Timing and Side-Channel Countermeasures for Quantum-Safe Cryptography

William Whyte, CTO, Onboard Security

OnBoardSecurity

Overview

- Combination of survey and original research
  - Natural places to be concerned about leakage
  - High-level notes about natural countermeasures
  - Goal is to provide implementers with information about what they need to research before implementation
- Won't go into "Why quantum-safe crypto"?
- What flavors of quantum safe crypto?
  - Code-based, lattice-based, MVQ, SIDH, Hash based signatures
  - We'll focus on lattice-based

LWE (Regev 2005; Ding 2012; Bos, Costello, Ducas, Mironov, Naehrig, Nikolaenko, Raghunathan, Stebila 2016)

ALICE

1	0	0	-1	1
1	0	-1	-1	-1
1	1	-1	0	0
0	-1	-1	-1	0
1	0	0	0	-1

*

38	87	99	8	14
119	2	115	12	117
58	32	58	30	18
45	64	111	44	58
28	105	103	90	62

+

1	-1	-1	0	1
0	1	-1	0	1
-1	1	1	-1	1
1	0	-1	1	-1
-1	1	0	0	1

=

22	0	90	54	19
34	14	80	98	4
98	58	30	116	114
33	29	96	42	60
9	110	123	45	80

BOB

38	87	99	8	14
119	2	115	12	117
58	32	58	30	18
45	64	111	44	58
28	105	103	90	62

*

1	0	-1	1	1
1	0	1	-1	1
0	1	1	-1	0
-1	-1	-1	-1	-1
1	1	-1	1	0

+

-1	1	1	0	1
0	0	0	1	-1
0	1	1	-1	-1
1	0	-1	1	1
1	0	0	1	1

=

3	106	0	112	118
99	93	123	108	108
78	47	112	82	59
124	125	27	12	66
106	75	28	47	44

ALICE

1	0	0	-1	1
1	0	-1	-1	-1
1	1	-1	0	0
0	-1	-1	-1	0
1	0	0	0	-1

*

3	106	0	112	118
99	93	123	108	108
78	47	112	82	59
124	125	27	12	66
106	75	28	47	44

=

112	56	1	20	96
76	113	87	98	76
24	25	11	11	40
80	116	119	52	21
24	31	99	65	74

BOB

22	0	90	54	19
34	14	80	98	4
98	58	30	116	114
33	29	96	42	60
9	110	123	45	80

*

1	0	-1	1	1
1	0	1	-1	1
0	1	1	-1	0
-1	-1	-1	-1	-1
1	1	-1	1	0

=

114	55	122	24	95
81	113	85	100	77
27	28	14	8	40
80	114	117	53	20
27	31	99	65	74

LWE

- Alice -> Bob: $E_A = A_A * H + B_A$
- Bob -> Alice: $E_B = H * A_B + B_B$
- Alice: $A_A * E_B = \underline{\mathbf{A_A * H * A_B}} + A_A * B_B$
- Bob: $E_A * A_B = \underline{\mathbf{A_A * H * A_B}} + B_A * A_B$
- Public value size : $N^2 \log_2 q$
- Naïve Multiplications: N^2
- N for 128-bit post-quantum security: 500-1000

R-LWE (Lyubashevsky, Peikert, Regev 2012; Peikert 2014; Alkim, Ducas, Pöppelmann, Schwabe 2016)

