A one-time single-bit fault leaks all previous NTRU-HRSS session keys to a chosen-ciphertext attack

D. J. Bernstein

University of Illinois at Chicago;
Ruhr University Bochum

---

cr.yp.to/papers.html#ntrw

Thanks to Lange for pointing out plaintext confirmation as a countermeasure to fault attacks.

PQ deployment and standards

2022.04: OpenSSH 9.0 uses
`x25519+sntrup761` by default.

PQ deployment and standards

2022.04: OpenSSH 9.0 uses `x25519+sntrup761` by default.

2022.07: NIST announces intent to standardize Kyber (+ sigs).

PQ deployment and standards

2022.04: OpenSSH 9.0 uses `x25519+sntrup761` by default.

2022.07: NIST announces intent to standardize Kyber (+ sigs).

2022.11: Google announces that all internal Google networking uses `x25519+ntruhrss701`.

PQ deployment and standards

2022.04: OpenSSH 9.0 uses `x25519+sntrup761` by default.

2022.07: NIST announces intent to standardize Kyber (+ sigs).

2022.11: Google announces that all internal Google networking uses `x25519+ntruhrss701`.

"Kyber has high performance . . . but still lacks some clarification from NIST about its Intellectual Property status", i.e., patents.

2010–2017 patents listed in NTRU Prime FAQ: US9094189, US9246675, CN107566121, CN108173643, KR101905689, US11050557, EP3698515.

2010–2017 patents listed in NTRU Prime FAQ: US9094189, US9246675, CN107566121, CN108173643, KR101905689, US11050557, EP3698515.

2022.11: NIST announces licenses for US9094189, US9246675 *for Kyber v2024 after Kyber v2024 is defined and standardized*.
No analysis of other patents.

2010–2017 patents listed in NTRU Prime FAQ: US9094189, US9246675, CN107566121, CN108173643, KR101905689, US11050557, EP3698515.

2022.11: NIST announces licenses for US9094189, US9246675 *for Kyber v2024 after Kyber v2024 is defined and standardized*.
No analysis of other patents.

For deploying software to protect users *now*, NTRU-HRSS is attractive: small, fast, unpatented.

Is NTRU-HRSS secure?

2017 HRSS paper says: NTRU proposal for OW-CPA encryption has “a track record of surviving 20 years of cryptanalysis”.

Is NTRU-HRSS secure?

2017 HRSS paper says: NTRU proposal for OW-CPA encryption has "a track record of surviving 20 years of cryptanalysis".

Make various changes, including: "We now show how to turn the above OW-CPA secure encryption into an IND-CCA2-secure KEM"— i.e., include extra defenses to stop chosen-ciphertext attacks.

Is NTRU-HRSS secure?

2017 HRSS paper says: NTRU proposal for OW-CPA encryption has "a track record of surviving 20 years of cryptanalysis".

Make various changes, including: "We now show how to turn the above OW-CPA secure encryption into an IND-CCA2-secure KEM"— i.e., include extra defenses to stop chosen-ciphertext attacks.

HRSS uses Fujisaki–Okamoto (FO) transform, specifically one of the variants from 2002 Dent.

Defense 1: After decrypting ciphertext C to obtain message m, reencrypt m and reject if $\neq C$.

Defense 1: After decrypting ciphertext C to obtain message m, reencrypt m and reject if $\neq C$.

This stops chosen-ciphertext attacks that probe variants of a legitimate C to see which variants decrypt to the same m.

Defense 1: After decrypting ciphertext C to obtain message m, reencrypt m and reject if $\neq C$.

This stops chosen-ciphertext attacks that probe variants of a legitimate C to see which variants decrypt to the same m.

If encryption is randomized, first derandomize it: obtain random bits as $H(m)$.

Defense 1: After decrypting ciphertext C to obtain message m, reencrypt m and reject if $\neq C$.

This stops chosen-ciphertext attacks that probe variants of a legitimate C to see which variants decrypt to the same m.

If encryption is randomized, first derandomize it: obtain random bits as $H(m)$. Make sure m has high entropy!

Defense 1: After decrypting ciphertext C to obtain message m, reencrypt m and reject if $\neq C$.

This stops chosen-ciphertext attacks that probe variants of a legitimate C to see which variants decrypt to the same m.

If encryption is randomized, first derandomize it: obtain random bits as $H(m)$. Make sure m has high entropy! See recent collapse of "FrodoKEM parameter sets comfortably match their target security levels with a large margin".

Defense 3 (in the numbering from `ntrw`'s survey of attacks and defenses): plaintext confirmation.

Instead of ciphertext $E(m)$, send ciphertext $(E(m), H'(m))$ where H' is a hash function. Also use (E, H') in reencryption.

Defense 3 (in the numbering from `ntrw`'s survey of attacks and defenses): plaintext confirmation.

Instead of ciphertext $E(m)$, send ciphertext $(E(m), H'(m))$ where H' is a hash function. Also use (E, H') in reencryption.

This stops chosen-ciphertext attacks that exploit structure of the public-key encryption function E to convert $E(m)$ for secret m into, e.g., $E(m + 1)$. Attacker has no way to convert $H'(m)$ into $H'(m+1)$ for "unstructured" H'.

Current NTRU-HRSS is different

2019 NTRU-HRSS proposal adopts changes proposed by 2017 Saito–Xagawa–Yamakawa.

Modified proposal **removes plaintext confirmation** and relies on another defense.

Current NTRU-HRSS is different

2019 NTRU-HRSS proposal adopts changes proposed by 2017 Saito–Xagawa–Yamakawa.

Modified proposal **removes plaintext confirmation** and relies on another defense.

