Flush, Gauss, and Reload

A Cache-Attack on the BLISS Lattice-Based Signature Scheme

Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange and Yuval Yarom

August 18th, 2016

Lattice-based Cryptography

- Lattice-based cryptography: promising post-quantum secure alternative.
- Active research on theoretical and practical security.
- But what about security of implementations?

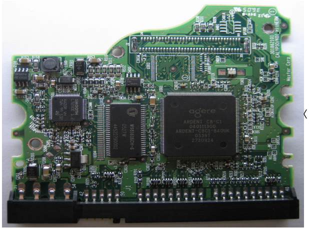
This work

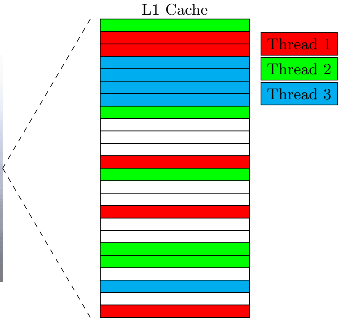
- The first side-channel attack on a lattice-based signature scheme.
- Exploits information leakage from the discrete Gaussian sampler via cache memory.
- Attack target: BLISS, an efficient lattice-based signature scheme.
- BLISS also included in strongSwan (library for IPsec-based VPN).

Cache Timing Attacks

Cache (Timing) Attacks

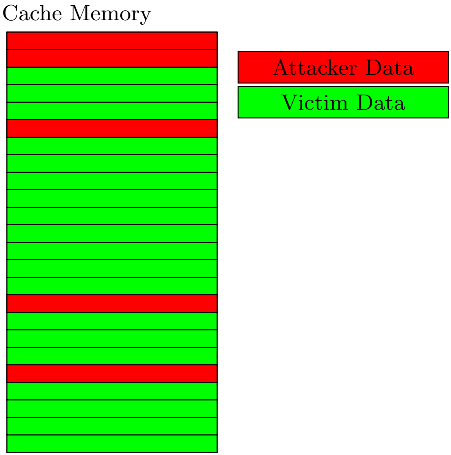
- Cache-memory: small, fast memory shared among all threads.
- Bridge the gap between processor speed and memory speed.
- Data is stored in cache-lines, typically 64 Bytes.





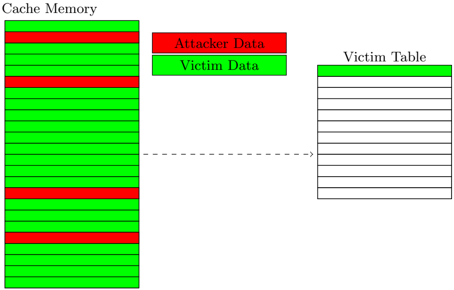
Cache (Timing) Attacks

- Attacker fills specific cache lines with his data.



Cache (Timing) Attacks

- Attacker notices that victim uses some part of cache.
- Learns cache-line of data used by victim.



BLISS

BLISS Lattice-based Signature Scheme

- Bimodal Lattice Signature Scheme (BLISS) (CRYPTO '13 by Ducas, Durmus, Lepoint and Lyubashevsky)
- Implementations available via NTRU lattices (polynomials in $R_q = \mathbb{Z}_q[x]/(x^n+1)$, $n = 2^r$, prime q).
- For $f, g \in R_q = \mathbb{Z}_q[x]/(x^n+1)$:

$$f \cdot g = \mathbf{f}\mathrm{G} = \mathbf{g}\mathrm{F}$$

where $\mathrm{F}, \mathrm{G} \in \mathbb{Z}_q^{n \times n}$, whose columns are rotations of $\mathbf{f}, \mathbf{g}$, with possibly opposite sign:

$$\mathrm{F} = \begin{bmatrix} f_0 & -f_{n-1} & \dots & -f_1 \\ f_1 & f_0 & \dots & -f_2 \\ \dots & \dots & \dots & \dots \\ f_{n-1} & f_{n-2} & \dots & f_0 \end{bmatrix}$$

BLISS Lattice-based Signature Scheme

- Secret key $\mathbf{S} = (f, 2g+1) \in R_q^2$ with f, g sparse and typically entries in $\{\pm 1, 0\}$
- Public key $\mathbf{A} = (a_1, a_2) \in R_q^2$ satisfying:

$$a_1 s_1 + a_2 s_2 \equiv q \bmod 2q$$

- Computed as $a_q = (2g+1)/f \bmod 2q$ (restart if f not invertible) and $\mathbf{A} = (2a_q, q-2)$.
- Attacker can validate correctness for candidate of key f with the public key and compute $2g+1$.
- Both $-\mathbf{S}$ and $\mathbf{S}$ are valid as secret key.