ALICE

1	0	0	-1	1
1	1	0	0	-1
-1	1	1	0	0
0	-1	1	1	0
0	0	-1	1	1

*

38	87	99	8	14
14	38	87	99	8
8	14	38	87	99
99	8	14	38	87
87	99	8	14	38

+

1	-1	-1	0	1
1	1	-1	-1	0
0	1	1	-1	-1
-1	0	1	1	-1
-1	-1	0	1	1

=

27	50	92	111	93
93	27	50	92	111
111	93	27	50	92
92	111	93	27	50
50	92	111	93	27

BOB

38	87	99	8	14
14	38	87	99	8
8	14	38	87	99
99	8	14	38	87
87	99	8	14	38

*

1	0	-1	1	1
1	1	0	-1	1
1	1	1	0	-1
-1	1	1	1	0
0	-1	1	1	1

+

-1	1	1	0	1
1	-1	1	1	0
0	1	-1	1	1
1	0	1	-1	1
1	1	0	1	-1

=

88	54	84	100	41
41	88	54	84	100
100	41	88	54	84
84	100	41	88	54
54	84	100	41	88

ALICE

1	0	0	-1	1
1	1	0	0	-1
-1	1	1	0	0
0	-1	1	1	0
0	0	-1	1	1

*

88	54	84	100	41
41	88	54	84	100
100	41	88	54	84
84	100	41	88	54
54	84	100	41	88

=

58	38	16	53	75
75	58	38	16	53
53	75	58	38	16
16	53	75	58	38
38	16	53	75	58

BOB

27	50	92	111	93
93	27	50	92	111
111	93	27	50	92
92	111	93	27	50
50	92	111	93	27

*

1	0	-1	1	1
1	1	0	-1	1
1	1	1	0	-1
-1	1	1	1	0
0	-1	1	1	1

=

58	33	15	54	78
78	58	33	15	54
54	78	58	33	15
15	54	78	58	33
33	15	54	78	58

R-LWE

- Alice -> Bob: $E_A = A_A * H + B_A$
- Bob -> Alice: $E_B = H * A_B + B_B$
- Alice: $A_A * E_B = \underline{\mathbf{A_A * H * A_B}} + A_A * B_B$
- Bob: $E_A * A_B = \underline{\mathbf{A_A * H * A_B}} + B_A * A_B$

- Keysize : $\underline{\mathbf{N}} \log_2 q$
- Naïve Multiplications: N^2
- N for 128-bit post-quantum security: 500-1000

NTRU (Hoffstein, Pipher, Silverman 1998)

