

Package: openfhe.R (via r-universe)

October 9, 2026

Title R Interface to the 'OpenFHE' Fully Homomorphic Encryption Library

Version 1.5.1.2

Description Provides an R interface to 'OpenFHE', the open-source C++ library for fully homomorphic encryption (Al Badawi and others, 2022) <<https://eprint.iacr.org/2022/915>>, which allows computation directly on encrypted data without access to the secret key. Supports the Brakerski-Fan-Vercauteren (BFV, 2012) <<https://eprint.iacr.org/2012/144>>, Brakerski-Gentry-Vaikuntanathan (BGV, 2014) <[doi:10.1145/2633600](https://eprint.iacr.org/2014/2633600)>, and Cheon-Kim-Kim-Song (CKKS, 2017) <<https://eprint.iacr.org/2016/421>> schemes for arithmetic on encrypted numbers, together with the Ducas-Micciancio (FHEW, 2015) <<https://eprint.iacr.org/2014/816>> and Chillotti-Gama-Georgieva-Izabachene (TFHE, 2020) <<https://eprint.iacr.org/2018/421>> schemes for evaluating arbitrary functions on encrypted bits.

License BSD_2_clause + file LICENSE

Copyright file inst/COPYRIGHTS

URL <https://openfheorg.github.io/openfhe.R/>,
<https://github.com/openfheorg/openfhe.R>

BugReports <https://github.com/openfheorg/openfhe.R/issues>

Depends R (>= 4.3.0)

Imports S7, cli, rlang, 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-extra.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 8.1.0

Config/pak/sysreqs cmake make

Repository <https://openfheorg.r-universe.dev>

Date/Publication 2026-10-06 13:45:15 UTC

RemoteUrl <https://github.com/openfheorg/openfhe.r>

RemoteRef HEAD

RemoteSha f88c5cc9b9b61b975ed4a5afdc877c6b6a33cc4d

Contents

openfhe.R-package	6
BFVParams	7
BGVParams	10
bin_bt_key_gen	12
bin_decrypt	12
bin_encrypt	13
bin_fhe_context	13
bin_key_gen	14
BinFHEContext	15
BinFHEMethod	15
BinFHEOutput	16
BinFHEParamSet	16
BinGate	17
ccparams_getters	17
Ciphertext	21
ckks_scaling_factor_bits	21
CKKSDataType	22
CKKSParams	22
clear_eval_automorphism_keys	24
clear_eval_mult_keys	25
clear_fhe_state	26
clear_static_maps_and_vectors	26
compress	27
CompressionLevel	27
CryptoContext	28
CryptoParameters	28
decrypt	29

DecryptionNoiseMode	30
deserialize_eval_keys	30
DistributionType	31
ElementParams	31
enable_feature	32
EncodingParams	32
encrypt	33
EncryptionTechnique	34
eval_add	34
eval_add_in_place	35
eval_add_mutable	35
eval_at_index_key_gen	36
eval_automorphism	36
eval_automorphism_key_gen	37
eval_bin_gate	38
eval_bootstrap	38
eval_bootstrap_key_gen	39
eval_bootstrap_setup	39
eval_chebyshev	40
eval_chebyshev_coefficients	41
eval_chebyshev_function	42
eval_chebyshev_linear	42
eval_chebyshev_ps	43
eval_cos	44
eval_divide	44
eval_fast_rotation	45
eval_fast_rotation_ext	45
eval_fast_rotation_precompute	46
eval_floor	46
eval_func	47
eval_logistic	48
eval_mult	48
eval_mult_and_relinearize	49
eval_mult_in_place	49
eval_mult_key_gen	50
eval_mult_mutable	50
eval_mult_no_relin	51
eval_negate	51
eval_negate_in_place	52
eval_not	52
eval_poly	53
eval_poly_linear	53
eval_poly_ps	54
eval_rotate	54
eval_rotate_key_gen	55
eval_sign	56
eval_sin	56
eval_square	57