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.
2. Construct vector $\mathbf{u}$, using $\mathbf{y}_1$ and public key $\mathbf{A}$.

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.
2. Construct vector $\mathbf{u}$, using $\mathbf{y}_1$ and public key $\mathbf{A}$.
3. Construct challenge $\mathbf{c} = H(\lfloor \mathbf{u} \rceil \bmod 2q, \mu) \in \{0,1\}^n$ with $||\mathbf{c}||_1 = \kappa$

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.
2. Construct vector $\mathbf{u}$, using $\mathbf{y}_1$ and public key $\mathbf{A}$.
3. Construct challenge $\mathbf{c} = H(\lfloor \mathbf{u} \rceil \bmod 2q, \mu) \in \{0,1\}^n$ with $||\mathbf{c}||_1 = \kappa$
4. Generate a random bit b. Set $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \cdot \mathbf{c} \bmod 2q$

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.
2. Construct vector $\mathbf{u}$, using $\mathbf{y}_1$ and public key $\mathbf{A}$.
3. Construct challenge $\mathbf{c} = H(\lfloor \mathbf{u} \rceil \bmod 2q, \mu) \in \{0,1\}^n$ with $||\mathbf{c}||_1 = \kappa$
4. Generate a random bit b. Set $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \cdot \mathbf{c} \bmod 2q$
5. Return signature $(\mathbf{z}_1, \mathbf{c})$ for μ.

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.
2. Construct vector $\mathbf{u}$, using $\mathbf{y}_1$ and public key $\mathbf{A}$.
3. Construct challenge $\mathbf{c} = H(\lfloor \mathbf{u} \rceil \bmod 2q, \mu) \in \{0,1\}^n$ with $||\mathbf{c}||_1 = \kappa$
4. Generate a random bit b. Set $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \cdot \mathbf{c} \bmod 2q$
5. Return signature $(\mathbf{z}_1, \mathbf{c})$ for μ.

- $\mathbf{s}_1 \cdot \mathbf{c} = \mathbf{s}_1 \mathrm{C}$ over $\mathbb{Z}$ for matrix $\mathrm{C} \in \{-1,0,1\}^{n \times n}$.

BLISS Lattice-based Signature Scheme

- Simplified version of the BLISS signature algorithm for message μ:

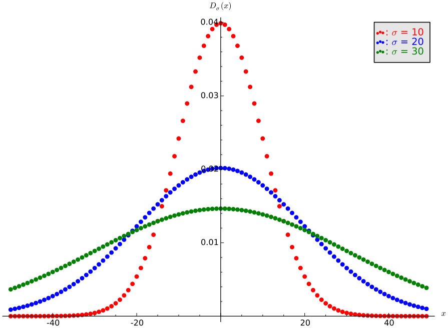
1. Sample $\mathbf{y}_1 \leftarrow D_{\mathbb{Z}^n,\sigma}$.
2. Construct vector $\mathbf{u}$, using $\mathbf{y}_1$ and public key $\mathbf{A}$.
3. Construct challenge $\mathbf{c} = H(\lfloor \mathbf{u} \rceil \bmod 2q, \mu) \in \{0,1\}^n$ with $||\mathbf{c}||_1 = \kappa$
4. Generate a random bit b. Set $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \cdot \mathbf{c} \bmod 2q$
5. Return signature $(\mathbf{z}_1, \mathbf{c})$ for μ.

- $\mathbf{s}_1 \cdot \mathbf{c} = \mathbf{s}_1 \mathrm{C}$ over $\mathbb{Z}$ for matrix $\mathrm{C} \in \{-1,0,1\}^{n \times n}$.
- Equation hidden in signature over $\mathbb{Z}$:

$$\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathrm{C}$$

where the unknowns for the attacker are $\mathbf{y}_1, b, \mathbf{s}_1$

Discrete Gaussian Distribution



- Step 1 in signature algorithm: $\mathbf{y} \leftarrow D_{\mathbb{Z}^m,\sigma}$
- This is required to achieve (provable) security and small signature size.
- Not straightforward to do in practice: high precision required.
- But how do we use additional knowledge of **y** to find **s**?

