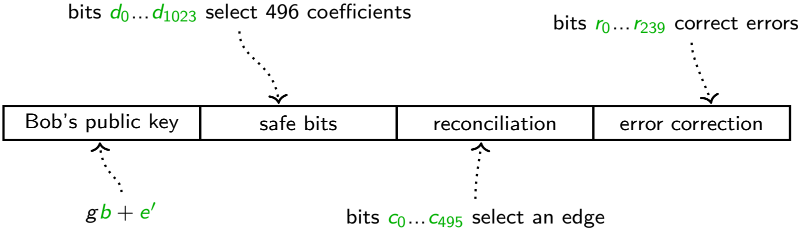
$d = 1$: Send reconciliation information c; use for key bit k.

Edges:

$c = 0$: $\lceil 3q/8 \rfloor \ldots \lceil 7q/8 \rfloor \rightsquigarrow k = 0.$
$\lceil 7q/8 \rfloor \ldots \lceil 3q/8 \rfloor \rightsquigarrow k = 1.$

$c = 1$: $\lceil q/8 \rfloor \ldots \lceil 5q/8 \rfloor \rightsquigarrow k = 0.$
$\lceil 5q/8 \rfloor \ldots \lceil \ q/8 \rfloor \rightsquigarrow k = 1.$

HILA5's packet format



bits $d_0 \ldots d_{1023}$ select 496 coefficients

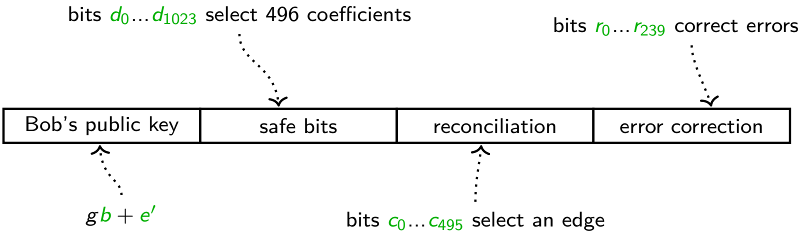
bits $r_0 \ldots r_{239}$ correct errors

Bob's public key	safe bits	reconciliation	error correction

$g b + e'$

bits $c_0 \ldots c_{495}$ select an edge

HILA5's packet format



bits $d_0 \ldots d_{1023}$ select 496 coefficients

bits $r_0 \ldots r_{239}$ correct errors

Bob's public key	safe bits	reconciliation	error correction

$g b + e'$

bits $c_0 \ldots c_{495}$ select an edge

We're going to manipulate each of these parts.

Unsafe bits

$gb + e'$	safe bits	reconciliation	error correction

We want to attack the first coefficient.

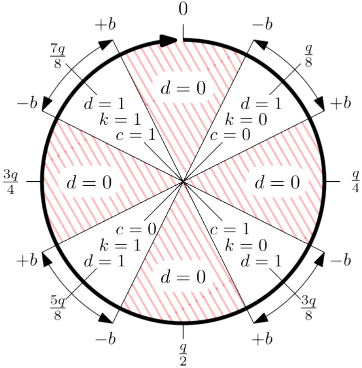
Unsafe bits

We want to attack the first coefficient.
$\Longrightarrow$ Force $d_0 = 1$ to make Alice use it.

Living on the edge

$gb + e'$	safe bits	reconciliation	error correction

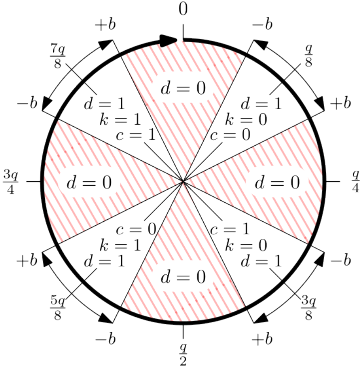
We want to attack the edge at $M = \lceil q/8 \rfloor$.



Living on the edge

$gb + e'$	safe bits	reconciliation	error correction

We want to attack the edge at $M = \lceil q/8 \rfloor$. $\Longrightarrow$ Force $c_0 = 1$.



Making errors

$gb + e'$	safe bits	reconciliation	error correction

- HILA5 uses a custom linear error-correcting code XE5.
- Encrypted (XOR) using part of Bob's shared secret S'.
- Ten variable-length codewords $R_0 \ldots R_9$.
- Alice corrects $S[0]$ using the first bit of each R_i.
- Capable of correcting (at least) 5-bit errors.