eval_square_mutable	57
eval_sub	58
eval_sub_in_place	58
eval_sub_mutable	59
eval_sum	59
eval_sum_key_gen	60
EvalKey	60
EvalKeyMap	61
ExecutionMode	62
FastRotationPrecomputation	62
Feature	63
fhe_ckks_tolerance	63
fhe_context	64
fhe_deserialize	65
fhe_serialize	66
find_automorphism_index	66
find_automorphism_indices	67
generate_lut_via_function	68
get_all_eval_automorphism_keys	68
get_all_eval_mult_keys	69
get_all_eval_sum_keys	69
get_bootstrap_depth	70
get_ckks_boot_correction_factor	70
get_complex_packed_value	71
get_crypto_context	72
get_crypto_parameters	72
get_cyclotomic_order	73
get_element_params	73
get_encoding_params	74
get_eval_automorphism_key_map	74
get_eval_mult_key_vector	75
get_eval_sum_key_map	75
get_key_gen_level	76
get_max_plaintext_space	76
get_native_int	77
get_num_threads	77
get_packed_value	78
get_real_packed_value	78
get_scaling_factor_real	79
insert_eval_automorphism_key	79
insert_eval_mult_key	80
insert_eval_sum_key	81
int_boot_add	81
int_boot_adjust_scale	82
int_boot_decrypt	82
int_boot_encrypt	83
int_mp_boot_add	83
int_mp_boot_adjust_scale	84

int_mp_boot_decrypt	84
int_mp_boot_encrypt	85
int_mp_boot_random_element_gen	85
is_good	86
key_gen	86
key_switch_down	87
key_tag	88
KeygenMode	88
KeyPair	89
KeySwitchTechnique	89
level_reduce	90
level_reduce_in_place	90
LWECiphertext	91
LWEPrivateKey	91
make_ckks_packed_plaintext	92
make_coef_packed_plaintext	93
make_packed_plaintext	94
mod_reduce	95
mod_reduce_in_place	95
multi_add_eval_automorphism_keys	96
multi_add_eval_keys	96
multi_add_eval_mult_keys	97
multi_add_eval_sum_keys	97
multi_add_pub_keys	98
multi_eval_at_index_key_gen	98
multi_eval_automorphism_key_gen	99
multi_eval_sum_key_gen	99
multi_key_switch_gen	100
multiparty_decrypt_fusion	101
multiparty_decrypt_lead	101
multiparty_decrypt_main	102
multiparty_key_gen	103
MultipartyMode	103
MultiplicationTechnique	104
OpenFHEObject	104
Plaintext	105
plaintext_accessors	105
plaintext_params_hash	108
PlaintextEncodings	109
PREMode	109
PrivateKey	110
PublicKey	110
recover_shared_key	111
relinearize	112
rescale	112
ring_dimension	113
ScalingTechnique	113
SchemeId	114

SecretKeyDist	114
SecretShareMap	115
SecurityLevel	115
serialize_eval_keys	116
set_ckks_boot_correction_factor	117
set_key_gen_level	117
set_length	118
set_num_threads	118
share_keys	119
threshold_decrypt	120
with_fhe_context	121

Index	122
--------------	------------

openfhe.R-package	<i>openfhe.R: R Interface to the 'OpenFHE' Fully Homomorphic Encryption Library</i>
-------------------	---

Description

Provides an R interface to 'OpenFHE', the open-source C++ library for fully homomorphic encryption (Al Badawi and others, 2022) <https://eprint.iacr.org/2022/915>, which allows computation directly on encrypted data without access to the secret key. Supports the Brakerski-Fan-Vercauteren (BFV, 2012) <https://eprint.iacr.org/2012/144>, Brakerski-Gentry-Vaikuntanathan (BGV, 2014) [doi:10.1145/2633600](https://doi.org/10.1145/2633600), and Cheon-Kim-Kim-Song (CKKS, 2017) <https://eprint.iacr.org/2016/421> schemes for arithmetic on encrypted numbers, together with the Ducas-Micciancio (FHEW, 2015) <https://eprint.iacr.org/2014/816> and Chillotti-Gama-Georgieva-Izabachene (TFHE, 2020) <https://eprint.iacr.org/2018/421> schemes for evaluating arbitrary functions on encrypted bits.

