Fast Lattice-Based Encryption: Stretching SPRING

Charles Bouillaguet[1] Claire Delaplace[1,2] Pierre-Alain Fouque[2] Paul Kirchner[3]

[1]CFHP team, CRIStAL, Université de Lille, France

[2]EMSEC team, IRISA, Université de Rennes 1, France

[3]École Normale Supérieure, Paris, France

PQCrypto, 2017

Motivation

Goal: Efficient (competitive with AES) PRF/PRG with strong design

Motivation

Goal: Efficient (competitive with AES) PRF/PRG with strong design

Lattice based PRF and PRG

Why?

- Strong design
- Proof of security assuming hard lattices problem
- Post Quantum Security

Issue

- PRF/PRG: deterministic primitives
- Lattice based cryptography: not deterministic

Motivation

Goal: Efficient (competitive with AES) PRF/PRG with strong design

Lattice based PRF and PRG

Why?

- Strong design
- Proof of security assuming hard lattices problem
- Post Quantum Security

Issue

- PRF/PRG: deterministic primitives
- Lattice based cryptography: not deterministic

Solution [BPR12]

- Derandomizing (Ring)-LWE
- Introduce a family of provably secure PRF under (Ring)-LWE assumption

Derandomizing RING-LWE

Polynomial Ring: $R_q = \mathbb{Z}_q[X]/(X^n + 1)$
$q \geq 2$ integer; n power of two

RING Learning With Error (RLWE)

- $s \in R_q$ secret
- e_i random independent errors (drawn from a discrete gaussian distribution)
- Distinguish $(a_i, a_i \cdot s + e_i)$ from uniform over $R_q \times R_q$

RING Learning With Rounding (RLWR)

- $2 \leq p \leq q$
- $S : R_q \to R_p$ rounding function
- $s \in R_q$ secret
- Distinguish $(a_i, S(a_i \cdot s))$ from uniform over $R_q \times R_p$

SPRING family of PRF

Polynomial Ring: $R_q = \mathbb{Z}_q[X]/(X^n + 1)$

Subset Product with Rounding over a RING

- Input: $x = (x_1, \ldots, x_k) \in \{0,1\}^k$
- Secrets: $(a, s_1, \ldots, s_k) \in R_q^* \times (R_q^*)^k$

$$F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} {s_i}^{x_i})$$

Rounding Function

Rounding of each coefficient of a polynomial b:

$$S_{coef}(b_i) = \lfloor p \cdot \bar{b}_i / q \rceil, \quad \bar{b}_i \equiv b_i \mod q$$

p power of two $\Rightarrow S_{coef}(b_i)$: $log_2(p)$ high-order bits of b_i

Parameters choice

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} {s_i}^{x_i})$

[BPR 12]

- q exponential in k
- s_i short

$\Rightarrow$ Proof of Security (Assuming hardness of $RLWE$) but not efficient

Parameters choice

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} s_i^{x_i})$

[BPR 12]

- q exponential in k
- s_i short

$\Rightarrow$ Proof of Security (Assuming hardness of *RLWE*) but not efficient

[BBLPR 14]

- $q = 257$, $n = 128$, $k = 64$, $p = 2$
- Efficient design but no proof of security
- Concrete security analysis required
- Output has a noticeable bias of $1/q$

Dealing with the Bias

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} {s_i}^{x_i})$

Parameters: $q = 257,\ n = 128,\ k = 64,\ p = 2$

SPRING-CRT [BBLPR 14]

Secrets drawn in $R^*_{2 \cdot q}$ instead of R^*_q

- Even modulus: no bias
- Attacks to recover the bias

Dealing with the Bias

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} {s_i}^{x_i})$

Parameters: $q = 257,\ n = 128,\ k = 64,\ p = 2$

SPRING-CRT [BBLPR 14]

Secrets drawn in $R^*_{2 \cdot q}$ instead of R^*_q

- Even modulus: no bias
- Attacks to recover the bias

SPRING-BCH [BBLPR 14]

