The libpqcrypto software library for post-quantum cryptography

Daniel J. Bernstein
and many contributors

libpqcrypto.org
Context
Redesigning crypto for security

New requirements for crypto software engineering to avoid real-world crypto disasters:

- No data flow from secrets to array indices. Stops, e.g., 2016 CacheBleed attack.
- No data flow from secrets to branch conditions. Stops, e.g., 2018 RSA key-generation attack by Aldaya–García–Tapia–Brumley.
- No padding oracles. Stops, e.g., 2017 ROBOT attack.
Redesigning crypto for security

New requirements for crypto software engineering to avoid real-world crypto disasters:

- No data flow from secrets to array indices. Stops, e.g., 2016 CacheBleed attack.
- No data flow from secrets to branch conditions. Stops, e.g., 2018 RSA key-generation attack by Aldaya–García–Tapia–Brumley.
Redesigning crypto for security

New requirements for crypto software engineering to avoid real-world crypto disasters:

▶ No data flow from secrets to array indices. Stops, e.g., 2016 CacheBleed attack.
▶ No data flow from secrets to branch conditions. Stops, e.g., 2018 RSA key-generation attack by Aldaya–García–Tapia–Brumley.
▶ No padding oracles. Stops, e.g., 2017 ROBOT attack.
Redesigning crypto for security

But wait, there’s more:

- Centralizing randomness: system has *one* central audited fast PRNG. Stops, e.g., Juniper fiasco discovered in 2015.
Redesigning crypto for security

But wait, there’s more:

- **Centralizing randomness:**
  system has *one* central audited fast PRNG.
  Stops, e.g., Juniper fiasco discovered in 2015.

- **Avoiding unnecessary randomness:**
  use audited deterministic functions.
  Stops, e.g., 2017 ROCA attack.
Redesigning crypto for security

But wait, there’s more:

- **Centralizing randomness:**
  system has *one* central audited fast PRNG.
  Stops, e.g., Juniper fiasco discovered in 2015.

- **Avoiding unnecessary randomness:**
  use audited deterministic functions.
  Stops, e.g., 2017 ROCA attack.

- **Eliminate low-security options.**
  Stops, e.g., 2015 Logjam attack.
Secure (and fast enough) crypto: Much simpler if we **upgrade crypto primitives and protocols**.

**Curve25519, Ed25519, etc.**

Example: Upgrading signatures.

- Use **ECC, not RSA**.
- Does the user really need "RSA signatures"? Or is the goal "high-security signatures"?
- Use **Curve25519**, not NSA (NIST) curves.
  - Simpler (and faster!) secure implementations.
- Use **EdDSA (Ed25519)**, not NSA signatures.
  - Avoid, e.g., hassle of implementing inversion.
Secure (and fast enough) crypto: Much simpler if we upgrade crypto primitives and protocols.

Example: Upgrading signatures.

- Use ECC, not RSA.
  Does the user really need “RSA signatures”? Or is the goal “high-security signatures”?

libpqcrypto.org

Daniel J. Bernstein
Secure (and fast enough) crypto: Much simpler if we *upgrade crypto primitives and protocols*. 

Example: Upgrading signatures.

- Use ECC, not RSA. 
  Does the user really need “RSA signatures”? Or is the goal “high-security signatures”? 
- Use Curve25519, not NSA (NIST) curves. Simpler (and faster!) secure implementations.
Secure (and fast enough) crypto: Much simpler if we **upgrade crypto primitives and protocols**.

Example: Upgrading signatures.

- Use ECC, not RSA.
  Does the user really need “RSA signatures”? Or is the goal “high-security signatures”?
- Use Curve25519, not NSA (NIST) curves.
  Simpler (and faster!) secure implementations.
- Use EdDSA (Ed25519), not NSA signatures.
  Avoid, e.g., hassle of implementing inversion.

libpqcrypto.org  Daniel J. Bernstein
A modern cryptographic API

Most libraries provide simple all-in-one hashing:

```c
const unsigned char m[...];
unsigned long long mlen;
unsigned char h[crypto_hash_BYTES];
crypto_hash_sha256(h, m, mlen);
```
A modern cryptographic API

Most libraries provide simple all-in-one hashing:

```
const unsigned char m[...];
unsigned long long mlen;
unsigned char h[crypto_hash_BYTES];
crypto_hash_sha256(h,m,mlen);
```

