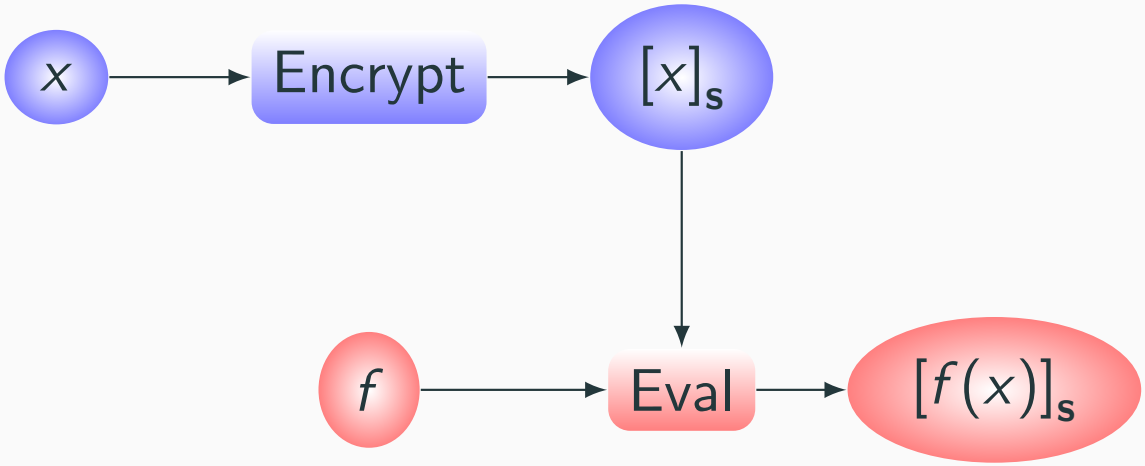
Plug-and-play Sanitization for TFHE

Florian Bourse and Malika Izabachène

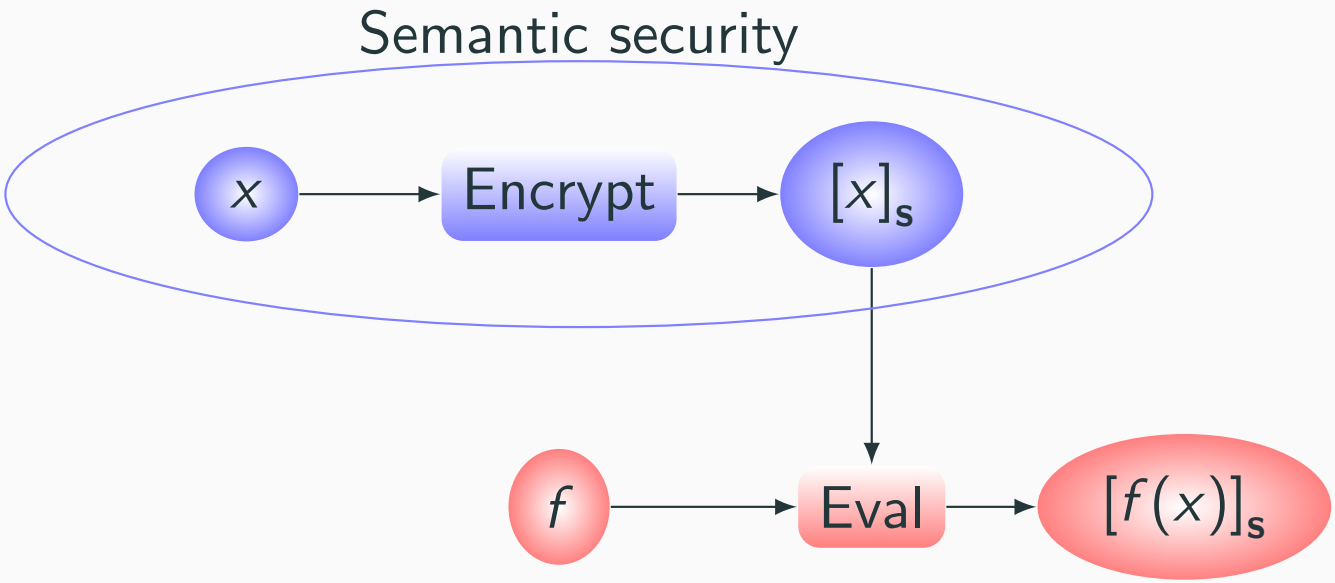
March 24, 2024: FHE.org Toronto 2024 Conference

