Enhanced Lattice-Based Signatures on Reconfigurable Hardware

Thomas Pöppelmann[1] Léo Ducas[2] Tim Güneysu[1]

[1]Horst Görtz Institute for IT-Security,
Ruhr-University Bochum, Germany

[2]University of California,
San-Diego, USA

September 2014, CHES Workshop

Enhanced Lattice-Based Signatures on Reconfigurable Hardware

Lattice Based Cryptography

Many theoretical advantages over ECC/RSA:

- Strong theoretical guarentee of hardness
- Resist known quantum algorithms
- Very versatile: PKE, Signatures, IBE, FHE ...
- Asymptotically efficient $O(n^2)$ or even $O(n \log n)$.

In practice ?

Simple and very fast PKE (NTRU-Encrypt, LWE Encryption).
Efficient signatures have been more problematic.

BLISS: An optimized Signature Scheme

Signature without Trapdoors (Fiat-Shamir transform).

[Lyu09] Fiat-Shamir with aborts ($\approx 50 Kbits$)

[Lyu12] Abort Rate improved using Gaussians ($\approx 12 Kbits$)

[DDLL13] Abort Rate improved using Bimodal Gaussians ($\approx 5 Kbits$)

BLISS vs. ECDSA vs. RSA on Software

Scheme.	Security	Sign Size	Sign./s	Ver./s
BLISS-I	**128 bits**	**5.5kbits**	**8k**	**33k**
RSA 2048	112 bits	2kbits	0.8k	27k
RSA 4096	$\geqslant$ **128 bits**	**4kbits**	**0.1k**	**7.5k**
ECDSA 256	**128 bits**	**512 bits**	**9.5k**	**2.5k**

BLISS [DDLL13] compared to `OpenSSL` implem. of RSA and ECDSA on `x86-64`.

Fiat-Shamir with aborts [Lyu09, Lyu12, DDLL13]

sk : $\mathbf{S} \in \mathbb{Z}_{2q}^{m\times k}$, short

pk : $\mathbf{A} \in \mathbb{Z}_{2q}^{n\times m}$, random

$$\mathbf{T} = \mathbf{AS} = q\,\mathbf{Id}$$

Fiat-Shamir with aborts [Lyu09, Lyu12, DDLL13]

sk : $\mathbf{S} \in \mathbb{Z}_{2q}^{m\times k}$, short

pk : $\mathbf{A} \in \mathbb{Z}_{2q}^{n\times m}$, random

$\mathbf{T} = \mathbf{AS} = q\,\mathbf{Id}$

Sample $\mathbf{y} \leftarrow D_{\mathbb{Z},\sigma}^{m}$, short

$$\xrightarrow{\mathbf{w} = \mathbf{Ay} \in \mathbb{Z}^n}$$

Fiat-Shamir with aborts [Lyu09, Lyu12, DDLL13]

sk : $\mathbf{S} \in \mathbb{Z}_{2q}^{m \times k}$, short

pk : $\mathbf{A} \in \mathbb{Z}_{2q}^{n \times m}$, random

$\mathbf{T} = \mathbf{AS} = q\,\mathbf{Id}$

Sample $\mathbf{y} \leftarrow D_{\mathbb{Z},\sigma}^{m}$, short

$$\xrightarrow{\mathbf{w} = \mathbf{A}\mathbf{y} \in \mathbb{Z}^n}$$

Sample $\mathbf{c} \in \mathbb{Z}^k$, short

$$\xleftarrow{\mathbf{c}}$$

Fiat-Shamir with aborts [Lyu09, Lyu12, DDLL13]

sk : $\mathbf{S} \in \mathbb{Z}_{2q}^{m\times k}$, short

pk : $\mathbf{A} \in \mathbb{Z}_{2q}^{n\times m}$, random

$\mathbf{T} = \mathbf{AS} = q\,\mathbf{Id}$

Sample $\mathbf{y} \leftarrow D_{\mathbb{Z},\sigma}^{m}$, short

$$\xrightarrow{\mathbf{w} = \mathbf{A}\mathbf{y} \in \mathbb{Z}^n}$$

Sample $\mathbf{c} \in \mathbb{Z}^k$, short

$$\xleftarrow{\mathbf{c}}$$

$\mathbf{z} = \mathbf{y} \pm \mathbf{Sc}$

Abort with proba.