Author(s)

Maintainer: Balasubramanian Narasimhan <naras@stanford.edu> ([ORCID](#)) [copyright holder]

Authors:

- Balasubramanian Narasimhan <naras@stanford.edu> ([ORCID](#)) [copyright holder]

Other contributors:

- New Jersey Institute of Technology (OpenFHE C++ library, inst/openfhe/src) [copyright holder]
- Duality Technologies, Inc. (OpenFHE C++ library, inst/openfhe/src) [contributor, copyright holder]
- Samuel Neves (BLAKE2 reference implementation, inst/openfhe/src/core) [contributor, copyright holder]
- Jean-Philippe Aumasson (BLAKE2X reference implementation, inst/openfhe/src/core) [contributor, copyright holder]
- Randolph Voorhies (cereal serialization library, inst/openfhe/cereal) [contributor, copyright holder]

- Shane Grant (cereal serialization library, inst/openfhe/cereal) [contributor, copyright holder]
- Juan Pedro Bolivar Puente (cereal std::optional and std::variant support) [contributor, copyright holder]
- THL A29 Limited, a Tencent company (RapidJSON, bundled in inst/openfhe/cereal) [copyright holder]
- Milo Yip (RapidJSON, bundled in inst/openfhe/cereal) [contributor, copyright holder]
- Alexander Chemeris (msinttypes, bundled in inst/openfhe/cereal) [contributor, copyright holder]
- Marcin Kalicinski (RapidXml, bundled in inst/openfhe/cereal) [contributor, copyright holder]
- Rene Nyffenegger (base64 encoder/decoder, bundled in inst/openfhe/cereal) [contributor, copyright holder]

See Also

Useful links:

- <https://openfheorg.github.io/openfhe.R/>
- <https://github.com/openfheorg/openfhe.R>
- Report bugs at <https://github.com/openfheorg/openfhe.R/issues>

BFVParams

BFV Parameters

Description

Constructor for the BFV scheme's CParams surface. Every argument maps 1:1 to an upstream CParams<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.

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 $\mathbb{Z}_t[x]/\Phi_m(x)$ 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	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	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	upstream noise-budget hint: maximum additions between multiplications. Used only by the noise-flooding path. Default 0.
key_switch_count	upstream noise-budget hint: maximum key-switch count. Default 0.
encryption_technique	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.

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	see the BFV entry.
pre_mode	see the BFV entry.
statistical_security	statistical security parameter (bits), used by the noise-flooding decryption path.
num_adversarial_queries	upper bound on the number of adversarial queries the noise-flooding path must survive.
eval_add_count	see the BFV entry.
key_switch_count	see the BFV entry.
pre_num_hops	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	<i>Generate BinFHE bootstrapping keys</i>
----------------	---

Description

Generate BinFHE bootstrapping keys

Usage

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

Arguments

ctx	A BinFHE context
sk	An LWESPrivateKey
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.

Value

Invisibly, the ctx BinFHE context that was passed in. Called for its side effect: the bootstrapping and key-switching keys are generated and stored inside the C++ context, after which [eval_bin_gate\(\)](#) and the other gate evaluations can be used.

bin_decrypt	<i>Decrypt a Binary FHE ciphertext</i>
-------------	--

Description

Decrypt a Binary FHE ciphertext

Usage

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

Arguments

ctx	A BinFHE context
sk	An LWESPrivateKey
ct	An LWECiphertext
p	Plaintext modulus (default: 4)

Value

Integer value

bin_encrypt	<i>Encrypt a value for Binary FHE</i>
-------------	---------------------------------------

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 LWEPriateKey
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	<i>Create a Binary FHE context</i>
-----------------	------------------------------------

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 (Eval-Sign, 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](#) object.

bin_key_gen	<i>Generate BinFHE secret key</i>
-------------	-----------------------------------

Description

Generate BinFHE secret key

Usage

bin_key_gen(ctx)

Arguments

ctx	A BinFHE context
-----	------------------

Value

An LWESecretKey

BinFHEContext	<i>BinFHE context (Binary FHE)</i>
---------------	------------------------------------

Description

BinFHE context (Binary FHE)

Usage

```
BinFHEContext(ptr = NULL)
```

Arguments

ptr	External pointer (internal use)
-----	---------------------------------

Value

An S7 object of class BinFHEContext, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ boolean-circuit context. It is the BinFHE counterpart of [CryptoContext](#) and is deliberately a different class, so a [CryptoContext](#) cannot be passed where a BinFHE context belongs or the other way round. Obtain one from [bin_fhe_context\(\)](#) rather than by calling this constructor directly.