Apply a BCH code with parameters $[128, 64, 22]$ to the biased output

- Reduce the bias to $1/q^{22} \simeq 2^{-176}$
- Halve the output length

Our Work: SPRING with Rejection Sampling

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} {s_i}^{x_i})$

Parameters: $q = 257,\ n = 128,\ k = 64,\ p \in \{2, 4, 8, 16\}$

Rounding Function

Rounding of each coefficient:

$$b_i \rightarrow \begin{cases} \perp & \text{if } b_i = 256 \\ S_{coef}(b_i) & \text{otherwise} \end{cases}$$

$S_{coef}(b_i)$: $\log_2(p)$ high order bits of b_i

Our Work: SPRING with Rejection Sampling

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} s_i^{x_i})$

Parameters: $q = 257$, $n = 128$, $k = 64$, $p \in \{2, 4, 8, 16\}$

Rounding Function

Rounding of each coefficient:

$$b_i \rightarrow \begin{cases} \perp & \text{if } b_i = 256 \\ S_{coef}(b_i) & \text{otherwise} \end{cases}$$

$S_{coef}(b_i)$: $\log_2(p)$ high order bits of b_i

SPRING-RS

- No bias
- Variable output length

Our Work: SPRING with Rejection Sampling

SPRING: $F(x_1, \ldots, x_k) = S(a \cdot \prod_{i=1}^{k} s_i^{x_i})$

Parameters: $q = 257,\ n = 128,\ k = 64,\ p \in \{2, 4, 8, 16\}$

Rounding Function

Rounding of each coefficient:

$$b_i \rightarrow \begin{cases} \perp & \text{if } b_i = 256 \\ S_{coef}(b_i) & \text{otherwise} \end{cases}$$

$S_{coef}(b_i)$: $\log_2(p)$ high order bits of b_i

SPRING-RS

- No bias
- Variable output length

⇒ Let's use it in counter mode (CTR) as a PRG.

Counter Mode

Using Gray Code Counter

	x	$F(x)$
0	0	$S(a)$
1	1	$S(a \cdot s_1)$
2	11	$S(a \cdot s_1 \cdot s_2)$
3	10	$S(a \cdot s_2)$
4	110	$S(a \cdot s_2 \cdot s_3)$
⋮	⋮	⋮

SPRING CTR

- b Internal state, y output
- **Initialization:** $b \leftarrow a$, $y \leftarrow \perp$
- **At Each Step:**
  - Update x
  - i flipped bit of x
  - $b \leftarrow b \cdot s_i$ if $x_i = 1$
  - $b \leftarrow b \cdot s_i^{-1}$ if $x_i = 0$
  - $y \leftarrow y || S(b)$
- **Return** y

Counter Mode

Using Gray Code Counter

	x	$F(x)$
0	0	$S(a)$
1	1	$S(a \cdot s_1)$
2	11	$S(a \cdot s_1 \cdot s_2)$
3	10	$S(a \cdot s_2)$
4	110	$S(a \cdot s_2 \cdot s_3)$
⋮	⋮	⋮

SPRING CTR

- b Internal state, y output
- **Initialization:** $b \leftarrow a$, $y \leftarrow \perp$
- **At Each Step:**
  - Update x
  - i flipped bit of x
  - $b \leftarrow b \cdot s_i$ if $x_i = 1$
  - $b \leftarrow b \cdot s_i^{-1}$ if $x_i = 0$
  - $y \leftarrow y || S(b)$
- **Return** y

- Only one polynomials product per step
- Require to store the s_i^{-1} polynomials as well