$D_{\mathbb{Z},\sigma}^{m}(\mathbf{z})/M \cdot D_{\mathbb{Z},\sigma}^{m}(\mathbf{z} \pm \mathbf{Sc})$

$$\xrightarrow{\mathbf{z}}$$

Rejection probability is such that $\mathbf{z} \sim D_{\mathbb{Z},\sigma}^{m}$ is independent from $\mathbf{S}$.

Fiat-Shamir with aborts [Lyu09, Lyu12, DDLL13]

sk : $\mathbf{S} \in \mathbb{Z}_{2q}^{m \times k}$, short

pk : $\mathbf{A} \in \mathbb{Z}_{2q}^{n \times m}$, random

$\mathbf{T} = \mathbf{AS} = q\,\mathbf{Id}$

Sample $\mathbf{y} \leftarrow D_{\mathbb{Z},\sigma}^m$, short

$\xrightarrow{\mathbf{w} = \mathbf{Ay} \in \mathbb{Z}^n}$

Sample $\mathbf{c} \in \mathbb{Z}^k$, short

$\xleftarrow{\mathbf{c}}$

$\mathbf{z} = \mathbf{y} \pm \mathbf{Sc}$

Abort with proba.

$D_{\mathbb{Z},\sigma}^m(\mathbf{z})/M \cdot D_{\mathbb{Z},\sigma}^m(\mathbf{z} \pm \mathbf{Sc})$

$\xrightarrow{\mathbf{z}}$

Check that

$\|\mathbf{z}\|$ is short, and

$\mathbf{Az} = \mathbf{Tc} + \mathbf{w} \bmod q$

Rejection probability is such that $\mathbf{z} \sim D_{\mathbb{Z},\sigma}^m$ is independent from $\mathbf{S}$.

Hardware Implementation of BLISS

The two costly steps are:

Hardware Implementation of BLISS

The two costly steps are:

- Polynomial multiplications $\mathbf{A}\mathbf{y}$ in $\mathbb{Z}_q[X]/(\Phi_{2n}(X))$
  Already addressed in [PG13] and [RVM$^+$14] (next talk)

Hardware Implementation of BLISS

The two costly steps are:

- Polynomial multiplications $\mathbf{Ay}$ in $\mathbb{Z}_q[X]/(\Phi_{2n}(X))$
  Already addressed in [PG13] and [RVM+14] (next talk)
- Discrete Gaussian Sampling $\mathbf{y} \leftarrow D^m_{\mathbb{Z},\sigma}$ for large σ
  Needs high precision sampling (learning attacks)
  - Long Floating Points Arith. [GPV08]
  - Slow algorithms [DDLL13]
  - Large tables/trees [DG14]

Hardware Implementation of BLISS

The two costly steps are:

- Polynomial multiplications $\mathbf{A}\mathbf{y}$ in $\mathbb{Z}_q[X]/(\Phi_{2n}(X))$
  Already addressed in [PG13] and [RVM$^+$14] (next talk)
- Discrete Gaussian Sampling $\mathbf{y} \leftarrow D_{\mathbb{Z},\sigma}^m$ for large σ
  Needs high precision sampling (learning attacks)
  - Long Floating Points Arith. [GPV08]
  - Slow algorithms [DDLL13]
  - Large tables/trees [DG14]

Most of our contributions deal with hardware implementation of **Discrete Gaussian Sampling**.

Algorithmic contributions to Gaussian samping

We focus on the fastest method to sample Discrete Gaussian using **Cumulative Distribution Tables (CDT)**.