BinFHEMethod	<i>Binary FHE Methods</i>
--------------	---------------------------

Description

Mirrors the C++ enum BINFHE_METHOD in binfhe/binfhe-constants.h.

Usage

```
BinFHEMethod
```

Value

A named list of 4 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, selecting the bootstrapping method a boolean-circuit context uses: AP (Alperin-Sheriff-Peikert), GINX (Gama-Izabachene-Nguyen-Xie, the default) or LMKCDEY. INVALID_METHOD marks an unset value. Pass an element as the method argument of [bin_fhe_context\(\)](#).

BinFHEOutput

Binary FHE Output Types

Description

Mirrors the C++ enum BINFHE_OUTPUT in binfhe/binfhe-constants.h.

Usage

BinFHEOutput

Value

A named list of 5 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, describing the form the ciphertext produced by a BinFHE operation should take: FRESH (noise reset to the level of a fresh encryption), BOOTSTRAPPED, LARGE_DIM and SMALL_DIM (the LWE dimension the result is expressed in). INVALID_OUTPUT marks an unset value. Pass an element as the output argument of [bin_encrypt\(\)](#).

BinFHEParamSet

Binary FHE Parameter Sets

Description

Mirrors the C++ enum BINFHE_PARAMSET in binfhe/binfhe-constants.h, whose enumerators are numbered sequentially from 0.

Usage

BinFHEParamSet

Value

A named list of 44 integer scalars, each the value of the OpenFHE C++ enumerator of the same name. An element names a pre-tabulated set of lattice parameters for the boolean-circuit (BinFHE) schemes: TOY and MEDIUM are insecure sizes for experimentation, and the STD128, STD192 and STD256 families give the corresponding bits of security, with the Q suffix denoting a larger ciphertext modulus, the _3 and _4 suffixes 3- and 4-input gates, LMKDEY the Lee-Micciancio-Kim-Choi-Deryabin-Eom-Yoo bootstrapping method and LPF a low-probability-of-failure variant. Pass an element as the paramset argument of [bin_fhe_context\(\)](#).

BinGate

*Binary Gate Types***Description**

Mirrors the C++ enum BINGATE in binfhe/binfhe-constants.h, whose enumerators are numbered sequentially from 0.

Usage

BinGate

Value

A named list of 14 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, naming the boolean gate to evaluate on encrypted bits: the two-input gates OR, AND, NOR, NAND, XOR, XNOR and the faster XOR_FAST, XNOR_FAST; the three- and four-input AND3, OR3, AND4, OR4; and MAJORITY and CMUX. Pass an element as the gate argument of [eval_bin_gate\(\)](#).

ccparams_getters

*CCParams getters (all schemes)***Description**

Retrieve a parameter from a BFVParams, BGVPParams, or CKKSPParams 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_multipart_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

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 $\mathbb{Z}_t[x]/\Phi_m(x)$ 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.
- `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/COMPOSITESCALINGMANUAL scaling techniques. Only meaningful when `scaling_technique = ScalingTechnique$COMPOSITESCALING*`; 0 under the default FLEXIBLEAUTO path.
- `get_scheme`: integer scheme identifier (see SchemeId for the enum values).
- `get_standard_deviation`: standard deviation of the Gaussian error distribution.
- `get_secret_key_dist`: secret-key distribution (see SecretKeyDist).
- `get_max_relin_sk_deg`: maximum relinearization secret-key degree.
- `get_pre_mode`: proxy re-encryption mode (see PREMode).
- `get_multipart_mode`: multipart mode (see MultipartMode).
- `get_execution_mode`: execution mode (see ExecutionMode).
- `get_decryption_noise_mode`: decryption noise mode (see DecryptionNoiseMode).

