Physical Attacks Against Lattice-Based Schemes

Thomas Espitau

LIP6 – Sorbonne Université

WRACH – Roscoff 04/2019

joint work with P.-A. Fouque, B. Gérard and M. Tibouchi

Outline

Outline

Introduction

- Implementation attacks
- Implementation attacks on lattice schemes

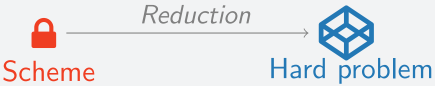
Physical attacks against BLISS

- A bird's eye view on lattices
- The BLISS signature scheme
- Fault attack on the Gaussian sampling
- SCA on the rejection sampling

Conclusion and countermeasures

Breaking provable crypto is hard

- Most crypto proposed in the last 15–20 years: provably secure



- Breaking it = provably as hard as solving some algorithmic problem like *integer factorization*, computing *discrete logarithms*(classical crypto) or *SVP, CVP, LWE,* (lattice based), …
  - Cryptanalysis = major algorithmic advance?

Breaking provable crypto is hard

- Most crypto proposed in the last 15–20 years: provably secure

- Breaking it = provably as hard as solving some algorithmic problem like *integer factorization*, computing *discrete logarithms*(classical crypto) or *SVP, CVP, LWE,* (lattice based), ...

  - Cryptanalysis = major algorithmic advance?

Breaking provable crypto is hard

- Most crypto proposed in the last 15–20 years: provably secure

- Breaking it = provably as hard as solving some algorithmic problem like *integer factorization*, computing *discrete logarithms*(classical crypto) or *SVP, CVP, LWE,* (lattice based), ...
  - Cryptanalysis = major algorithmic advance?

Yet, many attacks against deployed crypto

The crypto protocol that is perhaps most used in everyday life, TLS, is attacked all the time!

Internet Engineering Task Force (IETF) Y. Sheffer
Request for Comments: 7457 Porticor
Category: Informational R. Holz
ISSN: 2070-1721 Technische Universitaet Muenchen
P. Saint-Andre
&yet
February 2015

Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)

Abstract

Over the last few years, there have been several serious attacks on Transport Layer Security (TLS), including attacks on its most commonly used ciphers and modes of operation. This document summarizes these attacks, with the goal of motivating generic and protocol-specific recommendations on the usage of TLS and Datagram TLS (DTLS).

Yet, many attacks against deployed crypto

The crypto protocol that is perhaps most used in everyday life, TLS, is attacked all the time!

Table of Contents

So how do people actually break crypto?

- Very rarely: major algorithmic improvement
  - Big one recently: progress on small characteristic discrete logarithms/pairings [BaGaJoTh13]
- More commonly: non-provably secure schemes shown to be insecure
  - Several of the TLS attacks
  - Many legacy scheme still in use could be broken (e.g. PKCS#1v1.5 signatures?)
- Most importantly: implementation attacks!

So how do people actually break crypto?

- Very rarely: major algorithmic improvement
  - Big one recently: progress on small characteristic discrete logarithms/pairings [BaGaJoTh13]
- More commonly: non-provably secure schemes shown to be insecure
  - Several of the TLS attacks
  - Many legacy scheme still in use could be broken (e.g. PKCS#1v1.5 signatures?)
- Most importantly: implementation attacks!

So how do people actually break crypto?

- Very rarely: major algorithmic improvement
  - Big one recently: progress on small characteristic discrete logarithms/pairings [BaGaJoTh13]
- More commonly: non-provably secure schemes shown to be insecure
  - Several of the TLS attacks
  - Many legacy scheme still in use could be broken (e.g. PKCS#1v1.5 signatures?)
- Most importantly: implementation attacks!

Implementation attacks

- To break a real-world crypto implementation, no need to play by the rules of black-box security

- In particular, provably secure schemes can be broken by bypassing the (usually black-box) security model
  - Remark: some attempts to also capture non black-box attacks in security proofs (e.g. leakage-resilient crypto...)

- These are implementation attacks

Implementation attacks

- To break a real-world crypto implementation, no need to play by the rules of black-box security
- In particular, provably secure schemes can be broken by bypassing the (usually black-box) security model
  - Remark: some attempts to also capture non black-box attacks in security proofs (e.g. leakage-resilient crypto...)
- These are implementation attacks

Implementation attacks

- To break a real-world crypto implementation, no need to play by the rules of black-box security
- In particular, provably secure schemes can be broken by bypassing the (usually black-box) security model
  - Remark: some attempts to also capture non black-box attacks in security proofs (e.g. leakage-resilient crypto...)
- These are implementation attacks

Various types of implementation attacks

- Correctness attacks: use the implementation as a black box, but send malformed/incorrect/invalid/malicious inputs
  - think of fuzzing in software security for instance
- Side-channel attacks: passive physical attacks, exploiting information leakage about the computation or the keys
  - timing, electromagnetic emanations, heat production, power supply
- Fault attacks: active physical attacks, trying to extract secret information by tampering with the device to cause errors during the cryptographic computation
  - power tampering, laser beams

Various types of implementation attacks

- Correctness attacks: use the implementation as a black box, but send malformed/incorrect/invalid/malicious inputs
  - think of fuzzing in software security for instance