We want to keep errors in $S[0]$.

Making errors

- HILA5 uses a custom linear error-correcting code XE5.
- Encrypted (XOR) using part of Bob's shared secret S'.
- Ten variable-length codewords $R_0 \ldots R_9$.
- Alice corrects $S[0]$ using the first bit of each R_i.
- Capable of correcting (at least) 5-bit errors.

We want to keep errors in $S[0]$. $\implies$ Flip the first bit of $R_0 \ldots R_4$!

All coefficients for the price of one

Our binary search recovers $e'a[0]$ from $gab_\delta + e'a$ by varying δ.
How to get $a[1]$, $a[2]$, ..?

All coefficients for the price of one


$gb + e'$ | safe bits | reconciliation | error correction

Our binary search recovers $e'a[0]$ from $gab_\delta + e'a$ by varying δ.
How to get $a[1]$, $a[2]$, ..?

By construction of $R = \mathbf{Z}[x]/(q, x^{1024} + 1)$,
Evil Bob can rotate $a[i]$ into $e'a[0]$ by setting $e' = -x^{1024-i}$.

Running the search for all i yields all coefficients of a.

Evil Bob needs evil b_δ

Recall that Evil Bob needs b_δ such that $gab_\delta[0] = M + \delta$.
How to obtain b_δ without knowing a?

Evil Bob needs evil b_δ

Recall that Evil Bob needs b_δ such that $gab_\delta[0] = M + \delta$.
How to obtain b_δ without knowing a?

$\Longrightarrow$ Guess b_0 based on Alice's public key $A = ga + e$:

Evil Bob needs evil b_δ

Recall that Evil Bob needs b_δ such that $gab_\delta[0] = M + \delta$.
How to obtain b_δ without knowing a?

$\Longrightarrow$ Guess b_0 based on Alice's public key $A = ga + e$:

If b_0 has two entries ± 1 and $(Ab_0)[0] = M$, then

$$\Pr_{e \leftarrow \chi^n}[gab_0[0] = M] = \Pr_{x,y \leftarrow \Psi_{16}}[x + y = 0] \approx 9.9\%.$$

Evil Bob needs evil b_δ

Recall that Evil Bob needs b_δ such that $gab_\delta[0] = M + \delta$.
How to obtain b_δ without knowing a?

$\Longrightarrow$ Guess b_0 based on Alice's public key $A = ga + e$:
If b_0 has two entries ±1 and $(Ab_0)[0] = M$, then

$$\Pr_{e\leftarrow\chi^n}[gab_0[0] = M] = \Pr_{x,y\leftarrow\Psi_{16}}[x + y = 0] \approx 9.9\%.$$

For all other δ, set $b_\delta := (1 + \delta M^{-1} \bmod q) \cdot b_0$.
This works because $M^{-1} \bmod q = -8$ is small here.

Evil Bob needs evil b_δ

Recall that Evil Bob needs b_δ such that $gab_\delta[0] = M + \delta$.
How to obtain b_δ without knowing a?

$\Longrightarrow$ Guess b_0 based on Alice's public key $A = ga + e$:
If b_0 has two entries ± 1 and $(Ab_0)[0] = M$, then

$$\Pr_{e \leftarrow \chi^n}[gab_0[0] = M] = \Pr_{x,y \leftarrow \Psi_{16}}[x + y = 0] \approx 9.9\%.$$

For all other δ, set $b_\delta := (1 + \delta M^{-1} \bmod q) \cdot b_0$.
This works because $M^{-1} \bmod q = -8$ is small here.

If b_0 was wrong, the recovered coefficients are all 0 or -1.
$\Longrightarrow$ easily detectable.

Implementation

- Our code[1] attacks the HILA5 reference implementation.
- 100% success rate in our experiments.
- Less than 6000 queries (virtually always).

  (Note: Evil Bob could recover fewer coefficients and compute the rest by solving a lattice problem of reduced dimension.)

---

[1] https://helaas.org/hila5-20171218.tar.gz


TU/E KWALITEITSGARANTIE
SMEUÏG TOT OP DE BODEM
NOF
BRON VAN BOUWSTOFFEN*
NOF
HILA5
PINDAKAAS
STUKJES PINDA

HK17

"HK17 consists broadly in a Key Exchange Protocol (KEP) based on non-commutative algebra of hypercomplex numbers limited to quaternions and octonions. In particular, this proposal is based on non-commutative and non-associative algebra using octonions."