BOB

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

*

r

-7	0	0	-7	0
0	-7	0	0	-7
-7	0	-7	0	0
0	-7	0	-7	0
0	0	-7	0	-7

+

m

0	-1	1	-1	1
1	0	-1	1	-1
-1	1	0	-1	1
1	-1	1	0	-1
-1	1	-1	1	0

=

e

32	117	69	99	78
78	32	117	69	99
99	78	32	117	69
69	99	78	32	117
117	69	99	78	32

NTRU

ALICE

0	-1	1	0	1
1	0	-1	1	0
0	1	0	-1	1
1	0	1	0	-1
-1	1	0	1	0

*

104	12	58	35	46
46	104	12	58	35
35	46	104	12	58
58	35	46	104	12
12	58	35	46	104

+

=

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

ALICE

104	12	58	35	46
46	104	12	58	35
35	46	104	12	58
58	35	46	104	12
12	58	35	46	104

*

g

-1	1	-1	1	-1
-1	-1	1	-1	1
1	-1	-1	1	-1
-1	1	-1	-1	1
1	-1	1	-1	-1

+

=

h

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

ALICE

f

0	-1	1	0	1
1	0	-1	1	0
0	1	0	-1	1
1	0	1	0	-1
-1	1	0	1	0

*

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

+

=

126	1	126	1	126
126	126	1	126	1
1	126	126	1	126
126	1	126	126	1
1	126	1	126	126

BOB

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

*

r

-7	0	0	-7	0
0	-7	0	0	-7
-7	0	-7	0	0
0	-7	0	-7	0
0	0	-7	0	-7

+

m

0	-1	1	-1	1
1	0	-1	1	-1
-1	1	0	-1	1
1	-1	1	0	-1
-1	1	-1	1	0

=

e

32	117	69	99	78
78	32	117	69	99
99	78	32	117	69
69	99	78	32	117
117	69	99	78	32

NTRU

ALICE

0	-1	1	0	1
1	0	-1	1	0
0	1	0	-1	1
1	0	1	0	-1
-1	1	0	1	0

*

104	12	58	35	46
46	104	12	58	35
35	46	104	12	58
58	35	46	104	12
12	58	35	46	104

+

=

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

ALICE

104	12	58	35	46
46	104	12	58	35
35	46	104	12	58
58	35	46	104	12
12	58	35	46	104

*

g

-1	1	-1	1	-1
-1	-1	1	-1	1
1	-1	-1	1	-1
-1	1	-1	-1	1
1	-1	1	-1	-1

+

=

h

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

ALICE

f

0	-1	1	0	1
1	0	-1	1	0
0	1	0	-1	1
1	0	1	0	-1
-1	1	0	1	0

*

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

+

=

126	1	126	1	126
126	126	1	126	1
1	126	126	1	126
126	1	126	126	1
1	126	1	126	126

BOB

80	23	115	69	93
93	80	23	115	69
69	93	80	23	115
115	69	93	80	23
23	115	69	93	80

*

r

-7	0	0	-7	0
0	-7	0	0	-7
-7	0	-7	0	0
0	-7	0	-7	0
0	0	-7	0	-7

+

m

0	-1	1	-1	1
1	0	-1	1	-1
-1	1	0	-1	1
1	-1	1	0	-1
-1	1	-1	1	0

=

e

32	117	69	99	78
78	32	117	69	99
99	78	32	117	69
69	99	78	32	117
117	69	99	78	32

ALICE

0	-1	1	0	1
1	0	-1	1	0
0	1	0	-1	1
1	0	1	0	-1
-1	1	0	1	0

*

32	117	69	99	78
78	32	117	69	99
99	78	32	117	69
69	99	78	32	117
117	69	99	78	32

=

r*g+f*m

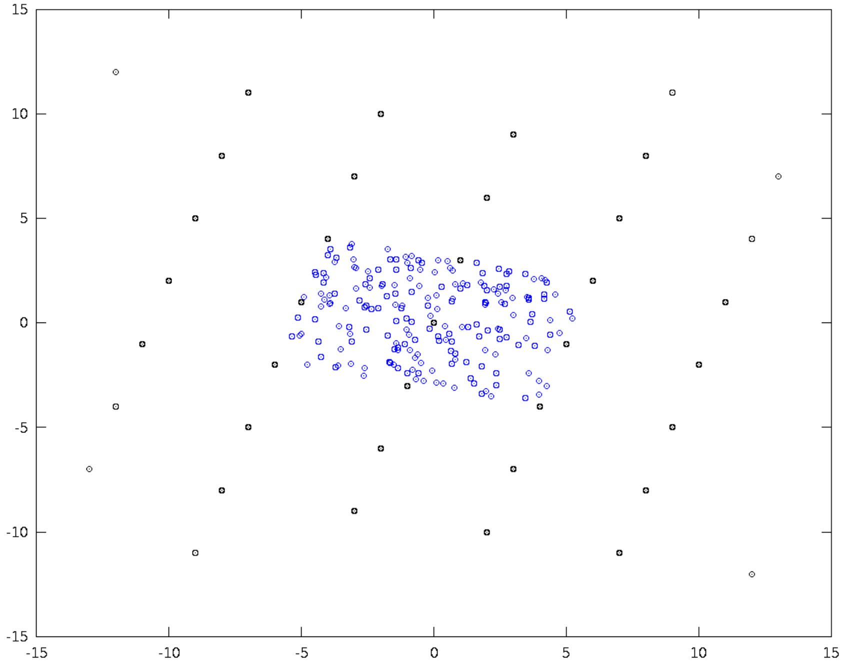
11	-12	14	-1	2
2	11	-12	14	-1
-1	2	11	-12	14
14	-1	2	11	-12
-12	14	-1	2	11

NTRU

- Alice -> Bob: h s.t. f * h = g
- Bob -> Alice: e = h * pr + m
- Alice: f * e = p*r*g + f*m
- = f*m mod p
- Keysize : **N** $\log_2 q$
- Naïve Multiplications: N^2
  - Index-based: N*d **adds**
- N for 128-bit post-quantum security: 400-1000