Why not the same simplicity for, e.g., signing?

```
crypto_sign_ed25519(sm,&smlen,m,mlen,sk);
```
A modern cryptographic API

Most libraries provide simple all-in-one hashing:

```c
const unsigned char m[...];
unsigned long long mlen;
unsigned char h[crypto_hash_BYTES];
crypto_hash_sha256(h,m,mlen);
```

Why not the same simplicity for, e.g., signing?

```c
crypto_sign_ed25519(sm,&smlen,m,mlen,sk);
```


libpqcrypto.org

Daniel J. Bernstein
Implementation and deployment

Curve25519: iOS starting 2010; WhatsApp starting 2016; formal verif in Firefox starting 2017; etc.

NaCl software library (forks: TweetNaCl, libsodium): Curve25519, audited implementations, modern API.

Competitions: Modern API required for submissions to CAESAR, NIST PQC, NIST Lightweight Crypto.

SUPERCOP benchmarking framework: Modern API, no requirement of constant-time etc. Currently 2556 implementations of 722 primitives.
All done?
The PQCRYPTO consortium

libpqcrypto.org
The PQCRYPTO portfolio

PQCRYPTO consortium, with many collaborators: 22 submissions to NIST. 1 damaged later, 0 broken.
The PQCRYPTO portfolio

PQCRYPTO consortium, with many collaborators: 22 submissions to NIST. 1 damaged later, 0 broken. 2 of the 22: gigabyte RSA encryption + signatures; submitted as baseline, not part of portfolio.
The PQCRYPTO portfolio

PQCRYPTO consortium, with many collaborators: 22 submissions to NIST. 1 damaged later, 0 broken.

2 of the 22: gigabyte RSA encryption + signatures; submitted as baseline, *not* part of portfolio.

47 non-PQCRYPTO submissions. 7 damaged, 13 broken. Most attacks by PQCRYPTO + collabs.

libpqcrypto.org

Daniel J. Bernstein
The PQCRYPTO portfolio

PQCRYPTO consortium, with many collaborators: 22 submissions to NIST. 1 damaged later, 0 broken. 2 of the 22: gigabyte RSA encryption + signatures; submitted as baseline, \textit{not} part of portfolio.

47 non-PQCRYPTO submissions. 7 damaged, 13 broken. Most attacks by PQCRYPTO + collabs.

Some broken systems in traditional PQ categories:
- Compact LWE, lattice-based encryption scheme.
- Edon-K, code-based encryption scheme.
- Giophantus, multivariate signature scheme.

Need detailed security analysis, not buzzwords.

libpqcrypto.org

Daniel J. Bernstein
50 signature systems in libpqcrypto

crypto_sign_dilithium{2,3,4}
crypto_sign_gui{184,312,448}
crypto_sign_luov{863256,890351,8117404,4849242,6468330,8086399}
crypto_sign_mqdss{48,64}
crypto_sign_picnicl{1,3,5}{fs,ur}
crypto_sign_qtesla{128,192,256}
crypto_sign_rainbow{1a,1b,1c,3b,3c,4a,5c,6a,6b}
crypto_sign_sphincs{f,s}{128,192,256}{haraka,sha256,shake256}
27 encryption systems in libpqcrypto

crypto_kem_bigquake{1,3,5}
crypto_kem_mceliece{6960119,8192128}
crypto_kem_kyber{512,768,1024}
crypto_kem_dags{3,5}
crypto_kem_frodokem{640,976}
crypto_kem_kindi{256342,256522,512222,512241,512321}
crypto_kem_newhope{512,1024}cca
crypto_kem_ntruhrss701
crypto_kem_{ntrulpr,sntrup}4591761
crypto_kem_ramstakers{216091,756839}
crypto_kem_{lightsaber,saber,firesaber}
NIST submissions vs. libpqcrypto

Each NIST submission includes software:
- a reference C implementation;
- in many cases, also fast implementations.
NIST submissions vs. libpqcrypto

Each NIST submission includes software:

- a reference C implementation;
- in many cases, also fast implementations.

libpqcrypto integrates this software with:

- a unified compilation framework;
NIST submissions vs. *libpqcrypto*

Each NIST submission includes software:
- a reference C implementation;
- in many cases, also fast implementations.