Security analysis: ". . . In our protocol, we could not find any ways to proceed with any abelianization of our octonions non-associative Moufang loop [29] or reducing of the GSDP problem of polynomial powers of octonions to a finitely generated nilpotent image of the given free group in the cryptosystem and a further nonlinear decomposition attack. We simply conclude that Roman'kov attacks do not affect our proposal."

What are octonions?

R: set of real numbers.

C: set of complex numbers; dim-2 **R**-vector space.

H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.

O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

What are octonions?

R: set of real numbers.
C: set of complex numbers; dim-2 **R**-vector space.
H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.
O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.

What are octonions?

R: set of real numbers.
C: set of complex numbers; dim-2 **R**-vector space.
H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.
O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.
- Conjugation $q \mapsto q^*$. (For **R**: the identity map.)

What are octonions?

R: set of real numbers.
C: set of complex numbers; dim-2 **R**-vector space.
H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.
O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.
- Conjugation $q \mapsto q^*$. (For **R**: the identity map.)
- Multiplication $q, r \mapsto qr$. (**R**, **C**: commutative. **R**, **C**, **H**: associative.)

What are octonions?

$\mathbf{R}$: set of real numbers.
$\mathbf{C}$: set of complex numbers; dim-2 $\mathbf{R}$-vector space.
$\mathbf{H}$: set of quaternions; dim-4 $\mathbf{R}$-vector space; 1843 Hamilton.
$\mathbf{O}$: set of octonions; dim-8 $\mathbf{R}$-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.
- Conjugation $q \mapsto q^*$. (For $\mathbf{R}$: the identity map.)
- Multiplication $q, r \mapsto qr$. ($\mathbf{R}, \mathbf{C}$: commutative. $\mathbf{R}, \mathbf{C}, \mathbf{H}$: associative.)

Simple unified definition from 1919 Dickson:

- $\mathbf{O} = \mathbf{H} \times \mathbf{H}$ with conjugation $(q, Q)^* = (q^*, -Q)$; multiplication $(q, Q)(r, R) = (qr - R^*Q, Rq + Qr^*)$.

What are octonions?

R: set of real numbers.
C: set of complex numbers; dim-2 **R**-vector space.
H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.
O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.
- Conjugation $q \mapsto q^*$. (For **R**: the identity map.)
- Multiplication $q, r \mapsto qr$. (**R**, **C**: commutative. **R**, **C**, **H**: associative.)

Simple unified definition from 1919 Dickson:

- $\mathbf{O} = \mathbf{H} \times \mathbf{H}$ with conjugation $(q, Q)^* = (q^*, -Q)$; multiplication $(q, Q)(r, R) = (qr - R^*Q, Rq + Qr^*)$.
- $\mathbf{H} = \mathbf{C} \times \mathbf{C}$ with same formulas.

What are octonions?

R: set of real numbers.
C: set of complex numbers; dim-2 **R**-vector space.
H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.
O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.
- Conjugation $q \mapsto q^*$. (For **R**: the identity map.)
- Multiplication $q, r \mapsto qr$. (**R**, **C**: commutative. **R**, **C**, **H**: associative.)

Simple unified definition from 1919 Dickson:

- $\mathbf{O} = \mathbf{H} \times \mathbf{H}$ with conjugation $(q, Q)^* = (q^*, -Q)$; multiplication $(q, Q)(r, R) = (qr - R^*Q, Rq + Qr^*)$.
- $\mathbf{H} = \mathbf{C} \times \mathbf{C}$ with same formulas.
- $\mathbf{C} = \mathbf{R} \times \mathbf{R}$ with same formulas.

What are octonions?

R: set of real numbers.
C: set of complex numbers; dim-2 **R**-vector space.
H: set of quaternions; dim-4 **R**-vector space; 1843 Hamilton.
O: set of octonions; dim-8 **R**-vector space; 1845 Cayley, 1845 Graves.

Each of these sets has a three-part definition:

- Elements.
- Conjugation $q \mapsto q^*$. (For **R**: the identity map.)
- Multiplication $q, r \mapsto qr$. (**R**, **C**: commutative. **R**, **C**, **H**: associative.)