- `get_noise_estimate`: noise-flooding noise estimate (double).
- `get_desired_precision`: noise-flooding target precision (double, bits).
- `get_statistical_security`: statistical security parameter. Return type is double per header inconsistency; see the Return-type note.
- `get_num_adversarial_queries`: upper bound on adversarial queries. Return type is double per header inconsistency; see the Return-type note.
- `get_threshold_num_of_parties`: threshold-FHE party count.
- `get_key_switch_technique`: key-switching technique (see `KeySwitchTechnique`).
- `get_scaling_technique`: scaling technique (see `ScalingTechnique`).
- `get_batch_size`: SIMD batch size.
- `get_first_mod_size`: bit size of the first (largest) prime in the CKKS modulus chain.
- `get_num_large_digits`: number of large digits for HYBRID key switching.
- `get_multiplicative_depth`: 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`: bit size of each CKKS scaling modulus.
- `get_security_level`: target security level (see `SecurityLevel`).
- `get_eval_add_count`: BFV/BGV noise-flooding hint: maximum additions between multiplications.
- `get_key_switch_count`: BFV/BGV noise-flooding hint: maximum key-switch count.
- `get_encryption_technique`: BFV encryption technique (see `EncryptionTechnique`).
- `get_multiplication_technique`: BFV multiplication technique (see `MultiplicationTechnique`).
- `get_pre_num_hops`: PRE hop count.
- `get_interactive_boot_compression_level`: CKKS interactive bootstrap compression level (see `CompressionLevel`).
- `get_register_word_size`: register word size for multi-precision arithmetic.
- `get_ckks_data_type`: 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.

Ciphertext	<i>Ciphertext class</i>
------------	-------------------------

Description

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

Usage

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

Arguments

<code>ptr</code>	External pointer (internal use)
------------------	---------------------------------

Value

An S7 object of class `Ciphertext`, inheriting from [OpenFHEObject](#), whose `ptr` property holds an external pointer to the C++ ciphertext. Ciphertexts are produced by [encrypt\(\)](#) and by the `eval_*`() family rather than by calling this constructor directly, and they carry the encrypted vector together with the level and scaling-factor bookkeeping the scheme needs.

<code>ckks_scaling_factor_bits</code>	<i>CKKS scaling-mod-size in bits</i>
---------------------------------------	--------------------------------------

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

<code>cc</code>	A <code>CryptoContext</code> (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

Description

Mirrors the C++ enum CKKSDataType in pke/constants-defs.h.

Usage

CKKSDataType

Value

A named list of 2 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, declaring whether the slots of a CKKS ciphertext hold REAL or COMPLEX numbers.

CKKSParams

CKKS Parameters

Description

Constructor for the CKKS scheme's CParams surface. Every argument maps 1:1 to an enabled CParams<CryptoContextCKKS>::Set* method. The 8 CKKS-disabled setters (SetPlaintextModulus, SetEvalAddCount, SetKeySwitchCount, SetEncryptionTechnique, SetMultiplicationTechnique, SetPREnumHops, SetMultipartyMode, SetThresholdNumOfParties) are not exposed. 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	see the BFV entry.
pre_mode	see the BFV entry.
execution_mode	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	one of DecryptionNoiseMode\$FIXED_NOISE_DECRYPT (default) or NOISE_FLOODING_DECRYPT.
noise_estimate	numeric noise estimate used by the noise-flooding path. Paired with execution_mode = EXEC_NOISE_ESTIMATION.
desired_precision	numeric target precision (in bits) for the noise-flooding path.
statistical_security	see the BGV entry.
num_adversarial_queries	see the BGV entry.
composite_degree	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	<i>Clear the EvalMult key cache</i>
----------------------	-------------------------------------

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	<i>Clear cached evaluation keys and contexts</i>
-----------------	--

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"
------	--

Value

No return value, called for side effects. Depending on what, it drops the process-wide caches of relinearization keys and of automorphism (rotation) keys, and releases the registry of crypto contexts, freeing the memory they hold.

clear_static_maps_and_vectors	<i>Clear OpenFHE static maps and vectors</i>
-------------------------------	--

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	<i>Compress a ciphertext to fewer towers</i>
----------	--

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.

Usage

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

Arguments

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

Value

A compressed Ciphertext.