- Side-channel attacks: passive physical attacks, exploiting information leakage about the computation or the keys
  - timing, electromagnetic emanations, heat production, power supply

- Fault attacks: active physical attacks, trying to extract secret information by tampering with the device to cause errors during the cryptographic computation
  - power tampering, laser beams

Various types of implementation attacks

- Correctness attacks: use the implementation as a black box, but send malformed/incorrect/invalid/malicious inputs
  - think of fuzzing in software security for instance
- Side-channel attacks: passive physical attacks, exploiting information leakage about the computation or the keys
  - timing, electromagnetic emanations, heat production, power supply
- Fault attacks: active physical attacks, trying to extract secret information by tampering with the device to cause errors during the cryptographic computation
  - power tampering, laser beams

Outline

Towards postquantum cryptography

- Quantum computers would break all currently deployed public-key crypto: RSA, discrete logs, elliptic curves

- Agencies warn that we should prepare the transition to quantum-resistant crypto
  - NSA deprecating Suite B (elliptic curves)
  - NIST is pursuing their postquantum competition (round 2 is going on)

- In theory, plenty of known schemes are quantum-resistant
  - Some primitives achieved with codes, hash trees, multivariate crypto, knapsacks, isogenies...
  - Almost everything possible with lattices

Towards postquantum cryptography

- Quantum computers would break all currently deployed public-key crypto: RSA, discrete logs, elliptic curves

- Agencies warn that we should prepare the transition to quantum-resistant crypto
  - NSA deprecating Suite B (elliptic curves)
  - NIST is pursuing their postquantum competition (round 2 is going on)

- In theory, plenty of known schemes are quantum-resistant
  - Some primitives achieved with codes, hash trees, multivariate crypto, knapsacks, isogenies…
  - Almost everything possible with lattices

Towards postquantum cryptography

- In practice, very few actual implementations
  - Secure parameters often unclear
  - Concrete software/hardware implementation papers quite rare
  - Almost no consideration for implementation attacks

- Serious issue if we want practical postquantum crypto

Implementations of lattice-based schemes (I)

- Implementation work on lattice-based crypto is limited and mostly academic, usually targeted towards efficiency
- Things tends to move a bit with NIST competition, but efforts are still made on efficiency rather than on code protection

Implementations of lattice-based schemes (I)

- Implementation work on lattice-based crypto is limited and mostly academic, usually targeted towards efficiency
- Things tends to move a bit with NIST competition, but efforts are still made on efficiency rather than on code protection

Implementations of lattice-based schemes (II)

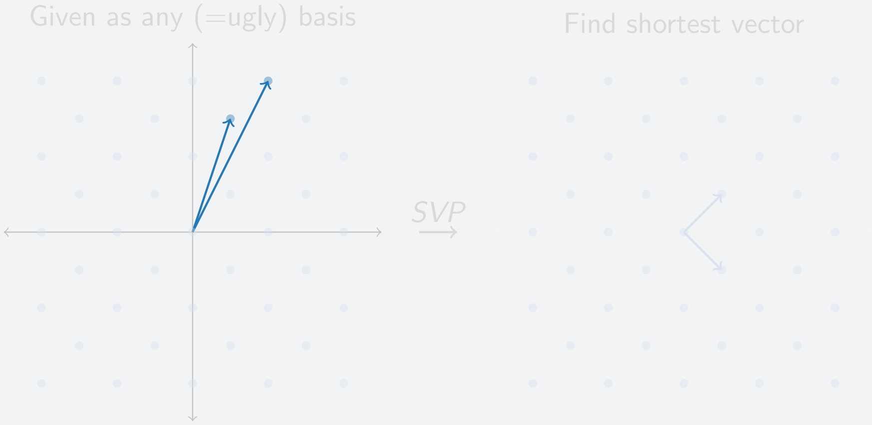
- One scheme has "industry" backing and quite a bit of code: NTRU
  - NTRUEncrypt is an ANSI standard, and believed to be okay
  - NTRUSign is a trainwreck that has been patched and broken many times
- In terms of practical schemes, other than NTRU, main efforts on signatures
  - GLP: improvement of Lyubashevsky signatures, efficient in SW and HW (CHES'12)
  - BLISS: improvement of GPL, even better (CRYPTO'13, CHES'14), and Dilithium (TCHES '18, NIST submitted)
  - DLP: hash-and-sign scheme using GPV sampling on NTRU lattices (AC'14)
  - A few others: PASSSign (ACNS'14), ~~(q)TESLA~~ (AFRICACRYPT'16), FALCON (NIST submitted)

Implementation attacks on lattice-based schemes

- Survey by Taha and Eisenbarth (eprint 2015/1083) on implementation attacks against postquantum schemes; thorough literature review
- Up to 2016, for lattice-based schemes, only referenced attacks are against NTRU
  - NTRUEncrypt: a few papers about timing attacks (CT-RSA'07), power analysis (RFIDSec'08+journals) and faults (JCEN, IEICE Trans.)
  - NTRUSign: one paper about faults (Cryptogr. and Comm.)
- On signatures: fault attacks (SAC 2016), side-channel analysis on lattice-based signatures (Groot Bruinderink et al. CHES 2016, CCS 2017, Pessl et al. CCS 2017),
- Impulsion in this direction with all the new NIST candidates.