Defense 4, implicit rejection (from 2017 Hofheinz–Hövelmanns–Kiltz, generalizing 2012 Persichetti): instead of having a KEM reject an invalid ciphertext C, have it output $H''(r, C)$ where r is a random string stored in secret key.

Is implicit rejection really an adequate substitute for plaintext confirmation as a defense against chosen-ciphertext attacks?

Is implicit rejection really an adequate substitute for plaintext confirmation as a defense against chosen-ciphertext attacks?

SXY+HRSS answer: Here's a **proof** of IND-CCA2 security from OW-CPA + implicit rejection.

Is implicit rejection really an adequate substitute for plaintext confirmation as a defense against chosen-ciphertext attacks?

SXY+HRSS answer: Here's a **proof** of IND-CCA2 security from OW-CPA + implicit rejection.

Issue 1: Proof is only in QROM; are there non-QROM attacks?

Is implicit rejection really an adequate substitute for plaintext confirmation as a defense against chosen-ciphertext attacks?

SXY+HRSS answer: Here's a **proof** of IND-CCA2 security from OW-CPA + implicit rejection.

Issue 1: Proof is only in QROM; are there non-QROM attacks?
Issue 2: Proof is tight only in ROM; can this be exploited?

Is implicit rejection really an adequate substitute for plaintext confirmation as a defense against chosen-ciphertext attacks?

SXY+HRSS answer: Here's a **proof** of IND-CCA2 security from OW-CPA + implicit rejection.

Issue 1: Proof is only in QROM; are there non-QROM attacks?
Issue 2: Proof is tight only in ROM; can this be exploited?
Issue 3, my focus today: Are there chosen-ciphertext attacks beyond the IND-CCA2 model?

2007 Koblitz, regarding HMQV: "Anyone working in cryptography should think very carefully before dropping a validation step that had been put in to prevent security problems. Certainly someone with Krawczyk's experience and expertise would never have made such a blunder if he hadn't been over-confident because of his 'proof' of security."

See also 2019 survey of failures.

2007 Koblitz, regarding HMQV: "Anyone working in cryptography should think very carefully before dropping a validation step that had been put in to prevent security problems. Certainly someone with Krawczyk's experience and expertise would never have made such a blunder if he hadn't been over-confident because of his 'proof' of security."

See also 2019 survey of failures.

Should think very carefully before dropping plaintext confirmation.

2018 Bernstein–Persichetti: implicit rejection "produces random-looking session keys" for invalid ciphertexts, "so it hides the pattern of valid ciphertexts"; plaintext confirmation "stops an earlier stage of the attack"; current proofs do not "show any advantages for the dual-defense construction" **but** it "seems difficult to justify a recommendation against the dual-defense construction" given that the defenses "target different aspects of attacks".

An attack against NTRU-HRSS

DRAM hardware is unreliable.
Often stored bits are corrupted.
Google statistics $\Rightarrow 10^9$ users,
each storing a 256-bit key in
DRAM, will have 50000–140000
keys corrupted each year.

An attack against NTRU-HRSS

DRAM hardware is unreliable. Often stored bits are corrupted. Google statistics $\Rightarrow 10^9$ users, each storing a 256-bit key in DRAM, will have 50000–140000 keys corrupted each year.

Main point of the `ntrw` paper: implicit rejection doesn't do its job if r is corrupted. Attacker detects invalid ciphertexts: changing r changes decryption output. See paper for application to NTRU-HRSS and full attack software.

What can we do in response?

Incompatible new NTRU-HRSS can re-add plaintext confirmation.

What can we do in response?

Incompatible new NTRU-HRSS can re-add plaintext confirmation.

Can fix corruption by applying an error-correcting code (ECC):

- `ntrw`'s `libsecded` software; or
- SECDED ECC DRAM hardware.

Many benefits beyond this attack.

What can we do in response?

Incompatible new NTRU-HRSS can re-add plaintext confirmation.

Can fix corruption by applying an error-correcting code (ECC):

- `ntrw`'s `libsecded` software; or
- SECDED ECC DRAM hardware.

Many benefits beyond this attack.

Specify ECC in secret-key format?
Use ECC in crypto libraries?
Use ECC in applications?
Programming language? OS?
Require SECDED ECC DRAM?

What can we do in response?

Incompatible new NTRU-HRSS can re-add plaintext confirmation.

Can fix corruption by applying an error-correcting code (ECC):

- `ntrw`'s `libsecded` software; or
- SECDED ECC DRAM hardware.

Many benefits beyond this attack.

Specify ECC in secret-key format?
Use ECC in crypto libraries?
Use ECC in applications?
Programming language? OS?
Require SECDED ECC DRAM?
Point fingers and do nothing?

Classic McEliece followup

2022.10: Classic McEliece recommends dropping plaintext confirmation "to proactively eliminate any concerns regarding U.S. patent 9912479".

Classic McEliece followup

2022.10: Classic McEliece recommends dropping plaintext confirmation "to proactively eliminate any concerns regarding U.S. patent 9912479".

Warns that this allows the `ntrw` attack whenever r is corrupted. Describes ECC as a defense.

Classic McEliece followup

2022.10: Classic McEliece recommends dropping plaintext confirmation "to proactively eliminate any concerns regarding U.S. patent 9912479".

Warns that this allows the `ntrw` attack whenever r is corrupted. Describes ECC as a defense.

Introduces principle of factoring "any generic transformation aiming at a goal beyond IND-CCA2" out of KEM specifications, to simplify design and review.