*libpqcrypto* integrates this software with
- a unified compilation framework;
- an automatic test framework;

[libpqcrypto.org](http://libpqcrypto.org)
Daniel J. Bernstein
NIST submissions vs. libpqcrypto

Each NIST submission includes software:

- a reference C implementation;
- in many cases, also fast implementations.

libpqcrypto integrates this software with:

- a unified compilation framework;
- an automatic test framework;
- automatic selection of fastest implementations;
- a unified C interface, modern API;
- a unified Python interface;
- command-line sig/verif/enc/dec tools;
- command-line benchmarking tools.

libpqcrypto.org

Daniel J. Bernstein
NIST submissions vs. libpqcrypto

Each NIST submission includes software:

- a reference C implementation;
- in many cases, also fast implementations.

libpqcrypto integrates this software with

- a unified compilation framework;
- an automatic test framework;
- automatic selection of fastest implementations;
- a unified C interface, modern API;
Each NIST submission includes software:
- a reference C implementation;
- in many cases, also fast implementations.

libpqcrypto integrates this software with
- a unified compilation framework;
- an automatic test framework;
- automatic selection of fastest implementations;
- a unified C interface, modern API;
- a unified Python interface;
NIST submissions vs. libpqcrypto

Each NIST submission includes software:
> a reference C implementation;
> in many cases, also fast implementations.

libpqcrypto integrates this software with
> a unified compilation framework;
> an automatic test framework;
> automatic selection of fastest implementations;
> a unified C interface, modern API;
> a unified Python interface;
> command-line sig/verif/enc/dec tools;

libpqcrypto.org
Daniel J. Bernstein
NIST submissions vs. libpqcrypto

Each NIST submission includes software:

- a reference C implementation;
- in many cases, also fast implementations.

libpqcrypto integrates this software with

- a unified compilation framework;
- an automatic test framework;
- automatic selection of fastest implementations;
- a unified C interface, modern API;
- a unified Python interface;
- command-line sig/verif/enc/dec tools;
- command-line benchmarking tools.

libpqcrypto.org

Daniel J. Bernstein
C interface

unsigned char pk[pqcrypto_sign_gui184_PUBLICKEYBYTES];
unsigned char sk[pqcrypto_sign_gui184_SECRETKEYBYTES];
#define mlen 7
unsigned char m[mlen] = "hello\n";
unsigned char sm[pqcrypto_sign_gui184_BYTES + mlen];
unsigned long long smlen;
unsigned char t[sizeof sm];
unsigned long long tlen;

int main()
{
    if (pqcrypto_sign_gui184_keypair(pk,sk)) abort();
    if (pqcrypto_sign_gui184(sm,&smlen,m,mlen,sk)) abort();
    if (pqcrypto_sign_gui184_open(t,&tlen,sm,smlen,pk)) abort();
    if (tlen != mlen) abort();
    if (memcmp(t,m,mlen)) abort();
    return 0;
}
Python interface

Generate key pair:

\[
\text{pk, sk} = \text{pqcrypto.sign.gui184.keypair()}
\]

Sign message \( \text{m} \):

\[
\text{sm} = \text{pqcrypto.sign.gui184.sign(}\text{m, sk})
\]

Recover message from signed message:

\[
\text{m} = \text{pqcrypto.sign.gui184.open(}\text{sm, pk})
\]

If verification fails: exception and no output.
A larger Python example

Test script to sign and recover a message under a random key pair:

```python
import pqcrypto
sig = pqcrypto.sign.gui184
pk,sk = sig.keypair()
m = b"hello world"
sm = sig.sign(m,sk)
assert m == sig.open(sm,pk)
```

libpqcrypto.org

Daniel J. Bernstein
Command-line signature interface

Generate key pair:

```
pq-keypair-gui184 5>publickey 9>secretkey
```

(Shell uses numbers to identify multiple outputs. Also makes tool easy to use from other languages.)

Sign message:

```
pq-sign-gui184 <message 8<secretkey >sm
```

Recover message from signed message:

```
pq-open-gui184 <sm 4<publickey >message
```
Benchmarking one system

$ pq-size-gui184
gui184 size
  publickey 422122
  secretkey 14985
  signature 45
$ pq-speed-gui184
gui184 speed
  keypair 375801649 378277969 389764325
  sign 13406823 18715903 40190324
  open 141531 141698 142025
$ pq-notes-gui184
gui184 implementation crypto_sign/gui184/pclmulqdq
gui184 version -
gui184 compiler gcc -fPIC -Wall -march=native
  -mtune=native -O3 -fomit-frame-pointer -fwrapv
Benchmarking all systems