Outline

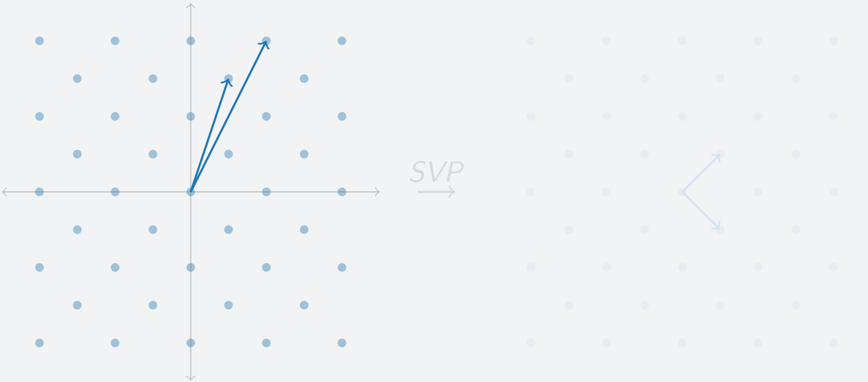
Shortest vector problem



Shortest vector problem

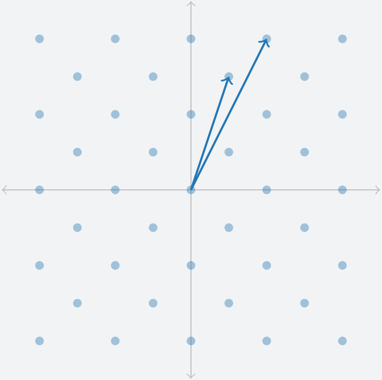
Given as any (=ugly) basis

Find shortest vector



Shortest vector problem

Given as any (=ugly) basis

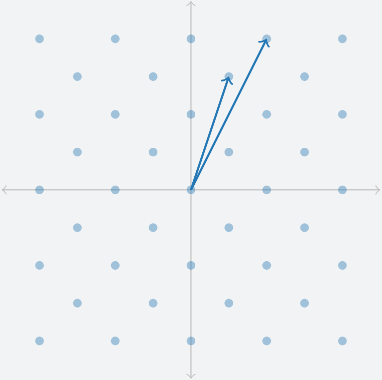


$\xrightarrow{SVP}$

Find shortest vector

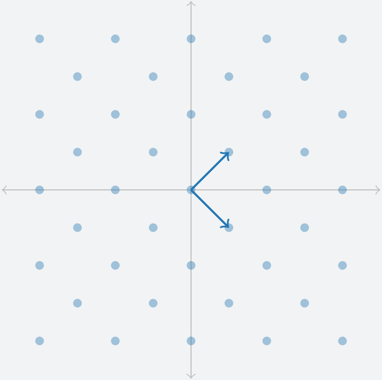
Shortest vector problem

Given as any (=ugly) basis



$\xrightarrow{SVP}$

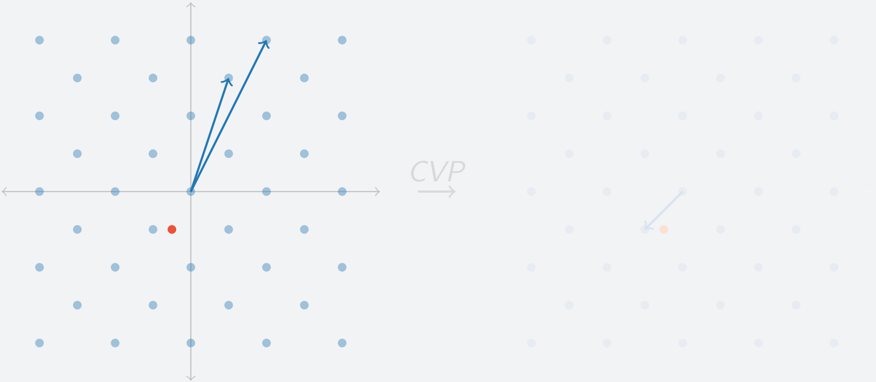
Find shortest vector



Closest vector problem

Given as any (=ugly) basis
and point outside the lattice

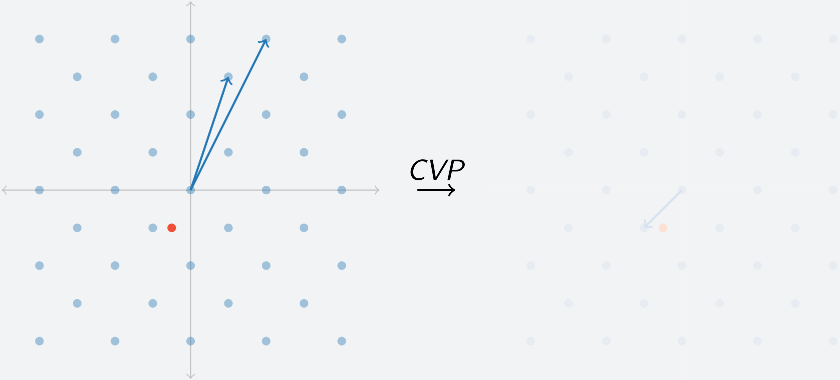
Find closest vector



Closest vector problem

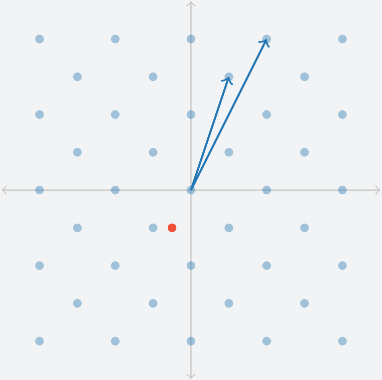
Given as any (=ugly) basis
and point outside the lattice

