Package: openfhe (via r-universe)

June 13, 2026

Title R Interface to the OpenFHE Fully Homomorphic Encryption Library

Version 1.5.1

Description Provides an R interface to the OpenFHE C++ library for fully homomorphic encryption (FHE). Supports BFV, BGV, and CKKS schemes for computation on encrypted data, as well as FHEW/TFHE for Boolean circuit evaluation.

License BSD_2_clause + file LICENSE

URL https://bnaras.github.io/openfhe/

BugReports https://github.com/bnaras/fhe/issues

Depends R (>= 4.3.0)

Imports S7, cli, methods

Suggests tinytest, knitr, rmarkdown

VignetteBuilder knitr

LinkingTo cpp11 (>= 0.4.2)

SystemRequirements cmake (>= 3.16), GNU make, C++17

Collate 'openfhe-package.R' 'cpp11.R' 'openfhe-object.R' 'enums.R' 'generics.R' 'element-params.R' 'crypto-parameters.R' 'encoding-params.R' 'params.R' 'params-getters.R' 'context.R' 'cryptocontext-getters.R' 'keys.R' 'plaintext.R' 'plaintext-accessors.R' 'ciphertext.R' 'ciphertext-accessors.R' 'methods-encrypt.R' 'methods-eval.R' 'methods-eval-9111.R' 'methods-ckks.R' 'multiparty.R' 'eval-key-map.R' 'secret-sharing.R' 'interactive-bootstrap.R' 'binfhe.R' 'serialization.R' 'operators.R' 'tolerance.R' 'utils.R' 'zzz.R'

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make

Repository https://bnaras.r-universe.dev

Date/Publication 2026-06-13 05:01:20 UTC

RemoteUrl https://github.com/bnaras/openfhe

RemoteRef HEAD

RemoteSha c960bff486d623896297e73326d9ffeff1a081e5

Contents

openfhe-package *openfhe: R Interface to the OpenFHE Fully Homomorphic Encryption Library*

Description

Provides an R interface to the OpenFHE C++ library for fully homomorphic encryption (FHE). Supports BFV, BGV, and CKKS schemes for computation on encrypted data, as well as FHEW/TFHE for Boolean circuit evaluation.

Author(s)

Maintainer: Balasubramanian Narasimhan <naras@stanford.edu> (ORCID)

Authors:

- Balasubramanian Narasimhan <naras@stanford.edu> (ORCID)

See Also

Useful links:

- https://bnaras.github.io/openfhe
- https://github.com/bnaras/fhe
- Report bugs at https://github.com/bnaras/fhe/issues

BFVParams *BFV Parameters*

Description

Constructor for the BFV scheme's `CCParams` surface. Every argument maps 1:1 to an upstream `CCParams<CryptoContextBFVRNS>::Set*` method whose override is *not* disabled in the BFV specialization. The 13 setters that BFV explicitly disables (`SetScalingTechnique`, `SetFirstModSize`, `SetPRENumHops`, `SetExecutionMode`, ...) are not exposed here; see discovery D013.

Usage

```
BFVParams(
  plaintext_modulus = NULL,
  multiplicative_depth = NULL,
  scaling_mod_size = NULL,
  batch_size = NULL,
  security_level = NULL,
  secret_key_dist = NULL,
  key_switch_technique = NULL,
```

```
  ring_dim = NULL,
  digit_size = NULL,
  num_large_digits = NULL,
  standard_deviation = NULL,
  multiparty_mode = NULL,
  threshold_num_of_parties = NULL,
  multiplication_technique = NULL,
  max_relin_sk_deg = NULL,
  pre_mode = NULL,
  eval_add_count = NULL,
  key_switch_count = NULL,
  encryption_technique = NULL
)
```

Arguments

`plaintext_modulus`
: Integer modulus `t` for the BFV plaintext space. BFV plaintexts are elements of `Z_t[x]/Phi_m(x)` and every homomorphic operation is performed modulo `t`. Must be a prime (or a composite supporting NTT in the batching case) and must be at least large enough that the end-to-end computation does not wrap modulo `t`. Typical values: `65537` (the smallest batching-friendly prime) for small-integer arithmetic, `786433` or larger when the operands or intermediate sums are larger. There is no scheme-level default — the user must supply one.

`multiplicative_depth`
: Integer depth of the multiplication circuit the context must support. BFV and BGV grow ciphertext noise with each multiplication, and the ring modulus `q` is sized by OpenFHE to survive exactly `multiplicative_depth` successive multiplications plus an `EvalAdd` chain between them. Set to the exact depth of your circuit; setting it too small causes correctness failure at decrypt, setting it too large inflates ring dimension and slows every operation. Couples tightly to `security_level` (together they pin `ring_dim`).

`scaling_mod_size`
: Integer bit-size of each intermediate scaling modulus in the RNS decomposition of `q`. Default leaves OpenFHE to choose from its internal tables (typically 60 bits). Override only when you know your depth budget needs a tighter value; the upstream simple-integers example uses the default.

`batch_size`
: Integer number of SIMD-batched plaintext slots. Default is `ring_dim / 2` under the full-packing convention. Set to a smaller power of two when your input vector is short and rotation cost dominates the circuit (the inner-product idiom). Must divide `ring_dim / 2`. Couples to `ring_dim`.

`security_level`
: One of `SecurityLevel$HEStd_128_classic` (default when unset), `HEStd_192_classic`, `HEStd_256_classic`, and their `_quantum` counterparts. Fixes the target hardness assumption and, together with `multiplicative_depth`, determines the minimum ring dimension via the upstream lattice-parameters tables in `core/lattice/stdlatticeparms`

`secret_key_dist`
: One of `SecretKeyDist$GAUSSIAN`, `UNIFORM_TERNARY` (default under classic lattice parameters), or `SPARSE_TERNARY`. Controls the coefficient distribution

of the secret key. Change only for research scenarios that demand a specific distribution; every production vignette uses the default.

`key_switch_technique`
: One of `KeySwitchTechnique$BV` (default for BFV) or `HYBRID`. `BV` produces smaller evaluation keys and is the simpler option; `HYBRID` is relevant only when you are also setting `num_large_digits` and benchmarking rotation cost.

`ring_dim`
: Integer power-of-two lattice ring dimension. Default (`NULL`) asks OpenFHE to compute the minimum ring dimension that satisfies `security_level` at the chosen `multiplicative_depth`. Override only to force a larger value than the auto-selection, typically to reserve headroom for later parameter sweeps.

`digit_size`
: Integer `r` for BV key-switching: the base-`2^r` digit decomposition of the ciphertext during a key-switch. Default `0L` selects the upstream default. Larger values decrease the number of digit multiplications at the cost of larger per-digit noise.

`num_large_digits`
: Integer number of digit groupings in HYBRID key switching. Only meaningful when `key_switch_technique = KeySwitchTechnique$HYBRID`; ignored otherwise. Default `0L` lets OpenFHE pick.

`standard_deviation`
: Numeric standard deviation of the Gaussian error distribution used during key generation and encryption. Default (`NULL`) uses the upstream default `3.19`. Override only in research scenarios.

`multiparty_mode`
: One of `MultipartyMode$FIXED_NOISE_MULTIPARTY` or `NOISE_FLOODING_MULTIPARTY`. Selects between threshold-FHE with fixed noise (no flooding) and noise-flooding-protected threshold-FHE. Both cox-threshold and CVXR-consensus-ADMM vignettes rely on threshold paths; leave at upstream default unless you are intentionally tuning the leakage/performance trade-off.

`threshold_num_of_parties`
: Integer count of parties in an n-of-n threshold protocol. Ignored in non-threshold contexts. The cox-threshold vignette uses 2; the threshold-fhe-5p Python example uses 5.

`multiplication_technique`
: One of `MultiplicationTechnique$BEHZ`, `HPS`, `HPSPOVERQ`, `HPSPOVERQLEVELED`. BFV-specific choice of how the plaintext modulus interacts with the ciphertext modulus during multiply. Default upstream is `HPSPOVERQLEVELED`; override only if you are benchmarking multiplication-path variants.

`max_relin_sk_deg`
: `parity-deferred:` maximum degree of the secret key that can be relinearized. Upstream default is 2. No current vignette or Python example exercises this; the cpp11 binding is in place so a later release can promote it without a recompile.

`pre_mode`
: `parity-deferred:` proxy re-encryption mode (`PREMode$NOT_SET`, `INDCPA`, `FIXED_NOISE_HRA`, `NOISE_FLOODING_HRA`). Only meaningful if the `PRE` feature is enabled on the context. No current vignette uses PRE.

`eval_add_count`
: `parity-deferred:` upstream noise-budget hint: maximum additions between multiplications. Used only by the noise-flooding path. Default 0.

key_switch_count
: parity-deferred: upstream noise-budget hint: maximum key-switch count. Default 0.

encryption_technique
: parity-deferred: BFV-specific encryption variant (EncryptionTechnique$STANDARD or EXTENDED). Default STANDARD.

Value

A BFVParams S7 object.

BGVParams *BGV Parameters*

Description

Constructor for the BGV scheme's CCParams surface. Every argument maps 1:1 to an enabled CCParams<CryptoContextBGVRNS>::Set* method. The 10 BGV-disabled setters (SetEncryptionTechnique, SetMultiplicationTechnique, SetExecutionMode, ...) are not exposed; see discovery D013.

Usage

```
BGVParams(
  plaintext_modulus = NULL,
  multiplicative_depth = NULL,
  scaling_mod_size = NULL,
  scaling_technique = NULL,
  batch_size = NULL,
  first_mod_size = NULL,
  security_level = NULL,
  secret_key_dist = NULL,
  key_switch_technique = NULL,
  ring_dim = NULL,
  digit_size = NULL,
  num_large_digits = NULL,
  standard_deviation = NULL,
  multiparty_mode = NULL,
  threshold_num_of_parties = NULL,
  max_relin_sk_deg = NULL,
  pre_mode = NULL,
  statistical_security = NULL,
  num_adversarial_queries = NULL,
  eval_add_count = NULL,
  key_switch_count = NULL,
  pre_num_hops = NULL
)
```

Arguments

`plaintext_modulus`
: Integer modulus `t` for the BGV plaintext space. See the BFV entry for full semantics; BGV shares the same `Z_t`-valued plaintext model.

`multiplicative_depth`
: Integer multiplicative depth; sizes the ring modulus `q`. See the BFV entry for the full coupling.

`scaling_mod_size`
: Integer bit-size per scaling modulus in the modulus chain.

`scaling_technique`
: One of `ScalingTechnique$FIXEDMANUAL`, `FIXEDAUTO`, `FLEXIBLEAUTO` (upstream default), `FLEXIBLEAUTOEXT`. Selects whether the scheme rescales automatically between multiplications (`*AUTO*`) or leaves it to the caller (`FIXEDMANUAL`). Every vignette uses an auto mode; pick `FIXEDMANUAL` only if you need deterministic control over rescale insertion.

`batch_size`
: Integer SIMD slot count; see the BFV entry.

`first_mod_size`
: Integer bit size of the first (largest) prime in the modulus chain. Default leaves OpenFHE to pick. Override only when tuning the noise budget at the top of the chain.

`security_level`
: See the BFV entry.

`secret_key_dist`
: See the BFV entry.

`key_switch_technique`
: See the BFV entry. BGV accepts both `BV` and `HYBRID`.

`ring_dim`
: See the BFV entry.

`digit_size`
: See the BFV entry.

`num_large_digits`
: See the BFV entry.

`standard_deviation`
: See the BFV entry.

`multiparty_mode`
: See the BFV entry.

`threshold_num_of_parties`
: See the BFV entry.

`max_relin_sk_deg`
: `parity-deferred:` see the BFV entry.

`pre_mode`
: `parity-deferred:` see the BFV entry.

`statistical_security`
: `parity-deferred:` statistical security parameter (bits), used by the noise-flooding decryption path.

`num_adversarial_queries`
: `parity-deferred:` upper bound on the number of adversarial queries the noise-flooding path must survive.

`eval_add_count`
: `parity-deferred:` see the BFV entry.

key_switch_count
: parity-deferred: see the BFV entry.

pre_num_hops
: parity-deferred: maximum number of hops for proxy re-encryption in BGV. Only meaningful if the PRE feature is enabled.

Value

A BGVParams S7 object.

---

bin_bt_key_gen *Generate BinFHE bootstrapping keys*

Description

Generate BinFHE bootstrapping keys

Usage

```
bin_bt_key_gen(ctx, sk, keygen_mode = KeygenMode$SYM_ENCRYPT)
```

Arguments

ctx
: A BinFHE context

sk
: An LWEPrivateKey

keygen_mode
: Integer from KeygenMode; controls whether the bootstrapping keys are generated under symmetric-key encryption (KeygenMode$SYM_ENCRYPT, the default and the C++ default per binfhe-constants.h line 133) or public-key encryption (KeygenMode$PUB_ENCRYPT), matching the BTKeyGen(sk, keyGenMode) overload at binfhecontext.h line 273.

---

bin_decrypt *Decrypt a Binary FHE ciphertext*

Description

Decrypt a Binary FHE ciphertext

Usage

```
bin_decrypt(ctx, sk, ct, p = 4L)
```

Arguments

ctx
: A BinFHE context

sk
: An LWEPrivateKey

ct
: An LWECiphertext

p
: Plaintext modulus (default: 4)

Value

Integer value

`bin_encrypt` *Encrypt a value for Binary FHE*

Description

Encrypt a value for Binary FHE

Usage

```
bin_encrypt(ctx, sk, message, output = 2L, p = 4L, mod = NULL)
```

Arguments

`ctx`	A BinFHE context
`sk`	An LWEPrivateKey
`message`	Integer (0 or 1 for Boolean; larger for integer FHE)
`output`	BINFHE_OUTPUT (default: 2 = BOOTSTRAPPED). Use `BinFHEOutput$LARGE_DIM` together with `mod` for the functional-bootstrapping path.
`p`	Plaintext modulus (default: 4)
`mod`	Optional large ciphertext modulus Q for the LARGE_DIM path. NULL (default) selects the standard small-modulus encrypt.

Value

An LWECiphertext

`bin_fhe_context` *Create a Binary FHE context*

Description

Create a Binary FHE context

Usage

```
bin_fhe_context(
  paramset = BinFHEParamSet$STD128,
  method = BinFHEMethod$GINX,
  arb_func = FALSE,
  log_q = 11L,
  n = 0L,
  time_optimization = FALSE
)
```

Arguments

paramset	A BinFHEParamSet value (default: STD128)
method	A BinFHEMethod value (default: GINX)
arb_func	If TRUE, build a context that supports arbitrary-function bootstrapping (EvalSign, EvalFunc). Selecting any of arb_func, log_q, n, or time_optimization activates the extended overload.
log_q	log2 of the large ciphertext modulus Q used by functional bootstrapping (default 11; use 17 for the eval-sign example).
n	Ring dimension override (0 lets OpenFHE pick).
time_optimization	Enable the GINX time-optimization variant.