Operation: Binary search accelerated by guide tables

Problem: Naïve implementation would require $42KB$

Algorithmic contributions to Gaussian samping

We focus on the fastest method to sample Discrete Gaussian using **Cumulative Distribution Tables (CDT)**.

Operation: Binary search accelerated by guide tables

Problem: Naïve implementation would require $42KB$

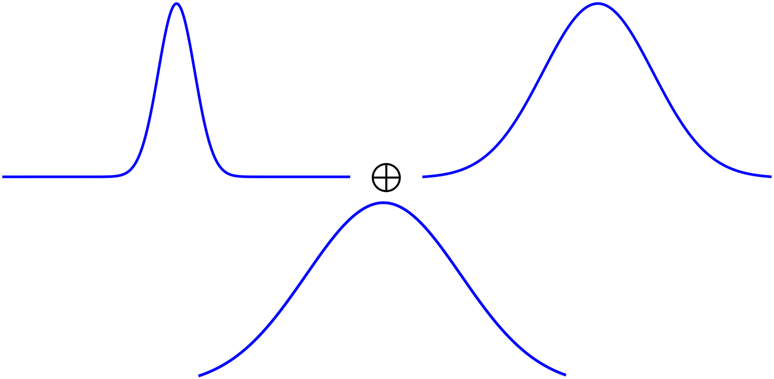
We introduce two new techniques:

- Gaussian sampling by **convolution**
- **Kullback-Leibler-divergence** based security argument

Our algorithm requires a table of $2.1KB$, and is almost as fast

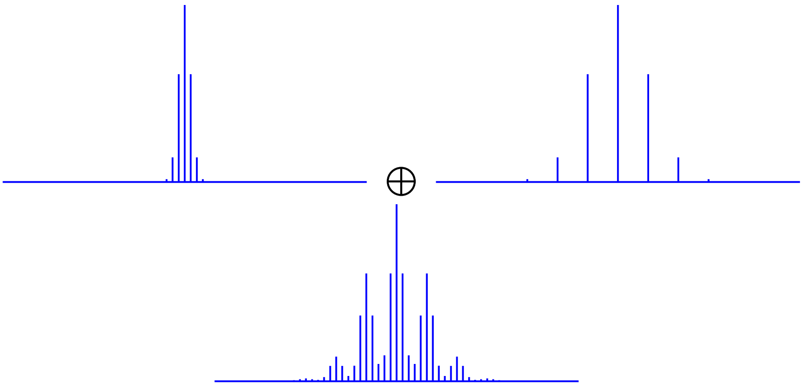
(Discrete) Gaussians convolutions

A convolution of gaussians is a gaussian.



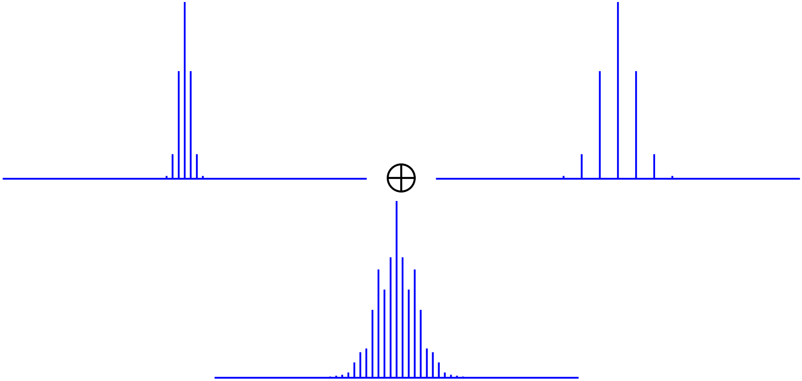
(Discrete) Gaussians convolutions

And what about discrete Gaussians ?