Signatures: transcript security

- In some "old" lattice-based signature schemes, each transcript of a signature-document pair reveals some information about the secret key;
- Collecting enough of those transcripts allows the attacker to learn a good basis of the lattice.
- Modern lattice-based signature schemes, following Lyubashevsky's insight, selectively reject otherwise valid signatures on signing to ensure the signature points are drawn from a known distribution.
- Use randomness to **select** a starting point for signing
- **Reject** with some probability based on the properties of the candidate siganture



Blue dots: transcripts

Black dots: lattice points

Enough blue dots gives a good approximation of the fundamental parallelepiped

BLISS (simplified) (Lyubashevsky 2009)

- Rejection sampling
  - Ensure that “transcript” of signatures doesn’t leak information
- BLISS:
  - Bimodal Gaussian sampling
    - Significantly reduces rejection probability
  - … plus other optimizations
- Note that revealing yi reveals si

- Private key $\mathbf{s}_1, \mathbf{s}_2$
- Public key $\mathbf{a}_1, \mathbf{a}_2, \mathbf{h} = a_1\mathbf{s}_1 + a_2\mathbf{s}_2$
- Sign:
  - **Select** $\mathbf{y}_1, \mathbf{y}_2$
  - Calculate $\mathbf{f} = H(\mathbf{a}_1\mathbf{y}_1 + \mathbf{a}_2\mathbf{y}_2, m)$
  - $(\mathbf{z}_1, \mathbf{z}_2) = (\mathbf{f}\mathbf{s}_1 + \mathbf{y}_1, \mathbf{f}\mathbf{s}_2 + \mathbf{y}_2)$
  - **Restart** if $(\mathbf{z}_1, \mathbf{z}_2)$ not consistent with specified distribution
  - Send $(\mathbf{f}, \mathbf{z}_1, \mathbf{z}_2)$
- Verify:
  - Check that $\mathbf{z}_1, \mathbf{z}_2$ are small and $\mathbf{f} = H(\mathbf{a}_1\mathbf{z}_1 + \mathbf{a}_2\mathbf{z}_2 - h\mathbf{f}, m)$

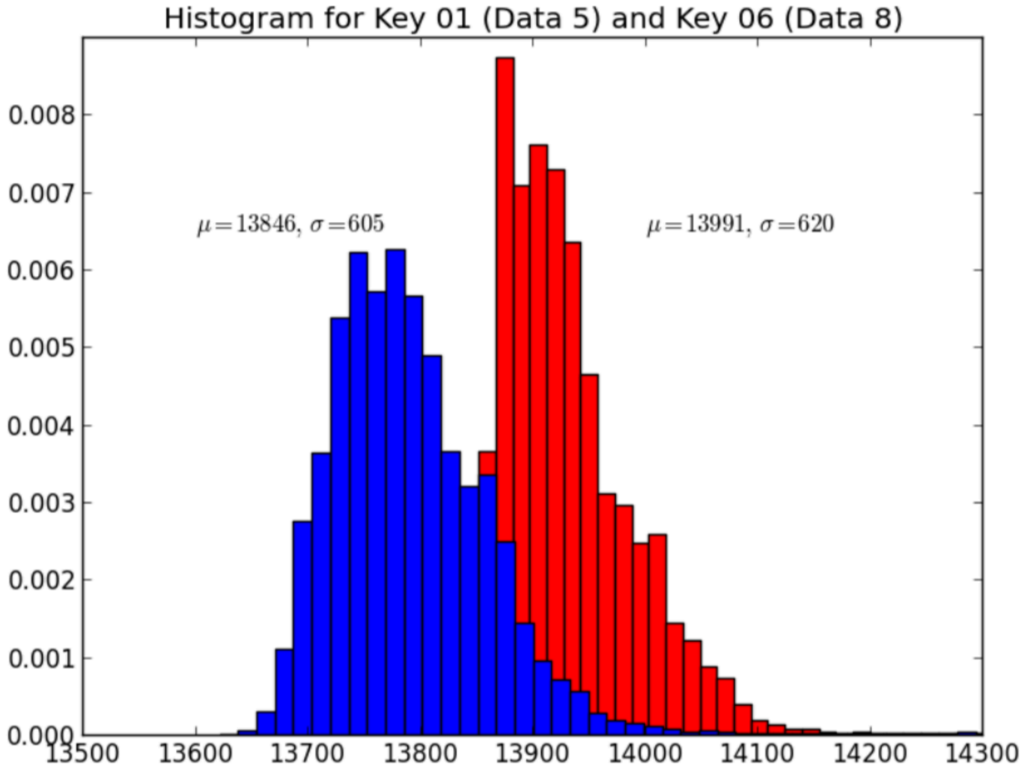
pqNTRUSign (Hoffstein et al 2014)