Motivation :
Circuit privacy and sanitization

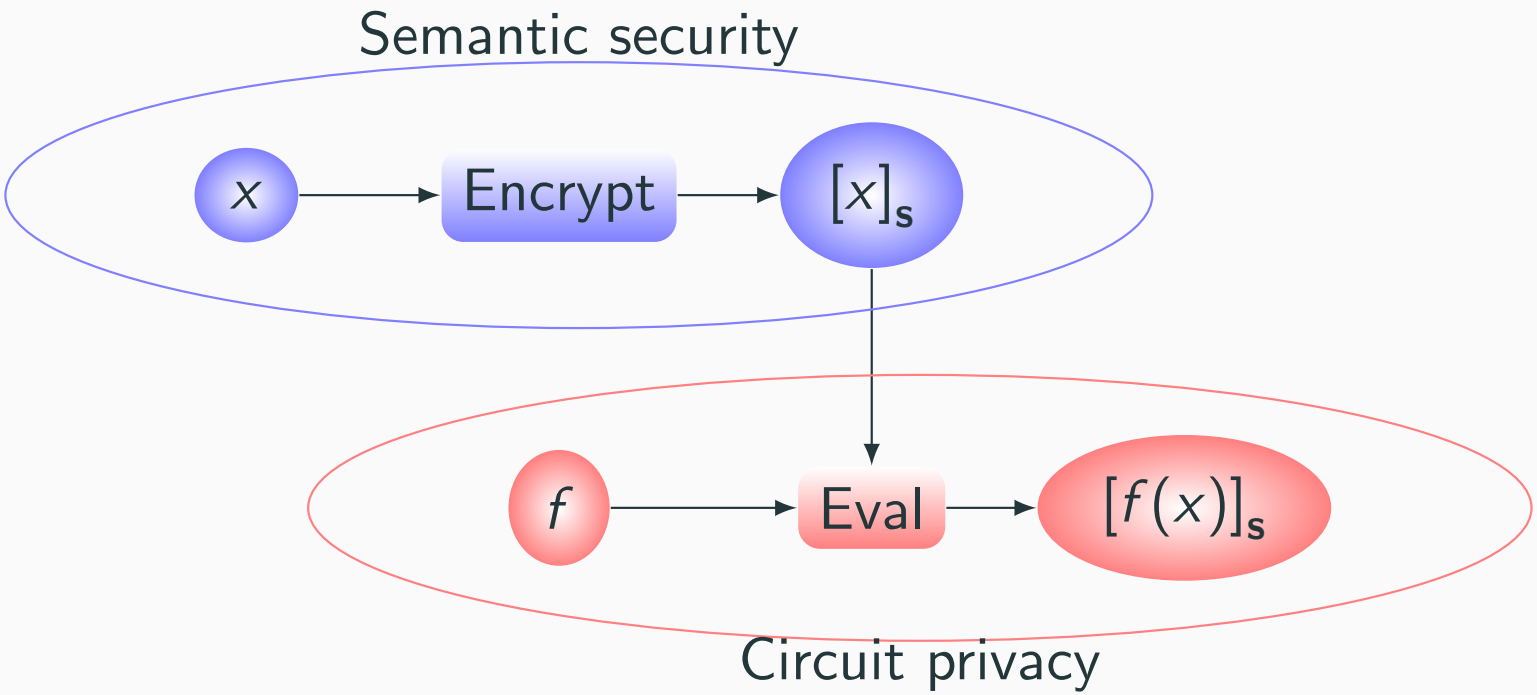
Motivation : Typical FHE application



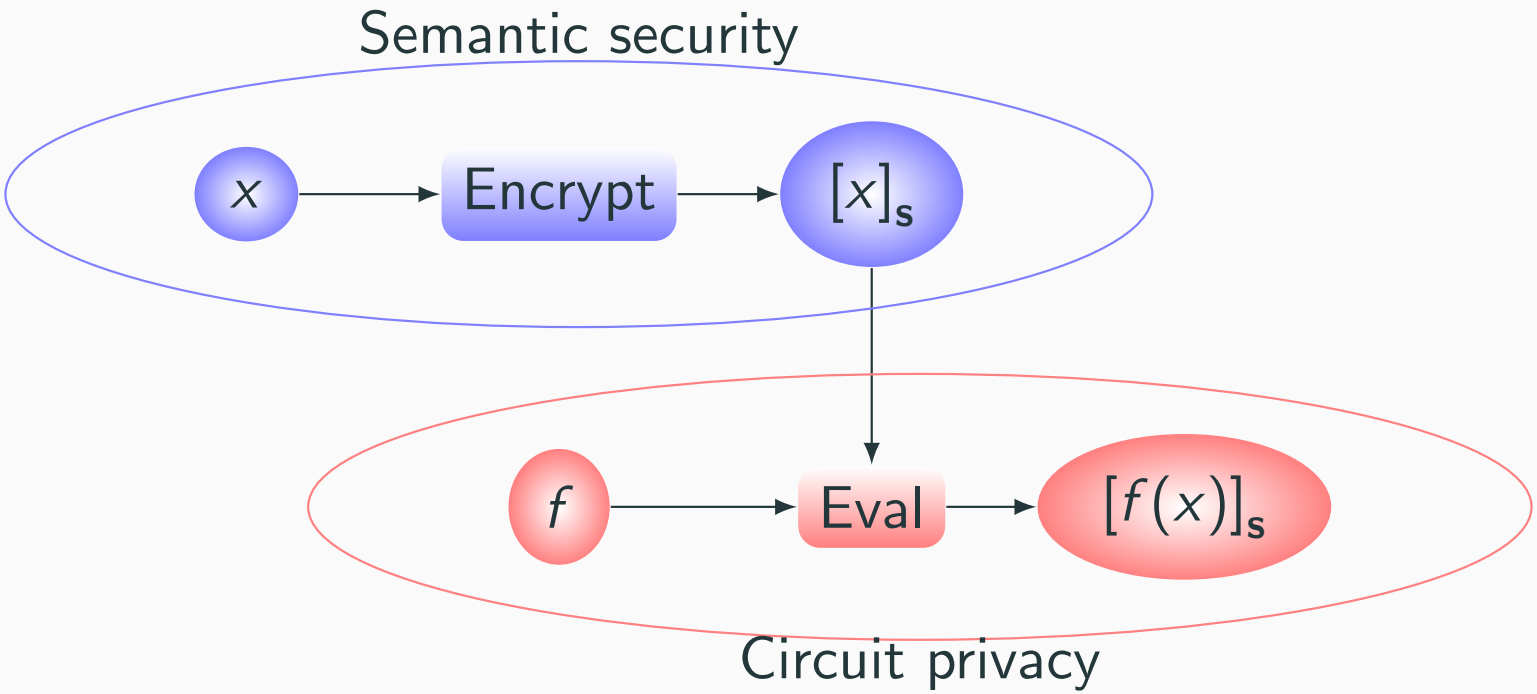
Motivation : Typical FHE application



Motivation : Typical FHE application



Motivation : Typical FHE application



Computational vs statistical.

Indistinguishability-based vs simulation-based.

Warm-up example : Circuit privacy

LWE in dimension $n = 1$, modulo $q = 1024$, secret key $\mathbf{s} = (0)$:

Warm-up example : Circuit privacy

LWE in dimension $n = 1$, modulo $q = 1024$, secret key $\mathbf{s} = (0)$:

$$C_0 = (42, 0) \qquad C_1 = (13, 37)$$

Warm-up example : Circuit privacy

LWE in dimension $n = 1$, modulo $q = 1024$, secret key $\mathbf{s} = (0)$:

$$C_0 = (42, 0) \qquad C_1 = (13, 37)$$

$$f_\beta(x_0, x_1) = \mathsf{mux}(\beta, x_0, x_1) = x_\beta$$

Warm-up example : Circuit privacy

LWE in dimension $n = 1$, modulo $q = 1024$, secret key $\mathbf{s} = (0)$:

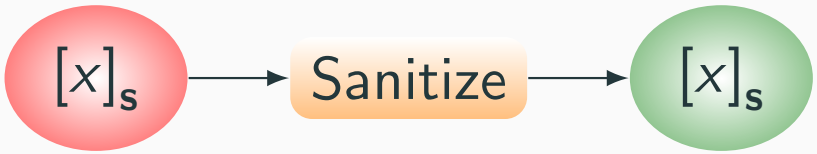
$$C_0 = (42, 0) \qquad C_1 = (13, 37)$$

$$f_\beta(x_0, x_1) = \mathsf{mux}(\beta, x_0, x_1) = x_\beta$$

$\mathsf{Eval}(f_\beta, C_0, C_1) = (42, 0)$, can you find β ?

Sanitization $\Rightarrow$ Circuit privacy

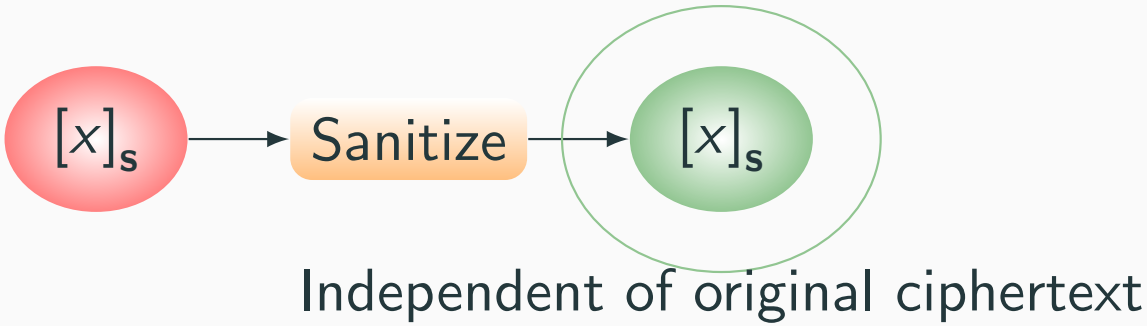
Sanitization :



Sanitization $\Rightarrow$ Circuit privacy

Sanitization :

$$\text{Sanitize}(C_0) \approx \text{Sanitize}(C_1) \text{ if } \text{Decrypt}(\mathbf{s}, C_0) = \text{Decrypt}(\mathbf{s}, C_1)$$



Sanitization $\Rightarrow$ Circuit privacy

Circuit Privacy for a class of function $\mathcal{F}$:

$$O_0 = \mathsf{Eval}(f_1, C_1, \ldots, C_k) \approx \mathsf{Eval}(f_2, C_1, \ldots, C_k) = O_1$$

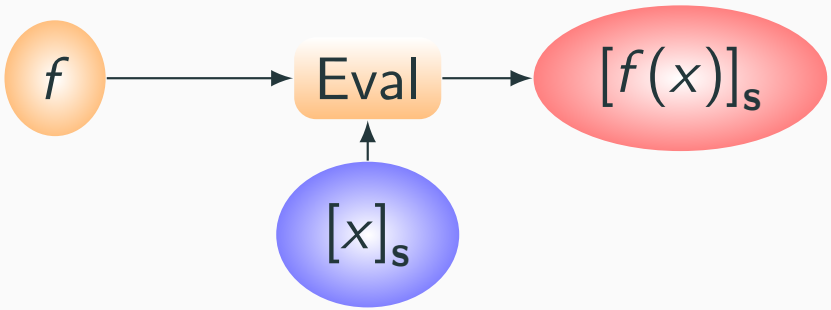
$$\text{if } \mathsf{Decrypt}(\mathbf{s}, O_0) = \mathsf{Decrypt}(\mathbf{s}, O_1)$$

Sanitization $\Rightarrow$ Circuit privacy

Circuit Privacy for a class of function $\mathcal{F}$:

$$O_0 = \mathsf{Eval}(f_1, C_1, \ldots, C_k) \approx \mathsf{Eval}(f_2, C_1, \ldots, C_k) = O_1$$

$$\text{if } \mathsf{Decrypt}(\mathbf{s}, O_0) = \mathsf{Decrypt}(\mathbf{s}, O_1)$$

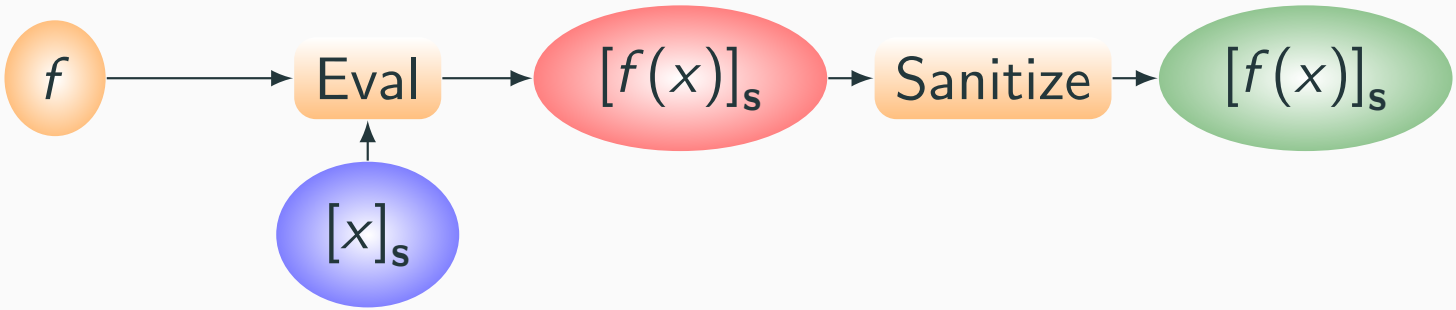


Sanitization $\Rightarrow$ Circuit privacy

Circuit Privacy for a class of function $\mathcal{F}$:

$$O_0 = \mathsf{Eval}(f_1, C_1, \ldots, C_k) \approx \mathsf{Eval}(f_2, C_1, \ldots, C_k) = O_1$$