Find closest vector



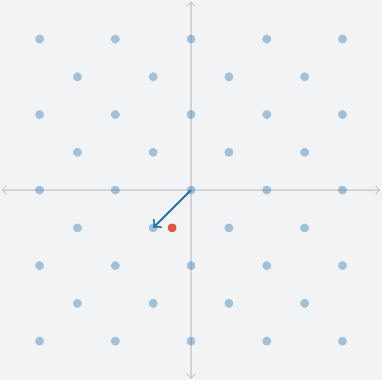
Closest vector problem

Given as any (=ugly) basis
and point outside the lattice



$\xrightarrow{CVP}$

Find closest vector



Outline

Introduction

Implementation attacks

Implementation attacks on lattice schemes

Physical attacks against BLISS

A bird's eye view on lattices

The BLISS signature scheme

Fault attack on the Gaussian sampling

SCA on the rejection sampling

Conclusion and countermeasures

BLISS: the basics

- Introduced by Ducas, Durmus, Lepoint and Lyubashevsky at CRYPTO'13
- Improvement of the earlier Ring-SIS-based scheme of Lyubashevsky (EC'12)
- Still following the structure of "Fiat–Shamir with aborts"
- Still defined over some ring $\mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$
- Main improvement: use bimodal Gaussian distributions to reduce the size of parameters

BLISS: the basics

- Introduced by Ducas, Durmus, Lepoint and Lyubashevsky at CRYPTO'13
- Improvement of the earlier Ring-SIS-based scheme of Lyubashevsky (EC'12)
- Still following the structure of "Fiat–Shamir with aborts"
- Still defined over some ring $\mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$
- Main improvement: use bimodal Gaussian distributions to reduce the size of parameters

BLISS: the basics

- Introduced by Ducas, Durmus, Lepoint and Lyubashevsky at CRYPTO'13
- Improvement of the earlier Ring-SIS-based scheme of Lyubashevsky (EC'12)
- Still following the structure of "Fiat–Shamir with aborts"
- Still defined over some ring $\mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$
- Main improvement: use bimodal Gaussian distributions to reduce the size of parameters

BLISS: the basics

- Introduced by Ducas, Durmus, Lepoint and Lyubashevsky at CRYPTO'13
- Improvement of the earlier Ring-SIS-based scheme of Lyubashevsky (EC'12)
- Still following the structure of "Fiat–Shamir with aborts"
- Still defined over some ring $\mathcal{R} = \mathbb{Z}[\mathbf{x}]/(\mathbf{x}^n + 1)$
- Main improvement: use bimodal Gaussian distributions to reduce the size of parameters

BLISS: key generation

```
function KeyGen()
    choose f, g as uniform polynomials with exactly d_1 = ⌈δ_1 n⌉ entries in {±1} and d_2 = ⌈δ_2 n⌉ entries in {±2}
    S = (s_1, s_2)^T ← (f, 2g + 1)^T
    if N_κ(S) ≥ C^2 · 5 · (⌈δ_1 n⌉ + 4⌈δ_2 n⌉) · κ then restart
    if f is not invertible then restart
    a_q = (2g + 1)/f mod q
    return (pk = A, sk = S) where A = (a_1 = 2a_q, q − 2) mod 2q
end function
```

BLISS: signature

function $\textsc{Sign}(\mu, pk = \mathbf{A},\ sk = \mathbf{S})$
- $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D^n_{\mathbb{Z},\sigma}$ ▷ Gaussian sampling
- $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$ ▷ $\zeta = 1/(q-2)$
- $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$ ▷ special hashing
- choose a random bit b
- $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$
- $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$
- **continue** with probability $1/\big(M \exp(-\|\mathbf{Sc}\|/(2\sigma^2)) \cosh(\langle \mathbf{z}, \mathbf{Sc} \rangle/\sigma^2\big)$ otherwise **restart**
- $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$
- **return** $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

end function

BLISS: signature

function $\textsc{Sign}(\mu, pk = \mathbf{A},\ sk = \mathbf{S})$

    $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D^n_{\mathbb{Z},\sigma}$ ▷ Gaussian sampling

    $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$ ▷ $\zeta = 1/(q-2)$

    $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$ ▷ special hashing

    choose a random bit b

    $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$

    $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$

    continue with probability
$1/\big(M \exp(-\|\mathbf{Sc}\|/(2\sigma^2)) \cosh(\langle \mathbf{z}, \mathbf{Sc} \rangle/\sigma^2\big)$ otherwise **restart**