- Rejection sampling
  - Ensure that "transcript" of signatures doesn't leak information
- Sampling:
  - **Uniform** starting point within set of lattice points with all coefficients a in range (–q/2, q/2]
  - Reject any signatures whose coefficients lie outside range (–(q-B)/2, (q-B)/2]
  - Ongoing research into other starting distributions
- Revealing($\mathbf{s}_0$, $\mathbf{t}_0$) doesn't directly reveal (**f**, **g**) but does allow a reasonably efficient transcript attack

- Private key **f**, **g**
- Public key $\mathbf{h} = \mathbf{g} * \mathbf{f}^{-1}$
- Sign:
  - Calculate Hash (m) -> ($\mathbf{s}_p$, $\mathbf{t}_p$)
  - **Select** lattice point ($\mathbf{s}_0$, $\mathbf{t}_0$)
  - Use (**f**, **g**) to find nearby (**s**, **t**) such that (**s**, **t**) = ($\mathbf{s}_p$, $\mathbf{t}_p$) mod p
  - **Restart** if (**s**, **t**) not consistent with specified distribution
  - Send (**s**)
- Verify:
  - Calculate t = s*h mod q
  - Calculate Hash (m) -> ($\mathbf{s}_p$, $\mathbf{t}_p$)
  - Check that $\mathbf{z}_1$, $\mathbf{z}_2$ are small and $\mathbf{f} = H(\mathbf{a}_1\mathbf{z}_1 + \mathbf{a}_2\mathbf{z}_2 - h\mathbf{f}, m)$