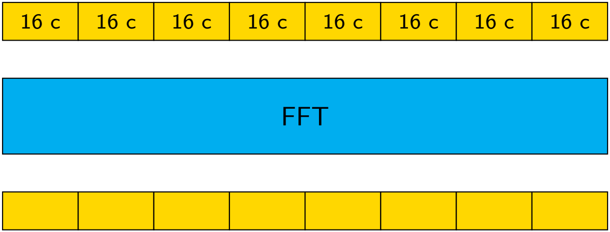
Implementation Tricks

- Store the a, s_i s_i^{-1} in FFT evaluated form a_{ev}, $s_{i,ev}$, $s_{i,ev}^{-1}$
  - Coefficient wise product
  - One FFT per step to get the internal state b
- Use SIMD vector instructions
  - Perform operation in one fell swoop on a vector of data
  - Intel core SSE2 and ARM Neon: 16 vectors of 8 coefficients per polynomials
  - Intel core AVX2: 8 vectors of 16 coefficients per polynomials

SPRING-RS in a Nutshell

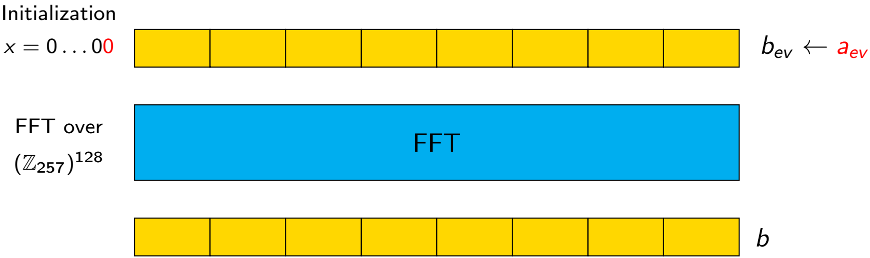
Initialization

$x = 0 \ldots 00$

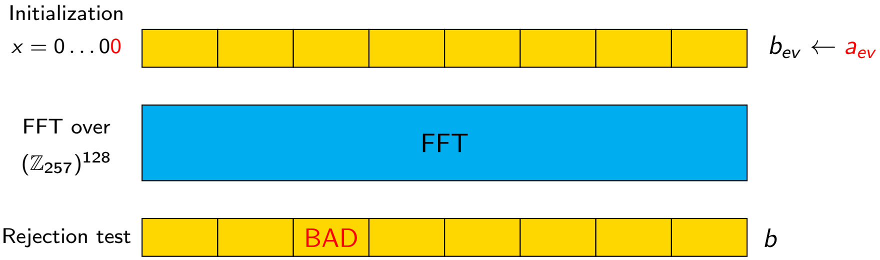


$b_{ev} \leftarrow a_{ev}$

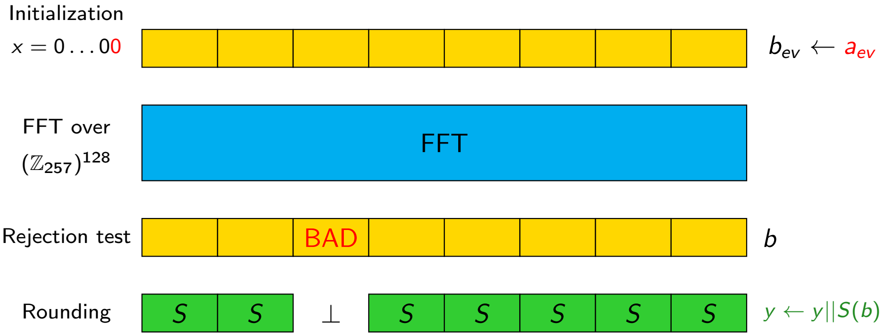
SPRING-RS in a Nutshell



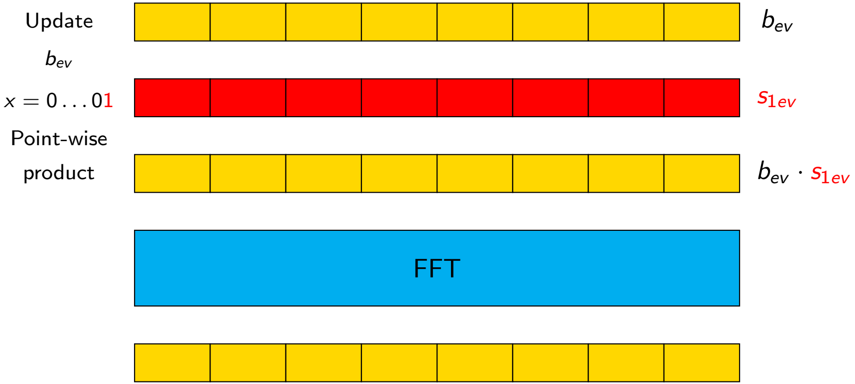
SPRING-RS in a Nutshell



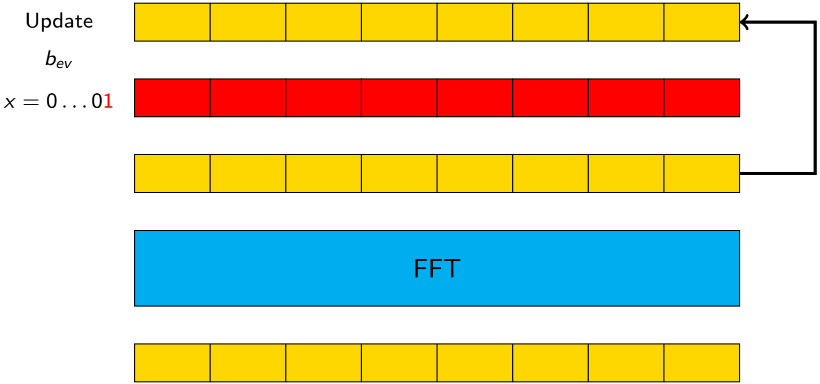
SPRING-RS in a Nutshell



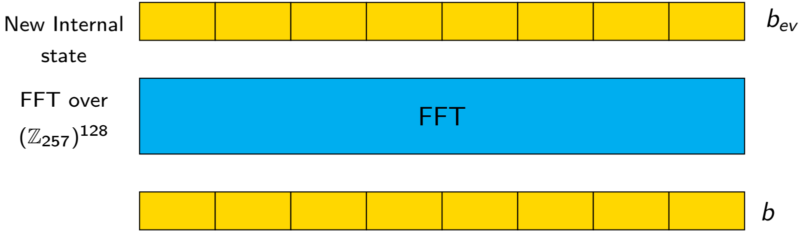
SPRING-RS in a Nutshell



SPRING-RS in a Nutshell



SPRING-RS in a Nutshell



Security Analysis of SPRING

- With BPR12 parameters: Security proof
- With efficient parameters
  - No security proof
  - Resistant against known RLWE attacks

SPRING-RS

- more output bits per coefficient returned
  - More information given to the adversaries
  - Does not seem to weaken the scheme though
- Using rejection sampling