Attack Scenario's

Attack Scenario 1

- Signature equation: $\mathbf{z} = \mathbf{y} + (-1)^b \mathbf{s} \mathrm{C}$

Scenario 1:
We can determine **y** completely from a side-channel attack

Attack Scenario 1

- Signature equation: $\mathbf{z} = \mathbf{y} + (-1)^b \mathbf{s}\mathrm{C}$

Scenario 1:
We can determine **y** completely from a side-channel attack

- Only need one signature.
- Solve equation $(-1)^b(\mathbf{z} - \mathbf{y}) = \mathbf{s}\mathrm{C}$ for **s**.
- But unlikely...(?)

Attack Scenario 2

- System of n equations over $\mathbb{Z}$:

$$\underbrace{\begin{bmatrix} z_0 \\ z_1 \\ \dots \\ z_{n-1} \end{bmatrix}}_{\text{Signature 1}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{bmatrix}}_{\text{Noise}} + \underbrace{(-1)^b}_{\text{Sign}} \underbrace{\begin{bmatrix} - & \mathbf{c}_0 & - \\ - & \mathbf{c}_1 & - \\ - & \dots & - \\ - & \mathbf{c}_{n-1} & - \end{bmatrix}}_{\text{Challenge}} \cdot \underbrace{\begin{bmatrix} s_0 \\ s_1 \\ \dots \\ s_{n-1} \end{bmatrix}}_{\text{Secret}}$$

Scenario 2:

There is a small set of values and an attacker can determine y_i when it is in this set.

Attack Scenario 2

- System of n equations over $\mathbb{Z}$:

$$\underbrace{\begin{bmatrix} z_0 \\ z_1 \\ \dots \\ z_{n-1} \end{bmatrix}}_{\text{Signature 1}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{bmatrix}}_{\text{Noise}} + \underbrace{(-1)^b}_{\text{Sign}} \underbrace{\begin{bmatrix} - & \mathbf{c}_0 & - \\ - & \mathbf{c}_1 & - \\ - & \dots & - \\ - & \mathbf{c}_{n-1} & - \end{bmatrix}}_{\text{Challenge}} \cdot \underbrace{\begin{bmatrix} s_0 \\ s_1 \\ \dots \\ s_{n-1} \end{bmatrix}}_{\text{Secret}}$$

Scenario 2:
There is a small set of values and an attacker can determine y_i when it is in this set.

- Since this set is small, we need more than one signature.
- Zoom in on coordinate-wise equations:

$$z_i = y_i + (-1)^b \langle \mathbf{c}_i, \mathbf{s} \rangle$$

- If we know y_i, we save $\zeta_k = \mathbf{c}_i$ in a list with y_i and z_i.

Attack Scenario 2

- We can acquire enough of these vectors from multiple signatures and form:

$$\begin{bmatrix} (-1)^{b_0}(z_0 - y_0) \\ (-1)^{b_1}(z_1 - y_1) \\ \dots \\ (-1)^{b_{n-1}}(z_{n-1} - y_{n-1}) \end{bmatrix} = \begin{bmatrix} - & \zeta_0 & - \\ - & \zeta_1 & - \\ - & \dots & - \\ - & \zeta_{n-1} & - \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ \dots \\ s_{n-1} \end{bmatrix}$$

- Unfortunately: all bits b_i are unknown.