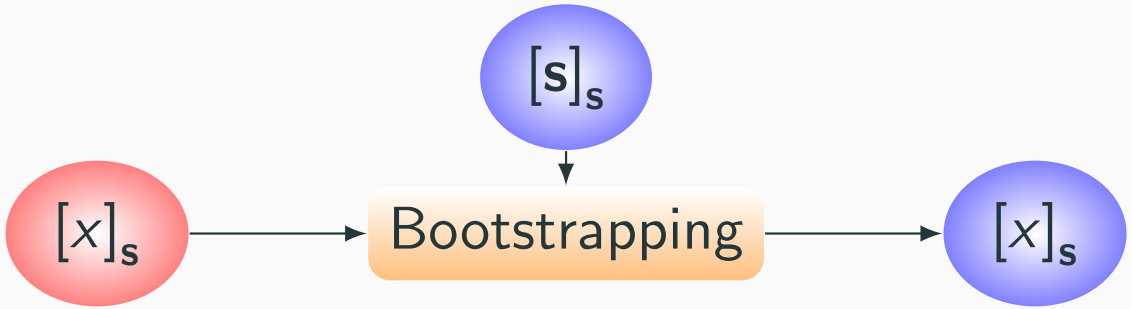
$$\text{if } \mathsf{Decrypt}(\mathbf{s}, O_0) = \mathsf{Decrypt}(\mathbf{s}, O_1)$$



Circuit privacy for bootstrapping $\Rightarrow$ Sanitization

Bootstrapping :

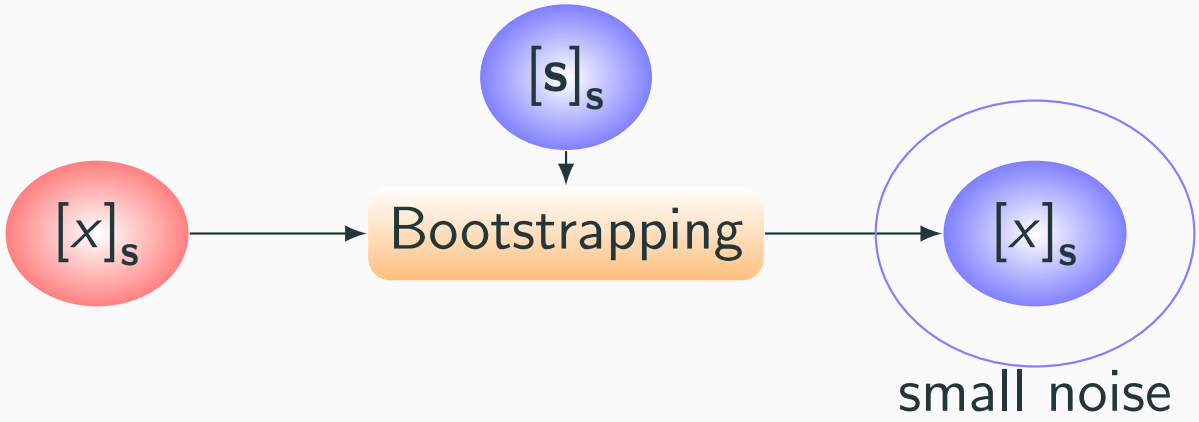
$$\mathsf{Bootstrap}(C) = \mathsf{Eval}(\mathsf{Decrypt}(\cdot, C), [\mathbf{s}]_{\mathbf{s}})$$



Circuit privacy for bootstrapping $\Rightarrow$ Sanitization

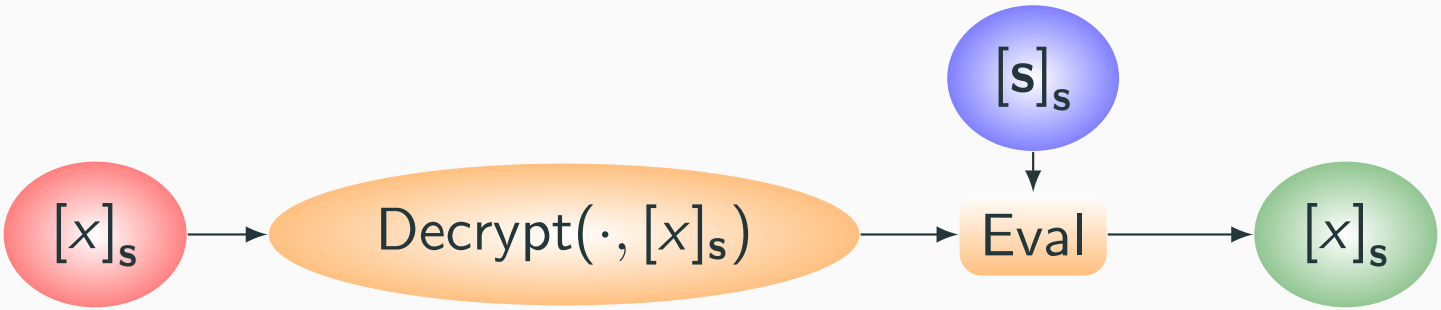
Bootstrapping :

$$\mathsf{Bootstrap}(C) = \mathsf{Eval}(\mathsf{Decrypt}(\cdot, C), [\mathbf{s}]_{\mathbf{s}})$$



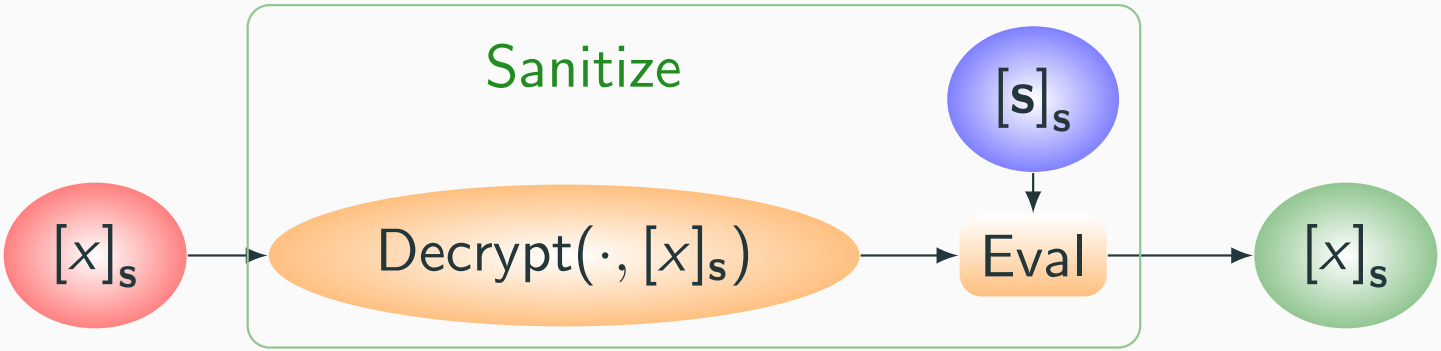
Circuit privacy for bootstrapping $\Rightarrow$ Sanitization

With circuit privacy for the class of functions $(\mathsf{Decrypt}(\cdot, C), [\mathbf{s}]_{\mathbf{s}})_C$:



Circuit privacy for bootstrapping $\Rightarrow$ Sanitization

With circuit privacy for the class of functions $(\mathsf{Decrypt}(\cdot, C), [\mathbf{s}]_{\mathbf{s}})_C$:



Randomization techniques

A common misconception : Circuit privacy is easy [Gen09], noise flooding

The idea is simple: to construct a random encryption ψ' of π from a particular encryption ψ of π, we simply add an encryption of 0 that has a **much larger random "error"** vector than ψ – **super-polynomially larger**, so that the new error vector statistically obliterates all information about ψ's error vector.

A common misconception : Circuit privacy is easy [Gen09], noise flooding

The idea is simple: to construct a random encryption ψ' of π from a particular encryption ψ of π, we simply add an encryption of 0 that has a **much larger random "error"** vector than ψ – **super-polynomially larger**, so that the new error vector statistically obliterates all information about ψ's error vector.

But : $(\mathbf{0}, e)$ is such an encryption of 0 :

$$C_0 + (0, 200) = (42, 200) \qquad C_1 + (0, 200) = (13, 237)$$

A common misconception : Circuit privacy is easy [Gen09], noise flooding

The idea is simple: to construct a random encryption ψ' of π from a particular encryption ψ of π, we simply add an encryption of 0 that has a **much larger random "error"** vector than ψ – **super-polynomially larger**, so that the new error vector statistically obliterates all information about ψ's error vector.

But : $(\mathbf{0}, e)$ is such an encryption of 0 :

$$C_0 + (0, 200) = (42, 200) \qquad C_1 + (0, 200) = (13, 237)$$

Requires additionnal properties !

Soak-and-Spin paradigm [DS16]

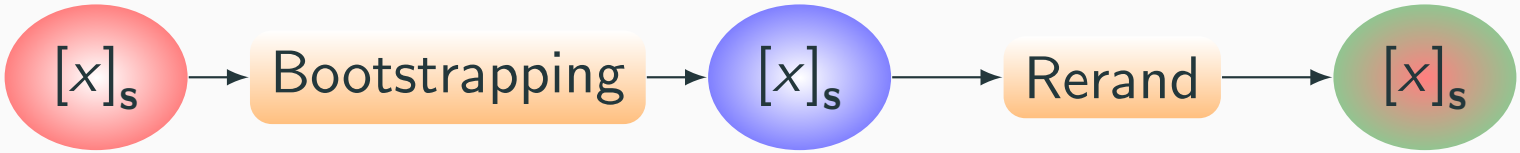
We assume that one may build an efficient randomized algorithm Rerand such that :

$$\Delta(\text{Rerand}(C_0), \text{Rerand}(C_1)) \leq \delta < 1$$

$$\text{if } \text{Decrypt}(\mathbf{s}, C_0) = \text{Decrypt}(\mathbf{s}, C_1)$$

Soak-and-Spin paradigm [DS16]

We assume that one may build an efficient randomized algorithm Rerand such that :



$$\Delta(\mathsf{Rerand}(C_0), \mathsf{Rerand}(C_1)) \leq \delta < 1$$

$$\text{if } \mathsf{Decrypt}(\mathbf{s}, C_0) = \mathsf{Decrypt}(\mathbf{s}, C_1)$$

Repeat for sanitization.