Value

A BinFHEContext (stored as OpenFHEObject)

---

bin_key_gen *Generate BinFHE secret key*

Description

Generate BinFHE secret key

Usage

```
bin_key_gen(ctx)
```

Arguments

ctx	A BinFHE context

Value

An LWEPrivateKey

BinFHEMethod *Binary FHE Methods Source: binfhe/binfhe-constants.h enum BINFHE_METHOD*

Description

Binary FHE Methods Source: binfhe/binfhe-constants.h enum BINFHE_METHOD

Usage

```
BinFHEMethod
```

`BinFHEOutput` *Binary FHE Output Types Source: binfhe/binfhe-constants.h enum BINFHE_OUTPUT*

Description

Binary FHE Output Types Source: binfhe/binfhe-constants.h enum BINFHE_OUTPUT

Usage

```
BinFHEOutput
```

`BinFHEParamSet` *Binary FHE Parameter Sets Source: binfhe/binfhe-constants.h enum BINFHE_PARAMSET (sequential from 0)*

Description

Binary FHE Parameter Sets Source: binfhe/binfhe-constants.h enum BINFHE_PARAMSET (sequential from 0)

Usage

```
BinFHEParamSet
```

`BinGate` *Binary Gate Types Source: binfhe/binfhe-constants.h enum BINGATE (sequential from 0)*

Description

Binary Gate Types Source: binfhe/binfhe-constants.h enum BINGATE (sequential from 0)

Usage

```
BinGate
```

ccparams_getters *CCParams getters (all schemes)*

Description

Retrieve a parameter from a BFVParams, BGVParams, or CKKSParams object. Each getter wraps the corresponding upstream CCParams<T>::Get* method and returns its value unchanged.

Usage

```
get_ring_dim(params, ...)

get_plaintext_modulus(params, ...)

get_digit_size(params, ...)

get_composite_degree(params, ...)

get_scheme(params, ...)

get_standard_deviation(params, ...)

get_secret_key_dist(params, ...)

get_max_relin_sk_deg(params, ...)

get_pre_mode(params, ...)

get_multiparty_mode(params, ...)

get_execution_mode(params, ...)

get_decryption_noise_mode(params, ...)

get_noise_estimate(params, ...)

get_desired_precision(params, ...)

get_statistical_security(params, ...)

get_num_adversarial_queries(params, ...)

get_threshold_num_of_parties(params, ...)

get_key_switch_technique(params, ...)

get_scaling_technique(params, ...)
```

```
get_batch_size(params, ...)

get_first_mod_size(params, ...)

get_num_large_digits(params, ...)

get_multiplicative_depth(params, ...)

get_scaling_mod_size(params, ...)

get_security_level(params, ...)

get_eval_add_count(params, ...)

get_key_switch_count(params, ...)

get_encryption_technique(params, ...)

get_multiplication_technique(params, ...)

get_pre_num_hops(params, ...)

get_interactive_boot_compression_level(params, ...)

get_register_word_size(params, ...)

get_ckks_data_type(params, ...)
```

Arguments

params	A BFVParams, BGVParams, or CKKSParams.
...	Reserved for future method-specific arguments (currently unused — all getters take only params).

Details

Per discovery D013, several parameters are "disabled" on specific schemes (their setters throw at runtime). For a disabled scheme/parameter combination the getter returns the default value of the underlying field (typically 0L for uint32_t, 0 for double, or the enum's zero sentinel) rather than throwing. This is benign — the field was never set so its default is all the information available — but it means e.g. calling get_plaintext_modulus(params) on a CKKSParams object returns 0 rather than a meaningful modulus.

Value

The underlying parameter value. Types vary per getter.

Functions

- get_ring_dim: Ring dimension n of the lattice the scheme operates over. For RLWE schemes this is the cyclotomic ring's degree n = phi(m) (Euler's totient of the cyclotomic order) and must be a power of two. The ring dimension is the load-bearing cryptographic parameter — security level and multiplicative depth together pin the minimum n, and most runtime costs scale as O(n log n) per homomorphic operation.
- get_plaintext_modulus: Plaintext modulus t for the BFV and BGV plaintext spaces. BFV/BGV plaintexts are elements of Z_t[x]/Phi_m(x) and every homomorphic operation is performed modulo t. On a CKKSParams object this returns the default field value (0) because CKKS does not use a plaintext modulus — see discovery D013.
- get_digit_size: Digit size r for BV key-switching: the base-2^r digit decomposition of the ciphertext during a key-switch. Larger values decrease the number of digit multiplications at the cost of larger per-digit noise.
- get_composite_degree: Composite-scaling degree for the CKKS COMPOSITESCALINGAUTO/COMPOSITESCALINGMANUA scaling techniques. Only meaningful when scaling_technique = ScalingTechnique$COMPOSITESCALING*; 0 under the default FLEXIBLEAUTO path.
- get_scheme: parity-deferred: integer scheme identifier (see SchemeId for the enum values).
- get_standard_deviation: parity-deferred: standard deviation of the Gaussian error distribution.
- get_secret_key_dist: parity-deferred: secret-key distribution (see SecretKeyDist).
- get_max_relin_sk_deg: parity-deferred: maximum relinearization secret-key degree.
- get_pre_mode: parity-deferred: proxy re-encryption mode (see PREMode).
- get_multiparty_mode: parity-deferred: multiparty mode (see MultipartyMode).
- get_execution_mode: parity-deferred: execution mode (see ExecutionMode).
- get_decryption_noise_mode: parity-deferred: decryption noise mode (see DecryptionNoiseMode).
- get_noise_estimate: parity-deferred: noise-flooding noise estimate (double).
- get_desired_precision: parity-deferred: noise-flooding target precision (double, bits).
- get_statistical_security: parity-deferred: statistical security parameter. Return type is double per header inconsistency; see the Return-type note.
- get_num_adversarial_queries: parity-deferred: upper bound on adversarial queries. Return type is double per header inconsistency; see the Return-type note.
- get_threshold_num_of_parties: parity-deferred: threshold-FHE party count.
- get_key_switch_technique: parity-deferred: key-switching technique (see KeySwitchTechnique).
- get_scaling_technique: parity-deferred: scaling technique (see ScalingTechnique).
- get_batch_size: parity-deferred: SIMD batch size.
- get_first_mod_size: parity-deferred: bit size of the first (largest) prime in the CKKS modulus chain.
- get_num_large_digits: parity-deferred: number of large digits for HYBRID key switching.

- get_multiplicative_depth: parity-deferred: configured multiplicative depth. Note: this is the depth the context was constructed to support, not the current depth budget after operations.
- get_scaling_mod_size: parity-deferred: bit size of each CKKS scaling modulus.
- get_security_level: parity-deferred: target security level (see SecurityLevel).
- get_eval_add_count: parity-deferred: BFV/BGV noise-flooding hint: maximum additions between multiplications.
- get_key_switch_count: parity-deferred: BFV/BGV noise-flooding hint: maximum key-switch count.
- get_encryption_technique: parity-deferred: BFV encryption technique (see EncryptionTechnique).
- get_multiplication_technique: parity-deferred: BFV multiplication technique (see MultiplicationTechnique).
- get_pre_num_hops: parity-deferred: PRE hop count.
- get_interactive_boot_compression_level: parity-deferred: CKKS interactive bootstrap compression level (see CompressionLevel).
- get_register_word_size: parity-deferred: register word size for multi-precision arithmetic.
- get_ckks_data_type: parity-deferred: CKKS data type (see CKKSDataType).

Return-type note

get_statistical_security and get_num_adversarial_queries return double rather than integer. This matches an inconsistency in the upstream Params base class: the corresponding setters take uint32_t but the getters return double. The underlying field is a double — R binds per the header, not per the setter signature.

get_plaintext_modulus returns an R integer on 32-bit-safe values; for moduli that exceed the 32-bit signed integer range the return value is a numeric (double) carrying a losslessly rounded 53-bit integer, per the design.md §7 return-type convention.

Ciphertext *Ciphertext class*

Description

Wraps an encrypted OpenFHE ciphertext. Supports arithmetic operators +, -, * which dispatch to homomorphic operations.

Usage

```
Ciphertext(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

ckks_scaling_factor_bits *CKKS scaling-mod-size in bits*

Description

Reads the real-valued scaling factor at level 0 from `cc` via `get_scaling_factor_real()` and returns its `log2` rounded to the nearest integer. For a CKKS context constructed with `set_scaling_mod_size(50L)` this returns `50L`.

Usage

```
ckks_scaling_factor_bits(cc)
```

Arguments

`cc`	A `CryptoContext` (should be CKKS — meaningful only for CKKS contexts).

Details

Used by the Stage 2 form of `fhe_ckks_tolerance()`.

Value

Integer.

CKKSDataType *CKKS Data Type Source: pke/constants-defs.h enum CKKSDataType*

Description

CKKS Data Type Source: pke/constants-defs.h enum CKKSDataType

Usage

```
CKKSDataType
```

CKKSParams *CKKS Parameters*

Description

Constructor for the CKKS scheme's CCParams surface. Every argument maps 1:1 to an enabled CCParams<CryptoContextCKKSRNS>::Set* method. The 8 CKKS-disabled setters (SetPlaintextModulus, SetEvalAddCount, SetKeySwitchCount, SetEncryptionTechnique, SetMultiplicationTechnique, SetPRENumHops, SetMultipartyMode, SetThresholdNumOfParties) are not exposed; see discovery D013. CKKS is a fixed-point scheme over the complex numbers and has no plaintext modulus; threshold_num_of_parties is currently CKKS-disabled upstream even though the scheme supports threshold variants via a separate code path.

Usage

```
CKKSParams(
  multiplicative_depth = NULL,
  scaling_mod_size = NULL,
  scaling_technique = NULL,
  batch_size = NULL,
  first_mod_size = NULL,
  security_level = NULL,
  secret_key_dist = NULL,
  key_switch_technique = NULL,
  ring_dim = NULL,
  digit_size = NULL,
  num_large_digits = NULL,
  interactive_boot_compression_level = NULL,
  standard_deviation = NULL,
  register_word_size = NULL,
  ckks_data_type = NULL,
  max_relin_sk_deg = NULL,
  pre_mode = NULL,
  execution_mode = NULL,
  decryption_noise_mode = NULL,
  noise_estimate = NULL,
  desired_precision = NULL,
  statistical_security = NULL,
  num_adversarial_queries = NULL,
  composite_degree = NULL
)
```

Arguments

multiplicative_depth
: See the BFV entry.

`scaling_mod_size`
: Integer bit-size of the CKKS rescaling factor (typically 50 or 59 bits). Together with `multiplicative_depth` this pins the modulus chain and therefore the precision budget available at the bottom of the circuit. Upstream default varies by scaling technique; when in doubt use the value the matching Python example uses.

`scaling_technique`
: See the BGV entry. CKKS additionally supports `NORESCALE` (debug-only) and the `COMPOSITESCALING*` modes.

`batch_size`
: See the BFV entry.

`first_mod_size`
: See the BGV entry.

`security_level`
: See the BFV entry.

`secret_key_dist`
: See the BFV entry.

`key_switch_technique`
: See the BFV entry. CKKS typically benefits from `HYBRID`.

`ring_dim`
: See the BFV entry.

`digit_size`
: See the BFV entry.

`num_large_digits`
: See the BFV entry.

`interactive_boot_compression_level`
: One of `CompressionLevel$COMPACT` (2) or `SLACK` (3). Controls the compression level to which the input ciphertext is brought before interactive multi-party bootstrapping. `COMPACT` is more efficient but assumes a stronger security model; `SLACK` is less efficient with weaker assumptions. Used by the `tckks- interactive-mp-bootstrapping*` Python examples.

`standard_deviation`
: See the BFV entry.

`register_word_size`
: Integer word size (in bits) for the register-based multi-precision arithmetic path. Default leaves it to upstream. Used by the simple-real-numbers-composite- scaling Python example.

`ckks_data_type`
: One of `CKKSDataType$REAL` (default) or `COMPLEX`. Selects whether CKKS plaintexts are modeled as real vectors or complex vectors. All current R vignettes use `REAL`.

`max_relin_sk_deg`
: `parity-deferred:` see the BFV entry.

`pre_mode`
: `parity-deferred:` see the BFV entry.

`execution_mode`
: `parity-deferred:` one of `ExecutionMode$EXEC_EVALUATION` (default) or `EXEC_NOISE_ESTIMATION`. The noise-estimation mode is only used by the adversarial-query noise-flooding path.

`decryption_noise_mode`
: `parity-deferred:` one of `DecryptionNoiseMode$FIXED_NOISE_DECRYPT` (default) or `NOISE_FLOODING_DECRYPT`.

`noise_estimate`
: `parity-deferred:` numeric noise estimate used by the noise-flooding path. Paired with `execution_mode = EXEC_NOISE_ESTIMATION`.

`desired_precision`
: `parity-deferred:` numeric target precision (in bits) for the noise-flooding path.

`statistical_security`
: `parity-deferred:` see the BGV entry.

`num_adversarial_queries`
: `parity-deferred:` see the BGV entry.

`composite_degree`
: `parity-deferred:` composite scaling degree for the `COMPOSITESCALING*` scaling techniques. Upstream default 0 means single-prime scaling.

Value

A `CKKSParams` S7 object.

`clear_eval_automorphism_keys` *Clear the EvalAutomorphism key cache*

Description

Companion to `clear_eval_mult_keys()` for the EvalAutomorphism key map (used by rotation and sum operations). `key_tag = NULL` clears everything (same as `clear_fhe_state()`'s `"automorphism_keys"` branch); a character scalar clears only that tag's entries.

Usage

```
clear_eval_automorphism_keys(key_tag = NULL)
```

Arguments

`key_tag`	`NULL` (default) to clear everything, or a character scalar to clear only one tag's entries.

Value

`NULL`, invisibly.

See Also

`clear_fhe_state()`, `clear_eval_mult_keys()`

clear_eval_mult_keys *Clear the EvalMult key cache*

Description

Clears the CryptoContextImpl internal EvalMult key map. With key_tag = NULL (the default), clears the entire cache — equivalent to the no-arg ClearEvalMultKeys() form used by clear_fhe_state(). With a non-NULL key_tag, clears only the entries registered under that tag, preserving everything else. Useful in checkpoint workflows where a single party's keys need to be evicted without wiping the whole registry.

Usage

```
clear_eval_mult_keys(key_tag = NULL)
```

Arguments

key_tag	NULL (default) to clear everything, or a character scalar to clear only one tag's entries.

Value

NULL, invisibly.

See Also

clear_fhe_state(), clear_eval_automorphism_keys()

clear_fhe_state *Clear cached evaluation keys and contexts*

Description

Clear cached evaluation keys and contexts

Usage

```
clear_fhe_state(what = c("mult_keys", "automorphism_keys", "contexts"))
```

Arguments

what	Character vector: subset of "mult_keys", "automorphism_keys", "contexts"

clear_static_maps_and_vectors *Clear OpenFHE static maps and vectors*

Description

Releases memory held in the upstream static eval-key maps and related global caches. Useful in long-running R sessions where repeated context construction accumulates static state. The call is idempotent; calling it when no static state is held is a no-op.

Usage

```
clear_static_maps_and_vectors()
```

Value

`invisible(NULL)`.

compress *Compress a ciphertext to fewer towers*

Description

Truncates the ciphertext's RNS modulus representation to `towers_left` towers and sets its noise-scale-degree to `noise_scale_deg`. Used by the interactive multi-party bootstrapping protocol to shrink a ciphertext before sending it across the network (see `notes/blocks/E-bindings-rewrite/gap-matrix.md` §21 for the bootstrap-side context).

Usage