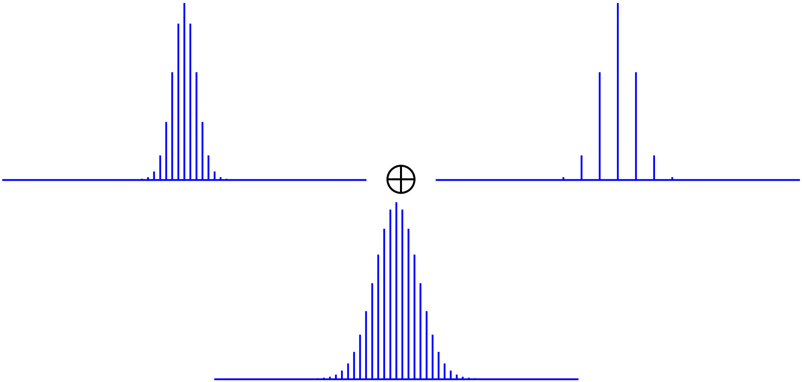
(Discrete) Gaussians convolutions

Well, depending on the parameter...



(Discrete) Gaussians convolutions

It may seem quite Gaussian.



Peikert's Convolution Theorem

Lemma (Adapted from [Pei10])

Let $x_1 \leftarrow D_{\mathbb{Z},\sigma_1}$, $x_2 \leftarrow \cdot D_{\mathbb{Z},\sigma_2}$ *and set* $\sigma_3^{-2} = \sigma_1^{-2} + \sigma_2^{-2}$, *and* $\sigma^2 = \sigma_1^2 + \sigma_2^2$. *If* $\sigma_1 \geqslant \omega(\sqrt{\log n})$ *and* $\sigma_3 \geqslant k \cdot \omega(\sqrt{\log n})$, *then:*

$$x_1 + kx_2 \simeq D_{\mathbb{Z},\sigma}.$$

Application to BLISS-I: We can sample two variables x_1, x_2 of deviation $\sigma' = 19.5$ to obtain $x = x_1 + 11x_2$, of deviation $\sigma = 215$.

Impact: Size of table is reduced from 42KB to 4.5KB.
Running time is less than doubled (binary search is faster for σ').

Kullback-Leibler divergence

Definition (Kullback-Leibler Divergence)

Let $\mathcal{P}$ and $\mathcal{Q}$ be distribution over S. The KL divergence, of $\mathcal{Q}$ from $\mathcal{P}$ is defined as:

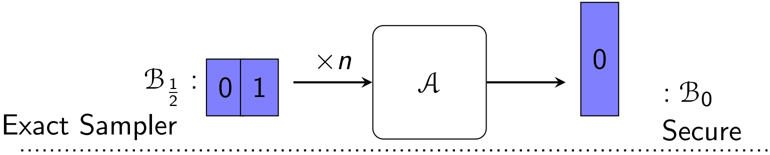
$$D_{\mathbf{KL}}(\mathcal{P}\|\mathcal{Q}) = \sum_{i\in S} \ln\left(\frac{\mathcal{P}(i)}{\mathcal{Q}(i)}\right)\mathcal{P}(i).$$

KL-divergence allows the same arguments as Statistical Distance:

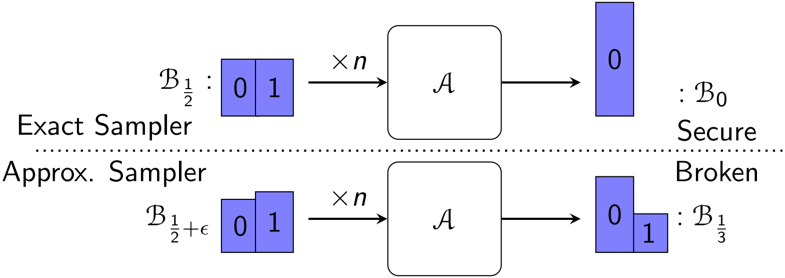
Fact (Addivity and Datta Processing inequality)