    $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$

    return $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

end function

BLISS: signature

function $\textsc{Sign}(\mu, pk = \mathbf{A},\ sk = \mathbf{S})$
    $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D^n_{\mathbb{Z},\sigma}$ ▷ Gaussian sampling
    $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$ ▷ $\zeta = 1/(q-2)$
    $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$ ▷ special hashing
    choose a random bit b
    $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$
    $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$
    continue with probability $1/\big(M \exp(-\|\mathbf{Sc}\|/(2\sigma^2)) \cosh(\langle \mathbf{z}, \mathbf{Sc} \rangle / \sigma^2\big)$ otherwise **restart**
    $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$
    return $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
end function

BLISS: signature

function $\textsc{Sign}(\mu, pk = \mathbf{A},\ sk = \mathbf{S})$

$\mathbf{y}_1, \mathbf{y}_2 \leftarrow D^n_{\mathbb{Z},\sigma}$ ▷ Gaussian sampling

$\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$ ▷ $\zeta = 1/(q-2)$

$\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$ ▷ special hashing

choose a random bit b

$\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$

$\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$

continue with probability $1/\Big(M \exp(-\|\mathbf{Sc}\|/(2\sigma^2)) \cosh(\langle \mathbf{z}, \mathbf{Sc} \rangle/\sigma^2\Big)$ otherwise **restart**

$\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$

return $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

end function

BLISS: verification

function $\textsc{Verify}(\mu, \mathbf{A}, (\mathbf{z}_1, \mathbf{z}_2^{\dagger}, \mathbf{c}))$
    if $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^{\dagger})\|_2 > B_2$ **then** reject
    if $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^{\dagger})\|_\infty > B_\infty$ **then** reject
    accept iff $\mathbf{c} = H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^{\dagger} \bmod p, \mu)$
end function

BLISS: parameters

- Parameters proposed by Ducas et al. for 128-bit security (BLISS–I & BLISS–II)
  - $n = 512$, $q = 12289$
  - $(\delta_1, \delta_2) = (0.3, 0)$ (density of $\mathbf{f}, \mathbf{g}$)
  - $\sigma = 215$ for BLISS–I, 107 for BLISS–II
  - $\kappa = 23$ (number of 1's in $\mathbf{c}$)

Outline

Attacking y

- The ring element $\mathbf{y}_1$, which acts as additive mask in the relation:

$$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod{q}$$

  is sampled according to a discrete Gaussian
- Sampling carried out coefficient by coefficient

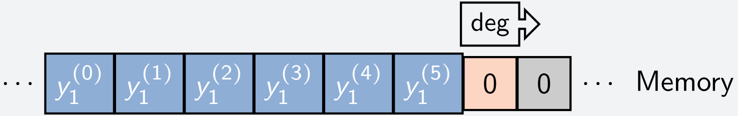
for i = 0 **to** n **do** $y_1^{(i)} \leftarrow \cdots$ **end**

Attacking y

- The ring element $\mathbf{y}_1$, which acts as additive mask in the relation:

$$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod{q}$$

  is sampled according to a discrete Gaussian
- Sampling carried out coefficient by coefficient



for i = 0 **to** n **do** $y_1^{(i)} \leftarrow \cdots$ **end**

Attacking $\mathbf{y}$

- Idea of the attack: use fault injection to abort the sampling early, so that a faulty signature will be generated with a low-degree $\mathbf{y}_1$

- Can be done by attacking the branching test of the loop (voltage spike, clock variation...), or the contents of the loop counter (lasers, x-rays...)

Attack details (I)

- So let's say we get a signature generated with $\mathbf{y}_1$ of degree $m \ll n$
- If $\mathbf{c}$ is invertible (probability around $(1 - 1/q)^n \approx 96\%$), we can compute:
  - $$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + (-1)^b \mathbf{s}_1 \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + \mathbf{s}_1 \pmod q$$
- WLOG, $b = 0$ (equivalent keys)

Attack details (I)

- So let's say we get a signature generated with $\mathbf{y}_1$ of degree $m \ll n$
- If $\mathbf{c}$ is invertible (probability around $(1-1/q)^n \approx 96\%$), we can compute:
  - $$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod{q}$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + (-1)^b \mathbf{s}_1 \pmod{q}$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + \mathbf{s}_1 \pmod{q}$$
- WLOG, $b = 0$ (equivalent keys)

Attack details (I)

- So let's say we get a signature generated with $\mathbf{y}_1$ of degree $m \ll n$
- If $\mathbf{c}$ is invertible (probability around $(1 - 1/q)^n \approx 96\%$), we can compute:
  - $$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + (-1)^b \mathbf{s}_1 \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + \mathbf{s}_1 \pmod q$$
- WLOG, $b = 0$ (equivalent keys)

Attack details (I)

- So let's say we get a signature generated with $\mathbf{y}_1$ of degree $m \ll n$
- If $\mathbf{c}$ is invertible (probability around $(1 - 1/q)^n \approx 96\%$), we can compute:
  - $$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + (-1)^b \mathbf{s}_1 \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + \mathbf{s}_1 \pmod q$$
- WLOG, $b = 0$ (equivalent keys)

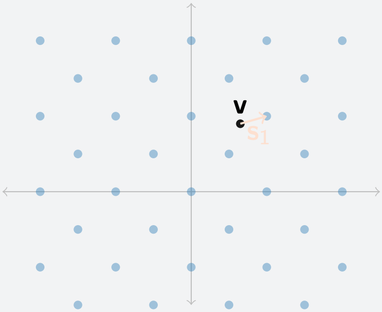
Attack details (I)