Simple unified definition from 1919 Dickson:

- $\mathbf{O} = \mathbf{H} \times \mathbf{H}$ with conjugation $(q, Q)^* = (q^*, -Q)$; multiplication $(q, Q)(r, R) = (qr - R^*Q, Rq + Qr^*)$.
- $\mathbf{H} = \mathbf{C} \times \mathbf{C}$ with same formulas.
- $\mathbf{C} = \mathbf{R} \times \mathbf{R}$ with same formulas.

Exercise: Every $q \in \mathbf{O}$ has $q^2 = tq - n$ and $q^* = t - q$ for some $t, n \in \mathbf{R}$.

How does HK17 work?

Use integers modulo prime p instead of real numbers.
HK17 submission claims 2^{256} security for $p = 2^{32} - 5$.

How does HK17 work?

Use integers modulo prime p instead of real numbers.
HK17 submission claims 2^{256} security for $p = 2^{32} - 5$.

Alice:

- Generate secret integers $m, n, f_0, f_1, \ldots, f_{32} > 0$.
- Generate public octonions q, r; secret $a = f_0 + f_1 q + \cdots + f_{32} q^{32}$.
- Send $q, r, a^m r a^n$ to Bob.

How does HK17 work?

Use integers modulo prime p instead of real numbers.
HK17 submission claims 2^{256} security for $p = 2^{32} - 5$.

Alice:

- Generate secret integers $m, n, f_0, f_1, \dots, f_{32} > 0$.
- Generate public octonions q, r; secret $a = f_0 + f_1 q + \cdots + f_{32} q^{32}$.
- Send $q, r, a^m r a^n$ to Bob.

Bob:

- Generate secret integers $k, \ell, h_0, h_1, \dots, h_{32} > 0$.
- Generate secret $b = h_0 + h_1 q + \cdots + h_{32} q^{32}$.
- Send $b^k r b^\ell$ to Alice.

How does HK17 work?

Use integers modulo prime p instead of real numbers.
HK17 submission claims 2^{256} security for $p = 2^{32} - 5$.

Alice:

- Generate secret integers $m, n, f_0, f_1, \ldots, f_{32} > 0$.
- Generate public octonions q, r; secret $a = f_0 + f_1 q + \cdots + f_{32} q^{32}$.
- Send $q, r, a^m r a^n$ to Bob.

Bob:

- Generate secret integers $k, \ell, h_0, h_1, \ldots, h_{32} > 0$.
- Generate secret $b = h_0 + h_1 q + \cdots + h_{32} q^{32}$.
- Send $b^k r b^\ell$ to Alice.

Shared secret: $a^m (b^k r b^\ell) a^n = b^k (a^m r a^n) b^\ell$.

Why does HK17 work?

Does $a^m r a^n$ mean $(a^m r)a^n$, or $a^m(ra^n)$?
Does a^m mean $a(a(\cdots))$, or $((\cdots)a)a$?

Why does HK17 work?

Does $a^m r a^n$ mean $(a^m r)a^n$, or $a^m(r a^n)$?
Does a^m mean $a(a(\cdots))$, or $((\cdots)a)a$?

Octonions satisfy some partial associativity rules:

- **Flexible identity**: $x(yx) = (xy)x$.
- **Alternative identity**: $x(xy) = (xx)y$ and $y(xx) = (yx)x$.
- **Moufang identities**: $z(x(zy)) = ((zx)z)y$; $x(z(yz)) = ((xz)y)z$; $(zx)(yz) = (z(xy))z = z((xy)z)$.

Why does HK17 work?

Does $a^m r a^n$ mean $(a^m r)a^n$, or $a^m(ra^n)$?
Does a^m mean $a(a(\cdots))$, or $((\cdots)a)a$?

Octonions satisfy some partial associativity rules:

- **Flexible identity**: $x(yx) = (xy)x$.
- **Alternative identity**: $x(xy) = (xx)y$ and $y(xx) = (yx)x$.
- **Moufang identities**: $z(x(zy)) = ((zx)z)y$; $x(z(yz)) = ((xz)y)z$; $(zx)(yz) = (z(xy))z = z((xy)z)$.

So $a(aa) = (aa)a$; $a(a(aa)) = (aa)(aa) = ((aa)a)a$; etc.

Why does HK17 work?

Does $a^m r a^n$ mean $(a^m r)a^n$, or $a^m(ra^n)$?
Does a^m mean $a(a(\cdots))$, or $((\cdots)a)a$?