```
compress(x, ...)
```

Arguments

`x`	A `Ciphertext`.
`...`	Method-specific arguments: `towers_left` (integer, target tower count), `noise_scale_deg` (integer, default `1L`).

Value

A compressed `Ciphertext`.

`CompressionLevel` *Compression Level (interactive multi-party bootstrap) Source: pke/constants-defs.h enum CompressionLevel NOTE: values start at 2, not 0. The header comment explains that compression levels 0 and 1 are not supported and the values are not renumbered.*

Description

Compression Level (interactive multi-party bootstrap) Source: pke/constants-defs.h enum CompressionLevel NOTE: values start at 2, not 0. The header comment explains that compression levels 0 and 1 are not supported and the values are not renumbered.

Usage

```
CompressionLevel
```

`CryptoContext` *Crypto Context*

Description

Crypto Context

Usage

```
CryptoContext(ptr = NULL)
```

Arguments

`ptr`	External pointer (internal use)

`CryptoParameters` *Crypto Parameters (opaque)*

Description

Wraps `std::shared_ptr<CryptoParametersBase<DCRTPoly>>` on the C++ side. Returned by `get_crypto_parameters(cc)` and used as an opaque token for RNS-level parameter accessors such as `get_scaling_factor_real`, `get_key_switch_technique`, etc. This class ships as scaffolding only: the S7 class definition is in place so that the getter wiring can treat it as already defined, but there is no constructor path from R.

Usage

```
CryptoParameters(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

---

decrypt *Decrypt a ciphertext*

Description

Decrypt a ciphertext

Usage

```
decrypt(ct, key, ...)
```

Arguments

ct	A Ciphertext
key	A PrivateKey
...	Additional arguments (cc = CryptoContext)

Value

A Plaintext

DecryptionNoiseMode *Decryption Noise Mode Source: pke/constants-defs.h enum DecryptionNoiseMode*

Description

Decryption Noise Mode Source: pke/constants-defs.h enum DecryptionNoiseMode

Usage

```
DecryptionNoiseMode
```

deserialize_eval_keys *Deserialize evaluation keys from file*

Description

Deserialize evaluation keys from file

Usage

```
deserialize_eval_keys(
  filename,
  type = c("mult", "automorphism", "sum"),
  format = "binary"
)
```

Arguments

filename	Path to serialized file
type	"mult", "automorphism", or "sum"
format	"binary" (default) or "json"

Value

TRUE on success (invisibly)

DistributionType *Distribution Type (lattice parameters) Source: core/lattice/stdlatticeparms.h enum DistributionType*

Description

Distribution Type (lattice parameters) Source: core/lattice/stdlatticeparms.h enum DistributionType

Usage

```
DistributionType
```

ElementParams *Element Parameters (opaque)*

Description

Wraps `std::shared_ptr<typename DCRTPoly::Params>` on the C++ side. Used by the `params` argument of CKKS plaintext factories and returned by `get_element_params()`. This class ships as scaffolding only: no constructor surface other than wrapping an existing external pointer.

Usage

```
ElementParams(ptr = NULL)
```

Arguments

`ptr`	External pointer (internal use)

enable_feature *Enable a feature on a CryptoContext*

Description

Accepts either a single `PKESchemeFeature` enum value or a `uint32_t` bitwise-OR mask of `PKESchemeFeature` values (for surface parity with the C++ `Enable(uint32_t)` overload). The two paths dispatch on the value itself: a mask is detected by being strictly larger than the largest single-feature value (`Feature$SCHEMESWITCH` = `0x80` = `128`) or by having more than one bit set.

Usage

```
enable_feature(cc, ...)
```

Arguments

`cc`	A `CryptoContext`.
`...`	`feature`: a `Feature` value (single) or an integer mask (e.g. `Feature$PKE` + `Feature$KEYSWITCH` + `Feature$LEVELEDSHE`, i.e. `11L`).

Value

`cc` invisibly.

EncodingParams *Encoding Parameters (opaque)*

Description

Wraps `std::shared_ptr<EncodingParamsImpl>` on the C++ side. Returned by `get_encoding_params(cc)` and by `Plaintext::GetEncodingParams()` once the corresponding Plaintext accessor lands. This class ships as scaffolding only: the S7 class definition is in place so that the getter wiring can treat it as already defined, but there is no constructor path from R.

Usage

```
EncodingParams(ptr = NULL)
```

Arguments

`ptr`	External pointer (internal use)

encrypt *Encrypt a plaintext*

Description

`encrypt` dispatches on both the key type and the plaintext. The `(PublicKey, Plaintext)` method performs public-key encryption and is the canonical path used by every vignette. The `(PrivateKey, Plaintext)` method performs symmetric / secret-key encryption using the private key directly; it is useful in protocols that want the secret key to serve as both encryption and decryption key (e.g. one-party tests, single-user benchmarks).

Usage

```
encrypt(key, pt, ...)
```

Arguments

`key`	A `PublicKey` or `PrivateKey`.
`pt`	A `Plaintext`.
`...`	Additional arguments (`cc` = `CryptoContext` is required).

Value

A `Ciphertext`.

EncryptionTechnique *Encryption Technique Source: pke/constants-defs.h enum EncryptionTechnique*

Description

Encryption Technique Source: pke/constants-defs.h enum EncryptionTechnique

Usage

```
EncryptionTechnique
```

eval_add *Homomorphic addition*

Description

Homomorphic addition

Usage

```
eval_add(x, y, ...)
```

Arguments

x, y	Ciphertext, Plaintext, or numeric values
...	Method-specific arguments

Value

A Ciphertext

eval_add_in_place *Homomorphic addition in place*

Description

Modifies the first argument in place to hold the result of adding the second argument. Avoids allocating a new Ciphertext, which matters in tight loops or when ciphertext memory footprint is a concern.

Usage

```
eval_add_in_place(x, ...)
```

Arguments

x	A Ciphertext (modified in place).
...	Method-specific arguments: y — a Ciphertext, Plaintext, or numeric scalar to add into x.

Value

x invisibly.

eval_add_mutable *Homomorphic add, mutable variant*

Description

The Mutable* family exists so that operations that can safely mutate their inputs during evaluation (e.g. a temporary intermediate in a longer circuit) can do so without forcing a defensive copy. Semantically equivalent to the non-Mutable counterpart from the user's perspective; the difference is performance under specific workloads.

Usage

```
eval_add_mutable(x, ...)
```

Arguments

x	A Ciphertext (may be modified internally).
...	Method-specific arguments: y — a Ciphertext (may be modified internally).

Value

A new Ciphertext holding x + y.

eval_at_index_key_gen *Generate at-index (rotation) keys for a secret key*

Description

Standalone wrapper around the `CryptoContext::EvalAtIndexKeyGen(privateKey, indexList)` C++ method. Functionally identical to `eval_rotate_key_gen()` (the C++ `EvalRotateKeyGen` is a thin inline wrapper around `EvalAtIndexKeyGen`) and provided for surface parity with the C++ header and openfhe-python, both of which bind the two names separately.

Usage

```
eval_at_index_key_gen(cc, sk, index_list)
```

Arguments

`cc`	A `CryptoContext`.
`sk`	A `PrivateKey` whose tag will be used to key the generated rotation keys in the cc's internal registry.
`index_list`	Integer vector of rotation indices.

Value

`NULL`, invisibly.

eval_automorphism *Apply an automorphism to a ciphertext*

Description

Evaluates the automorphism at the given index on `ct` using the eval-key map returned by `eval_automorphism_key_gen()`. The result is a new ciphertext whose decrypted slot vector is a permutation of `ct`'s slot vector (the permutation determined by the automorphism group element).

Usage

```
eval_automorphism(ct, index, eval_key_map)
```

Arguments

`ct`	A `Ciphertext`.
`index`	Integer; the automorphism index (must match one of the indices passed to `eval_automorphism_key_gen(`
`eval_key_map`	An `EvalKeyMap` from `eval_automorphism_key_gen()`.

Value

A transformed Ciphertext.

See Also

eval_automorphism_key_gen(), eval_rotate()

---

eval_automorphism_key_gen *Generate automorphism evaluation keys for a set of indices*

Description

Generates the eval-key map needed to apply eval_automorphism() at the given set of slot indices. The generated keys are both inserted into the CryptoContext's internal eval-automorphism-key registry (keyed by sk's tag) **and** returned as an EvalKeyMap handle that the caller can pass directly to eval_automorphism().

Usage

```
eval_automorphism_key_gen(cc, sk, indices)
```

Arguments

cc	A CryptoContext.
sk	A PrivateKey.
indices	Integer vector of automorphism indices (not slot indices — use find_automorphism_indices() to compute them from slot indices).

Details

On the C++ side this is equivalent to calling EvalAutomorphismKeyGen(sk, indices) which internally calls CryptoContextImpl::InsertEvalAutomorphismKey with the generated map (cryptocontext.h line 2237). The dual return / registry-insert pattern matches the openfhe-python behavior at the equivalent entry point.

Companion to eval_rotate_key_gen() (reached via key_gen()'s rotations argument): both populate the same cc-internal storage. The automorphism form gives raw access to the automorphism group element (bypassing the rotate-to-automorphism slot mapping that eval_rotate_key_gen performs internally).

Value

An EvalKeyMap with one entry per input index.

See Also

eval_automorphism(), find_automorphism_indices()

eval_bin_gate *Evaluate a binary gate on encrypted values*

Description

Evaluate a binary gate on encrypted values

Usage

```
eval_bin_gate(ctx, gate, ct1, ct2 = NULL)
```

Arguments

ctx	A BinFHE context
gate	A BinGate value. Two-input gates: OR, AND, NOR, NAND, XOR, XNOR, XOR_FAST, XNOR_FAST. Three or more input gates (vector form): MAJORITY, AND3, OR3, AND4, OR4, CMUX.
ct1	An LWECiphertext, OR a list of LWECiphertext objects for the 3+-input vector form. When a list is supplied, ct2 must be left at its default (NULL).
ct2	An LWECiphertext for the 2-input form, or NULL (the default) when ct1 is a list. The vector dispatch follows binfhecontext.h line 322.

Value

An LWECiphertext

eval_bootstrap *Perform CKKS bootstrapping*

Description

Refreshes the ciphertext to allow further computation.

Usage

```
eval_bootstrap(ct, num_iterations = 1L, precision = 0L)
```

Arguments

ct	A Ciphertext
num_iterations	Number of bootstrap iterations (default: 1)
precision	Target precision (default: 0 = automatic)

Value

A refreshed Ciphertext

eval_bootstrap_key_gen *Generate bootstrapping keys*

Description

Generate bootstrapping keys

Usage

```
eval_bootstrap_key_gen(cc, sk, slots)
```

Arguments

cc	A CryptoContext
sk	A PrivateKey
slots	Number of slots

eval_bootstrap_setup *Set up CKKS bootstrapping*

Description

Set up CKKS bootstrapping

Usage

```
eval_bootstrap_setup(
  cc,
  level_budget = c(5L, 4L),
  dim1 = c(0L, 0L),
  slots = 0L,
  correction_factor = 0L,
  precompute = TRUE,
  bt_slots_encoding = FALSE
)
```

Arguments

cc	A CryptoContext
level_budget	Integer vector of length 2 (default: c(5, 4))
dim1	Integer vector of length 2 (default: c(0, 0))
slots	Number of slots (default: 0 = automatic)

`correction_factor`
: Correction factor (default: 0)

`precompute`
: Precompute rotation keys (default: TRUE)

`bt_slots_encoding`
: Logical; controls whether the bootstrap precomputes with slot-count encoding (the `BTSlotsEncoding` tail argument on cryptocontext.h line 3513). Default `FALSE` matching the C++ default.

`eval_chebyshev` *Evaluate a Chebyshev series on a ciphertext*

Description

Uses OpenFHE's default algorithm selector, which routes to `eval_chebyshev_linear()` for degree < 5 and `eval_chebyshev_ps()` (Paterson-Stockmeyer) for higher degrees. Call the variants directly to force one algorithm or the other.

Usage

```
eval_chebyshev(ct, coefficients, a, b)
```

Arguments

`ct`	A Ciphertext
`coefficients`	Numeric vector of Chebyshev coefficients
`a`	Lower bound of the approximation interval
`b`	Upper bound of the approximation interval

Value

A Ciphertext

See Also

`eval_chebyshev_coefficients()` and `eval_chebyshev_function()` for constructing the coefficient vector from a user-supplied function; `eval_chebyshev_linear()` / `eval_chebyshev_ps()` for forcing the algorithm.

eval_chebyshev_coefficients

Chebyshev coefficients for a real-valued function

Description

Computes the Chebyshev (first-kind) coefficients that approximate a univariate function `func` on the interval $[a, b]$ using the discrete orthogonality formula at `degree + 1` Chebyshev nodes. This is a direct port of the upstream routine `EvalChebyshevCoefficients()` in `core/lib/math/chebyshev.cpp` of openfhe-development, and the returned vector is in exactly the form that `eval_chebyshev()` (and the upstream `EvalChebyshevSeries`) expect — the zeroth coefficient is not halved.

Usage

```
eval_chebyshev_coefficients(func, a, b, degree)
```

Arguments

`func`	An R function taking a single numeric argument and returning a numeric scalar.
`a`	Lower bound of the approximation interval.
`b`	Upper bound of the approximation interval.
`degree`	Chebyshev polynomial degree (must be >= 1).

Value

Numeric vector of length `degree + 1`.

See Also

`eval_chebyshev()`, `eval_chebyshev_function()`.

eval_chebyshev_function

Evaluate a user-supplied function on a CKKS ciphertext via Chebyshev approximation

Description

Computes the Chebyshev coefficients for `func` on `[a, b]` via `eval_chebyshev_coefficients()` and applies the resulting series to `ct` via `eval_chebyshev()`. This mirrors the upstream `CryptoContext::EvalChebyshevFu` helper, which is a thin wrapper around `EvalChebyshevCoefficients` followed by `EvalChebyshevSeries` on the C++ side.

Usage

```
eval_chebyshev_function(ct, func, a, b, degree)
```

Arguments

ct	A Ciphertext holding CKKS-encoded real values in [a, b].
func	An R function taking a single numeric argument and returning a numeric scalar. It is evaluated on cleartext Chebyshev nodes inside [a, b] to produce the approximating coefficients.
a	Lower bound of the approximation interval.
b	Upper bound of the approximation interval.
degree	Chebyshev polynomial degree (must be >= 1).

Value

A Ciphertext holding the elementwise approximation of func applied to the plaintext slots of ct.

See Also

eval_chebyshev(), eval_chebyshev_coefficients(), eval_logistic(), eval_sin(), eval_cos(), eval_divide().

---

eval_chebyshev_linear *Evaluate a Chebyshev series via the linear evaluator*

Description

Forces the "linear" Chebyshev evaluator regardless of degree. Shallower circuit depth than Paterson-Stockmeyer but uses more multiplications at high degree.

Usage

```
eval_chebyshev_linear(ct, coefficients, a, b)
```

Arguments

ct	A Ciphertext
coefficients	Numeric vector of Chebyshev coefficients
a	Lower bound of the approximation interval
b	Upper bound of the approximation interval

Value

A Ciphertext

See Also

eval_chebyshev(), eval_chebyshev_ps()

eval_chebyshev_ps *Evaluate a Chebyshev series via the Paterson-Stockmeyer method*

Description

Forces the Paterson-Stockmeyer Chebyshev evaluator regardless of degree. Efficient for high-degree polynomials.