- So let's say we get a signature generated with $\mathbf{y}_1$ of degree $m \ll n$
- If $\mathbf{c}$ is invertible (probability around $(1-1/q)^n \approx 96\%$), we can compute:
  - $$\mathbf{z}_1 \equiv \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c} \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + (-1)^b \mathbf{s}_1 \pmod q$$
  - $$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \mathbf{c}^{-1}\mathbf{y}_1 + \mathbf{s}_1 \pmod q$$
- WLOG, $b = 0$ (equivalent keys)

Attack details (I)

$$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \underbrace{\mathbf{c}^{-1} \overbrace{\mathbf{y}_1}^{\in\mathbb{Z}[\mathbf{x}^i]}}_{\in\mathbb{Z}[\mathbf{c}^{-1}\mathbf{x}^i]=\mathbb{Z}[\mathbf{w}_i]} + \mathbf{s}_1 \pmod q$$

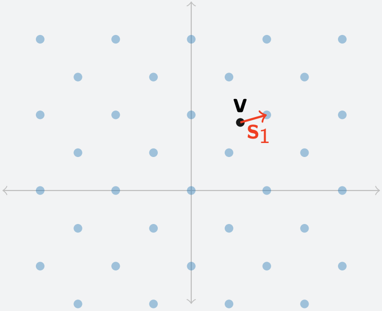
- Since $\mathbf{s}_1$ is very short, $\mathbf{v}$ very close to the lattice L generated by $q\mathbb{Z}^n$ and $\mathbf{w}_i = \mathbf{c}^{-1}\mathbf{x}^i$, $i = 0, \ldots, m$



Attack details (I)

$$\mathbf{v} = \mathbf{c}^{-1}\mathbf{z}_1 \equiv \underbrace{\mathbf{c}^{-1}\overbrace{\mathbf{y}_1}^{\in\mathbb{Z}[\mathbf{x}^i]}}_{\in\mathbb{Z}[\mathbf{c}^{-1}\mathbf{x}^i]=\mathbb{Z}[\mathbf{w}_i]} + \mathbf{s}_1 \pmod{q}$$

- Since $\mathbf{s}_1$ is very short, $\mathbf{v}$ very close to the lattice L generated by $q\mathbb{Z}^n$ and $\mathbf{w}_i = \mathbf{c}^{-1}\mathbf{x}^i$, $i = 0, \ldots, m$



Attack details (I)

- L of dimension n: too large to apply lattice reduction
- However, we have the same relation on arbitrary subset of coefficients: we can reduce the dimension

Attack details (I)

- L of dimension n: too large to apply lattice reduction
- However, we have the same relation on arbitrary subset of coefficients: we can reduce the dimension

Attack details (II)

- More precisely, fix a subset $I \subset \{0, \ldots, n-1\}$ of ℓ indices, and let $\varphi_I \colon \mathbb{Z}^n \to \mathbb{Z}^I$ be the obvious projection
- $\varphi_I(\mathbf{v})$ is close to the lattice generated by $\varphi_I(\mathbf{w}_i)$ and $q\mathbb{Z}^I$, and if ℓ is large enough, the difference should be $\varphi_I(\mathbf{s}_1)$.
- Solve this close vector problem using Babai nearest plane algorithm. Condition on ℓ to recover $\varphi_I(\mathbf{s}_1)$:

$$\ell + 1 \gtrsim \frac{m + 2 + \frac{\log \sqrt{\delta_1 + 4\delta_2}}{\log q}}{1 - \frac{\log \sqrt{2\pi e(\delta_1 + 4\delta_2)}}{\log q}}$$

Attack details (II)

- More precisely, fix a subset $I \subset \{0, \ldots, n-1\}$ of ℓ indices, and let $\varphi_I : \mathbb{Z}^n \to \mathbb{Z}^I$ be the obvious projection

- $\varphi_I(\mathbf{v})$ is close to the lattice generated by $\varphi_I(\mathbf{w}_i)$ and $q\mathbb{Z}^I$, and if ℓ is large enough, the difference should be $\varphi_I(\mathbf{s}_1)$.

- Solve this close vector problem using Babai nearest plane algorithm. Condition on ℓ to recover $\varphi_I(\mathbf{s}_1)$:

$$\ell + 1 \gtrsim \frac{m + 2 + \frac{\log\sqrt{\delta_1 + 4\delta_2}}{\log q}}{1 - \frac{\log\sqrt{2\pi e(\delta_1 + 4\delta_2)}}{\log q}}$$

Attack details (II)

- More precisely, fix a subset $I \subset \{0, \ldots, n-1\}$ of ℓ indices, and let $\varphi_I : \mathbb{Z}^n \to \mathbb{Z}^I$ be the obvious projection
- $\varphi_I(\mathbf{v})$ is close to the lattice generated by $\varphi_I(\mathbf{w}_i)$ and $q\mathbb{Z}^I$, and if ℓ is large enough, the difference should be $\varphi_I(\mathbf{s}_1)$.
- Solve this close vector problem using Babai nearest plane algorithm. Condition on ℓ to recover $\varphi_I(\mathbf{s}_1)$:

$$\ell + 1 \gtrsim \frac{m + 2 + \frac{\log\sqrt{\delta_1 + 4\delta_2}}{\log q}}{1 - \frac{\log\sqrt{2\pi e(\delta_1 + 4\delta_2)}}{\log q}}$$

Attack details (III)

- For BLISS–I and BLISS–II, this says $\ell \approx 1.09 \cdot m$
- In practice: works fine with LLL for $m \lesssim 60$ and with BKZ with $m \lesssim 100$
- Just apply the attack for several choices of I to recover all of $\mathbf{s}_1$, and subsequently $\mathbf{s}_2$: full key recovery with one faulty signature!