Octonions satisfy some partial associativity rules:

- **Flexible identity**: $x(yx) = (xy)x$.
- **Alternative identity**: $x(xy) = (xx)y$ and $y(xx) = (yx)x$.
- **Moufang identities**: $z(x(zy)) = ((zx)z)y$; $x(z(yz)) = ((xz)y)z$; $(zx)(yz) = (z(xy))z = z((xy)z)$.

So $a(aa) = (aa)a$; $a(a(aa)) = (aa)(aa) = ((aa)a)a$; etc.
Also $(ar)(aa) = a((ra)a) = a(r(aa))$;
$(ar)((aa)a) = a((r(aa))a) = a(((ra)a)a) = a(r(a(aa)))$; etc.

Why does HK17 work?

Does $a^m r a^n$ mean $(a^m r)a^n$, or $a^m(ra^n)$?
Does a^m mean $a(a(\cdots))$, or $((\cdots)a)a$?

Octonions satisfy some partial associativity rules:

- **Flexible identity**: $x(yx) = (xy)x$.
- **Alternative identity**: $x(xy) = (xx)y$ and $y(xx) = (yx)x$.
- **Moufang identities**: $z(x(zy)) = ((zx)z)y$; $x(z(yz)) = ((xz)y)z$; $(zx)(yz) = (z(xy))z = z((xy)z)$.

So $a(aa) = (aa)a$; $a(a(aa)) = (aa)(aa) = ((aa)a)a$; etc.
Also $(ar)(aa) = a((ra)a) = a(r(aa))$;
$(ar)((aa)a) = a((r(aa))a) = a(((ra)a)a) = a(r(a(aa)))$; etc.

$q^m(q^k r q^\ell)q^n = q^k(q^m r q^n)q^\ell$.
$a^m(b^k r b^\ell)a^n = b^k(a^m r a^n)b^\ell$ because a, b are polynomials in q.

A fast attack, and a faster attack

Remember the exercise: q^2 is a linear combination of $1, q$.
So every polynomial in q is a linear combination of $1, q$.
There are only p^2 of these combinations!

A fast attack, and a faster attack

Remember the exercise: q^2 is a linear combination of $1, q$.
So every polynomial in q is a linear combination of $1, q$.
There are only p^2 of these combinations!

Attacker sees $a^m r a^n$, tries p^2 possibilities for a^m.
Recognizing correct possibility: a^n is linear combination of $1, q$.
"Fake" solutions aren't a problem: good enough for decryption.

A fast attack, and a faster attack

Remember the exercise: q^2 is a linear combination of $1, q$.
So every polynomial in q is a linear combination of $1, q$.
There are only p^2 of these combinations!

Attacker sees $a^m r a^n$, tries p^2 possibilities for a^m.
Recognizing correct possibility: a^n is linear combination of $1, q$.
"Fake" solutions aren't a problem: good enough for decryption.

Even faster: Attacker tries only $q, q+1, q+2, q+3, \ldots$.
Finds integer multiple of a^m; good enough for decryption.
This was the first attack script: 2^{32} fast computations.

A fast attack, and a faster attack

Remember the exercise: q^2 is a linear combination of $1, q$.
So every polynomial in q is a linear combination of $1, q$.
There are only p^2 of these combinations!

Attacker sees $a^m r a^n$, tries p^2 possibilities for a^m.
Recognizing correct possibility: a^n is linear combination of $1, q$.
"Fake" solutions aren't a problem: good enough for decryption.

Even faster: Attacker tries only $q, q+1, q+2, q+3, \ldots$.
Finds integer multiple of a^m; good enough for decryption.
This was the first attack script: 2^{32} fast computations.

Even faster: Attacker solves $a^m r a^n = (q+x)r(yq+z)$.
Eight equations in three variables x, y, z; linearize.
This was the second attack script: practically instantaneous.

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- Why are these equal?

$$v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc$$

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- Why are these equal?

$$v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$$

- Why does the weight restriction hold?

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- Why are these equal?

$$v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$$

- Why does the weight restriction hold?
  S and y are sparse, but each entry in Sc is sum over n positions

$$z_i = y_i + \sum_{j=1}^{n} S_{ij} c_j .$$

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- Why are these equal?

$$v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$$

- Why does the weight restriction hold?
  S and y are sparse, but each entry in Sc is sum over n positions

$$z_i = y_i + \sum_{j=1}^{n} S_{ij} c_j .$$

This needs a special hash function so that c is sparse.