How to choose your Rerand

Previous work:

- Public key: $O(n \log q)$ LWE encryptions of 0.
- Rerand: random subset sum of those (+ noise).
- Statistical security.

How to choose your Rerand

Previous work:

- Public key: $O(n \log q)$ LWE encryptions of 0.
- Rerand: random subset sum of those (+ noise).
- Statistical security.

This work:

- Public key: 1 RLWE encryption of 0: $(a, as + e)$.
- Rerand: 1 RLWE encryption (+ noise):

$$(ra + e', r(as + e) + e'').$$

- Security under decisional RLWE assumption.

How to choose your Rerand

Previous work:

- Public key: $O(n \log q)$ LWE encryptions of 0.
- Rerand: random subset sum of those (+ noise).
- Statistical security.

This work:

- Public key: 1 RLWE encryption of 0: $(a, as + e)$.
- Rerand: 1 RLWE encryption (+ noise):

$$(ra + e', r(as + e) + e'').$$

- Security under decisional RLWE assumption.
- Ciphertext is well distributed even if the secret key is leaked.

Circuit private mux operation [BdMW16]

GSW ciphertexts:

$$[\beta]_{\mathbf{s}} = (a, as + e) + \beta\mathbf{G}, \qquad \beta \in \{0, 1\}$$

Circuit privacy for $(\mathsf{mux}(\cdot, C_0, C_1))_{C_0, C_1}$**:**

Circuit private mux operation [BdMW16]

GSW ciphertexts:

$$[\beta]_{\mathbf{s}} = (a, as + e) + \beta \mathbf{G}, \qquad \beta \in \{0, 1\}$$

Circuit privacy for $(\mathrm{mux}(\cdot, C_0, C_1))_{C_0, C_1}$**:**

$$C_0 + \mathbf{G}^{-1}_{\mathsf{rand}}(C_1 - C_0) \cdot [\beta]_{\mathbf{s}} + \mathsf{Rerand} \approx C_\beta + \mathcal{D}$$

where $\mathbf{G}^{-1}_{\mathsf{rand}}(\cdot)$ is a Gaussian, Rerand is just Gaussian noise and $\mathcal{D}$ is independent of C_0, C_1

Circuit private mux operation [BdMW16]

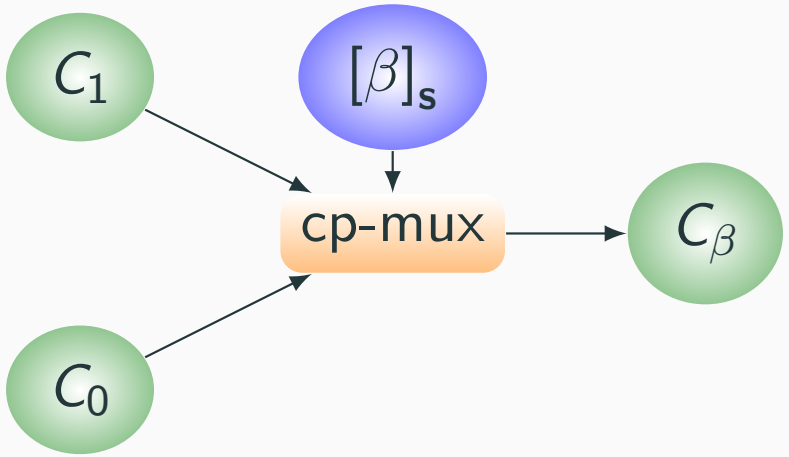
GSW ciphertexts:

$$[\beta]_{\mathbf{s}} = (a, as + e) + \beta \mathbf{G}, \qquad \beta \in \{0, 1\}$$

Circuit privacy for $(\mathrm{mux}(\cdot, C_0, C_1))_{C_0, C_1}$**:**

$$C_0 + \mathbf{G}^{-1}_{\mathsf{rand}}(C_1 - C_0) \cdot [\beta]_{\mathbf{s}} + \mathsf{Rerand} \approx C_\beta + \mathcal{D}$$

where $\mathbf{G}^{-1}_{\mathsf{rand}}(\cdot)$ is a Gaussian, Rerand is just Gaussian noise and $\mathcal{D}$ is independent of C_0, C_1



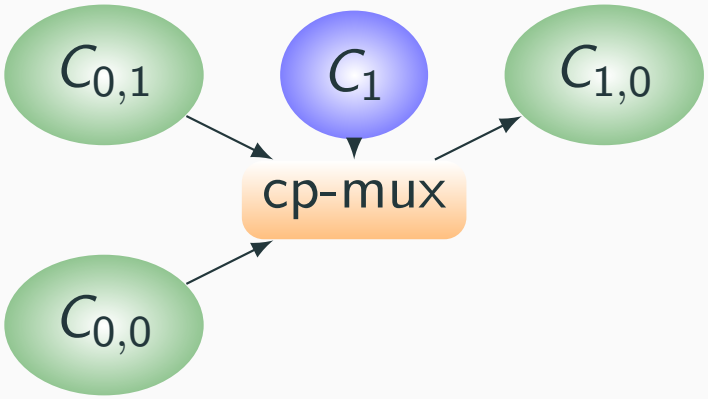
Circuit private BlindRotate

Circuit privacy for branching program evaluation [BdMW16]

Application to branching programs :

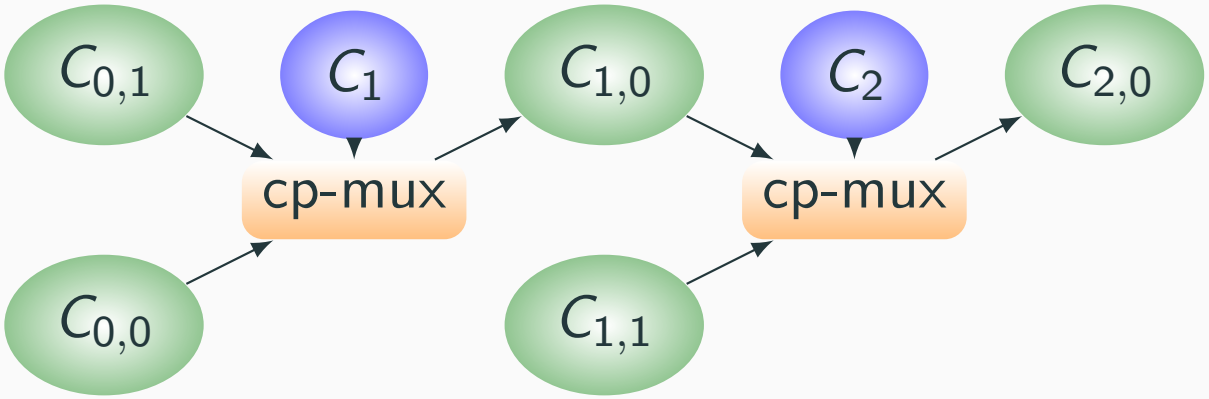
Circuit privacy for branching program evaluation [BdMW16]

Application to branching programs :



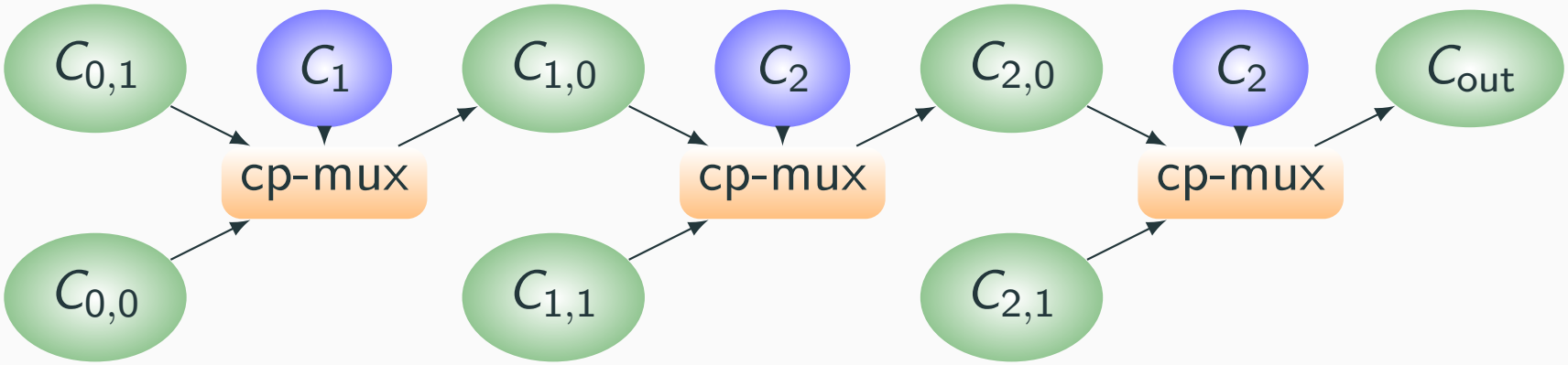
Circuit privacy for branching program evaluation [BdMW16]

Application to branching programs :