CompressionLevel	<i>Compression Level (interactive multi-party bootstrap)</i>
------------------	--

Description

Mirrors the C++ enum `CompressionLevel` in `pke/constants-defs.h`. Note that the values start at 2, not 0: the header explains that compression levels 0 and 1 are not supported and that the remaining values are deliberately not renumbered.

Usage

```
CompressionLevel
```

Value

A named list of 2 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, naming how far a ciphertext is compressed before the interactive bootstrapping round-trip: `COMPACT` (2) sends the least data but is only safe when the result is decrypted immediately, while `SLACK` (3) leaves room for further computation on the result. Pass an element as the `interactive_boot_compression_level` argument of `CKKSParams()` or `fhe_context()`.

CryptoContext	<i>Crypto Context</i>
---------------	-----------------------

Description

Crypto Context

Usage

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

Arguments

ptr	External pointer (internal use)
-----	---------------------------------

Value

An S7 object of class `CryptoContext`, inheriting from [OpenFHEObject](#), whose `ptr` property holds an external pointer to the C++ crypto context. The context owns the scheme parameters and the registry of evaluation keys, and is the first argument to most operations. Obtain one from [fhe_context\(\)](#) rather than by calling this constructor directly.

CryptoParameters	<i>Crypto Parameters (opaque)</i>
------------------	-----------------------------------

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)
-----	---------------------------------

Value

An S7 object of class `CryptoParameters`, inheriting from [OpenFHEObject](#), whose `ptr` property holds an external pointer to the C++ `CryptoParametersBase<DCRTPoly>`. It is an opaque token: its contents are read with the RNS-level accessors rather than from R, and it is obtained from `get_crypto_parameters()` rather than by calling this constructor directly.

decrypt

*Decrypt a ciphertext***Description**

decrypt dispatches on both arguments and accepts them in either order, as the C++ header does: Decrypt(ciphertext, privateKey) is the primary form and Decrypt(privateKey, ciphertext) forwards to it (cryptocontext.h, both overloads). The key-first order lets code that is written around the holder of the key, such as a party object in a protocol, put that holder first, the same way [encrypt\(\)](#) puts the key first.

Usage

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

Arguments

ct	A Ciphertext, or a PrivateKey when the key-first order is used.
key	A PrivateKey, or a Ciphertext when the key-first order is used.
...	Additional arguments (cc = CryptoContext)

Value

A Plaintext

Examples

```
cc <- fhe_context("CKKS", multiplicative_depth = 2L,
                  scaling_mod_size = 50L, batch_size = 8L)
kp <- key_gen(cc, eval_mult = TRUE)
pt <- make_ckks_packed_plaintext(cc, c(0.25, 0.5, 0.75, 1))
ct <- encrypt(kp@public, pt, cc = cc)

## Add the encrypted vector to itself, then decrypt. Decryption
## returns a Plaintext padded out to the full slot count, so trim it
## back to the length that was encoded.
out <- decrypt(eval_add(ct, ct), kp@secret, cc = cc)
out <- set_length(out, 4L)
get_real_packed_value(out)

## The same call with the key first:
out2 <- decrypt(kp@secret, eval_add(ct, ct), cc = cc)
get_real_packed_value(set_length(out2, 4L))
```

DecryptionNoiseMode	<i>Decryption Noise Mode</i>
---------------------	------------------------------

Description

Mirrors the C++ enum DecryptionNoiseMode in pke/constants-defs.h.

Usage

DecryptionNoiseMode

Value

A named list of 2 integer scalars, each the value of the OpenFHE C++ enumerator of the same name: FIXED_NOISE_DECRYPT decrypts with a fixed noise estimate, while NOISE_FLOODING_DECRYPT floods the result with extra noise so that the decryption itself leaks nothing about the circuit. Pass an element as the decryption_noise_mode argument of [CKKSParams\(\)](#) or [fhe_context\(\)](#).

deserialize_eval_keys	<i>Deserialize evaluation keys from file</i>
-----------------------	--

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	<i>Distribution Type (lattice parameters)</i>
------------------	---

Description

Mirrors the C++ enum DistributionType in core/lattice/stdlatticeparms.h.

Usage

DistributionType

Value

A named list of 3 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, naming the secret distribution a row of the HomomorphicEncryption.org standard parameter tables applies to: HESTd_uniform, HESTd_error or HESTd_ternary.