RaCoSS – Random Code-based Signature Schemes

- System parameters: $n = 2400$, $k = 2060$.
  Random matrix $H \in \mathbf{F}_2^{(n-k)\times n}$.
- Secret key: sparse $S \in \mathbf{F}_2^{n\times n}$.
- Public key: $T = H \cdot S$. (looks pretty random).
- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.
- Why are these equal?

$$v' = Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Hy = v$$

- Why does the weight restriction hold?
  S and y are sparse, but each entry in Sc is sum over n positions

$$z_i = y_i + \sum_{j=1}^{n} S_{ij} c_j.$$

This needs a special hash function so that c is very sparse.

PQCRYPTO
ICT-645622

The weight-restricted hash function (`wrhf`)

- Maps to 2400-bit strings of weight 3.

The weight-restricted hash function (`wrhf`)

- Maps to 2400-bit strings of weight 3.
- Only

  $$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

  possible outputs.

The weight-restricted hash function (`wrhf`)

- Maps to 2400-bit strings of weight 3.
- Only
$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$
possible outputs.
- Slow: 600 to 800 hashes per second and core.
- Expected time for a preimage on $\approx$ 100 cores: 10 hours.

RaCoSS

Implementation bug:

```
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```

RaCoSS

Implementation bug:

```
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```

RaCoSS

Implementation bug:

```
unsigned char  c[RACOSS_N];
unsigned char  c2[RACOSS_N];

/* ... */

for( i=0 ; i<(RACOSS_N/8) ; i++ )
    if( c2[i] != c[i] )
        /* fail */

return 0; /* accept */
```

...compares only the first 300 coefficients!
Thus, a signature with $c[0\ldots299] = 0$ is accepted for

$$\binom{2100}{3} \Big/ \binom{2400}{3} \approx 67\%$$

of all messages.

The weight-restricted hash function (`wrhf`)

- Maps to 2400-bit strings of weight 3.
- Only

  $$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

  possible outputs.
- Slow: 600 to 800 hashes per second and core.
- Expected time for a preimage on ≈ 100 cores: 10 hours.
- crashed while brute-forcing: memory leaks
- another message signed by the first KAT:

NISTPQC is so much fun! 10900qmmP

Wait, there is more!

- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} = \begin{pmatrix} & & H & & \end{pmatrix} \begin{pmatrix} \\ \\ z \\ \\ \\ \end{pmatrix}$$

- Sign without knowing S: $(c, y, z \in \mathbf{F}_2^n,\ v, Tc \in \mathbf{F}_2^{n-k})$.

Wait, there is more!

- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \\ \quad \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \qquad\qquad H \qquad\qquad \\ \\ \end{pmatrix} \begin{pmatrix} \\ \\ z \\ \\ \\ \end{pmatrix}$$

- Sign without knowing S: $(c, y, z \in \mathbf{F}_2^n,\ v, Tc \in \mathbf{F}_2^{n-k})$.
  Pick a low weight $y \in \mathbf{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.

Wait, there is more!

- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \quad \\ \\ \end{pmatrix} = \begin{pmatrix} \quad & H & \quad \\ \\ \end{pmatrix} \begin{pmatrix} \\ z \\ \\ \end{pmatrix}$$

- Sign without knowing S: $(c, y, z \in \mathbf{F}_2^n,\ v, Tc \in \mathbf{F}_2^{n-k})$.
  Pick a low weight $y \in \mathbf{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.

Wait, there is more!

- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that weight$(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \begin{pmatrix} \quad \end{pmatrix} = \begin{pmatrix} H_1 & H_2 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

- Sign without knowing S: $(c, y, z \in \mathbf{F}_2^n,\ v, Tc \in \mathbf{F}_2^{n-k})$.
  Pick a low weight $y \in \mathbf{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.
  Pick $n - k$ columns of H that form an invertible matrix H_1.

Wait, there is more!