Usage

```
eval_chebyshev_ps(ct, coefficients, a, b)
```

Arguments

ct	A Ciphertext
coefficients	Numeric vector of Chebyshev coefficients
a	Lower bound of the approximation interval
b	Upper bound of the approximation interval

Value

A Ciphertext

See Also

eval_chebyshev(), eval_chebyshev_linear()

eval_cos *Evaluate cosine on a ciphertext*

Description

Evaluate cosine on a ciphertext

Usage

```
eval_cos(ct, a, b, degree)
```

Arguments

ct	A Ciphertext
a	Lower bound of the approximation interval
b	Upper bound of the approximation interval
degree	Chebyshev polynomial degree

eval_divide *Evaluate division approximation on a ciphertext*

Description

Evaluate division approximation on a ciphertext

Usage

```
eval_divide(ct, a, b, degree)
```

Arguments

ct	A Ciphertext
a	Lower bound of the approximation interval
b	Upper bound of the approximation interval
degree	Chebyshev polynomial degree

eval_fast_rotation *Hoisted slot rotation using precomputed digits*

Description

Hoisted slot rotation using precomputed digits

Usage

```
eval_fast_rotation(ct, ...)
```

Arguments

ct	A Ciphertext
...	Method-specific arguments: index (rotation amount, positive = left, negative = right), m (cyclotomic order, typically 2 * ring_dimension(ctx)), precomp (a FastRotationPrecomputation from eval_fast_rotation_precompute())

Value

A Ciphertext

eval_fast_rotation_ext

Extended hoisted slot rotation

Description

Applies a rotation using precomputed digit decomposition like `eval_fast_rotation()`, but with the extension that the first digit of the decomposition can be folded into the output before the rotation is applied (controlled by `add_first`). Used inside the CKKS bootstrap fast-rotation inner loop per `openfhe-development`'s `scheme/base-scheme.cpp`. The eval-key map is pulled from the `CryptoContext` internal registry via the ciphertext's key tag, so there is no `EvalKeyMap` argument at the R boundary — the automorphism keys must already be resident on the cc (call `key_gen(cc, rotations = ...)` to populate them, then reuse the `ct` here).

Usage

```
eval_fast_rotation_ext(ct, ...)
```

Arguments

`ct`	A `Ciphertext`.
`...`	Method-specific arguments: `index` (rotation amount, positive = left, negative = right), `precomp` (a `FastRotationPrecomputation` from `eval_fast_rotation_precompute()`), `add_first` (logical, default `FALSE`).

Value

A `Ciphertext`.

See Also

`eval_fast_rotation()`

eval_fast_rotation_precompute

Precompute digit decomposition for hoisted fast rotations

Description

Computes the digit decomposition of a ciphertext once so that multiple `eval_fast_rotation()` calls against the same ciphertext avoid redoing it. The cyclotomic order `m` (typically `2 * N`, where `N` is the ring dimension) is required by `eval_fast_rotation()`.

Usage

```
eval_fast_rotation_precompute(ct, ...)
```

Arguments

ct	A Ciphertext
...	Reserved for future method-specific arguments

Value

A FastRotationPrecomputation

---

eval_floor *Evaluate a floor (rounding) function on an LWE ciphertext*

Description

Performs the LWE equivalent of `floor(ct / 2^roundbits)` via functional bootstrapping. Used as a primitive in arbitrary- function evaluation pipelines where the bit-level rounding operation is needed separately from `eval_func()`'s LUT path.

Usage

```
eval_floor(ctx, ct, roundbits)
```

Arguments

ctx	A BinFHEContext
ct	An LWECiphertext
roundbits	Integer; the number of low-order bits to round off.

Details

Binding-level note: the underlying `BinFHEContext__EvalFloor` cpp11 binding has been present since the earliest BinFHE work; the R wrapper was added later to close the latent gap (cpp11-only entry with no R path).

Value

A new LWECiphertext holding the rounded value.

See Also

`eval_func()`, `eval_sign()`

eval_func *Evaluate an arbitrary function on an encrypted value*

Description

Functional bootstrapping with a precomputed lookup table. The context must have been created with `arb_func = TRUE`.

Usage

```
eval_func(ctx, ct, lut)
```

Arguments

`ctx`	A BinFHE context built with `arb_func = TRUE`
`ct`	An LWECiphertext encrypted with `output = BinFHEOutput$LARGE_DIM`
`lut`	A numeric vector of length `p` (the plaintext modulus) typically produced by `generate_lut_via_function()`

Value

An LWECiphertext encrypting `lut[plaintext(ct) + 1]`

eval_logistic *Evaluate logistic function on a ciphertext*

Description

Evaluate logistic function on a ciphertext

Usage

```
eval_logistic(ct, a, b, degree)
```

Arguments

`ct`	A Ciphertext
`a`	Lower bound of the approximation interval
`b`	Upper bound of the approximation interval
`degree`	Chebyshev polynomial degree

eval_mult *Homomorphic multiplication*

Description

Homomorphic multiplication

Usage

```
eval_mult(x, y, ...)
```

Arguments

x, y	Ciphertext, Plaintext, or numeric values
...	Method-specific arguments

Value

A Ciphertext

eval_mult_and_relinearize *Fused multiply-and-relinearize*

Description

Equivalent to `relinearize(eval_mult_no_relin(x, y))` but slightly more efficient in OpenFHE's implementation. Use this at the end of a multiplication chain.

Usage

```
eval_mult_and_relinearize(x, ...)
```

Arguments

x	A `Ciphertext`.
...	Method-specific arguments: `y` — a `Ciphertext` to multiply `x` by in a fused multiply-and-relinearize step.

Value

A relinearized `Ciphertext`.

eval_mult_in_place *Homomorphic multiplication in place*

Description

Modifies the first argument in place to hold the result of multiplying by a numeric scalar. `CryptoContextImpl` only declares `EvalMultInPlace` scalar overloads in the v1.5.1.0 header surface — the ct/ct and ct/pt variants that upstream-defects P1 refers to live on `SchemeBase` and are not exposed on `CryptoContextImpl`, so R's `eval_mult_in_place` supports only the scalar case. The ct/ct multiplication continues to work via the non-in-place `eval_mult()` generic.

Usage

```
eval_mult_in_place(x, ...)
```

Arguments

`x`	A `Ciphertext` (modified in place).
`...`	Method-specific arguments: `y` — a numeric scalar to multiply `x` by.

Value

`x` invisibly.

eval_mult_key_gen *Generate relinearization (eval-mult) keys for a secret key*

Description

Standalone wrapper around the `CryptoContext::EvalMultKeyGen(privateKey)` C++ method. Populates the `CryptoContext`'s internal eval-mult registry (keyed by the secret key's tag) so that ciphertext × ciphertext multiplication can be relinearized.

Usage

```
eval_mult_key_gen(cc, sk)
```

Arguments

`cc`	A `CryptoContext`.
`sk`	A `PrivateKey` whose tag will be used to key the generated eval-mult key in the cc's internal registry.

Details

key_gen() folds this into its eval_mult = TRUE branch as a convenience for fresh keypairs. The standalone wrapper is the right entry point when the secret key already exists — for example in any threshold or multi-party flow that holds a secret-key share but did not generate it through key_gen().

Value

NULL, invisibly.

---

eval_mult_mutable *Homomorphic multiply, mutable variant*

Description

Homomorphic multiply, mutable variant

Usage

```
eval_mult_mutable(x, ...)
```

Arguments

x	A Ciphertext (may be modified internally).
...	Method-specific arguments: y — a Ciphertext (may be modified internally).

Value

A new Ciphertext holding x * y.

---

eval_mult_no_relin *Homomorphic multiplication without relinearization*

Description

Returns the raw product of two ciphertexts as a higher-degree ciphertext (the result has n1 + n2 - 1 polynomial components where the inputs had n1 and n2). The standard eval_mult() automatically relinearizes the result back to 2 components; this variant skips the relinearization step so that multiple multiplications can be chained at higher polynomial degree before a single relinearize() call at the end. Used by the EvalMultAndRelinearize fused variant and by EvalPolyWithPrecomp for noise-optimal polynomial evaluation.

Usage

```
eval_mult_no_relin(x, ...)
```

Arguments

x	A Ciphertext.
...	Method-specific arguments: y — a Ciphertext to multiply x by without relinearization.

Value

A Ciphertext at higher polynomial degree.

---

eval_negate *Homomorphic negation*

Description

Homomorphic negation

Usage

```
eval_negate(x, ...)
```

Arguments

x	A Ciphertext
...	Method-specific arguments

Value

A Ciphertext

---

eval_negate_in_place *Homomorphic negation in place*

Description

Homomorphic negation in place

Usage

```
eval_negate_in_place(x, ...)
```

Arguments

x	A Ciphertext (modified in place).
...	Reserved for future method-specific arguments.

Value

x invisibly.

eval_not *Evaluate NOT on an encrypted value*

Description

Evaluate NOT on an encrypted value

Usage

```
eval_not(ctx, ct)
```

Arguments

ctx	A BinFHE context
ct	An LWECiphertext

Value

An LWECiphertext

eval_poly *Evaluate a polynomial on a ciphertext*

Description

Evaluates p(x) = c0 + c1*x* + *c2*x^2 + ... on encrypted x. Uses OpenFHE's default algorithm selector, which routes to `eval_poly_linear()` for degree < 5 and `eval_poly_ps()` (Paterson-Stockmeyer) for higher degrees. Call the variants directly to force one algorithm or the other — Linear is shallower for low-degree polynomials; PS has fewer multiplications for high-degree polynomials.

Usage

```
eval_poly(ct, coefficients)
```

Arguments

ct	A Ciphertext
coefficients	Numeric vector of polynomial coefficients

Value

A Ciphertext

See Also

`eval_poly_linear()`, `eval_poly_ps()`

eval_poly_linear *Evaluate a polynomial via the linear evaluator*

Description

Forces the "linear" Horner-style polynomial evaluator regardless of degree. Shallower circuit depth than the Paterson-Stockmeyer variant but uses more multiplications at high degree. Cheaper for degree < 5; fall over to `eval_poly_ps()` above that.

Usage

```
eval_poly_linear(ct, coefficients)
```

Arguments

`ct`	A Ciphertext
`coefficients`	Numeric vector of polynomial coefficients

Value

A Ciphertext

See Also

`eval_poly()`, `eval_poly_ps()`

eval_poly_ps *Evaluate a polynomial via the Paterson-Stockmeyer method*

Description

Forces the Paterson-Stockmeyer polynomial evaluator regardless of degree. Efficient for high-degree polynomials (fewer multiplications at the cost of more additions and a deeper circuit); for degree < 5 prefer `eval_poly_linear()`.

Usage

```
eval_poly_ps(ct, coefficients)
```

Arguments

`ct`	A Ciphertext
`coefficients`	Numeric vector of polynomial coefficients

Value

A Ciphertext

See Also

eval_poly(), eval_poly_linear()

---

eval_rotate *Rotate ciphertext slots*

Description

Rotate ciphertext slots

Usage

```
eval_rotate(ct, ...)
```

Arguments

ct	A Ciphertext
...	Method-specific arguments (index)

Value

A Ciphertext

---

eval_rotate_key_gen *Generate rotation keys for a secret key*

Description

Standalone wrapper around the CryptoContext::EvalRotateKeyGen(privateKey, indexList) C++ method. Populates the CryptoContext's internal automorphism key registry for the supplied rotation indices so that eval_rotate() can consume them.

Usage

```
eval_rotate_key_gen(cc, sk, index_list)
```

Arguments

cc	A CryptoContext.
sk	A PrivateKey whose tag will be used to key the generated rotation keys in the cc's internal registry.
index_list	Integer vector of rotation indices.

Details

key_gen() folds this into its rotations = ... argument as a convenience for fresh keypairs. The standalone wrapper is the right entry point when the secret key already exists — for example as the lead-party rotation-key generation step in a multi-party rotation protocol, where subsequent parties contribute via multi_eval_at_index_key_gen().

Value

NULL, invisibly.

---

eval_sign *Evaluate sign on an encrypted value (functional bootstrapping)*

Description

Extracts the most-significant bit of an LWE ciphertext encrypted under the large modulus Q. The context must have been created with arb_func = TRUE.

Usage

```
eval_sign(ctx, ct, scheme_switch = FALSE)
```

Arguments

ctx	A BinFHE context built with arb_func = TRUE
ct	An LWECiphertext encrypted via bin_encrypt(..., mod = Q)
scheme_switch	Logical; when TRUE, the output ciphertext is encoded compatibly with the CKKS<->FHEW scheme-switching pipeline (the schemeSwitch flag at binfhecontext.h line 367). Default FALSE for the standalone FHEW path. Per the upstream header description, this is the "flag that indicates if it should be compatible to scheme switching".

Value

An LWECiphertext encrypting 0 if the input was negative (i.e. lay in the upper half of [0, Q)), 1 otherwise

eval_sin *Evaluate sine on a ciphertext (Chebyshev approximation)*

Description

Evaluate sine on a ciphertext (Chebyshev approximation)

Usage

```
eval_sin(ct, a, b, degree)
```

Arguments

ct	A Ciphertext
a	Lower bound of the approximation interval
b	Upper bound of the approximation interval
degree	Chebyshev polynomial degree

Value

A Ciphertext

eval_square *Homomorphic squaring*

Description

Homomorphic squaring

Usage

```
eval_square(x, ...)
```

Arguments

x	A Ciphertext
...	Method-specific arguments

Value

A Ciphertext

eval_square_mutable *Homomorphic square, mutable variant*

Description

Homomorphic square, mutable variant

Usage

```
eval_square_mutable(x, ...)
```

Arguments

x	A Ciphertext (may be modified internally).
...	Reserved for future method-specific arguments.

Value

A new Ciphertext holding x * x.

eval_sub *Homomorphic subtraction*

Description

Homomorphic subtraction

Usage

```
eval_sub(x, y, ...)
```

Arguments

x, y	Ciphertext, Plaintext, or numeric values
...	Method-specific arguments

Value

A Ciphertext

eval_sub_in_place *Homomorphic subtraction in place*

Description

Modifies the first argument in place to hold the result of subtracting the second argument.

Usage

```
eval_sub_in_place(x, ...)
```

Arguments

x	A Ciphertext (modified in place).
...	Method-specific arguments: y — a Ciphertext, Plaintext, or numeric scalar to subtract from x.

Value

x invisibly.

eval_sub_mutable *Homomorphic subtract, mutable variant*

Description

Homomorphic subtract, mutable variant

Usage

```
eval_sub_mutable(x, ...)
```

Arguments

x	A Ciphertext (may be modified internally).
...	Method-specific arguments: y — a Ciphertext (may be modified internally).

Value

A new Ciphertext holding x - y.

eval_sum *Sum all slots in a ciphertext*

Description

Sum all slots in a ciphertext

Usage

```
eval_sum(ct, ...)
```

Arguments

ct	A Ciphertext
...	Method-specific arguments (batch_size)

Value

A Ciphertext

eval_sum_key_gen *Generate sum keys for a secret key*

Description

Populates the CryptoContext's internal sum-key registry (keyed by the secret key's tag) so that eval_sum() and the multi-party sum-key protocol can consume the generated entries. Closes a long-standing gap: the underlying CryptoContext__EvalSumKeyGen cpp11 binding has been present since the early phases but had no standalone R wrapper — users had to go through key_gen()'s side-effects only. The wrapper lands so that the multi-party sum-key flow (which needs to call this on each party's secret share) has a direct R-level entry point.

Usage

```
eval_sum_key_gen(cc, sk)
```

Arguments

cc	A CryptoContext.
sk	A PrivateKey whose tag will be used to key the generated sum-key map in the cc's internal registry.