ElementParams	<i>Element Parameters (opaque)</i>
---------------	------------------------------------

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)

Value

An S7 object of class ElementParams, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ DCRTPoly::Params. It is an opaque token describing the ring the polynomials live in, obtained from get_element_params() and passed on to the CKKS plaintext factories.

enable_feature	<i>Enable a feature on a CryptoContext</i>
----------------	--

Description

Accepts either a single PKESchemeFeature enum value or a uint32_t bitwise-OR mask of PKESchemeFeature values (for matching 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\$LEVELEDSE, i.e. 11L).

Value

cc invisibly.

EncodingParams	<i>Encoding Parameters (opaque)</i>
----------------	-------------------------------------

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)
-----	---------------------------------

Value

An S7 object of class `EncodingParams`, inheriting from [OpenFHEObject](#), whose `ptr` property holds an external pointer to the C++ `EncodingParamsImpl`. It is an opaque token describing how values are packed into a plaintext, obtained from `get_encoding_params()` rather than by calling this constructor directly.

 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

<code>key</code>	A <code>PublicKey</code> or <code>PrivateKey</code> .
<code>pt</code>	A <code>Plaintext</code> .
<code>...</code>	Additional arguments (<code>cc = CryptoContext</code> is required).

Value

A `Ciphertext`.

Examples

```
cc <- fhe_context("CKKS", multiplicative_depth = 2L,
                  scaling_mod_size = 50L, batch_size = 8L)
kp <- key_gen(cc, eval_mult = TRUE)
pt <- make_ckks_packed_plaintext(cc, c(0.5, 1.5, 2.5, 3.5))

## Public-key encryption -- the usual path, and the one that lets a
## party encrypt without holding the secret key:
ct <- encrypt(kp@public, pt, cc = cc)
ct

## Symmetric encryption with the secret key, for protocols where the
## same party encrypts and decrypts:
ct2 <- encrypt(kp@secret, pt, cc = cc)
ct2
```

EncryptionTechnique	<i>Encryption Technique</i>
---------------------	-----------------------------

Description

Mirrors the C++ enum EncryptionTechnique in pke/constants-defs.h.

Usage

EncryptionTechnique

Value

A named list of 2 integer scalars, each the value of the OpenFHE C++ enumerator of the same name: STANDARD encrypts in the ciphertext modulus, while EXTENDED encrypts in an enlarged modulus, which lowers the noise BFV multiplication starts from. Pass an element as the encryption_technique argument of [BFVParams\(\)](#) or [fhe_context\(\)](#).

eval_add	<i>Homomorphic addition</i>
----------	-----------------------------

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	<i>Homomorphic addition in place</i>
-------------------	--------------------------------------

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	<i>Homomorphic add, mutable variant</i>
------------------	---

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 both names are provided to mirror the C++ API.

Usage

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

Arguments

<code>cc</code>	A <code>CryptoContext</code> .
<code>sk</code>	A <code>PrivateKey</code> whose tag will be used to key the generated rotation keys in the cc's internal registry.
<code>index_list</code>	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

<code>ct</code>	A <code>Ciphertext</code> .
<code>index</code>	Integer; the automorphism index (must match one of the indices passed to eval_automorphism_key_gen()).
<code>eval_key_map</code>	An <code>EvalKeyMap</code> 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 mirrors the equivalent C++ 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	<i>Evaluate a binary gate on encrypted values</i>
---------------	---

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	<i>Perform CKKS bootstrapping</i>
----------------	-----------------------------------

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

Value

Invisibly, the cc crypto context that was passed in. Called for its side effect: the rotation and evaluation keys bootstrapping needs are generated from sk and stored inside the C++ context, after which [eval_bootstrap\(\)](#) can be called.

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 BSlotsEncoding tail argument on cryptocontext.h line 3513). Default FALSE matching the C++ default.