- $D_{\mathbf{KL}}(\mathcal{P}_0 \times \mathcal{P}_1 \| \mathcal{Q}_0 \times \mathcal{Q}_1) = D_{\mathbf{KL}}(\mathcal{P}_0\|\mathcal{Q}_0) + D_{\mathbf{KL}}(\mathcal{P}_1\|\mathcal{Q}_1)$
- *for any function* f*:* $D_{\mathbf{KL}}(f(\mathcal{P})\|f(\mathcal{Q})) \leqslant D_{\mathbf{KL}}(\mathcal{P}\|\mathcal{Q})$

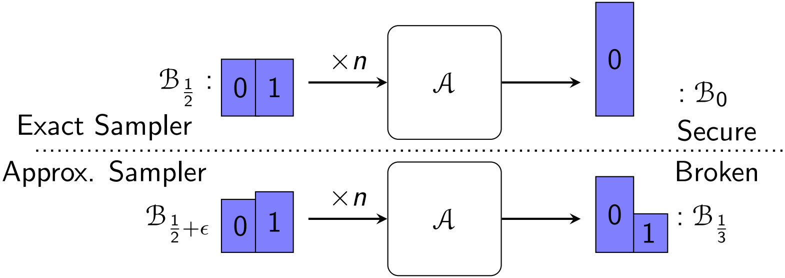
Kullback-Leibler vs. Statistical distance: Example



Kullback-Leibler vs. Statistical distance: Example



Kullback-Leibler vs. Statistical distance: Example

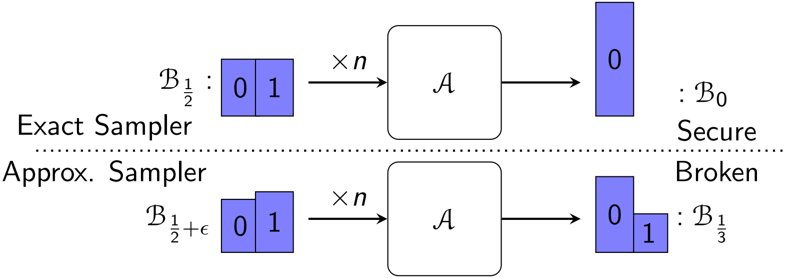


Statistical distance argument

$$\Delta(\mathcal{B}_{\frac{1}{2}}, \mathcal{B}_{\frac{1}{2}+\epsilon}) = \Theta(\epsilon), \qquad \Delta(\mathcal{B}_0, \mathcal{B}_{\frac{1}{3}}) = \Theta(1)$$

$$n \geqslant \Theta(1/\epsilon)$$

Kullback-Leibler vs. Statistical distance: Example



KL-Divergence argument

$$D_{\mathbf{KL}}(\mathcal{B}_{\frac{1}{2}} \| \mathcal{B}_{\frac{1}{2}+\epsilon}) = \Theta(\epsilon^2), \qquad D_{\mathbf{KL}}(\mathcal{B}_0 \| \mathcal{B}_{\frac{1}{3}}) = \Theta(1)$$

$$n \geqslant \Theta(1/\epsilon^2)$$

Kullback-Leibler vs. Statistical distance: Rule of Thumb

Δ Averaged **absolute error**

$$\Delta(\mathcal{P},\mathcal{Q}) = \frac{1}{2}\sum_i |\mathcal{P}(i) - \mathcal{Q}(i)|$$

$D_{\mathbf{KL}}$ Averaged squared **relative error**

$$D_{\mathbf{KL}}(\mathcal{P}\|\mathcal{Q}) \leqslant 2\sum_i \left|\frac{\mathcal{P}(i) - \mathcal{Q}(i)}{\mathcal{P}(i)}\right|^2 \mathcal{P}(i).$$

Limits of KL-divergence

- $D_{\mathbf{KL}}$ is not symmetric
- Can be worse than Δ (e.g. Tailcutting)
- Improvements only for reduction to **Search Problems**