$ pq-size-all

dilithium2 size publickey 1184 secretkey 2800 signature 2044
dilithium3 size publickey 1472 secretkey 3504 signature 2701
dilithium4 size publickey 1760 secretkey 3856 signature 3366
gui184 size publickey 422122 secretkey 14985 signature 45
gui312 size publickey 1990045 secretkey 41755 signature 63
gui448 size publickey 5903405 secretkey 94757 signature 83
luov4849242 size publickey 7536 secretkey 32 signature 1746
luov6468330 size publickey 19973 secretkey 32 signature 3184
luov8086399 size publickey 40248 secretkey 32 signature 4850
luov8117404 size publickey 100989 secretkey 32 signature 521
luov863256 size publickey 15908 secretkey 32 signature 319
luov890351 size publickey 46101 secretkey 32 signature 441
mqdss48 size publickey 62 secretkey 32 signature 32882
mqdss64 size publickey 88 secretkey 48 signature 67800
picnic11fs size publickey 33 secretkey 49 signature 34004
## Benchmarking all systems

<table>
<thead>
<tr>
<th>System</th>
<th>Size</th>
<th>Public Key</th>
<th>Secret Key</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>picnic1ur</td>
<td>33</td>
<td>49</td>
<td>97</td>
<td>209</td>
</tr>
<tr>
<td>picnic13fs</td>
<td>49</td>
<td>73</td>
<td>121</td>
<td>6176</td>
</tr>
<tr>
<td>picnic13ur</td>
<td>49</td>
<td>73</td>
<td>121</td>
<td>6176</td>
</tr>
<tr>
<td>picnic15fs</td>
<td>65</td>
<td>97</td>
<td>132</td>
<td>6176</td>
</tr>
<tr>
<td>picnic15ur</td>
<td>65</td>
<td>97</td>
<td>132</td>
<td>6176</td>
</tr>
<tr>
<td>qtesla128</td>
<td>4128</td>
<td>2112</td>
<td>3104</td>
<td>6176</td>
</tr>
<tr>
<td>qtesla192</td>
<td>8224</td>
<td>8256</td>
<td>6176</td>
<td>6176</td>
</tr>
<tr>
<td>qtesla256</td>
<td>8224</td>
<td>8256</td>
<td>6176</td>
<td>6176</td>
</tr>
<tr>
<td>rainbow1a</td>
<td>152097</td>
<td>100209</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>rainbow1b</td>
<td>163185</td>
<td>114308</td>
<td>78</td>
<td>78</td>
</tr>
<tr>
<td>rainbow1c</td>
<td>192241</td>
<td>143385</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>rainbow3b</td>
<td>564535</td>
<td>409463</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>rainbow3c</td>
<td>720793</td>
<td>537781</td>
<td>156</td>
<td>156</td>
</tr>
<tr>
<td>rainbow4a</td>
<td>565489</td>
<td>376141</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>rainbow5c</td>
<td>1723681</td>
<td>1274317</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>rainbow6a</td>
<td>1351361</td>
<td>892079</td>
<td>118</td>
<td>118</td>
</tr>
</tbody>
</table>
Benchmarking all systems

rainbow6b size publickey 1456225 secretkey 1016868 signature 147
sphincsf128haraka size publickey 32 secretkey 64 signature 16976
sphincsf128sha256 size publickey 32 secretkey 64 signature 16976
sphincsf128shake256 size publickey 32 secretkey 64 signature 16976
sphincsf192haraka size publickey 48 secretkey 96 signature 35664
sphincsf192sha256 size publickey 48 secretkey 96 signature 35664
sphincsf192shake256 size publickey 48 secretkey 96 signature 35664
sphincsf256haraka size publickey 64 secretkey 128 signature 49216
sphincsf256sha256 size publickey 64 secretkey 128 signature 49216
sphincsf256shake256 size publickey 64 secretkey 128 signature 49216
sphincss128haraka size publickey 32 secretkey 64 signature 8080
sphincss128sha256 size publickey 32 secretkey 64 signature 8080
sphincss128shake256 size publickey 32 secretkey 64 signature 8080
sphincss192haraka size publickey 48 secretkey 96 signature 17064
sphincss192sha256 size publickey 48 secretkey 96 signature 17064
sphincss192shake256 size publickey 48 secretkey 96 signature 17064
Benchmarking all systems