Value

NULL, invisibly.

EvalKey *EvalKey class for multi-party key operations*

Description

EvalKey class for multi-party key operations

Usage

```
EvalKey(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

EvalKeyMap *Map of homomorphic evaluation keys*

Description

Opaque S7 wrapper around a shared_ptr<std::map<uint32_t, EvalKey<DCRTPoly>>>. Produced by the multi_eval_*_key_gen() family and by get_eval_sum_key_map() / get_eval_automorphism_key_map(); consumed by multi_add_eval_sum_keys(), multi_add_eval_automorphism_keys(), insert_eval_sum_key(), and insert_eval_automorphism_key(). The map is keyed by a rotation/automorphism index and carries one EvalKey per index.

Usage

```
EvalKeyMap(ptr = NULL)
```

Arguments

ptr	External pointer (internal use).

Details

Users do not construct or index into an EvalKeyMap directly — it is a transport format for the multi-party eval-key protocols. In a single-user protocol the same data is tracked inside the CryptoContext's internal key registry (populated by EvalSumKeyGen() / EvalRotateKeyGen()) and is only exposed as an EvalKeyMap when the distributed-party flow needs to exchange it.

`ExecutionMode` *Execution Mode Source: pke/constants-defs.h enum ExecutionMode*

Description

Execution Mode Source: pke/constants-defs.h enum ExecutionMode

Usage

```
ExecutionMode
```

`FastRotationPrecomputation` *Precomputed digit decomposition for hoisted rotations*

Description

Returned by `eval_fast_rotation_precompute()` and consumed by `eval_fast_rotation()`. Hoisting amortizes the per-rotation decomposition over many rotations of the same source ciphertext.

Usage

```
FastRotationPrecomputation(ptr = NULL)
```

Arguments

`ptr`	External pointer (internal use)

`Feature` *PKE Scheme Features (bitmask) Source: pke/constants-defs.h enum PKESchemeFeature*

Description

PKE Scheme Features (bitmask) Source: pke/constants-defs.h enum PKESchemeFeature

Usage

```
Feature
```

`fhe_ckks_tolerance` *Per-test CKKS precision tolerance*

Description

Computes a tolerance value suitable for comparing the cleartext result of a CKKS circuit against a decrypted ciphertext. The value is a function of the scheme's scaling factor, the circuit's multiplicative depth at the point of decryption, and the `ScalingTechnique`-specific precision loss per level.

Usage

```
fhe_ckks_tolerance(x, ...)
```

Arguments

`x`	For the numeric method: integer multiplicative depth at the point of decryption. For the Ciphertext method: a `Ciphertext` object whose associated context supplies all parameters.
`...`	Method-specific arguments. The numeric method accepts `scaling_factor_bits` (integer bit size of the scaling modulus, typical values 50/59/78), `scaling_technique` (a character string like `"FLEXIBLEAUTO"` or an integer from the `ScalingTechnique` enum), and `k` (conservative multiplicative factor, default 8). The Ciphertext method accepts only `k`.

Details

Two dispatch forms:

- **Stage 1 (numeric)**: pass parameters as direct arguments. Useful when the ciphertext isn't yet constructed — e.g. in a fixture setup block that has to produce a tolerance before calling `encrypt()`.
- **Stage 2 (Ciphertext)**: pass a `Ciphertext` directly. The helper reads the associated `CryptoContext` via `get_crypto_context()`, pulls the multiplicative depth (from context) minus the ciphertext's current level, computes the scaling factor bits via `ckks_scaling_factor_bits()`, and looks up the scaling technique. This form is preferred at test sites where a ciphertext exists.

Value

Numeric scalar — the tolerance value to use as `tolerance` in `tinytest::expect_equal()` or as `atol` in a manual diff.

Examples

```
# Stage 1 — pass parameters directly:
tol1 <- fhe_ckks_tolerance(4L, 50L, "FLEXIBLEAUTO")

# Stage 2 — pass a ciphertext (requires a live CKKS context):
```

```
# cc <- fhe_context("CKKS", multiplicative_depth = 4L, scaling_mod_size = 50L)
# kp <- key_gen(cc, eval_mult = TRUE)
# pt <- make_ckks_packed_plaintext(cc, c(0.1, 0.2, 0.3, 0.4))
# ct <- encrypt(kp@public, pt, cc)
# tol2 <- fhe_ckks_tolerance(ct)
```

`fhe_context` *Create a fully homomorphic encryption context*

Description

High-level constructor that creates a `CryptoContext` with sensible defaults. `PKE`, `KEYSWITCH`, and `LEVELEDSHE` features are enabled automatically.

Usage

```
fhe_context(scheme = c("BFV", "BGV", "CKKS"), ..., features = NULL)
```

Arguments

`scheme`	Character: "BFV", "BGV", or "CKKS".
`...`	Scheme-specific `CCParams` setter arguments. Forwarded to `BFVParams()`, `BGVParams()`, or `CKKSParams()`. Example: `fhe_context("BFV", plaintext_modulus = 65537, multiplicative_depth = 2)` or `fhe_context("CKKS", multiplicative_depth = 4, scaling_mod_size = 50, scaling_technique = ScalingTechnique$FLEXIBLEAUTO)`.
`features`	Additional `Feature` values to enable on the context beyond the default `PKE|KEYSWITCH|LEVELEDSHE` triple.

Details

All scheme-specific `CCParams` setter arguments are accepted via `...` and forwarded to the appropriate per-scheme constructor (`BFVParams()`, `BGVParams()`, or `CKKSParams()`). See those functions' argument lists for the valid per-scheme setter surface — each scheme accepts only the setters that are *not* disabled in its upstream `CCParams<T>` specialization (see discovery D013). Passing an invalid scheme-specific argument produces an R-level "unused argument" error at the underlying `*Params()` call site.

Value

A `CryptoContext` object.

See Also

`BFVParams()`, `BGVParams()`, `CKKSParams()`

fhe_deserialize *Deserialize an OpenFHE object from file*

Description

Deserialize an OpenFHE object from file

Usage

```
fhe_deserialize(
  filename,
  type = c("CryptoContext", "PublicKey", "PrivateKey", "Ciphertext"),
  format = "binary"
)
```

Arguments

filename	Path to serialized file
type	One of "CryptoContext", "PublicKey", "PrivateKey", "Ciphertext"
format	"binary" (default) or "json"

Value

The deserialized object

fhe_serialize *Serialize an OpenFHE object to file*

Description

Serialize an OpenFHE object to file

Usage

```
fhe_serialize(x, ...)
```

Arguments

x	An OpenFHE object (CryptoContext, PublicKey, PrivateKey, Ciphertext)
...	Method-specific arguments (filename, format)

Value

TRUE on success (invisibly)

find_automorphism_index *Compute the automorphism index for a single slot index*

Description

Maps a CKKS slot index to the corresponding automorphism index in the cyclotomic ring `Z[X]/(X^N + 1)`. The automorphism group of the ring is isomorphic to the multiplicative group `(Z/2N)*`; this function returns the representative of that group that corresponds to rotating the plaintext slots by the given amount.

Usage

```
find_automorphism_index(cc, index)
```

Arguments

`cc`	A `CryptoContext`.
`index`	Integer; the slot index (positive = left rotation, negative = right rotation).

Details

Used as a primitive by `find_automorphism_indices()` and by code that needs to address automorphism keys directly (for example, selectively generating eval keys for a sparse set of rotation amounts).

R-first binding: `openfhe-python` does not bind `FindAutomorphismIndex`. Logged in `notes/upstream-defects.md` under R-only surface.

Value

Integer; the automorphism group element corresponding to that rotation.

See Also

`find_automorphism_indices()` for the vector form.

find_automorphism_indices

Compute the automorphism indices for a list of slot indices

Description

Vector form of `find_automorphism_index()`. Takes a vector of slot indices and returns the corresponding automorphism indices in the same order. **R-first binding** — `openfhe-python` does not bind `FindAutomorphismIndices`.

Usage

```
find_automorphism_indices(cc, indices)
```

Arguments

`cc`	A `CryptoContext`.
`indices`	Integer vector of slot indices.

Value

Integer vector of automorphism indices.

See Also

`find_automorphism_index()`

generate_lut_via_function

Generate a lookup table for an arbitrary plaintext function

Description

Computes `f(0:(p-1), p)` and returns it as a numeric vector suitable for `eval_func()`. This is the R-side analogue of OpenFHE's `GenerateLUTviaFunction` — we don't bind the C++ helper because its signature takes a raw function pointer that can't capture an R closure, and R is natively vectorised so a pure-R helper is both simpler and faster than wiring an R callback through cpp11.

Usage

```
generate_lut_via_function(f, p)
```

Arguments

`f`	A function `function(m, p)` that returns the table entry for input `m` under plaintext modulus `p`. Vectorised functions are supported.
`p`	The plaintext modulus (typically `get_max_plaintext_space(ctx)`)

Value

A length-p numeric vector of LUT entries

`get_all_eval_automorphism_keys`
Retrieve all registered EvalAutomorphism key maps

Description

Reads the entire `CryptoContextImpl` internal EvalAutomorphism key map — a named R list keyed by secret-key tag, where each element is an `EvalKeyMap` (the rotation/automorphism key map for that party). Used for rotation and EvalAtIndex under the EvalKeyMap wire format.

Usage

```
get_all_eval_automorphism_keys()
```

Value

A named list keyed by key-tag string. Each element is an `EvalKeyMap` (opaque wrapper around `shared_ptr<map<uint32_t, EvalKey<DCRTPoly>>>`).

See Also

`get_eval_automorphism_key_map()` for per-tag lookup, `insert_eval_automorphism_key()` for the write path.

`get_all_eval_mult_keys`
Retrieve all registered EvalMult key vectors

Description

Reads the entire `CryptoContextImpl` internal EvalMult key map — a named R list keyed by secret-key tag, where each element is itself a list of `EvalKey` objects (the vector of multiplication-eval keys registered under that tag).

Usage

```
get_all_eval_mult_keys()
```

Details

The returned list is a **snapshot**: each `EvalKey` wraps a fresh `shared_ptr` copy, so retained references survive subsequent `clear_eval_mult_keys()` calls. The underlying keys are still shared with the cc registry — modifications through other paths remain visible.

Primary consumer: checkpoint/resume workflows that need to audit which parties have keys registered before serializing the cc.

Value

A named list keyed by key-tag string. Each element is a list of `EvalKey` objects.

See Also

`get_eval_mult_key_vector()` for per-tag lookup, `insert_eval_mult_key()` for the write path.

`get_all_eval_sum_keys` *Retrieve all registered EvalSum key maps*

Description

Reads the entire `CryptoContextImpl` internal EvalSum key map. Structurally identical to `get_all_eval_automorphism_ke` both share backing storage on the C++ side, but the two accessors are exposed separately so that fixture authors can match whichever OpenFHE doc they are reading.

Usage

```
get_all_eval_sum_keys()
```

Value

A named list keyed by key-tag string. Each element is an `EvalKeyMap`.

See Also

`get_eval_sum_key_map()`, `insert_eval_sum_key()`

get_bootstrap_depth *Get the required multiplicative depth for CKKS bootstrapping*

Description

Get the required multiplicative depth for CKKS bootstrapping

Usage

```
get_bootstrap_depth(level_budget, secret_key_dist = 1L)
```

Arguments

level_budget	Integer vector of length 2
secret_key_dist	Secret key distribution (default: UNIFORM_TERNARY = 1)

Value

Integer: required depth

get_ckks_boot_correction_factor *Get the CKKS bootstrap correction factor*

Description

Reads the current correction factor the scheme uses during the bootstrap `EvalModReduceInternal` step. Companion of `set_ckks_boot_correction_factor()`. Changes here affect all subsequent bootstrap operations on this `CryptoContext` until another call to `set_ckks_boot_correction_factor()`.

Usage

```
get_ckks_boot_correction_factor(cc)
```

Arguments

cc	A `CryptoContext`.

Value

Integer; the current correction factor.

See Also

`set_ckks_boot_correction_factor()`, `eval_bootstrap_setup()`

get_complex_packed_value *Get complex values from a CKKS plaintext*

Description

Reads the CKKS plaintext's internal slot vector as complex numbers. Every CKKS plaintext slot carries a `std::complex<double>` internally; when a plaintext was constructed from a real-valued vector via `make_ckks_packed_plaintext()`, the imaginary parts are all zero (up to CKKS encoding noise). When a plaintext was constructed from a complex vector (the is-complex dispatch path), both real and imaginary parts carry information.

Usage

```
get_complex_packed_value(pt)
```

Arguments

`pt`	A `Plaintext`.

Value

A native R `complex` vector.

See Also

`get_real_packed_value()` for the real-only view, `make_ckks_packed_plaintext()` for the matching constructor.

get_crypto_context *Associated CryptoContext of a Ciphertext*

Description

Returns the `CryptoContext` that was used to construct `ct`. OpenFHE ciphertexts carry a back-pointer to their context so that homomorphic operations can dispatch to the correct scheme implementation without requiring the user to pass the context explicitly.

Usage

```
get_crypto_context(ct, ...)
```

Arguments

`ct`	A `Ciphertext`.
`...`	Reserved for future method-specific arguments.

Details

Naming note: `get_crypto_context` is distinct from `get_crypto_parameters`. The former returns the high-level `CryptoContext` S7 wrapper; the latter returns the opaque `CryptoParameters` S7 wrapper.

Value

A `CryptoContext` S7 object.

`get_crypto_parameters` *Crypto parameters of a CryptoContext*

Description

Returns the `CryptoParameters` S7 object carrying the opaque `std::shared_ptr<CryptoParametersBase<DCRTPoly>>` at the C++ level. Useful for introspection; most R users will prefer to call the individual lambda-routed getters (e.g. `get_scaling_technique(cc)`, `get_batch_size(cc)`) directly instead of going through the CryptoParameters object.

Usage

```
get_crypto_parameters(cc, ...)
```

Arguments

`cc`	A `CryptoContext`.
`...`	Reserved for future method-specific arguments.

Value

A `CryptoParameters` S7 object.

`get_cyclotomic_order` *Cyclotomic order of a CryptoContext*

Description

Integer `m` such that the underlying polynomial ring is `Z[x]/(x^n + 1)` with `n = m/2` (the ring dimension). Always `2 * ring_dimension(cc)` for power-of-two cyclotomics.

Usage

```
get_cyclotomic_order(cc, ...)
```

Arguments

cc	A CryptoContext.
...	Reserved for future method-specific arguments.

Value

Integer.

---

get_element_params *Element parameters of a CryptoContext*

Description

Returns the ElementParams S7 object wrapping std::shared_ptr<typename DCRTPoly::Params>. This is the object that can be passed as the params argument to make_ckks_packed_plaintext() to build a plaintext against a specific parameter set rather than the context default.

Usage

```
get_element_params(cc, ...)
```

Arguments

cc	A CryptoContext.
...	Reserved for future method-specific arguments.

Details

This is the first R-side way to obtain a non-default ElementParams (the class itself has existed as a scaffold but had no constructor path until now).

Value

An ElementParams S7 object.

get_encoding_params *Encoding parameters of a CryptoContext*

Description

Returns the EncodingParams S7 object wrapping std::shared_ptr<EncodingParamsImpl>. Holds the plaintext modulus, batch size, and other encoding-level parameters.

Usage