Implementation results

Fault after iteration number $m =$	5	10	20	40	80	100
Theoretical minimum dimension $\ell_{\min}$	6	11	22	44	88	110
Dimension ℓ in our experiment	6	12	24	50	110	150
Lattice reduction algorithm	LLL	LLL	LLL	BKZ–20	BKZ–25	BKZ–25
Avg. CPU time to recover ℓ coeffs. (s)	0.005	0.022	0.23	7.3	941	33655
Avg. CPU time for full key recovery	0.5 s	1 s	5 s	80 s	80 min	38 h

Outline

Attack overview

- The rejection sampling step is the cornerstone of BLISS security (difference with NTRUSign) and efficient (the bimodal aspect)
- In practice: difficult to implement on constrained devices, so some tricks have to be used
- The optimized version of the rejection sampling used iterated Bernoulli trials on each of the bits of $\|\mathbf{Sc}\|^2$; as a result, we can read that value on an SPA trace
- This yields to the recovery of the relative algebraic norm $\mathbf{s} \cdot \bar{\mathbf{s}}$ of the secret key. Algorithmic number theoretic techniques (Howgrave-Graham–Szydlo) can then be used to retrieve $\mathbf{s}$!

Attack overview

- The rejection sampling step is the cornerstone of BLISS security (difference with NTRUSign) and efficient (the bimodal aspect)
- In practice: difficult to implement on constrained devices, so some tricks have to be used
- The optimized version of the rejection sampling used iterated Bernoulli trials on each of the bits of $\|\mathbf{Sc}\|^2$; as a result, we can read that value on an SPA trace
- This yields to the recovery of the relative algebraic norm $\mathbf{s} \cdot \bar{\mathbf{s}}$ of the secret key. Algorithmic number theoretic techniques (Howgrave-Graham–Szydlo) can then be used to retrieve $\mathbf{s}$!

Attack overview

- The rejection sampling step is the cornerstone of BLISS security (difference with NTRUSign) and efficient (the bimodal aspect)
- In practice: difficult to implement on constrained devices, so some tricks have to be used
- The optimized version of the rejection sampling used iterated Bernoulli trials on each of the bits of $\|\mathbf{Sc}\|^2$; as a result, we can read that value on an SPA trace
- This yields to the recovery of the relative algebraic norm $\mathbf{s} \cdot \bar{\mathbf{s}}$ of the secret key. Algorithmic number theoretic techniques (Howgrave-Graham–Szydlo) can then be used to retrieve $\mathbf{s}$!

Attack overview

- The rejection sampling step is the cornerstone of BLISS security (difference with NTRUSign) and efficient (the bimodal aspect)
- In practice: difficult to implement on constrained devices, so some tricks have to be used
- The optimized version of the rejection sampling used iterated Bernoulli trials on each of the bits of $\|\mathbf{Sc}\|^2$; as a result, we can read that value on an SPA trace
- This yields to the recovery of the relative algebraic norm $\mathbf{s} \cdot \bar{\mathbf{s}}$ of the secret key. Algorithmic number theoretic techniques (Howgrave-Graham–Szydlo) can then be used to retrieve $\mathbf{s}$!

BLISS rejection sampling

function SampleBernExp($x \in [0, 2^\ell) \cap \mathbb{Z}$) **for** $i = 0$ to $\ell - 1$ **do** **if** $x_i = 1$ **then** Sample $a \leftarrow \mathcal{B}_{c_i}$ **if** $a = 0$ **then return** 0 **end if** **end for** **return** 1 **end function** $\triangleright$ $x = K - \lVert \mathbf{Sc} \rVert^2$	**function**SampleBernCosh(x) **if** $a = 1$ **then return** 1 Sample $b \leftarrow \mathcal{B}_{1/2}$ **if** $b = 1$ **then restart** Sample $c \leftarrow \mathcal{B}_{\exp(-x/f)}$ **if** $c = 1$ **then restart** **return** 0 **end function** $\triangleright$ $x = 2 \cdot \langle \mathbf{z}, \mathbf{Sc} \rangle$

Sampling algorithms for the distributions $\mathcal{B}_{\exp(-x/f)}$ and $\mathcal{B}_{1/\cosh(x/f)}$ ($c_i = 2^i/f$ precomputed)