Truncating Cumulative Distribution Table

Full CDT

```
11111100001010111001011011000010110110100011001010100001110011001110111010110000
10110001100110100001010001110011010010101110001010101101010100101111110000011001
01110001001010001100101100010001011100100100101111111101000001111011101011001101
01000000101100011101100111000110101110111111001110001010100011110001010111011011
00100000111111001111111010000000100101001110100001101010110101110111100101010111 1
00001110111010011000100111111100111010111000000101000100011010001001111001000011
00000101111110101101001101100001000000010001011000100010001010111010011111010101 0
00000010000110010011101100010101011110000001100100011001101100110100011101000101
00000000101001100100101111001101010110100101100011111001000000000100100100110011
00000000001011010010101010001111000000000100100000100110001100011011000011010000 0
00000000000010101011111110110011101010101110110000001110110110010010110011111011 1
00000000000000100011110011101111011010001010101000100001001111011001111000000001
00000000000000000110100001001100111010111100000000010110101011110100110101001000
00000000000000000001000010010110000110100110111111001011000111000011001011101001
00000000000000000000001001001101011101001100100011111100000101010011001010010100
00000000000000000000000001000111011001010010111101010100110110111111111101101011
00000000000000000000000000001111000101010001101101000001100110110010000111101 11
00000000000000000000000000000000101100011100001011100101100001100100000000111010
00000000000000000000000000000000011100100001101111010101111010001011110010111
00000000000000000000000000000000000001111111101000000101011011111110111001101
00000000000000000000000000000000000000001111100010100110111010000111001111 10
00000000000000000000000000000000000000000001101001011101011100110111010000 1
00000000000000000000000000000000000000000000000010011011110000111000000111011
00000000000000000000000000000000000000000000000000001100100001000010010010 1
00000000000000000000000000000000000000000000000000000000111000000001010000
00000000000000000000000000000000000000000000000000000000000001101101000110
0000000000000000000000000000000000000000000000000000000000000000000010111000
```

Truncating Cumulative Distribution Table

Truncating left-most zeros

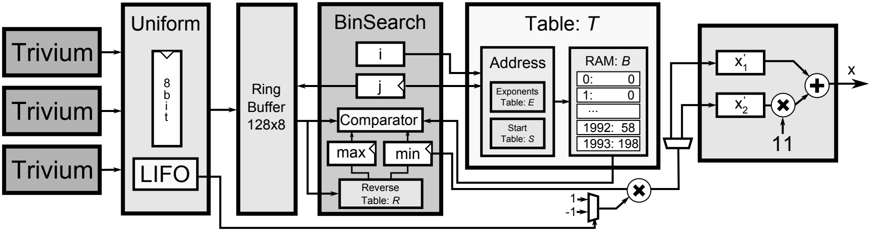
```
111111000010101110010110110000101101101000110010101000011100110011101110101100000
1011000110011010000101000111001101001010111000101010110101010010111111000001100
1
01110001001010001100101100010001011100100100101111111101000001111011101011001101
01000000101100011101100111000110101110111110011100010101000111100010101110111011
00100000111110011111110100000001001010011101000011010101101011101111001010101111
00001110111010011000100111111100111010111000000101000100011010001001111001000011
00000101111110101101001101100001000000010001011000100010001010111010011111010101
0
00000010000110010011101100010101011110000001100100011001101100110100011101000101
00000000101001100100101111001101010110100101100011111001000000000100100100110011
00000000001011010010101010001111000000000100100000100110001100011011000011010000
0
00000000000010101011111101100111010101011101100000011101101100100101100111110111
00000000000000100011110011101111011010001010101000100001001111011001111000000001
00000000000000000110100001001100111010111100000000010110101011110100110101001000
00000000000000000010000100101100001101001101111110010110001110000110010111010
01
00000000000000000000000100100110101110100110010001111110000010101001100101001010
0
00000000000000000000000001000111011001010010111101010100110110111111111101101011
00000000000000000000000000001111000101010001101101000001100110110010000111101
11
00000000000000000000000000000001011000111000010111001011000011001000000000111010
00000000000000000000000000000000000111001000011011110101011110100010111100101
11
00000000000000000000000000000000000000011111111010000001010110111111101110011
01
00000000000000000000000000000000000000000111110001010011011101000011100111110
00000000000000000000000000000000000000000000011010010111010111001101110100001
00000000000000000000000000000000000000000000000000100110111110000111000001110
11
00000000000000000000000000000000000000000000000000000000110010000100001001001
01
00000000000000000000000000000000000000000000000000000000000001110000000010100
00
00000000000000000000000000000000000000000000000000000000000000000001101101000
110
000000000000000000000000000000000000000000000000000000000000000000000000000010111000
```