```
get_encoding_params(cc, ...)
```

Arguments

cc	A CryptoContext.
...	Reserved for future method-specific arguments.

Value

An EncodingParams S7 object.

get_eval_automorphism_key_map *Retrieve the automorphism-key map for a given key tag*

Description

Accessor for the cc-internal static automorphism-key map. The underlying C++ call returns a shared_ptr directly (no copy), so the returned EvalKeyMap is a live view of the cc registry.

Usage

```
get_eval_automorphism_key_map(key_tag)
```

Arguments

key_tag	Character; the key tag used when the map was originally generated.

Value

An EvalKeyMap.

get_eval_mult_key_vector *Retrieve the EvalMult key vector for a given key tag*

Description

Reads the vector of EvalMult keys registered under `key_tag`. Errors (via `catch_openfhe`) if the tag is not present in the registry.

Usage

```
get_eval_mult_key_vector(key_tag)
```

Arguments

`key_tag`	Character; the tag to look up (typically `get_key_tag(sk)` of a generated `PrivateKey`).

Value

A list of `EvalKey` objects.

See Also

`get_all_eval_mult_keys()`, `insert_eval_mult_key()`

get_eval_sum_key_map *Retrieve the sum-key map for a given key tag*

Description

Accessor for the cc-internal static map populated by `eval_sum_key_gen()`. Used by the multi-party sum protocol to pull the lead party's initial eval-sum map so other parties can produce their shares.

Usage

```
get_eval_sum_key_map(key_tag)
```

Arguments

`key_tag`	Character; the key tag used when the map was originally generated. Typically the `get_key_tag()` of the secret key that produced it.

Details

The underlying C++ call returns a `const std::map&`; the R wrapper copies the map into a fresh `shared_ptr` to give the returned `EvalKeyMap` owning semantics. The returned map is a snapshot: subsequent modifications to the cc registry are not reflected.

Value

An `EvalKeyMap`.

`get_key_gen_level` *Key-generation level of a CryptoContext*

Description

Integer level at which subsequent `key_gen()` calls will generate keys. Defaults to `0L`. Useful when generating keys at a non-fresh level for deep circuit protocols.

Usage

```
get_key_gen_level(cc, ...)
```

Arguments

`cc`	A `CryptoContext`.
`...`	Reserved for future method-specific arguments.

Value

Integer.

`get_max_plaintext_space` *Maximum supported plaintext space for functional bootstrapping*

Description

Maximum supported plaintext space for functional bootstrapping

Usage

```
get_max_plaintext_space(ctx)
```

Arguments

`ctx`	A BinFHE context built with `arb_func = TRUE`

Value

A numeric scalar (q / (2 * beta))

---

get_native_int *Get the native integer size of the OpenFHE build*

Description

Returns 64 or 128 depending on how OpenFHE was compiled.

Usage

```
get_native_int()
```

Value

integer

---

get_packed_value *Get packed integer values from a plaintext*

Description

Get packed integer values from a plaintext

Usage

```
get_packed_value(pt)
```

Arguments

pt	A Plaintext

Value

Integer vector

get_real_packed_value *Get real values from a CKKS plaintext*

Description

Get real values from a CKKS plaintext

Usage

```
get_real_packed_value(pt)
```

Arguments

pt	A Plaintext

Value

Numeric vector

See Also

get_complex_packed_value() for the complex-view accessor on the same underlying plaintext (each slot internally carries a complex pair — this function returns only the real parts).

get_scaling_factor_real *Real-valued CKKS scaling factor at a modulus-chain level*

Description

Returns cryptoParams->GetScalingFactorReal(level) — the double-valued scaling factor at the given level of the CKKS modulus chain. Meaningful only for CKKS contexts; BFV/BGV contexts return the default field value (effectively 1.0).

Usage

```
get_scaling_factor_real(cc, ...)
```

Arguments

cc	A CryptoContext.
...	Method-specific arguments. The CryptoContext method accepts level (integer level in the RNS modulus chain, default 0L — the top of the chain = the scaling factor at fresh encryption time).

Details

Used by `ckks_scaling_factor_bits()` (which takes `log2` of the level-0 value to recover the bit size originally set via `set_scaling_mod_size()`) and by the Stage 2 form of `fhe_ckks_tolerance()`.

Value

Numeric scalar.

`insert_eval_automorphism_key` *Insert a joined automorphism-key map into the cc registry*

Description

After combining multi-party automorphism-key shares via `multi_add_eval_automorphism_keys()`, insert the joined map into the cc-internal registry so that `eval_rotate()` / `eval_fast_rotation()` can consume it.

Usage

```
insert_eval_automorphism_key(eval_key_map, key_tag = "")
```

Arguments

`eval_key_map`	An `EvalKeyMap`.
`key_tag`	Character; default `""`.

Value

`NULL`, invisibly.

`insert_eval_mult_key` *Insert an EvalMult key vector into the cc registry*

Description

Adds a vector of `EvalKey` objects to the CryptoContext's internal EvalMult-key map under `key_tag`. Silently replaces any existing matching keys. If `key_tag` is the empty string (`""`, the default), the tag is retrieved from the eval-key vector itself (each `EvalKey` carries its own tag).

Usage

```
insert_eval_mult_key(eval_keys, key_tag = "")
```

Arguments

eval_keys	A list of EvalKey objects (from multi_key_switch_gen(), multi_add_eval_mult_keys(), or from a deserialization).
key_tag	Character; the tag to register the vector under. Default "" (auto-detect from the first eval key in the vector).

Details

Used in checkpoint/resume workflows: after fhe_deserialize_eval_keys() or multi_add_eval_mult_keys() produces a combined eval-mult key vector, this function registers it into the cc's internal storage so that subsequent eval_mult() calls on ciphertexts encrypted under the associated party's key can consume it.

Value

NULL, invisibly.

See Also

insert_eval_sum_key(), insert_eval_automorphism_key()

---

insert_eval_sum_key *Insert a joined sum-key map into the cc registry*

Description

After combining multi-party shares via multi_add_eval_sum_keys(), the joined map has to be inserted back into the cc's internal static registry before eval_sum() can consume it. insert_eval_sum_key() routes the map through CryptoContextImpl::InsertEvalSumKey (which delegates internally to InsertEvalAutomorphismKey — the same static storage is shared between the two surfaces).

Usage

```
insert_eval_sum_key(eval_key_map, key_tag = "")
```

Arguments

eval_key_map	An EvalKeyMap from multi_add_eval_sum_keys() or constructed through the distributed key-gen flow.
key_tag	Character; the tag to register the map under. Default "".

Value

NULL, invisibly.

int_boot_add *Combine encrypted and unencrypted masked decryptions*

Description

Final step of the two-party interactive bootstrap protocol. Adds the server's masked decryption to the client's re-encryption to produce the refreshed ciphertext.

Usage

```
int_boot_add(ct1, ct2)
```

Arguments

ct1, ct2	Ciphertext objects — typically the outputs of int_boot_decrypt() and int_boot_encrypt().

Value

A refreshed Ciphertext.

int_boot_adjust_scale *Prepare a ciphertext for interactive bootstrap*

Description

Adjusts a ciphertext's scale to meet the scheme's requirements before entering the interactive bootstrap protocol. Typically called before int_boot_decrypt().

Usage

```
int_boot_adjust_scale(ct)
```

Arguments

ct	A Ciphertext.

Value

A Ciphertext ready for int_boot_decrypt().

int_boot_decrypt *Server-side masked decryption for interactive bootstrap*

Description

First step of the single-party interactive bootstrap protocol. The server applies its secret key share to produce a "masked" partial decryption that the client can finish off-line. Pairs with `int_boot_encrypt()` / `int_boot_add()` / `int_boot_adjust_scale()` to complete the refresh.

Usage

```
int_boot_decrypt(sk, ct)
```

Arguments

`sk`	A `PrivateKey` (server's share).
`ct`	A `Ciphertext` to refresh.

Value

A `Ciphertext` holding the masked decryption.

See Also

`int_boot_encrypt()`, `int_boot_add()`, `int_boot_adjust_scale()`

int_boot_encrypt *Client-side re-encryption for interactive bootstrap*

Description

Encrypts the client's masked decryption result under the public key, raising the ciphertext modulus back to a fresh level.

Usage

```
int_boot_encrypt(pk, ct)
```

Arguments

`pk`	A `PublicKey`.
`ct`	A `Ciphertext` from the client (typically the masked-decryption output processed off-line).

Value

A refreshed `Ciphertext`.

See Also

`int_boot_decrypt()`

`int_mp_boot_add` *Aggregate multi-party shares pairs*

Description

Combines the shares-pair lists produced by each party's call to `int_mp_boot_decrypt()` into a single aggregated shares pair for use in `int_mp_boot_encrypt()`. The input is a list of per-party shares-pair lists (a list of lists of `Ciphertext`).

Usage

```
int_mp_boot_add(cc, shares_pair_list)
```

Arguments

`cc`	A `CryptoContext`.
`shares_pair_list`	A list where each element is a list of `Ciphertext` objects from one party's `int_mp_boot_decrypt()` call.

Value

A list of `Ciphertext` objects — the aggregated shares pair.

`int_mp_boot_adjust_scale` *Prepare a ciphertext for multi-party interactive bootstrap*

Description

Multi-party analogue of `int_boot_adjust_scale()`. Adjusts the ciphertext's scale before entering the distributed bootstrap protocol.

Usage

```
int_mp_boot_adjust_scale(ct)
```

Arguments

ct	A Ciphertext.

Value

A Ciphertext ready for the multi-party bootstrap protocol.

---

int_mp_boot_decrypt *Multi-party masked decryption for interactive bootstrap*

Description

Each party calls this with their own secret share, the ciphertext being refreshed, and the common random element from int_mp_boot_random_element_gen(). Returns a list of two Ciphertext objects — the party's masked-decryption "shares pair". Each party's shares pair gets collected and fed into int_mp_boot_add().

Usage

```
int_mp_boot_decrypt(sk, ct, a)
```

Arguments

sk	A PrivateKey (this party's share).
ct	A Ciphertext to refresh.
a	A Ciphertext holding the common random element from int_mp_boot_random_element_gen().

Value

A list of two Ciphertext objects (the party's shares pair).

---

int_mp_boot_encrypt *Final re-encryption for multi-party interactive bootstrap*

Description

Lead party's final step in the multi-party interactive bootstrap. Takes the aggregated shares pair from int_mp_boot_add() plus the common random element and the original ciphertext, produces the refreshed ciphertext at a fresh modulus level.

Usage

```
int_mp_boot_encrypt(pk, shares_pair, a, ct)
```

Arguments

pk	The lead party's PublicKey.
shares_pair	A list of Ciphertext objects — the aggregated shares pair from int_mp_boot_add().
a	The common random element Ciphertext used in the per-party int_mp_boot_decrypt() calls.
ct	The original Ciphertext being refreshed.

Value

A refreshed Ciphertext.

---

int_mp_boot_random_element_gen

Generate a common random element for multi-party bootstrap

Description

Generates a common random polynomial used by all parties in a multi-party interactive bootstrap round. Two overloads:

Usage

```
int_mp_boot_random_element_gen(cc, source)
```

Arguments

cc	A CryptoContext. Only used by the PublicKey overload; the Ciphertext overload ignores it and uses the source's internal cc.
source	Either a PublicKey or a Ciphertext.

Details

- When source is a PublicKey (the lead party's public key), routes to the (publicKey) C++ overload.
- When source is a Ciphertext, routes to the (ciphertext) overload which derives the cc and parameters from the ciphertext directly — convenient when a ciphertext is already in scope.

Value

A Ciphertext holding the common random element.

is_good *Is a KeyPair valid?*

Description

Returns TRUE when both the public and secret keys of a KeyPair are non-null external pointers. The C++ KeyPair::good() predicate performs the same check on the C++ side; because R's KeyPair is a pure-R aggregate that wraps an already-constructed PublicKey and PrivateKey, the R-level check is equivalent.

Usage

```
is_good(kp, ...)
```

Arguments

kp	A KeyPair.
...	Reserved for future method-specific arguments (currently unused).

Value

TRUE or FALSE.

key_gen *Generate key pair*

Description

Generate key pair

Usage

```
key_gen(cc, ...)
```

Arguments

cc	A CryptoContext
...	Method-specific arguments (eval_mult, rotations)

Value

A KeyPair

key_switch_down *Scale a ciphertext down from extended CRT basis to Q*

Description

Brings a ciphertext that lives in the extended P*Q basis (for example, the output of `eval_fast_rotation_ext()` with hybrid key switching) back to the standard Q basis. Only supported when the scheme is configured with hybrid key switching — other key-switching techniques have no round-trip to extended PQ and therefore nothing to scale back from.

Usage

```
key_switch_down(ct)
```

Arguments

`ct`	A `Ciphertext` in the extended P*Q basis.

Details

R-first binding: openfhe-python v1.5.1.0 does not expose `KeySwitchDown` at all. See `notes/upstream-defects.md` for the R-only surface tracking.

Value

A `Ciphertext` in the Q basis.

See Also

`eval_fast_rotation_ext()`

key_tag *Key tag accessors*

Description

Every `PublicKey` / `PrivateKey` carries a string "key tag" identifying which key pair it belongs to. The tag is set at key-generation time by OpenFHE and can be inspected or overwritten via these accessors. In threshold / multiparty protocols the tag is used to associate a key with the party that owns it; in single-user protocols it is typically left at its default.

Usage

```
get_key_tag(key, ...)

set_key_tag(key, ...)
```

Arguments

key	A PublicKey or PrivateKey.
...	Reserved for future method-specific arguments. set_key_tag accepts a value argument here.

Value

get_key_tag: character scalar. set_key_tag: the key invisibly.

KeygenMode *Key Generation Mode Source: binfhe/binfhe-constants.h enum KEYGEN_MODE*

Description

Key Generation Mode Source: binfhe/binfhe-constants.h enum KEYGEN_MODE

Usage

```
KeygenMode
```

KeyPair *Key Pair*

Description

Contains a public key and a secret (private) key.

Usage

```
KeyPair(public = NULL, secret = NULL)
```

Arguments

public	A PublicKey
secret	A PrivateKey

KeySwitchTechnique — *Key Switching Techniques Source: pke/constants-defs.h enum KeySwitchTechnique*

Description

Key Switching Techniques Source: pke/constants-defs.h enum KeySwitchTechnique

Usage

```
KeySwitchTechnique
```

level_reduce — *Reduce the modulus chain by multiple levels*

Description

Drops `levels` levels from `x`'s modulus chain in a single operation. Useful when `x` is at a deeper level than the ciphertext it will interact with next; level-reducing brings them onto the same rung of the chain.

Usage

```
level_reduce(x, ...)
```

Arguments

`x`	A `Ciphertext`.
`...`	Method-specific arguments: `eval_key` (an `EvalKey` from `key_gen`), `levels` (integer, default `1L`).

Details

An evaluation key is required — supply it from `key_gen(cc, eval_mult = TRUE)`.

Value

A `Ciphertext` at `x$level + levels`.

level_reduce_in_place *Reduce the modulus chain by multiple levels, in place*

Description

Reduce the modulus chain by multiple levels, in place

Usage

```
level_reduce_in_place(x, ...)
```

Arguments

x	A Ciphertext (modified in place).
...	Method-specific arguments: eval_key (an EvalKey), levels (integer, default 1L).

Value

x invisibly.

LWECiphertext *LWE Ciphertext (Binary FHE)*

Description

LWE Ciphertext (Binary FHE)

Usage