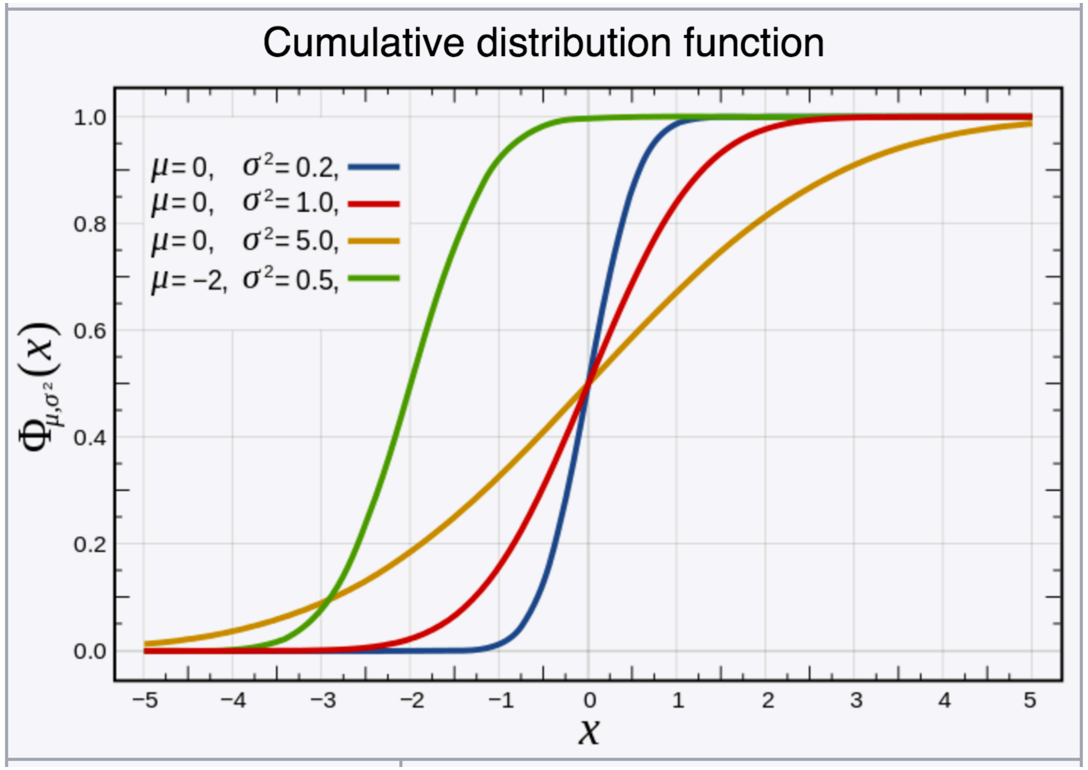
Natural sources of side channel leakage

- Multiplication
  - Coefficient-based
  - Index-based
  - ***In all algorithms, there is a multiplication that depends on private information***
    - ==> timing variation could leak private information
- Sampling
  - Uniform
  - Fixed-weight
  - Gaussian
  - ***In some cases, sampling depends on private information***
    - ==> timing variation could leak private information in these cases
- Rejection
- Fault attacks



Sampling

- Uniform binary
- Uniform mod p (prime) or mod q (composite)
- Fixed-weight
  - Some parameterized number of 1s / -1s
- Gaussian
  - Draw from a Gaussian distribution
  - Can be implemented using floating-point arithmetic or lookup tables
    - Floating-point: issues with accuracy
    - Lookup: issues with size



Examples

- RSA Signing Fault Attack
- Timing attack on NTRUEncrypt via Variation in Number of Hash Calls
- Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme
- Loop-Abort Faults on Lattice-Based Fiat–Shamir and Hash-and-Sign Signatures
- Generalized Howgrave-Graham–Szydlo and Side-Channel Attacks Against BLISS

- Cause fault at time of signing
- Use CRT to recover private key

Examples

- RSA Signing Fault Attack
- Timing attack on NTRUEncrypt via Variation in Number of Hash Calls
- Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme
- Loop-Abort Faults on Lattice-Based Fiat–Shamir and Hash-and-Sign Signatures
- Generalized Howgrave-Graham–Szydlo and Side-Channel Attacks Against BLISS

- CCA2 scheme for NTRUEncrypt
  - Generates r from (plaintext + salt)
- Attack uses timing information leaked by method to get fixed-weight r
- Create ciphertexts that will decrypt to extremely sparse candidate plaintexts and precalculate time to obtain r for the plaintexts
- Match times to determine which candidate plaintexts were obtained
- Recover private key!

Examples

- RSA Signing Fault Attack
- Timing attack on NTRUEncrypt via Variation in Number of Hash Calls
- Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme
- Loop-Abort Faults on Lattice-Based Fiat–Shamir and Hash-and-Sign Signatures
- Generalized Howgrave-Graham–Szydlo and Side-Channel Attacks Against BLISS

- Bruinderink, Hulsing, Lange, Yarom 2016
- Applies when Gaussian sampling is implemented by table look-up
  - Large tables are known to be vulnerable to cache misses – see similar attacks on AES
- Recovers private key with 90% success
- "These attacks require significant power over device... more research is needed to get security for implementations."