Truncating Cumulative Distribution Table

Truncating right-most bits using KL-divergence

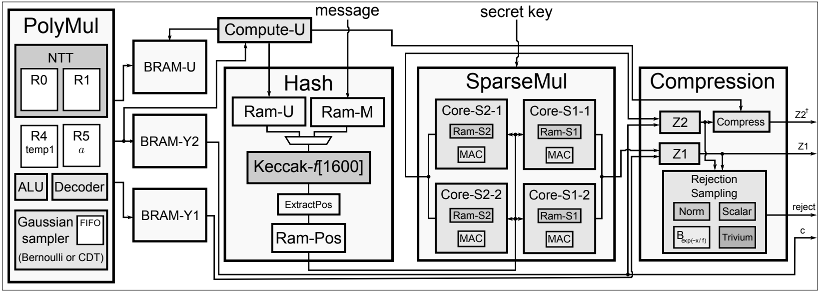
```
1111110000101011100101101100001011011010001100101010000111001100111011101011000 0
10110001100110100001010001110011010010101110001010101101010100101111110000011001
01110001001010001100101100010001011100100100101111111101000001111011101011001101
01000000101100011101100111000110101110111110011100010101000111100010101110111011
00100000111110011111110100000001001010011101000011010101101011101111001010101111
00001110111010011000100111111100111010111000000101000100011010001001111001000011
00000101111101011010011011000010000000100010110001000100010101110100111110101010
00000010000110010011101100010101011110000001100100011001101100110100011101000101
00000000101001100100101111001101010110100101100011111001000000000100100100110011
00000000001011010010101010001111000000000100100000100110001100011011000011010000 0
00000000000010101011111101100111010101011101100000011101101100100101100111110111
00000000000000100011110011101111011010001010101000100001001111011001111000000001
00000000000000000110100001001100111010111100000000010110101011110100110101001000
00000000000000000010000100101100001101001101111110010110001110000110010111010 01
00000000000000000000001001001101011101001100100011111100000101010011001010010100
00000000000000000000000010001110110010100101111010101001101101111111110110101 1
00000000000000000000000000001111000101010001101101000001100110110010000111101 11
00000000000000000000000000000000101100011100001011100101100001100100000000111010
00000000000000000000000000000000000111001000011011110101011110100010111100101 11
00000000000000000000000000000000000000111111111010000001010110111111101110011 01
00000000000000000000000000000000000000000011111000101001101110100001110011111 0
00000000000000000000000000000000000000000000011010010111010111001101110100001
00000000000000000000000000000000000000000000000010011011110000111000000111011
00000000000000000000000000000000000000000000000000001100100001000010010010 1
00000000000000000000000000000000000000000000000000000001100000000101000 0
00000000000000000000000000000000000000000000000000000000000101101000110
00000000000000000000000000000000000000000000000000000000000001011 1000
```

FPGA Implementation of BLISS-I : CDT Sampler



- Ring-buffer to store random numbers generated by Trivium instantiation
- Binary-search component operates on block RAM *B*
- **Biggest challenge:** Critical path of binary search