Attack Scenario 2

- We can acquire enough of these vectors from multiple signatures and form:

$$\begin{bmatrix} (-1)^{b_0}(z_0 - y_0) \\ (-1)^{b_1}(z_1 - y_1) \\ \dots \\ (-1)^{b_{n-1}}(z_{n-1} - y_{n-1}) \end{bmatrix} = \begin{bmatrix} - & \zeta_0 & - \\ - & \zeta_1 & - \\ - & \dots & - \\ - & \zeta_{n-1} & - \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ \dots \\ s_{n-1} \end{bmatrix}$$

- Unfortunately: all bits b_i are unknown.
- Trick: if we know y_i, we can *be selective* and ensure that $z_i = y_i$, before saving $\zeta_k = \mathbf{c}_i$ in our list.
- We can eliminate b:

$$(-1)^b(z_i - y_i) = 0 = \langle \zeta_k, \mathbf{s} \rangle$$

Attack Scenario 2

- If we know y_i and $z_i = y_i$: we save $\zeta_k = \mathbf{c}_i$.
- Acquire enough of these vectors from multiple signatures and we have equation:

$$\mathbf{s}\mathrm{L} = \mathbf{0}$$

- With very high probability: secret vector $\mathbf{s}$ is the only vector in the integer (left) kernel of L.

Attack Scenario 3

- Signature equation over $\mathbb{Z}$: $\mathbf{z} = \mathbf{y} + (-1)^b \mathrm{C}\mathbf{s}$.
- Let us go one step further:

Scenario 3:

There is a small set of tuples $\{\gamma, \gamma + 1\}$ and an attacker can determine the tuple for y_i when it is in this set.

With high probability, $y_i = \gamma$

Attack Scenario 3

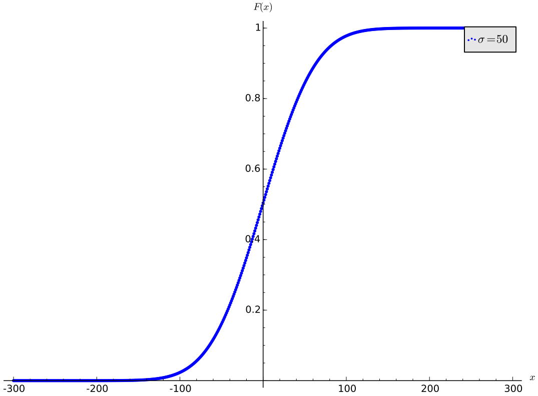
- Apply same method as previous:
- If we know $y_i \in \{\gamma, \gamma + 1\}$ and $z_i = \gamma$: we save $\zeta_k = \mathbf{c}_i$.
- Now $\mathbf{s}\mathrm{L}$ is not an all-zero vector, but it is small.
- Use LLL-algorithm to compute small vectors, search for **s** in the unitary transformation matrix.
- Verify correctness with public key.

Cache-Attacking BLISS with CDT Sampling

Cache-attacks on BLISS

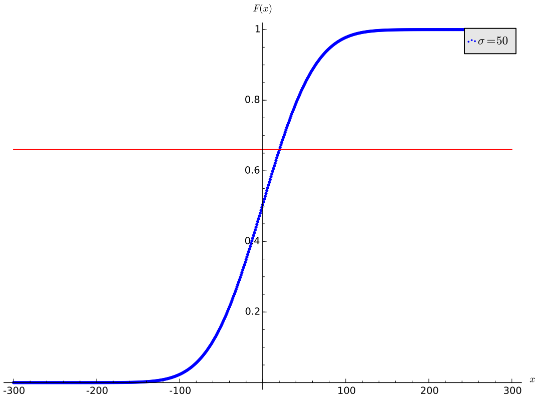
- $\mathbf{y} \leftarrow D_{\mathbb{Z}^m,\sigma}$
- Three attack scenario's using additional knowledge of **y**.
- Implemented cache-attacks on two discrete Gaussian samplers: CDT sampling and Bernoulli-based sampling, which both use table look-ups.

CDT Sampling with Guide Table



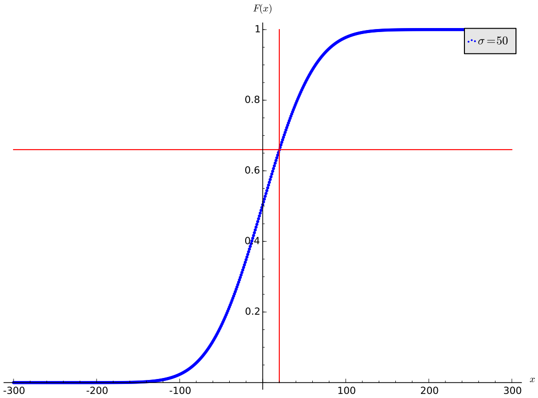
1. Save values of the discrete Gaussian CDF in table T.