```
LWECiphertext(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

LWEPrivateKey *LWE Private Key (Binary FHE)*

Description

LWE Private Key (Binary FHE)

Usage

```
LWEPrivateKey(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

make_ckks_packed_plaintext *Make a CKKS packed plaintext from real numbers*

Description

Encode a real-valued numeric vector as a CKKS packed plaintext. The result is an unencrypted `Plaintext` object that can then be passed to `encrypt()`.

Usage

```
make_ckks_packed_plaintext(
  cc,
  values,
  noise_scale_deg = 1L,
  level = 0L,
  params = NULL,
  slots = 0L
)
```

Arguments

cc	A `CryptoContext` built with `scheme = "CKKS"`.
values	A numeric vector to pack. Length must not exceed `slots` (or `batch_size / 2` when `slots` is left at `0L`).
noise_scale_deg	Integer degree of the initial scaling factor applied to the encoded plaintext, expressed as a power of the scheme's scaling factor. Defaults to `1L`, meaning "scale by the base scaling factor once", which is the value every current vignette implicitly uses. Setting `noise_scale_deg = 2L` encodes at scaling factor squared,

which is occasionally useful when the plaintext is about to be subtracted from a ciphertext that has already been rescaled once and the two noise levels must agree. Under `FLEXIBLEAUTO` scaling the scheme's auto-rescale logic overrides this argument at context-generation time — see discovery D011 — so this parameter is only meaningful under `FIXEDMANUAL`. Couples tightly to `level`: a plaintext at `(noise_scale_deg = k, level = L)` may only interact with ciphertexts at the same `(k, L)`.

`level` Integer target encryption level of the encoded plaintext. Defaults to `0L`, meaning "encode at the fresh level, matching a just-encrypted ciphertext". When performing an operation between an encoded plaintext and a ciphertext that has already been rescaled `L` times, the plaintext must be encoded at `level = L` so the two sit at the same level of the modulus chain; otherwise the evaluator will silently mismatch or error depending on the scheme variant. The canonical case for setting this is encoding a constant vector for use inside a deep CKKS circuit, not for fresh-input computation. Couples to `noise_scale_deg` and to the multiplicative depth of the crypto context, which bounds the maximum valid `level`.

`params` Advanced. An `ElementParams` object to encode against, or `NULL` to use the `cryptocontext`'s own parameters at the chosen `level`. Defaults to `NULL`, which is the only value any current vignette or test uses. There is no R-side way to construct a fresh `ElementParams` (only to wrap one returned by `get_element_params(cc)`), so in practice the argument is accepted for surface parity with openfhe-python but is normally left at its default.

`slots` Integer number of CKKS slots to pack into. Defaults to `0L`, which is the upstream sentinel for "use the `batch_size` set on the context's params". Setting `slots` to a smaller power of two produces a plaintext that leaves the upper half of the packing register zero, which the evaluator exploits to reduce rotation cost in algorithms like `eval_sum`. The value must be a power of two and must not exceed `batch_size`. The natural time to set this is when you have a short input vector and rotations dominate the circuit cost, as in the CKKS-inner-product idiom; the natural time to leave it at `0L` is when you are encoding a full-width vector.

Value

A `Plaintext`.

`make_coef_packed_plaintext` *Make a coefficient-packed integer plaintext*

Description

Encode an integer vector as a coefficient-packed plaintext. Coefficient packing places each input value in a separate polynomial coefficient and is the alternative to the SIMD batched packing produced by `make_packed_plaintext()`. Used by the integer-modulus Ring-LWE vignettes where per-coefficient access is needed.

Usage

```
make_coef_packed_plaintext(cc, values, noise_scale_deg = 1L, level = 0L)
```

Arguments

cc	A CryptoContext (BFV or BGV).
values	An integer vector whose length must not exceed the ring dimension of cc.
noise_scale_deg	See the make_packed_plaintext() entry.
level	See the make_packed_plaintext() entry.

Value

A Plaintext.

---

make_packed_plaintext *Make a packed integer plaintext*

Description

Encode an integer vector as a BFV / BGV packed plaintext. The result is an unencrypted Plaintext object that can then be passed to encrypt().

Usage

```
make_packed_plaintext(cc, values, noise_scale_deg = 1L, level = 0L)
```

Arguments

cc	A CryptoContext.
values	An integer vector to pack. Length must not exceed batch_size set at context creation.
noise_scale_deg	Integer degree of the initial scaling factor applied to the encoded plaintext. Defaults to 1L; only meaningful under FIXEDMANUAL scaling (under FLEXIBLEAUTO the scheme overrides this value — see discovery D011). Every current vignette leaves it at the default.
level	Integer target level in the RNS modulus chain. Defaults to 0L, meaning "fresh level, matching a just-encrypted ciphertext". Set to match the level of a ciphertext the plaintext will interact with if the ciphertext has already been rescaled.

Value

A Plaintext.

mod_reduce *Reduce the modulus chain by one level*

Description

Synonym for `rescale()`. Both names dispatch to the same C++ operation (`CryptoContextImpl::Rescale` delegates to `ModReduce` internally); the R binding keeps both so fixture and vignette authors can use whichever name matches the OpenFHE documentation they're following.

Usage

```
mod_reduce(x, ...)
```

Arguments

`x`	A `Ciphertext`.
`...`	Reserved for future method-specific arguments.

Value

A `Ciphertext` at one lower level.

See Also

`rescale()`, `mod_reduce_in_place()`

mod_reduce_in_place *Reduce the modulus chain by one level, in place*

Description

Reduce the modulus chain by one level, in place

Usage

```
mod_reduce_in_place(x, ...)
```

Arguments

`x`	A `Ciphertext` (modified in place).
`...`	Reserved for future method-specific arguments.

Value

`x` invisibly.

multi_add_eval_automorphism_keys *Combine two automorphism-key map shares*

Description

Combine two automorphism-key map shares

Usage

```
multi_add_eval_automorphism_keys(
  cc,
  eval_key_map1,
  eval_key_map2,
  key_tag = ""
)
```

Arguments

cc	A CryptoContext.
eval_key_map1, eval_key_map2	EvalKeyMap shares from two parties.
key_tag	Character; default "".

Value

A combined EvalKeyMap suitable for insertion into the cc registry via insert_eval_automorphism_key().

multi_add_eval_keys *Combine evaluation keys from multiple parties*

Description

Combines two partial key-switching eval keys into a joint eval key. See multi_add_eval_mult_keys() for the eval-mult variant — the two functions consume keys produced by different generators and are not interchangeable.

Usage

```
multi_add_eval_keys(cc, ek1, ek2, key_tag = "")
```

Arguments

cc	A CryptoContext
ek1, ek2	EvalKey objects to combine
key_tag	Character; optional tag to associate with the combined key. Default "".

Value

A combined EvalKey

`multi_add_eval_mult_keys` *Combine partial eval-mult keys from multiple parties*

Description

The eval-mult flavor of `multi_add_eval_keys()`. Consumes keys produced by a multi-party eval-mult key generator rather than by `multi_key_switch_gen()`. Where the R generator is not yet exposed, the wrapper still lets downstream code exercise the add-keys flow against keys constructed through the underlying cpp11 binding.

Usage

```
multi_add_eval_mult_keys(cc, ek1, ek2, key_tag = "")
```

Arguments

`cc`	A CryptoContext
`ek1`, `ek2`	EvalKey objects (eval-mult partials) to combine
`key_tag`	Character; optional tag to associate with the combined key. Default `""`.

Value

A combined EvalKey

`multi_add_eval_sum_keys` *Combine two sum-key map shares into a joint sum-key map*

Description

Combine two sum-key map shares into a joint sum-key map

Usage

```
multi_add_eval_sum_keys(cc, eval_key_map1, eval_key_map2, key_tag = "")
```

Arguments

`cc`	A `CryptoContext`.
`eval_key_map1`, `eval_key_map2`	`EvalKeyMap` shares from two parties.
`key_tag`	Character; default `""`.

Value

A combined EvalKeyMap suitable for insertion into the cc registry via insert_eval_sum_key().

---

multi_add_pub_keys *Combine public keys from multiple parties*

Description

Combine public keys from multiple parties

Usage

```
multi_add_pub_keys(cc, pk1, pk2, key_tag = "")
```

Arguments

cc	A CryptoContext
pk1, pk2	PublicKey objects to combine
key_tag	Character; optional tag to associate with the combined key. Default "" (empty tag) to match the C++ header default. Round-trips through get_key_tag().

Value

A combined PublicKey

---

multi_eval_at_index_key_gen *Generate a joint rotation-at-index key share*

Description

The EvalAtIndex flavor of multi_eval_automorphism_key_gen(); takes signed rotation indices rather than automorphism indices. Semantically equivalent but lives on a distinct C++ entry point, matching the openfhe-python surface.

Usage

```
multi_eval_at_index_key_gen(cc, sk, eval_key_map, index_list, key_tag = "")
```

Arguments

cc	A CryptoContext.
sk	This party's PrivateKey share.
eval_key_map	An existing EvalKeyMap.
index_list	Integer vector of signed rotation indices.
key_tag	Character; default "".

Value

An EvalKeyMap.

---

multi_eval_automorphism_key_gen

Generate a joint automorphism-key share for multi-party rotation

---

Description

Produces this party's share of the joined automorphism eval key map for the supplied index_list. Each other party calls the same method with their own secret share, and the shares are combined via multi_add_eval_automorphism_keys() to produce the final joined map.

Usage

```
multi_eval_automorphism_key_gen(cc, sk, eval_key_map, index_list, key_tag = "")
```

Arguments

cc	A CryptoContext with the MULTIPARTY feature enabled.
sk	This party's PrivateKey share.
eval_key_map	An existing EvalKeyMap carrying the prior-party automorphism key state, obtained via get_eval_automorphism_key_map() after the lead party has populated the cc registry through key_gen(cc, rotations = ...).
index_list	Integer vector of rotation indices.
key_tag	Character; optional tag to associate with the produced map. Default "".

Value

An EvalKeyMap holding this party's joint share.

---

multi_eval_sum_key_gen

Generate a joint sum-key share for multi-party EvalSum

Description

Generate a joint sum-key share for multi-party EvalSum

Usage

```
multi_eval_sum_key_gen(cc, sk, eval_key_map, key_tag = "")
```

Arguments

cc	A CryptoContext.
sk	This party's PrivateKey share.
eval_key_map	An existing EvalKeyMap carrying the prior-party sum-key state, obtained via get_eval_sum_key_map() after the lead party has populated the cc registry through eval_sum_key_gen().
key_tag	Character; default "".

Value

An EvalKeyMap.

---

multi_key_switch_gen *Multi-party key-switch eval-key generation*

Description

Generates an eval key that switches ciphertexts encrypted under sk_orig into a form decryptable by sk_new, starting from an existing eval key that carries the key-switch auxiliary information. Used by threshold protocols to route partial decryptions across a re-keyed party set.

Usage

```
multi_key_switch_gen(cc, sk_orig, sk_new, eval_key)
```

Arguments

cc	A CryptoContext
sk_orig	The original party's PrivateKey
sk_new	The new party's PrivateKey
eval_key	An EvalKey carrying key-switch auxiliary data

Value

An EvalKey suitable for routing through multi_add_eval_keys() to combine with other parties' key-switch shares

`multiparty_decrypt_fusion` *Fuse partial decryptions into final plaintext*

Description

Combines partial decryptions from any number of parties (n >= 2). The lead party's partial decryption (from `multiparty_decrypt_lead()`) must be supplied first; subsequent partials (from `multiparty_decrypt_main()`) follow in any order.

Usage

```
multiparty_decrypt_fusion(cc, ...)
```

Arguments

`cc`	A CryptoContext
`...`	Two or more partially decrypted Ciphertext objects. The first must be from the lead party.

Value

A Plaintext with the final decrypted result

`multiparty_decrypt_lead` *Lead party's partial decryption*

Description

In threshold decryption, the lead party calls this first. Accepts either a single `Ciphertext` or a list of `Ciphertext` objects:

Usage

```
multiparty_decrypt_lead(cc, sk, ct)
```

Arguments

`cc`	A CryptoContext
`sk`	This party's PrivateKey
`ct`	A Ciphertext or a list of Ciphertexts

Details

- single `Ciphertext`: returns a single partially decrypted `Ciphertext`, matching the original single-ciphertext signature.
- list of `Ciphertext`: returns a list of partially decrypted `Ciphertext` objects of the same length, routed through the C++ `MultipartyDecryptLead(vector<Ciphertext>, PrivateKey)` overload (cryptocontext.h line 3115). Useful when a protocol round needs to partially decrypt a batch in one trip.

Value

A partially decrypted Ciphertext or list of Ciphertexts, mirroring the input shape.

`multiparty_decrypt_main`

Non-lead party's partial decryption

Description

Other parties call this after the lead. Accepts either a single `Ciphertext` or a list of `Ciphertext` objects with the same semantics as `multiparty_decrypt_lead()`.

Usage

```
multiparty_decrypt_main(cc, sk, ct)
```

Arguments

`cc`	A CryptoContext
`sk`	This party's PrivateKey
`ct`	A Ciphertext or a list of Ciphertexts

Value

A partially decrypted Ciphertext or list of Ciphertexts, mirroring the input shape.

multiparty_key_gen *Generate a key pair for a secondary party in threshold FHE*

Description

The lead party uses `key_gen()` to generate the initial keypair. Subsequent parties call this with the lead's public key.

Usage

```
multiparty_key_gen(cc, lead_pk, make_sparse = FALSE, fresh = FALSE)
```

Arguments

`cc`	A CryptoContext (must have MULTIPARTY feature enabled)
`lead_pk`	The lead party's PublicKey
`make_sparse`	Logical; if `TRUE`, produce an LWE-sparse secret. Default `FALSE` to match the C++ header default. RLWE-only semantics; BFV/BGV/CKKS accept both values.
`fresh`	Logical; if `TRUE`, sample a fresh secret rather than deriving one from the existing key material. Default `FALSE`.

Value

A KeyPair for this party

MultipartyMode *Multiparty Mode Source: pke/constants-defs.h enum MultipartyMode*

Description

Multiparty Mode Source: pke/constants-defs.h enum MultipartyMode

Usage

```
MultipartyMode
```

MultiplicationTechnique

Multiplication Technique (BFV) Source: pke/constants-defs.h enum MultiplicationTechnique

Description

Multiplication Technique (BFV) Source: pke/constants-defs.h enum MultiplicationTechnique

Usage

```
MultiplicationTechnique
```

OpenFHEObject

Base class for OpenFHE objects

Description

All OpenFHE objects (CryptoContext, Ciphertext, Plaintext, Keys, etc.) inherit from this class. It holds an external pointer to a C++ shared_ptr.

Usage

```
OpenFHEObject(ptr = NULL)
```

Arguments

ptr	External pointer to C++ object (internal use)

Plaintext

Plaintext

Description

Plaintext

Usage

```
Plaintext(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

plaintext_accessors *Plaintext accessors*

Description

Retrieve or set fields on a `Plaintext` object. Each accessor wraps the corresponding upstream `PlaintextImpl::Get*/Set*` method and returns its value unchanged.