FPGA Implementation of BLISS-I : Signing



- Number theoretic transform (NTT) multiplier ($\mathbf{a}\mathbf{y}_1$)
- Keccak-1600 hash function
- Fast sparse multiplier ($\mathbf{z}_1 = \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 = \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$)

FPGA Implementation of BLISS-I : Results

Algorithm	LUT	FF	BRAM	DSP	OPs/s
BLISS-1[Sign]	7,491	7,033	7.5	6	7.9k signs/s
BLISS-1[Ver]	5,275	4,488	4.5	3	14,4k verifs/s
CDT-Sampler	928	1,121	1	0	17,4 M samp./s
Bernoulli Sampler	1,178	1,183	0	1	7,4 M samp./s

- Results are given for a 1024-bit message on Spartan6-LX25-3
- High-speed signing and verification
- DT sampler is twice as fast as Bernoulli sampler for similar resource consumption

FPGA Implementation of BLISS-I : Comparision

Algorithm	LUT	FF	BRAM	DSP	OPs/s
BLISS-1[Sign]	7,491	7,033	7.5	6	7,958 signs/sec
BLISS-1[Ver]	5,275	4,488	4.5	3	14,438 signs/sec
GLP-1[Sign/Ver]	6,088	6,804	19.5	4	1,627/7,438
RSA-2048 [Sign]	4190 slices		7	17	79
Curve25519	2783	3592	2	20	2518
ECDSA-256 [Sign/Ver]	32,299 LUT/FF pairs		0	0	139/110

- Implementation faster that RSA and prime curve ECC/ECDSA
- Faster, shorter and more secure than GLP lattice-signature
- Reasonable area consumption

Conclusion

Lattice-based Crypto is ready.

- BLISS compares to standardized signature schemes, in both Software and Hardware
- New geometric analysis will make it **even faster**
  [D14, To appear, see Rump Session]

Time has come for **standardization** of lattice-based cryptography.

- Provide alternative/fallbacks to ECC/RSA
- Mostly **unpatended**
- Motivate more work:
  Comprehensive Cryptanalysis, Improved Algorithms,
  Lightweight Implementation, Side Channel Attacks, . . .

Open Access, Open-Sources

Full and updated paper:

`http://eprint.iacr.org/2014/254`

Software implementation:

`http://bliss.di.ens.fr/`

Hardware implementation:

`http://www.sha.rub.de/research/projects/lattice/`

Thanks !

Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky.
Lattice signatures and bimodal gaussians.
In Ran Canetti and Juan A. Garay, editors, CRYPTO 2013, Part I, volume 8042 of LNCS, pages 40–56, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

Nagarjun C. Dwarakanath and Steven D. Galbraith.
Sampling from discrete Gaussians for lattice-based cryptography on a constrained device.
Applicable Algebra in Engineering, Communication and Computing, pages 1–22, 2014.

Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan.
Trapdoors for hard lattices and new cryptographic constructions.
In Richard E. Ladner and Cynthia Dwork, editors, 40th ACM STOC, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

Vadim Lyubashevsky.
Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures.
In Mitsuru Matsui, editor, ASIACRYPT 2009, volume 5912 of LNCS, pages 598–616, Tokyo, Japan, December 6–10, 2009. Springer, Berlin, Germany.

Vadim Lyubashevsky.
Lattice signatures without trapdoors.
In David Pointcheval and Thomas Johansson, editors, EUROCRYPT 2012, volume 7237 of LNCS, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

Chris Peikert.
An efficient and parallel gaussian sampler for lattices.
In Tal Rabin, editor, CRYPTO 2010, volume 6223 of LNCS, pages 80–97, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Germany.

Thomas Pöppelmann and Tim Güneysu.
Towards practical lattice-based public-key encryption on reconfigurable hardware.
In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, SAC 2013, volume 8282 of LNCS, pages 68–85, Burnaby, BC, Canada, August 14–16, 2013. Springer, Berlin, Germany.

Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede.
Compact Ring-LWE based cryptoprocessor.
In CHES'14, 2014.