sphincss256haraka size 64 publickey 128 secretkey 128 signature 29792
sphincss256sha256 size 64 publickey 128 secretkey 128 signature 29792
sphincss256shake256 size 64 publickey 128 secretkey 128 signature 29792
bigquake1 size 25482 secretkey 14772 ciphertext 201 sessionkey 32
bigquake3 size 84132 secretkey 30860 ciphertext 406 sessionkey 32
bigquake5 size 149800 secretkey 41804 ciphertext 492 sessionkey 32
dags3 size 11616 secretkey 2973704 ciphertext 2144 sessionkey 64
dags5 size 11616 secretkey 2973704 ciphertext 2144 sessionkey 64
firesaber size 1312 secretkey 3040 ciphertext 1472 sessionkey 32
frodokem640 size 9616 secretkey 19872 ciphertext 9736
frodokem976 size 15632 secretkey 31272 ciphertext 1576
kindi256342 size 1184 secretkey 1472 ciphertext 1824
kindi256522 size 1984 secretkey 2304 ciphertext 2752
kindi512222 size 1456 secretkey 1712 ciphertext 2544
kindi512241 size 1728 secretkey 2112 ciphertext 2752
kindi512321 size 2368 secretkey 2752 ciphertext 3392

libpqcrypto.org Daniel J. Bernstein
Benchmarking all systems

kyber1024 size: publickey 1440, secretkey 3168, ciphertext 1504
kyber512 size: publickey 736, secretkey 1632, ciphertext 800
kyber768 size: publickey 1088, secretkey 2400, ciphertext 1152
lightsaber size: publickey 672, secretkey 1568, ciphertext 736
mceliece6960119 size: publickey 1047319, secretkey 13908, ciphertext 226
mceliece8192128 size: publickey 1357824, secretkey 14080, ciphertext 240
newhope1024cca size: publickey 1824, secretkey 3680, ciphertext 220
newhope512cca size: publickey 928, secretkey 1888, ciphertext 1120
ntruhrss701 size: publickey 1138, secretkey 1418, ciphertext 1278
sntrulpr4591761 size: publickey 1047, secretkey 1238, ciphertext 117
ramstakers216091 size: publickey 27044, secretkey 54056
ramstakers756839 size: publickey 94637, secretkey 189242
saber size: publickey 992, secretkey 2304, ciphertext 1088
sntrup4591761 size: publickey 1218, secretkey 1600, ciphertext 1047

libpqcrypto.org

Daniel J. Bernstein
Signature size \((y)\) vs. public-key size \((x)\)
The future

Various libpqcrypto goals and ongoing work:

- Following constant-time rules.
  Already done for some implementations.
The future

Various libpqcrypto goals and ongoing work:

- Following constant-time rules.
  Already done for some implementations.
- More tests, audits. Everything already passes Valgrind and ASan except NTRU-HRSS-KEM.
The future

Various libpqcrypto goals and ongoing work:

- Following constant-time rules. Already done for some implementations.
- More tests, audits. Everything already passes Valgrind and ASan except NTRU-HRSS-KEM.
- Formal verification.
The future

Various libpqcrypto goals and ongoing work:

- Following constant-time rules. Already done for some implementations.
- More tests, audits. Everything already passes Valgrind and ASan except NTRU-HRSS-KEM.
- Formal verification.
- Faster installation.
- Less CPU time. Already many speedups.

libpqcrypto.org
Daniel J. Bernstein
The future

Various libpqcrypto goals and ongoing work:

- Following constant-time rules.
  Already done for some implementations.
- More tests, audits. Everything already passes Valgrind and ASan except NTRU-HRSS-KEM.
- Formal verification.
- Faster installation.
- Less CPU time. Already many speedups.
- Reducing code volume: e.g., SHA-3 merge.
The future

Various libpqcrypto goals and ongoing work:

- Following constant-time rules. Already done for some implementations.
- More tests, audits. Everything already passes Valgrind and ASan except NTRU-HRSS-KEM.
- Formal verification.
- Faster installation.
- Less CPU time. Already many speedups.
- Reducing code volume: e.g., SHA-3 merge.
- Long term: Reduce number of primitives.
Some links

https://libpqcrypto.org: libpqcrypto
https://pqcrypto.eu.org: PQCRYPTO

https://github.com/mupq/pqhw: PQCRYPTO’s FPGA implementations of NewHope-1024, BLISS