CDT Sampling with Guide Table



2. Generate a random value $r \in [0, 1)$

CDT Sampling with Guide Table



3. Perform a binary search to find sample x with $T(x-1) \leq r < T(x)$.

CDT Sampling with Guide Table

- Some speed-ups used in practice:
  - Use only non-negative values and pick a random sign at the end.
  - Use additional table I with intervals, to speed-up the binary search

CDT Sampling with Guide Table

- Some speed-ups used in practice:
  - Use only non-negative values and pick a random sign at the end.
  - Use additional table I with intervals, to speed-up the binary search
- Two types of cache weaknesses:

CDT Sampling with Guide Table

- Some speed-ups used in practice:
  - Use only non-negative values and pick a random sign at the end.
  - Use additional table I with intervals, to speed-up the binary search
- Two types of cache weaknesses:
  - Intersection (use knowledge of accesses in I and T)

CDT Sampling with Guide Table

- Some speed-ups used in practice:
  - Use only non-negative values and pick a random sign at the end.
  - Use additional table I with intervals, to speed-up the binary search
- Two types of cache weaknesses:
  - Intersection (use knowledge of accesses in I and T)
  - Last-jump (track the binary search using knowledge of multiple accesses in T)

CDT Sampling with Guide Table

- Some speed-ups used in practice:
  - Use only non-negative values and pick a random sign at the end.
  - Use additional table I with intervals, to speed-up the binary search
- Two types of cache weaknesses:
  - Intersection (use knowledge of accesses in I and T)
  - Last-jump (track the binary search using knowledge of multiple accesses in T)
- Find all cache weaknesses for tables T and I for specific parameter set.

CDT Sampling with Guide Table

- Some speed-ups used in practice:
  - Use only non-negative values and pick a random sign at the end.
  - Use additional table I with intervals, to speed-up the binary search
- Two types of cache weaknesses:
  - Intersection (use knowledge of accesses in I and T)
  - Last-jump (track the binary search using knowledge of multiple accesses in T)
- Find all cache weaknesses for tables T and I for specific parameter set.
- Use only those weaknesses satisfying:

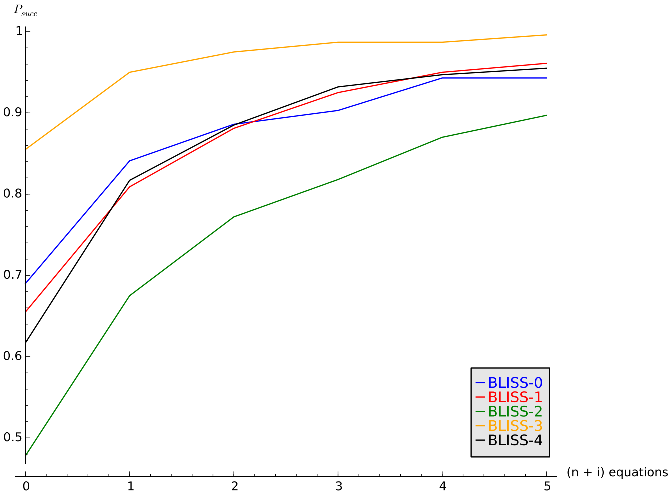
Scenario 3:

There is a small set of tuples $\{\gamma, \gamma + 1\}$ and an attacker can determine the tuple for y_i when it is in this set.

With high probability, $y_i = \gamma$

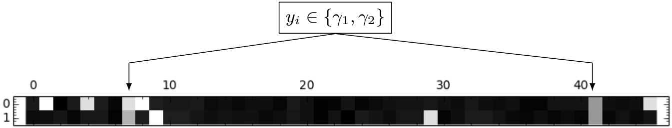
Experiments

- Results (modelled) cache-attack with perfect side-channel.
- BLISS with CDT sampling:



Experiments

- Proof-of-concept attack using FLUSH+RELOAD technique.
- Visualization of last-jump weakness:



- Experiments with BLISS-I succeeded 90% of the time.

Details in full paper

- Similar method and results achieved for Bernoulli-based sampling method, including experiments.
- Full paper includes analysis of weaknesses of Knuth-Yao and discrete Ziggurat samplers.
- Details in *https://eprint.iacr.org/2016/300*.