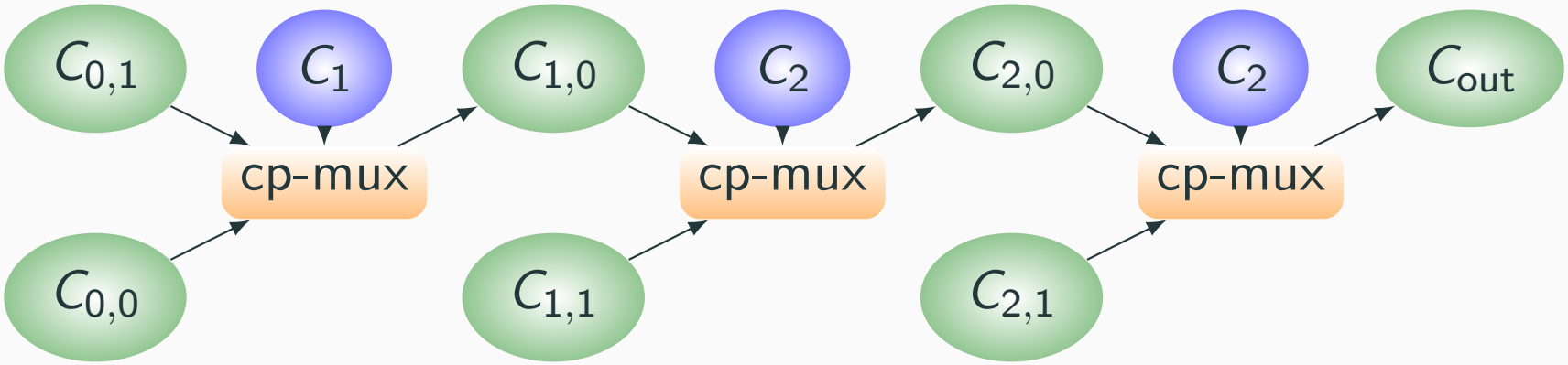
Circuit privacy for branching program evaluation [BdMW16]

Application to branching programs :



Circuit privacy for branching program evaluation [BdMW16]

Application to branching programs :



Proof by induction, adding Rerand noise at each step :

Lemma requires fresh Gaussian samples,

but each step depends on the previous one.

TFHE BlindRotate algorithm

Building block of the Bootstrapping,
computes $\left[X^{b-\mathbf{sa}}\right]_{\mathbf{s}}$ from $(\mathbf{a}, b)$ and $[\mathbf{s}]_{\mathbf{s}}$.

TFHE BlindRotate algorithm

Building block of the Bootstrapping,
computes $\left[X^{b-\mathbf{sa}}\right]_{\mathbf{s}}$ from $(\mathbf{a}, b)$ and $[\mathbf{s}]_{\mathbf{s}}$.

BlindRotate

ACC $= (0, \ldots, 0, \text{testVector} \cdot X^{-b})$
for $i = 1$ **to** $n - 1$
    ACC $=$ cp-mux $([s_i]_{\mathbf{s}}, \text{ACC}, \text{ACC} \times (X^{a_i} - 1))$
Return ACC

testVector allows for functionnal bootstrapping.

TFHE BlindRotate algorithm

Building block of the Bootstrapping,
computes $\left[X^{b-\mathbf{sa}}\right]_{\mathbf{s}}$ from $(\mathbf{a}, b)$ and $[\mathbf{s}]_{\mathbf{s}}$.

BlindRotate

ACC $= (0, \ldots, 0, \text{testVector} \cdot X^{-b})$
for $i = 1$ **to** $n - 1$
    ACC $=$ cp-mux $([s_i]_{\mathbf{s}}, \text{ACC}, \text{ACC} \times (X^{a_i} - 1))$
Return ACC

testVector allows for functionnal bootstrapping.
RGSW ciphertexts, over $\mathcal{R} = \mathbb{Z}/q\mathbb{Z}[X]/(X^N + 1)$,
q and N powers-of-two.

[BdMW16] over rings

$$\mathbf{e}\mathbf{x} + y \approx e'$$

holds over rings for any $\mathbf{v}$, $\mathbf{x} = \mathbf{G}_{rand}^{-1}(\mathbf{v})$.

[BdMW16] over rings

$$\mathbf{ex} + y \approx e'$$

holds over rings for any $\mathbf{v}$, $\mathbf{x} = \mathbf{G}_{rand}^{-1}(\mathbf{v})$.

$$C_0 + \mathbf{G}_{\mathsf{rand}}^{-1}\left(C_1 - C_0\right) \cdot [\beta]_{\mathbf{s}} + \mathsf{Rerand} \approx C_\beta + \mathcal{D}$$

where $\mathbf{G}_{\mathsf{rand}}^{-1}(\cdot)$ is a Gaussian, Rerand is just Gaussian noise and $\mathcal{D}$ is independent of C_0, C_1.

[BdMW16] over rings

$$\mathbf{ex} + y \approx e'$$

holds over rings for any $\mathbf{v}$, $\mathbf{x} = \mathbf{G}_{rand}^{-1}(\mathbf{v})$.

$$C_0 + \mathbf{G}_{\mathsf{rand}}^{-1}\left(C_1 - C_0\right) \cdot [\beta]_{\mathbf{s}} + \mathsf{Rerand} \approx C_\beta + \mathcal{D}$$

where $\mathbf{G}_{\mathsf{rand}}^{-1}(\cdot)$ is a Gaussian, Rerand is just Gaussian noise and $\mathcal{D}$ is independent of C_0, C_1.

Holds, but only for impractical parameters (Leftover Hash Lemma).

Lower bound for required parameters

Leftover Hash Lemma : If $\mathbf{a}, \mathbf{r}, u$ uniform

$$\mathbf{a}, \mathbf{ar} \approx \mathbf{a}, u$$

Lower bound for required parameters

Leftover Hash Lemma : If $\mathbf{a}, \mathbf{r}, u$ uniform

$$\mathbf{a}, \mathbf{ar} \approx \mathbf{a}, u$$

However : in $\mathcal{R}$: $\frac{1}{2}$ invertible, $\frac{1}{2}$ not invertible

Lower bound for required parameters

Leftover Hash Lemma : If $\mathbf{a}, \mathbf{r}, u$ uniform

$$\mathbf{a}, \mathbf{ar} \approx \mathbf{a}, u$$

However : in $\mathcal{R}$: $\frac{1}{2}$ invertible, $\frac{1}{2}$ not invertible

$$\Delta(\mathbf{ar}, u) \geq \frac{1}{2^{\ell}}$$

with ℓ length of $\mathbf{a}$.

Exemple in dimension 1 :

a	0	1	1	0	0	1	1	0
r	0	0	1	1	1	1	0	0
u	0	0	0	0	1	1	1	1
ar	0	0	1	0	0	1	0	0

Our version of cp-mux

Public key : $\mathsf{pk} = (a, as + e)$

$$\text{cp-mux}(C, C_0, C_1) = C_0 + \mathbf{G}^{-1}_{\text{rand}}(C_1 - C_0) \cdot C + \text{Rerand}$$

with $\text{Rerand} = (ra + e', r(as + e) + e'')$

Our version of cp-mux

Public key : $\mathsf{pk} = (a, as + e)$

$$\text{cp-mux}(C, C_0, C_1) = C_0 + \mathbf{G}^{-1}_{\text{rand}}(C_1 - C_0) \cdot C + \text{Rerand}$$

with $\text{Rerand} = (ra + e', r(as + e) + e'')$

$$\text{cp-mux}([\beta]_{\mathbf{s}}, C_0, C_1) \approx C_\beta + \mathcal{D}$$

Our version of cp-mux

Public key : $\mathsf{pk} = (a, as + e)$

$$\text{cp-mux}(C, C_0, C_1) = C_0 + \mathbf{G}^{-1}_{\text{rand}}(C_1 - C_0) \cdot C + \text{Rerand}$$

with $\text{Rerand} = (ra + e', r(as + e) + e'')$

$$\begin{aligned}\text{cp-mux}([\beta]_{\mathbf{s}}, C_0, C_1) &\approx C_\beta + \mathcal{D} \\ &\approx \text{cp-mux}([\beta]_{\mathbf{s}}, C_\beta, C_\beta)\end{aligned}$$