  - Possible side channel leaks
  - Seems hard to recover the exact position of rejected coefficients
  - The adversary would need to solve a polynomial system

Performance

Performance (counter mode) in cycle per output bytes

	SPRING-BCH	SPRING-CRT	AES-CTR	**SPRING-RS**
ARM Cortex A7	445		41	**59**
Core i7 Ivy Bridge	46	23.5	1.3 (NI)	**6**
Core i5 Haswell	19.5 (AVX2)		0.68 (NI)	**2.8** (AVX2)

Other Points of the Paper

Reducing Key Size

- Using an other PRG
- Using a smaller instantiation of SPRING-RS

SPRING-RS PRF

- Return the rounding of the first non-rejected 96 coefficients of the product
- If less than 96 coefficients are returned pad the output with zeros

To Conclude

- This work proposes a version of SPRING using rejection sampling
- Efficient as a PRG when used in counter mode
- No security proof
- Seems to be resistant to known attacks

Open questions

- Is there a security proof for SPRING with efficient parameters?
- Are there other attacks?

To Conclude

- This work proposes a version of SPRING using rejection sampling
- Efficient as a PRG when used in counter mode
- No security proof
- Seems to be resistant to known attacks

Open questions

- Is there a security proof for SPRING with efficient parameters?
- Are there other attacks?

Thank you for your time !