Usage

```
get_noise_scale_deg(x, ...)

get_length(x, ...)

get_level(x, ...)

get_scaling_factor(x, ...)

get_log_precision(x, ...)

get_formatted_values(x, ...)

set_ckks_data_type(x, ...)

get_encoding_type(x, ...)

get_scaling_factor_int(x, ...)

get_scheme_id(x, ...)

is_encoded(x, ...)

low_bound(x, ...)

high_bound(x, ...)

get_slots(x, ...)

get_log_error(x, ...)

get_coef_packed_value(x, ...)

get_string_value(x, ...)

get_element_ring_dimension(x, ...)

set_scaling_factor(x, ...)
```

```
set_scaling_factor_int(x, ...)

set_noise_scale_deg(x, ...)

set_level(x, ...)

set_slots(x, ...)

set_string_value(x, ...)

set_int_vector_value(x, ...)
```

Arguments

x	A Plaintext.
...	Reserved for future method-specific arguments. Setters accept a value argument here; get_formatted_values accepts a precision integer.

Details

Several base-class accessors are declared virtual and throw OPENFHE_THROW in the base implementation (e.g. get_string_value on a non-string plaintext, get_log_precision on a non-CKKS plaintext). Calling an accessor on the wrong kind of plaintext therefore raises an R error via cli::cli_abort, not a silent wrong value.

Value

The underlying field value. Types vary per accessor.

Functions

- get_noise_scale_deg: Integer noise-scale-degree of a CKKS plaintext. After construction the degree is the value passed to make_ckks_packed_plaintext(..., noise_scale_deg) under FIXEDMANUAL scaling; under FLEXIBLEAUTO the scheme overrides the user-supplied value at context-generation time (see discovery D011). Incremented by each multiplication before a rescale. The harness Signal 2 differential fixture for MakeCKKSPackedPlaintext reads this value to verify the noise_scale_deg argument reached the C++ call site.
- get_length: Integer effective length of a packed plaintext — the number of slots that hold user-supplied values. Defaults to the full batch size at construction; set_length() can shorten it for display or decryption purposes.
- get_level: Integer level of the plaintext in the RNS modulus chain. 0 for a fresh plaintext; incremented by each rescale() the plaintext survives. For CKKS-packed plaintexts the level must match the ciphertext level at every homomorphic operation, or the evaluator rejects the pair. The harness Signal 2 fixture reads this value to verify the level argument reached the C++ call site.

- get_scaling_factor: Numeric scaling factor for CKKS-based plaintexts — the multiplier the real vector is scaled by before encoding. Equals 2^scaling_mod_size for a freshly-encoded plaintext under FLEXIBLEAUTO; halves after each rescale. For BFV/BGV plaintexts this is returned as a default value (no rescaling applies).
- get_log_precision: Numeric log2 of the precision lost during CKKS encoding. Only meaningful for CKKS plaintexts; throws via OPENFHE_THROW for BFV/BGV and is surfaced as an R error by cli::cli_abort.
- get_formatted_values: String-format the plaintext's encoded values with precision decimal digits. Implemented on the concrete plaintext subclass (packed, CKKS, string, coefficient); throws on plaintexts whose subclass does not override.
- set_ckks_data_type: Set the CKKS data type to CKKSDataType$REAL or CKKSDataType$COMPLEX. Only meaningful for CKKS plaintexts. Most R vignettes use the default REAL type; set to COMPLEX only when you are constructing a CKKS plaintext from a complex vector.
- get_encoding_type: parity-deferred: integer encoding type (see PlaintextEncodings for the enum values).
- get_scaling_factor_int: parity-deferred: BGV integer scaling factor. Returned as a double carrying a losslessly rounded 53-bit integer per design.md §7.
- get_scheme_id: parity-deferred: scheme identifier (see SchemeId enum).
- is_encoded: parity-deferred: logical "has the plaintext been encoded yet?" The factory methods typically encode plaintexts eagerly, so this is TRUE for fresh plaintexts. Returns FALSE only for plaintexts constructed in a two-step uninitialised form.
- low_bound: parity-deferred: integer lower bound that can be encoded with the current plaintext modulus: -floor(t / 2).
- high_bound: parity-deferred: integer upper bound that can be encoded with the current plaintext modulus: floor(t / 2).
- get_slots: parity-deferred: integer CKKS slot count. For Plaintext dispatch this is the GetSlots() value set at construction. The harness Signal 2 fixture reads this value to verify the slots argument reached the C++ call site.
- get_log_error: parity-deferred: numeric log2 of the error estimate. Only meaningful for CKKS plaintexts; throws for BFV/BGV.
- get_coef_packed_value: parity-deferred: integer vector of the underlying coef-packed encoding. Throws for plaintexts whose subclass is not coef-packed.
- get_string_value: parity-deferred: string value of a string-encoded plaintext. Throws for plaintexts whose subclass is not string-encoded.
- get_element_ring_dimension: parity-deferred: integer ring dimension of the underlying Element. This is the plaintext's view of the lattice ring dimension; typically matches the crypto context's ring dimension.
- set_scaling_factor: parity-deferred: set the CKKS plaintext scaling factor.
- set_scaling_factor_int: parity-deferred: set the BGV plaintext integer scaling factor.
- set_noise_scale_deg: parity-deferred: set the plaintext noise scale degree. Most users should not call this directly — the factory methods and the evaluator manage noise_scale_deg automatically.
- set_level: parity-deferred: set the plaintext level. As with set_noise_scale_deg, most users should not call this directly.

- `set_slots`: `parity-deferred`: set the CKKS slot count. As with `set_length`, this is typically managed by the factory methods.
- `set_string_value`: `parity-deferred`: set the string value of a string-encoded plaintext. Throws for plaintexts whose subclass is not string-encoded.
- `set_int_vector_value`: `parity-deferred`: set the integer-vector value of an integer-encoded plaintext. Throws for plaintexts whose subclass does not support an integer vector.

`plaintext_params_hash` *Deterministic hash of a plaintext's identifying parameters*

Description

Combines the plaintext's encoding type, scaling factor, noise scale degree, level, and slot count into a deterministic string representation. Two plaintexts with identical parameters produce identical strings; any parameter change produces a different string.

Usage

```
plaintext_params_hash(x)
```

Arguments

`x`	A `Plaintext`.

Details

Named `plaintext_params_hash` in design.md §10 as the fourth "harness unblocker" for the Signal 2 differential fixture for `make_ckks_packed_plaintext`. The fixture calls `plaintext_params_hash()` on a default and a perturbed plaintext and expects different strings when the perturbation reaches the C++ call site.

Not a cryptographic hash — equality comparison is the only guarantee. The returned string's format is implementation detail and may change without notice.

Value

Character scalar.

`PlaintextEncodings` *Plaintext Encoding Types Source: pke/constants-defs.h enum PlaintextEncodings*

Description

Plaintext Encoding Types Source: pke/constants-defs.h enum PlaintextEncodings

Usage

```
PlaintextEncodings
```

PREMode *Proxy Re-encryption Mode Source: pke/constants-defs.h enum ProxyReEncryptionMode R-side name* PREMode *matches the design.md §8 shortening convention (same pattern as* Feature *for* PKESchemeFeature*).*

Description

Proxy Re-encryption Mode Source: pke/constants-defs.h enum ProxyReEncryptionMode R-side name PREMode matches the design.md §8 shortening convention (same pattern as Feature for PKESchemeFeature).

Usage

```
PREMode
```

PrivateKey *Private Key*

Description

Private Key

Usage

```
PrivateKey(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

PublicKey *Public Key*

Description

Public Key

Usage

```
PublicKey(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)

recover_shared_key *Recover a secret key from distributed shares*

Description

Inverse of share_keys(). Given a SecretShareMap holding at least threshold shares, reconstructs a PrivateKey equivalent to the original secret at the point of sharing. The reconstructed key participates in distributed decryption identically to the original, so a dropped-out party's share of a threshold decryption can still be completed by the remaining parties.

Usage

```
recover_shared_key(
  cc,
  share_map,
  n_parties,
  threshold,
  sharing_scheme = "additive"
)
```

Arguments

cc	A CryptoContext. Used to construct the empty placeholder key that the scheme routine fills in.
share_map	A SecretShareMap from share_keys().
n_parties	Integer; must match the value used at share_keys() time.
threshold	Integer; must match the value used at share_keys() time.
sharing_scheme	Character; must match the value used at share_keys() time.

Details

Under the hood, the C++ API takes a mutable PrivateKey reference that must be pre-allocated as an empty PrivateKeyImpl bound to cc. The R wrapper constructs that empty placeholder internally so R users do not have to know about the in-place-fill convention.

Value

A PrivateKey holding the reconstructed secret.

See Also

share_keys()

relinearize *Relinearize a higher-degree ciphertext*

Description

Reduces a ciphertext to 2 polynomial components. Needed after a sequence of `eval_mult_no_relin()` calls to restore a decryptable form. A relinearization key must have been generated via `key_gen(cc, eval_mult = TRUE)` before calling this.

Usage

```
relinearize(x, ...)
```

Arguments

`x`	A `Ciphertext` (may have more than 2 components).
`...`	Reserved for future method-specific arguments.

Value

A `Ciphertext` with exactly 2 components.

rescale *Rescale a CKKS ciphertext (alias for ModReduce)*

Description

Reduces the modulus chain by one level. Required after `eval_mult` under `FIXEDMANUAL` scaling; automatic under `FIXEDAUTO` / `FLEXIBLEAUTO`.

Usage

```
rescale(ct, ...)
```

Arguments

`ct`	A Ciphertext
`...`	Reserved for future method-specific arguments

Value

A Ciphertext at one lower level

ring_dimension *Ring dimension of a CryptoContext*

Description

Returns N, the cyclotomic ring dimension. The cyclotomic order M used by `eval_fast_rotation()` is `2 * N`.

Usage

```
ring_dimension(cc, ...)
```

Arguments

`cc`	A CryptoContext
`...`	Reserved for future method-specific arguments

Value

Integer ring dimension

ScalingTechnique *Scaling Techniques (CKKS) Source: pke/constants-defs.h enum ScalingTechnique*

Description

Scaling Techniques (CKKS) Source: pke/constants-defs.h enum ScalingTechnique

Usage

```
ScalingTechnique
```

SchemeId *Scheme Identifier*

Description

Returned by `get_scheme()` on any `CCParams` object. R-side name `SchemeId` matches the upstream `pke/scheme/scheme-id.h` header filename and avoids colliding with a future `Scheme` S7 class (design.md §6 mentions a potential wrapper around `std::shared_ptr<SchemeBase<DCRTPoly>>` with that name).

Usage

```
SchemeId
```

Details

Source: pke/scheme/scheme-id.h enum SCHEME

SecretKeyDist *Secret Key Distribution Source: core/lattice/constants-lattice.h enum SecretKeyDist*

Description

Secret Key Distribution Source: core/lattice/constants-lattice.h enum SecretKeyDist

Usage

```
SecretKeyDist
```

SecretShareMap *Map of secret-key shares for threshold-FHE abort recovery*

Description

Opaque S7 wrapper around a `shared_ptr<std::unordered_map<uint32_t, DCRTPoly>>`. Produced by `share_keys()` — each call returns one party's contribution to the distributed shares of their own secret key. Consumed by `recover_shared_key()`, which reconstructs the original secret from `threshold` or more shares when a party drops out.

Usage

```
SecretShareMap(ptr = NULL)
```

Arguments

ptr	External pointer (internal use).

Details

The map is keyed by party index (1-based uint32). Users do not index into it directly; it is a transport format for the secret-sharing protocol.

SecurityLevel *Security Levels Source: core/lattice/stdlatticeparms.h enum SecurityLevel*

Description

Security Levels Source: core/lattice/stdlatticeparms.h enum SecurityLevel

Usage

```
SecurityLevel
```

---

serialize_eval_keys *Serialize evaluation keys to file*

Description

Serialize evaluation keys to file

Usage

```
serialize_eval_keys(
  filename,
  type = c("mult", "automorphism", "sum"),
  format = "binary",
  key_tag = ""
)
```

Arguments

filename	Path to output file
type	"mult" or "automorphism"
format	"binary" (default) or "json"
key_tag	Key tag (default: "")

Value

TRUE on success (invisibly)

set_ckks_boot_correction_factor *Set the CKKS bootstrap correction factor*

Description

Sets the scheme-level correction factor used by subsequent bootstraps. Normally this is set via the `correction_factor` argument to `eval_bootstrap_setup()` at bootstrap setup time; this pair exists for post-setup programmatic control.

Usage

```
set_ckks_boot_correction_factor(cc, cf)
```

Arguments

`cc`	A `CryptoContext`.
`cf`	Integer; the new correction factor.

Value

The `cc`, invisibly.

See Also

`get_ckks_boot_correction_factor()`

set_key_gen_level *Set the key-generation level of a CryptoContext*

Description

Set the key-generation level of a CryptoContext

Usage

```
set_key_gen_level(cc, ...)
```

Arguments

`cc`	A `CryptoContext`.
`...`	Method-specific arguments; the `level` integer is passed here.

Value

`cc` invisibly.

set_length *Set the effective length of a plaintext*

Description

Set the effective length of a plaintext

Usage

```
set_length(pt, len)
```

Arguments

pt	A Plaintext
len	Integer length

share_keys *Distribute a secret key into shares*

Description

Produces the set of shares that party `index` would distribute to the other parties under the chosen `sharing_scheme`. The returned `SecretShareMap` is opaque; in a real deployment the shares would be serialized and routed over the network to each receiving party, and the receiving parties would store them for use in an abort recovery.

Usage

```
share_keys(cc, sk, n_parties, threshold, index, sharing_scheme = "additive")
```

Arguments

cc	A `CryptoContext` with the `MULTIPARTY` feature.
sk	The `PrivateKey` to share.
n_parties	Integer; total number of parties.
threshold	Integer; minimum number of shares needed to reconstruct. For `"additive"` this must be `n_parties - 1`; for `"shamir"` it is typically `floor(n_parties/2) + 1`.
index	Integer; the 1-based index of the party owning `sk` (the "my share index").
sharing_scheme	Character; either `"additive"` (default) or `"shamir"`.

Details

Two sharing schemes are supported:

- `"additive"` — N-1 threshold; every party must contribute their share to reconstruct. Robust against corruption of any single party's storage but not against any party dropping out.
- `"shamir"` — `floor(N/2) + 1` threshold; the secret can be reconstructed from any majority subset of the distributed shares. Robust against up to `floor((N-1)/2)` parties dropping out.

Value

A `SecretShareMap` suitable for passing to `recover_shared_key()`.

See Also

`recover_shared_key()`

`threshold_decrypt` *Threshold decryption convenience: lead + main + fusion in one call*

Description

Performs a full n-of-n threshold decryption of `ct` given the ordered list of party secret keys. The first key in `sks` is used for `multiparty_decrypt_lead()`; the remaining keys for `multiparty_decrypt_main()`; the resulting partials are then fused with `multiparty_decrypt_fusion()`.

Usage

```
threshold_decrypt(cc, sks, ct)
```

Arguments

`cc`	A CryptoContext
`sks`	A list of PrivateKey objects, lead first
`ct`	The Ciphertext to decrypt

Details

Use this when you have all secret keys in one place (testing, simulation, single-process demos). In a real distributed deployment each site holds only its own secret key and the partials travel over the network — that flow uses the lead / main / fusion functions directly.

Value

A Plaintext

`with_fhe_context` *Execute code with automatic cleanup of FHE state*

Description

Clears eval keys and releases contexts on exit (even on error).

Usage

```
with_fhe_context(expr)
```

Arguments

`expr`	Expression to evaluate

Value

Result of expr

Index