Our version of cp-mux

Public key : $\mathsf{pk} = (a, as + e)$

$$\text{cp-mux}(C, C_0, C_1) = C_0 + \mathbf{G}^{-1}_{\text{rand}}(C_1 - C_0) \cdot C + \text{Rerand}$$

with $\text{Rerand} = (ra + e', r(as + e) + e'')$

$$\begin{aligned} \text{cp-mux}([\beta]_{\mathbf{s}}, C_0, C_1) &\approx C_\beta + \mathcal{D} \\ &\approx \text{cp-mux}([\beta]_{\mathbf{s}}, C_\beta, C_\beta) \\ C_0 + \mathbf{G}^{-1}_{\text{rand}}(C_1 - C_0) \cdot [\beta]_{\mathbf{s}} + \text{Rerand} &\approx C_\beta + \mathbf{G}^{-1}_{\text{rand}}(\mathbf{0}) \cdot [\beta]_{\mathbf{s}} + \text{Rerand} \end{aligned}$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
X^{b-\mathbf{sa}} + & \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_1(X^{a_1}-1)\right)\cdot[s_1]_{\mathbf{s}} \\
& + \dots \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_{n-1}(X^{a_{n-1}}-1)\right)\cdot[s_{n-1}]_{\mathbf{s}} \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_{n}(X^{a_{n}}-1)\right)\cdot[s_{n}]_{\mathbf{s}} \\
& + \mathsf{Rerand}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
X^{b-\mathbf{sa}} + & \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_1(X^{a_1}-1)\right)\cdot[s_1]_{\mathbf{s}} \\
& + \dots \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_{n-1}(X^{a_{n-1}}-1)\right)\cdot[s_{n-1}]_{\mathbf{s}} \\
\mathsf{Rerand} + & \ \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_n(X^{a_n}-1)\right)\cdot[s_n]_{\mathbf{s}}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
X^{b-\mathbf{sa}} + & \\
& + \mathbf{G}^{-1}_{\text{rand}}\left(\text{ACC}_1(X^{a_1}-1)\right)\cdot[s_1]_{\mathbf{s}} \\
& + \dots \\
& + \mathbf{G}^{-1}_{\text{rand}}\left(\text{ACC}_{n-1}(X^{a_{n-1}}-1)\right)\cdot[s_{n-1}]_{\mathbf{s}} \\
\text{Rerand} + & \ \mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{0}\right)\cdot[s_n]_{\mathbf{s}}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
& X^{b-\mathbf{sa}} + \\
& \qquad + \mathbf{G}_{\mathsf{rand}}^{-1}\left(\mathsf{ACC}_1(X^{a_1}-1)\right)\cdot[s_1]_{\mathbf{s}} \\
& \qquad + \dots \\
\mathsf{Rerand} \quad & + \mathbf{G}_{\mathsf{rand}}^{-1}\left(\mathsf{ACC}_{n-1}(X^{a_{n-1}}-1)\right)\cdot[s_{n-1}]_{\mathbf{s}} \\
& + \mathbf{G}_{\mathsf{rand}}^{-1}\left(\mathbf{0}\right)\cdot[s_n]_{\mathbf{s}}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
X^{b-\mathbf{sa}} + & \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_1(X^{a_1}-1)\right)\cdot[s_1]_{\mathbf{s}} \\
& + \dots \\
\mathsf{Rerand} + & \ \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathbf{0}\right)\cdot[s_{n-1}]_{\mathbf{s}} \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathbf{0}\right)\cdot[s_{n}]_{\mathbf{s}}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
& X^{b-\mathbf{sa}} + \\
\mathsf{Rerand} + & \; \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathsf{ACC}_1(X^{a_1}-1)\right) \cdot [s_1]_{\mathbf{s}} \\
& + \dots \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathbf{0}\right) \cdot [s_{n-1}]_{\mathbf{s}} \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}\left(\mathbf{0}\right) \cdot [s_n]_{\mathbf{s}}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
& X^{b-\mathbf{sa}} + \\
\mathsf{Rerand} + & \; \mathbf{G}^{-1}_{\mathsf{rand}}(\mathbf{0}) \cdot [s_1]_{\mathbf{s}} \\
& + \dots \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}(\mathbf{0}) \cdot [s_{n-1}]_{\mathbf{s}} \\
& + \mathbf{G}^{-1}_{\mathsf{rand}}(\mathbf{0}) \cdot [s_n]_{\mathbf{s}}
\end{aligned}
$$

Circuit private BlindRotate

New proof technique : reuse the Rerand

Unfolding the loop :

$$
\begin{aligned}
X^{b-\mathbf{sa}} &+ \mathsf{Rerand} \\
&+ \mathbf{G}_{\mathsf{rand}}^{-1}(\mathbf{0}) \cdot [s_1]_{\mathbf{s}} \\
&+ \dots \\
&+ \mathbf{G}_{\mathsf{rand}}^{-1}(\mathbf{0}) \cdot [s_{n-1}]_{\mathbf{s}} \\
&+ \mathbf{G}_{\mathsf{rand}}^{-1}(\mathbf{0}) \cdot [s_n]_{\mathbf{s}}
\end{aligned}
$$

Building Sanitize

Putting together Sanitize

TFHE Bootstrapping :

ModSwitch

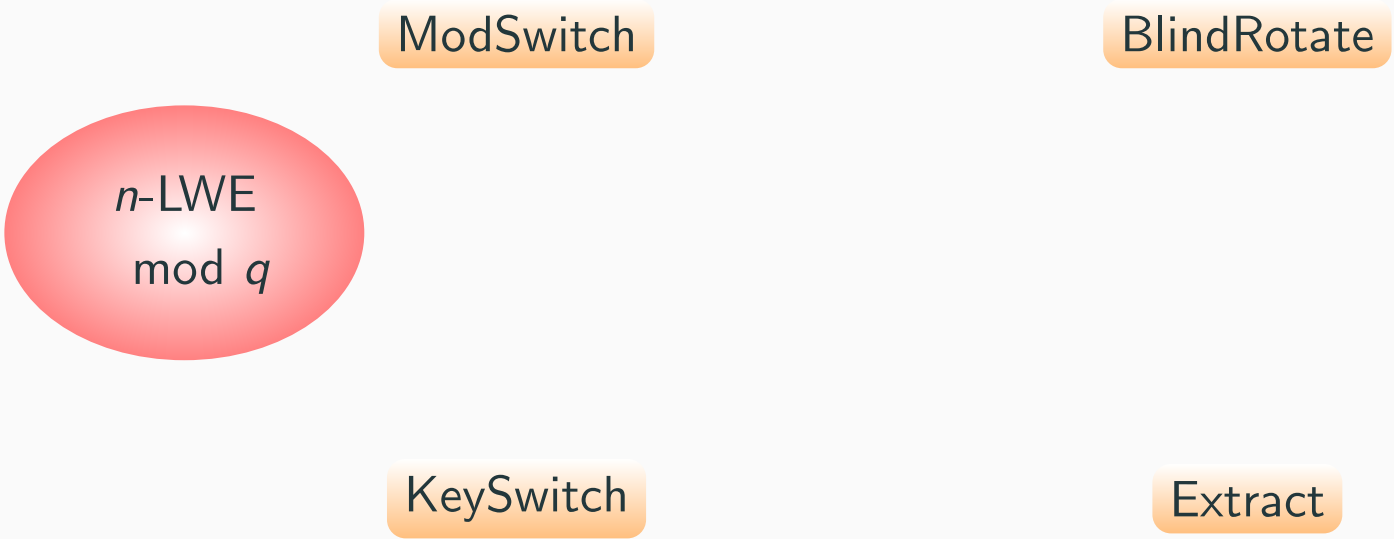
BlindRotate

KeySwitch

Extract

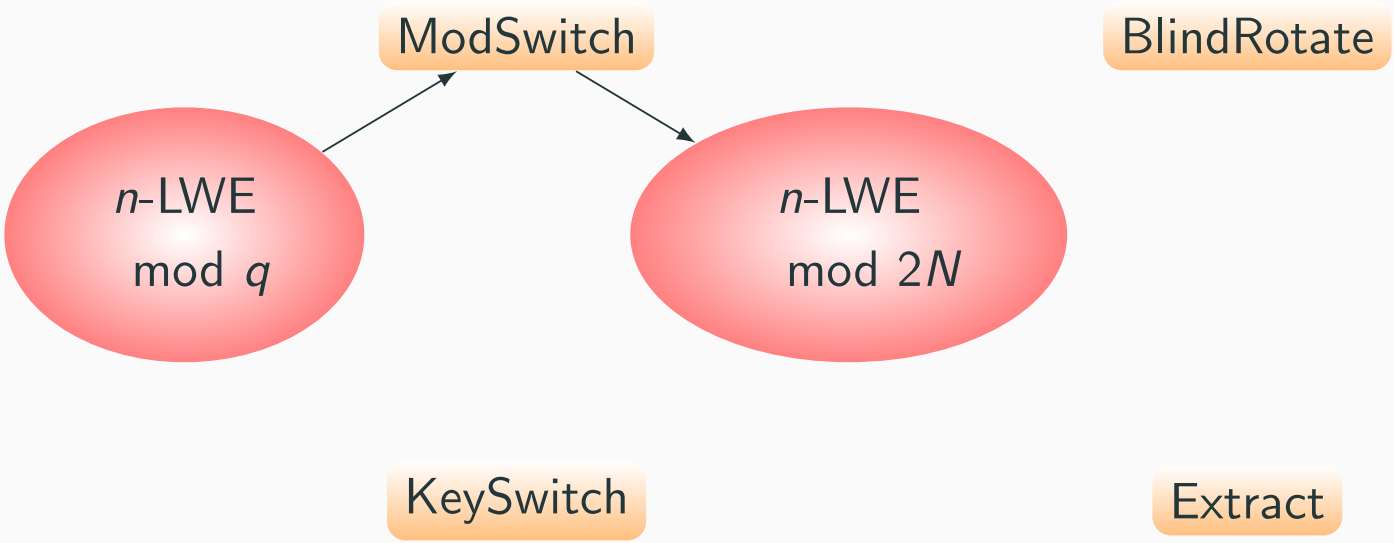
Putting together Sanitize

TFHE Bootstrapping :