- Sign m: Pick a low weight $y \in \mathbf{F}_2^n$.
  Compute $v = Hy$, $c = h(v, m)$, $z = Sc + y$. Output (z, c).
- Verify $m, (z, c)$: Check that $\text{weight}(z) \leq 1564$.
  Compute $v' = Hz + Tc$. Check that $h(v', m) = c$.

$$v + Tc = \left(\quad \right) = \begin{pmatrix} H_1 & H_2 \end{pmatrix} \begin{pmatrix} z_1 \\ \\ z_2 \end{pmatrix}$$

- Sign without knowing S: $(c, y, z \in \mathbf{F}_2^n,\ v, Tc \in \mathbf{F}_2^{n-k})$.
  Pick a low weight $y \in \mathbf{F}_2^n$. Compute $v = Hy$, $c = h(v, m)$.
  Pick $n - k$ columns of H that form an invertible matrix H_1.
- Compute $z = (z_1 || 00 \ldots 0)$ by linear algebra.
- Expected weight of z is $\approx (n-k)/2 = 170 \ll 1564$.
- Properly generated signatures have $\text{weight}(z) \approx 261$.

RaCoSS – Summary

- Bug in code: bit vs. byte confusion meant only every 8th bit verified.
- Preimages for RaCoSS' special hash function: only

  $$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

  possible outputs.
- The code dimensions give a lot of freedom to the attacker – our forged signature is better than a real one!

Code-based encryption

BIG QUAKE
Classic McEliece
LAKE
LOCKER
DAGS
LEDAkem
LEDApkc
Lepton
McNie

Edon-K✝
BIKE☢
HQC☢
NTS-KEM☢
Ouroboros-R☢
QC-MDPC KEM☢
RQC☢
RLCE-KEM☢

✝: submitter has withdrawn submission.

Lattice-based encryption

CRYSTALS-KYBER
EMBLEM and R.EMBLEM
FrodoKEM
KINDI
LAC
LIMA
LOTUS
NewHope
NTRUEncrypt
NTRU-HRSS-KEM

NTRU Prime
Odd Manhattan
SABER
Titanium
HILA5
Ding Key Exchange☢
Lizard☢
KCL OKCN/AKCN/CNKE☢
Round2☢
Compact LWE☢

Other encryption

SIKE: isogeny-based encryption

Other encryption

SIKE: isogeny-based encryption

Mersenne-756839: integer-ring encryption
Ramstake: integer-ring encryption
Three Bears: integer-ring encryption

Other encryption

SIKE: isogeny-based encryption

Mersenne-756839: integer-ring encryption
Ramstake: integer-ring encryption
Three Bears: integer-ring encryption

pqRSA: factoring-based encryption

Other encryption

SIKE: isogeny-based encryption

Mersenne-756839: integer-ring encryption
Ramstake: integer-ring encryption
Three Bears: integer-ring encryption

pqRSA: factoring-based encryption

CFPKM: multivariate encryption
SRTPI✝: multivariate encryption
DME☢: multivariate encryption

Other encryption

SIKE: isogeny-based encryption

Mersenne-756839: integer-ring encryption
Ramstake: integer-ring encryption
Three Bears: integer-ring encryption

pqRSA: factoring-based encryption

CFPKM: multivariate encryption
SRTPI✝: multivariate encryption
DME☢: multivariate encryption

Guess Again: hard to classify
HK17✝: hard to classify
RVB✝: hard to classify

Signatures

Gravity-SPHINCS: hash-based
Picnic: hash-based
SPHINCS+: hash-based

DualModeMS: multivariate
GeMSS: multivariate
HiMQ-3: multivariate
LUOV: multivariate
Giophantus: multivariate
Gui☢: multivariate
MQDSS☢: multivariate
Rainbow☢: multivariate

pqRSA: factoring-based

CRYSTALS-DILITHIUM: lattice-based
qTESLA: lattice-based
DRS: lattice-based
FALCON☢: lattice-based
pqNTRUSign☢: lattice-based

pqsigRM: code-based
RaCoSS: code-based
RankSign✝: code-based

WalnutDSA☢: braid-group

Further resources

- https://2017.pqcrypto.org/school: PQCRYPTO summer school with 21 lectures on video + slides + exercises.
- https://2017.pqcrypto.org/exec: Executive school (12 lectures), less math, more overview. So far slides, soon videos.
- https://pqcrypto.org: Our survey site.
  - Many pointers: e.g., to PQCrypto conferences.
  - Bibliography for 4 major PQC systems.
- https://pqcrypto.eu.org: PQCRYPTO EU project.
  - Expert recommendations.
  - Free software libraries.
  - More video presentations, slides, papers.
- https://twitter.com/pqc_eu: PQCRYPTO Twitter feed.
- https://twitter.com/PQCryptoConf: PQCrypto conference Twitter feed.
- https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions NIST PQC competition.