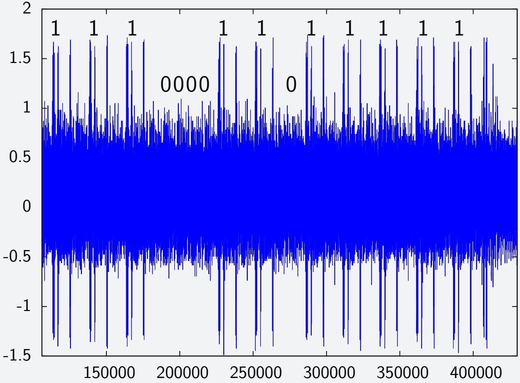
BLISS rejection sampling

function $\textsc{SampleBernExp}(x \in [0, 2^\ell) \cap \mathbb{Z})$
    for $i = 0$ to $\ell - 1$ **do**
        if $x_i = 1$ **then**
            Sample $a \leftarrow \mathcal{B}_{c_i}$
            if $a = 0$ **then return** 0
        end if
    end for
    return 1
end function $\triangleright\ x = K - \|\mathbf{Sc}\|^2$

function $\textsc{SampleBernCosh}(x)$
    if $a = 1$ **then return** 1
    Sample $b \leftarrow \mathcal{B}_{1/2}$
    if $b = 1$ **then restart**
    Sample $c \leftarrow \mathcal{B}_{\exp(-x/f)}$
    if $c = 1$ **then restart**
    return 0
end function $\triangleright\ x = 2 \cdot \langle \mathbf{z}, \mathbf{Sc} \rangle$

Sampling algorithms for the distributions $\mathcal{B}_{\exp(-x/f)}$ and $\mathcal{B}_{1/\cosh(x/f)}$ ($c_i = 2^i/f$ precomputed)

Experimental leakage



Electromagnetic measure of BLISS rejection sampling for norm $\|\mathbf{Sc}\|^2 = 14404$. One reads the value:

$$K - \|\mathbf{Sc}\|^2 = 46539 - 14404 = \overline{11100001101111}_2$$

Exploiting the leakage

- After collecting around 1024 traces, one obtains the value of $\mathbf{S} \cdot \overline{\mathbf{S}}$
- Algorithmic number theory (HGS) allows to deduce $\mathbf{S}$ itself (up to a root of unity):
  - Compute the norm of $\mathbf{S}$ over $\mathbb{Z}$, factor it.
  - Construct part of candidates secrets from the prime factors.
  - Combine each of them to get a candidate.
  - Enumerate the candidates.

Exploiting the leakage

- Attack is in polynomial time **IF** the (absolute) algebraic norm of **S** is easy to factor (e.g. semismooth: happens in a significant fraction of cases!)

- This is a full key recovery!

Security of BLISS implementation (PQ scheme) ... relies on integer factorization problem

Exploiting the leakage

- Attack is in polynomial time **IF** the (absolute) algebraic norm of **S** is easy to factor (e.g. semismooth: happens in a significant fraction of cases!)

- This is a full key recovery!

Security of BLISS implementation (PQ scheme) ... relies on integer factorization problem

Exploiting the leakage

- Attack is in polynomial time **IF** the (absolute) algebraic norm of **S** is easy to factor (e.g. semismooth: happens in a significant fraction of cases!)

- This is a full key recovery!

Security of BLISS implementation (PQ scheme) ... relies on integer factorization problem

Efficiency of the attack

Field size n	32	64	128	256	512
CPU time	0.6 s	13 s	21 min.	17h 22 min.	1.2 months (est.)
Clock cycles	$\approx 2^{30}$	$\approx 2^{35}$	$\approx 2^{41}$	$\approx 2^{47}$	$\approx 2^{53}$

Average running time of the attack for various field sizes n
BLISS parameters: $n = 256$ or 512

Cosh is also leaking... (WIP)

function SampleBernExp($x \in [0, 2^\ell) \cap \mathbb{Z}$)
  for $i = 0$ to $\ell - 1$ **do**
    if $x_i = 1$ **then**
      Sample $a \leftarrow \mathscr{B}_{c_i}$
      if $a = 0$ **then return** 0
    end if
  end for
  return 1
end function

function SampleBernCosh(x)
  Sample $a \leftarrow \mathscr{B}_{\exp(-x/f)}$
  if $a = 1$ **then return** 1
  Sample $b \leftarrow \mathscr{B}_{1/2}$
  if $b = 1$ **then restart**
  Sample $c \leftarrow \mathscr{B}_{\exp(-x/f)}$
  if $c = 1$ **then restart**
  return 0
end function ▷ $x = 2 \cdot \langle \mathbf{z}, \mathbf{Sc} \rangle$

Sampling algorithms for the distributions $\mathscr{B}_{\exp(-x/f)}$ and $\mathscr{B}_{1/\cosh(x/f)}$ ($c_i = 2^i/f$ precomputed)

Conclusion and countermeasures

- Important to investigate implementation attacks on lattice schemes
- Physical attack resistance should be part of the design goals for practical schemes
- We described faults and SCA against BLISS signatures, implementation is vulnerable to various leakage (timing, SPA)

Conclusion and countermeasures

- Possible countermeasures?

- Against faults:
  - check that the result has $> (1 - \varepsilon) \cdot n$ non zero coeffs.
  - randomize the order of generation of the coefficients? (still risky)
  - use double loop counters!

- Against side-channels:
  - compute rejection probability with floating point arithmetic (slow)
  - use a constant-time Bernoulli sampling (doable)
  - prefer a scheme with simpler structure (GLP) and use masking