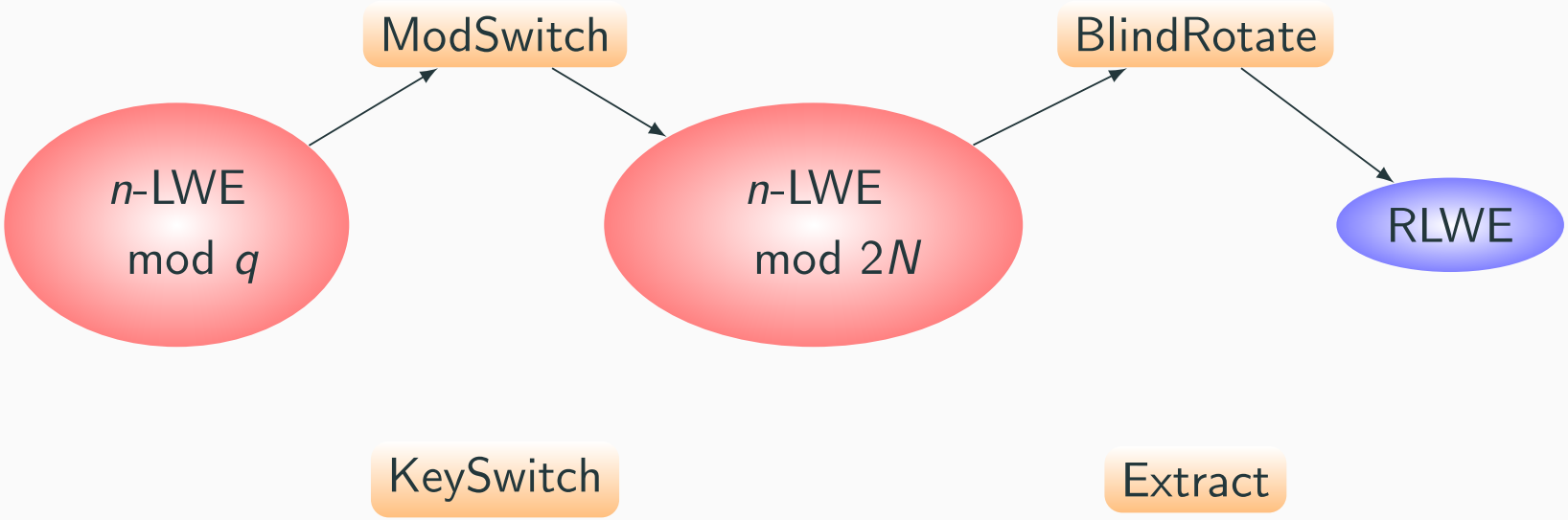
Putting together Sanitize

TFHE Bootstrapping :



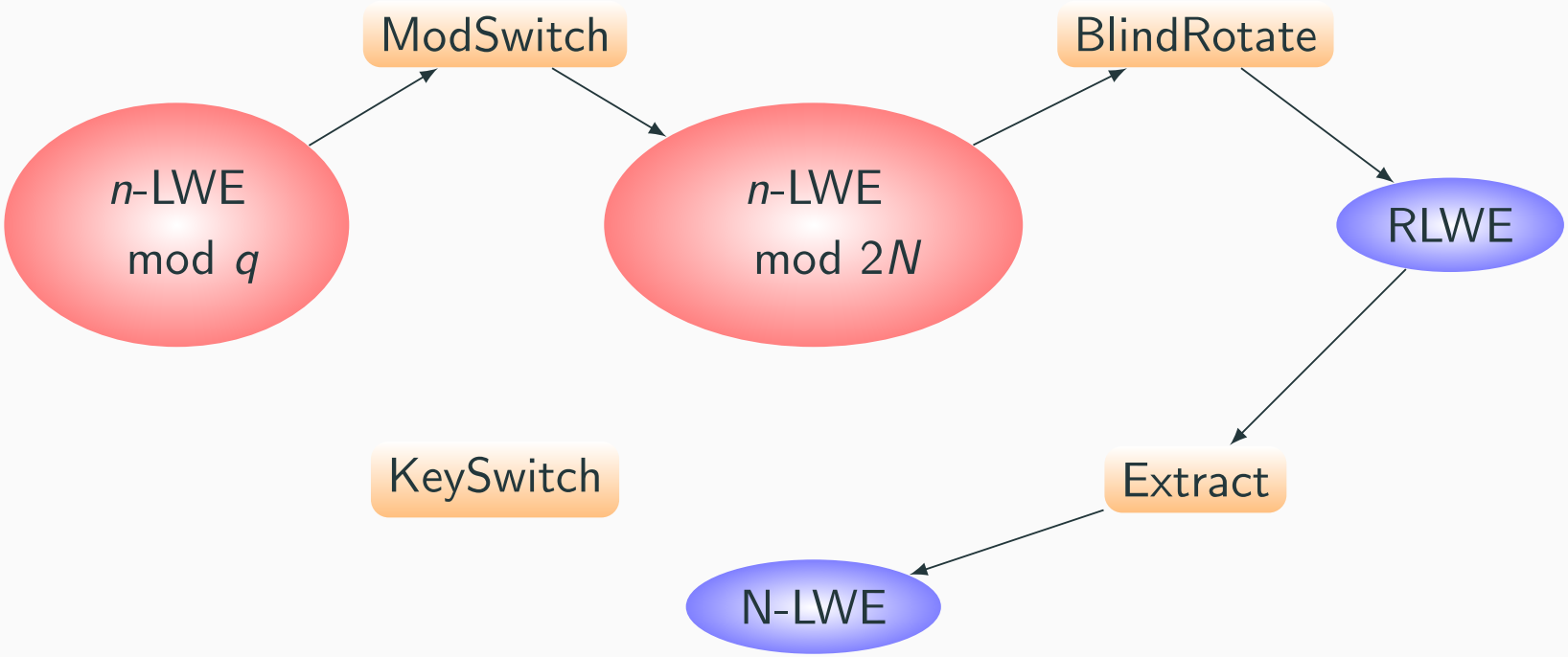
Putting together Sanitize

TFHE Bootstrapping :



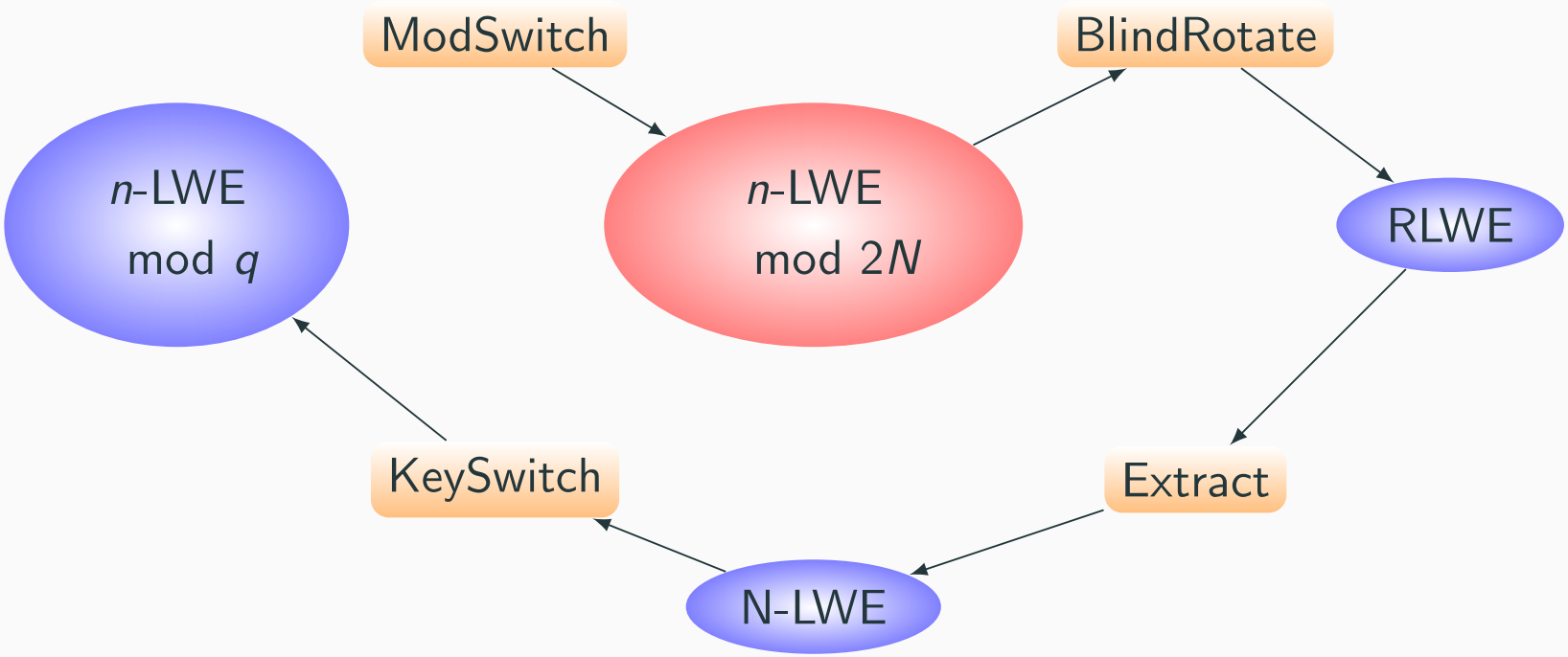
Putting together Sanitize

TFHE Bootstrapping :