Examples

- RSA Signing Fault Attack
- Timing attack on NTRUEncrypt via Variation in Number of Hash Calls
- Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme
- Loop-Abort Faults on Lattice-Based Fiat–Shamir and Hash-and-Sign Signatures
- Generalized Howgrave-Graham–Szydlo and Side-Channel Attacks Against BLISS

- Espiteau, Fouque, Girard, Tibouchi, 2016
- "We present several fault attacks against those schemes yielding a full key recovery with only a few or even a single faulty signature"
- Strong attack model – requires attacker to have control over PRNG output

Examples

- RSA Signing Fault Attack
- Timing attack on NTRUEncrypt via Variation in Number of Hash Calls
- Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme
- Loop-Abort Faults on Lattice-Based Fiat–Shamir and Hash-and-Sign Signatures
- Generalized Howgrave-Graham–Szydlo and Side-Channel Attacks Against BLISS

- Espiteau, Fouque, Girard, Tibouchi, 2017

Our contributions. Our goal is to look at the security of embedded implementations of BLISS (particularly the 8-bit AVR microcontroller implementation of Pöppelmann et al. [POG15]) against side-channel attacks. Our main target is the clever algorithm proposed in the original BLISS paper [DDLL13] to perform the rejection sampling, which is intervenes in a crucial way in those embedded implementations. To achieve the correct output distribution, the signature generation algorithm has to be restarted with probability:

$$1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2} \right) \cosh\left(\frac{\langle \mathbf{z}, \mathbf{Sc} \rangle}{\sigma^2} \right) \right),$$

where $(\mathbf{z}, \mathbf{c})$ is the signature generated so far, $\mathbf{S}$ the secret key, σ the Gaussian standard deviation and M a scaling factor ensuring that this probability is always at most 1.

It turns out that the clever algorithm for rejection sampling, based on iterated Bernoulli trials, traverses the bits of the two values $\langle \mathbf{z}, \mathbf{Sc} \rangle$ and $K - \|\mathbf{Sc}\|^2$ (where K is defined such that $M = \exp\left(K/(2\sigma^2)\right)$) in much the same way as a square-and-multiply algorithm traverses the bits of its exponent: one can basically read those bits on a power or electromagnetic trace! This makes it possible to mount an SPA/SEMA attack on the rejection sampling using either of these values.

Concretely, for the parameter set BLISS-0 (resp. BLISS-I and above), it takes a few CPU hours (resp. a little over a CPU-month) to recover the exact secret key up to multiplication by a *root of unity*, i.e. the exact key with its coefficients possibly rotated around with appropriate sign flips. This recovered key is functionally equivalent to the original one for the signature algorithm, and we thus achieve full key recovery for the aforementioned "weak" keys.

Countermeasures

Constant time algorithms

- No conditional branches
  - If … then … else …
- No memory access w.r.t. secret data
- All secret data are only used as operands of constant time arithmetic operations
  - r = f h mod q ✓
  - If f =1, r += h ✗

Avoid large/secret/non-constant time look-up tables

- Look up tables can be used to (significantly) reduce on-line computation
  - Small
  - Public
  - Constant time
- They may also leak side channel information
  - Gaussian look up tables vs BLISS

```
+/* maps index = a|b into a%3|b%3 where a and b are 4 bits each */
+uint8_t mod3map[256] = {
+        0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,
+       16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16,
+       32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32,
+        0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,
+       16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16,
+       32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32,
+        0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,
+       16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16,
+       32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32,
+        0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,
+       16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16,
+       32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32,
+        0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,
+       16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16, 17, 18, 16,
+       32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32, 33, 34, 32,
+        0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0,  1,  2,  0};
```

Avoid large/secret/non-constant time look-up tables

- Constant-time Gaussian sampling is an active area of research

Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time

Daniele Micciancio
UCSD
daniele@cs.ucsd.edu

Michael Walter
UCSD
miwalter@eng.ucsd.edu

March 21, 2017

Abstract

Sampling integers with Gaussian distribution is a fundamental problem that arises in almost every application of lattice cryptography, and it can be both time consuming and challenging to implement. Most previous work has focused on the optimization and implementation of integer Gaussian sampling in the context of specific applications, with fixed sets of parameters. We present new algorithms for discrete Gaussian sampling that are both generic (application independent), efficient, and more easily implemented in constant time without incurring a substantial slow-down, making them more resilient to side-channel (e.g., timing) attacks. As an additional contribution, we present new analytical techniques that can be used to simplify the precision/security evaluation of floating point cryptographic algorithms, and an experimental comparison of our algorithms with previous algorithms from the literature.