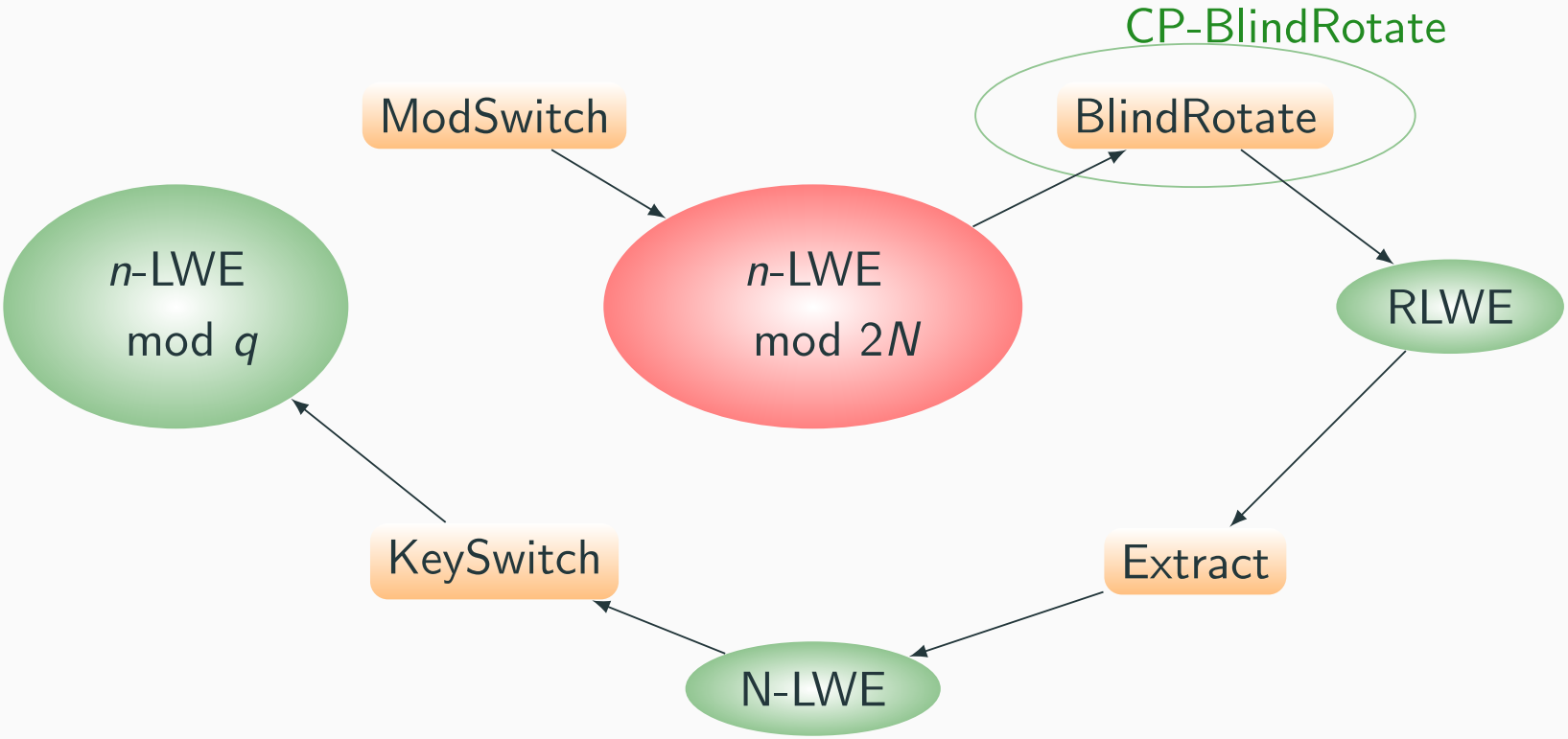
Putting together Sanitize

TFHE Bootstrapping :



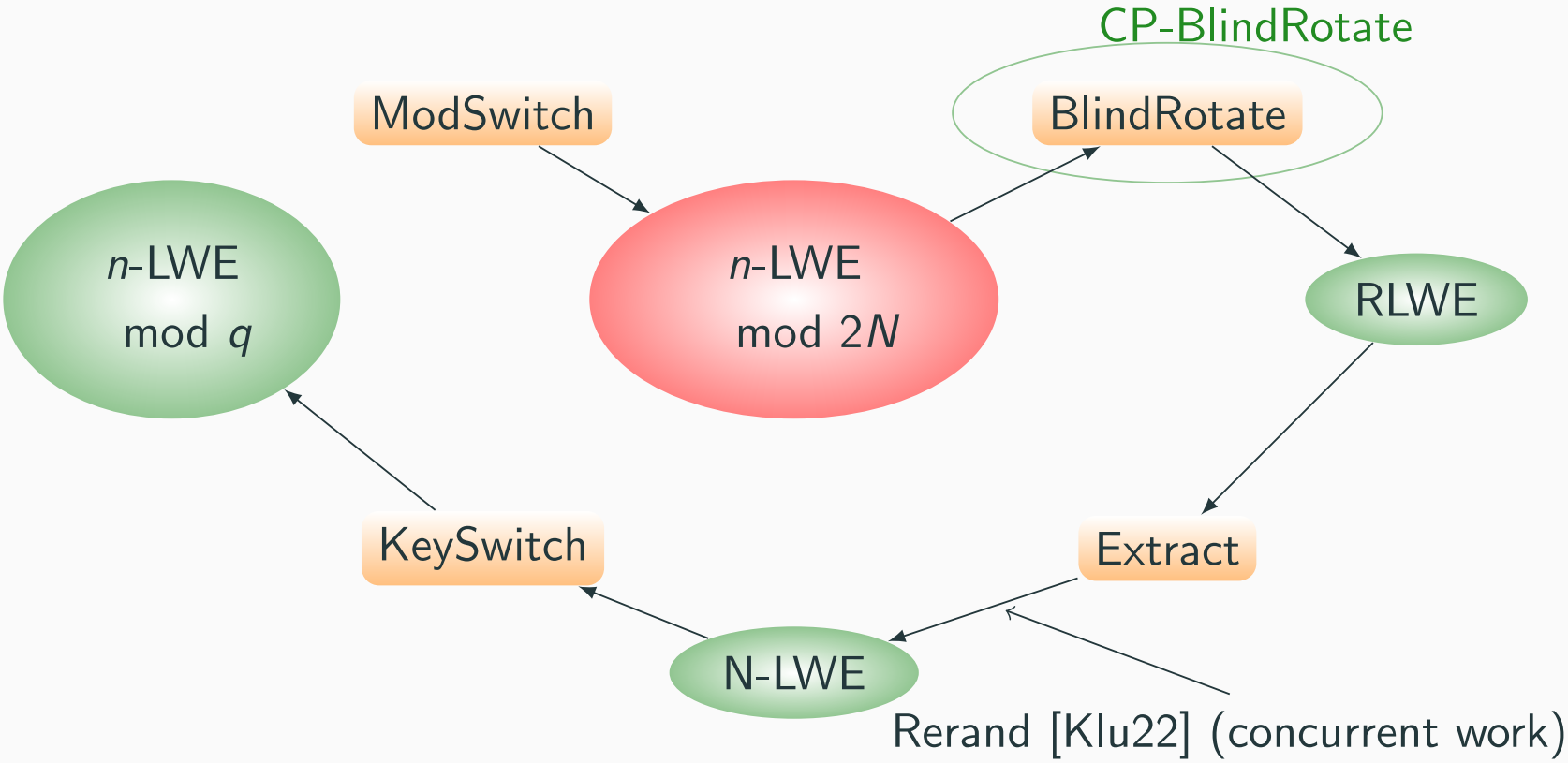
Putting together Sanitize

TFHE Bootstrapping :



Putting together Sanitize

TFHE Bootstrapping :



Empirical results and conclusion

Performance

LWE: $q = 2^{35}, \vartheta_{lwe} = 2^{-27.2}, B_{\mathsf{ks}} = 4, \ell_{\mathsf{ks}} = 7$

RLWE: $N = 1024, \vartheta_{rlwe} = 2^{-67.6}, \vartheta_{pk} = 2^{-65.6}, 2^{-49.5}$

RGSW: $B_{\mathbf{G}} = 32, \ell_{\mathbf{G}} = 7, \vartheta_{\mathbf{G}} = 2^{-39.4}$

Bootstrap -	Sanitize w/ pre-computation	Sanitize w/o pre-computation	Rerand -
0.42 s	0.52 s	7.5 s	1 ms

100 bits of security

Conclusion

- [BdMW16] over rings :
  multi-hop circuit privacy for branching programs for RGSW without circular security

Conclusion

- [BdMW16] over rings :
  multi-hop circuit privacy for branching programs for RGSW without circular security
- Better induction proof :
  reduces parameters and improve efficiency for [BdMW16], even on GSW

Conclusion

- [BdMW16] over rings :
  multi-hop circuit privacy for branching programs for RGSW without circular security
- Better induction proof :
  reduces parameters and improve efficiency for [BdMW16], even on GSW
- Alternative Rerand implementation :
  greatly reduces the size of public key, applicable to RLWE and LWE

Conclusion

- [BdMW16] over rings :
  multi-hop circuit privacy for branching programs for RGSW without circular security
- Better induction proof :
  reduces parameters and improve efficiency for [BdMW16], even on GSW
- Alternative Rerand implementation :
  greatly reduces the size of public key, applicable to RLWE and LWE
- Sanitization with one round of bootstrapping :
  works with power-of-two modulus, adaptable to other modulus

Conclusion

- [BdMW16] over rings :
  multi-hop circuit privacy for branching programs for RGSW without circular security
- Better induction proof :
  reduces parameters and improve efficiency for [BdMW16], even on GSW
- Alternative Rerand implementation :
  greatly reduces the size of public key, applicable to RLWE and LWE
- Sanitization with one round of bootstrapping :
  works with power-of-two modulus, adaptable to other modulus

Thanks for listening !