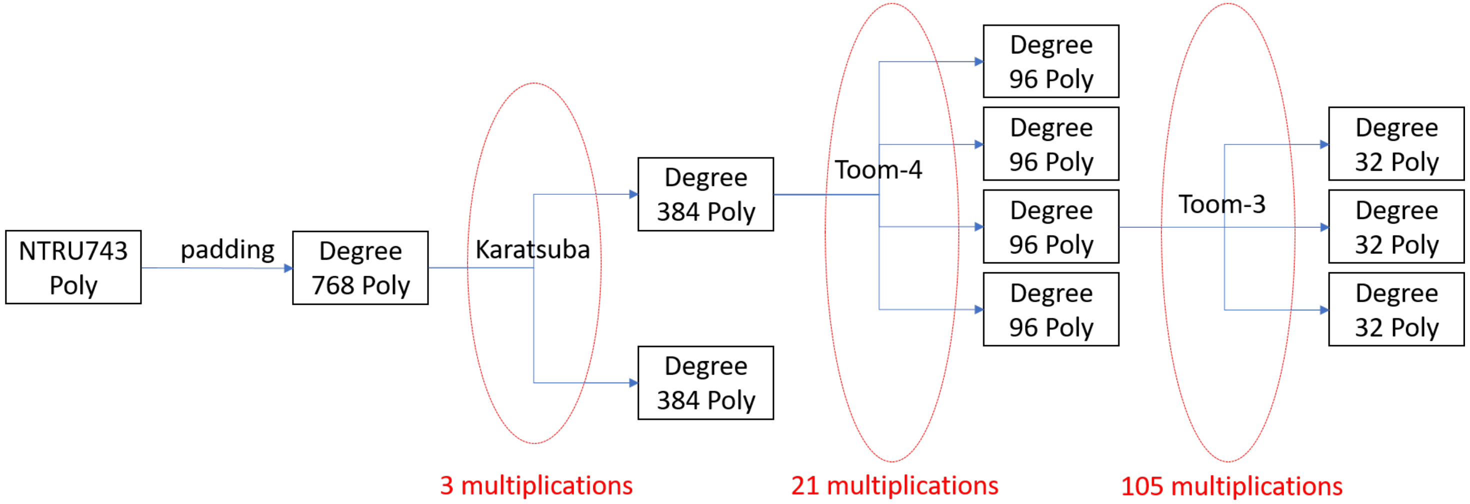
Time-Independent Discrete Gaussian Sampling For Post-Quantum Cryptography

A. Khalid, J. Howe, C. Rafferty, and M. O'Neill

Centre for Secure Information Technologies (CSIT),
Queen's University Belfast, UK.

This research proposes countermeasures against timing information leakage with FPGA-based designs of the CDT-based discrete Gaussian samplers with constant response time, targeting encryption and signature scheme parameters. The proposed designs are compared against the state-of-the-art and are shown to significantly outperform existing implementations. For encryption, the proposed sampler is 9x faster in comparison to the only other existing time-independent CDT sampler design. For signatures, the first time-independent CDT sampler in hardware is proposed.

Tailored hierarchical multiplication



- Constant time
- 105 multiplications (degree 32) vs 243 multiplications (degree 23) using Karatsuba only
- Gain 2.3x if degree 32 mul = degree 23 mul

NIST's FIPS-140 hybrid statement

- http://csrc.nist.gov/groups/ST/post-quantum-crypto/faq.html
- ***Q: The call for proposals briefly mentions hybrid modes that combine quantum-resistant cryptographic algorithms with existing cryptographic algorithms (which may not be quantum-resistant). Can these hybrid modes be FIPS-validated?***
- A: Assuming one of the components of the hybrid mode in question is a NIST-approved cryptographic primitive, such hybrid modes can be approved for use for key establishment or digital signatures.
- Conclusion: Customers may be in a position to request side-channel resistant quantum-safe implementations sooner than you think!

Questions / Discussion