Value

Invisibly, the cc crypto context that was passed in. Called for its side effect: the linear transforms bootstrapping needs are precomputed and cached inside the C++ context. Call it before [eval_bootstrap_key_gen\(\)](#).

eval_chebyshev	<i>Evaluate a Chebyshev series on a ciphertext</i>
----------------	--

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

<code>func</code>	An R function taking a single numeric argument and returning a numeric scalar.
<code>a</code>	Lower bound of the approximation interval.
<code>b</code>	Upper bound of the approximation interval.
<code>degree</code>	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	<i>Evaluate a Chebyshev series via the Paterson-Stockmeyer method</i>
-------------------	---

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	<i>Evaluate cosine on a ciphertext</i>
----------	--

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

Value

A [Ciphertext](#) holding the encrypted, slot-wise cosine of ct, approximated by a Chebyshev polynomial of the given degree on the interval [a, b]. Accuracy degrades outside that interval, and the result sits $\text{ceiling}(\log_2(\text{degree})) + 1$ levels below ct.

eval_divide	<i>Evaluate division approximation on a ciphertext</i>
-------------	--

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

Value

A [Ciphertext](#) holding the encrypted, slot-wise reciprocal of ct, approximated by a Chebyshev polynomial of the given degree on the interval [a, b]. The interval must exclude zero, accuracy degrades outside it, and the result sits $\text{ceiling}(\log_2(\text{degree})) + 1$ levels below ct.

eval_fast_rotation	<i>Hoisted slot rotation using precomputed digits</i>
--------------------	---

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 $2 * \text{ring_dimension}(\text{ctx})$), precomp (a FastRotationPrecomputation from eval_fast_rotation_precompute())

Value

A Ciphertext

eval_fast_rotation_ext	<i>Extended hoisted slot rotation</i>
------------------------	---------------------------------------

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

<code>ct</code>	A Ciphertext
<code>...</code>	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\$ <code>LARGE_DIM</code>
lut	A numeric vector of length <code>p</code> (the plaintext modulus) typically produced by <code>generate_lut_via_function()</code>

Value

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

eval_logistic	<i>Evaluate logistic function on a ciphertext</i>
---------------	---

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

Value

A [Ciphertext](#) holding the encrypted, slot-wise logistic function $1 / (1 + \exp(-x))$ of ct, approximated by a Chebyshev polynomial of the given degree on the interval $[a, b]$. Accuracy degrades outside that interval, and the result sits $\text{ceiling}(\log_2(\text{degree})) + 1$ levels below ct.

eval_mult	<i>Homomorphic multiplication</i>
-----------	-----------------------------------

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

<code>x</code>	A Ciphertext.
<code>...</code>	Method-specific arguments: <code>y</code> — a Ciphertext to multiply <code>x</code> 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

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

Value

`x` invisibly.

eval_mult_key_gen	<i>Generate relinearization (eval-mult) keys for a secret key</i>
-------------------	---

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 $\times$ ciphertext multiplication can be relinearized.

Usage

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

Arguments

cc	A <code>CryptoContext</code> .
sk	A <code>PrivateKey</code> 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	<i>Homomorphic multiply, mutable variant</i>
-------------------	--

Description

Homomorphic multiply, mutable variant

Usage

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

Arguments

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

Value

A new Ciphertext holding $x * y$.

eval_mult_no_relin	<i>Homomorphic multiplication without relinearization</i>
--------------------	---

Description

Returns the raw product of two ciphertexts as a higher-degree ciphertext (the result has $n_1 + n_2 - 1$ polynomial components where the inputs had n_1 and n_2). 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

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

Value

A Ciphertext at higher polynomial degree.

eval_negate	<i>Homomorphic negation</i>
-------------	-----------------------------

Description

Homomorphic negation

Usage

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

Arguments

<code>x</code>	A Ciphertext
<code>...</code>	Method-specific arguments

Value

A Ciphertext

eval_negate_in_place	<i>Homomorphic negation in place</i>
----------------------	--------------------------------------

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	<i>Evaluate NOT on an encrypted value</i>
----------	---

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	<i>Evaluate a polynomial on a ciphertext</i>
-----------	--

Description

Evaluates $p(x) = c_0 + c_1x + c_2x^2 + \dots$ 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	<i>Evaluate a polynomial via the linear evaluator</i>
------------------	---

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	<i>Generate rotation keys for a secret key</i>
---------------------	--

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 <code>CryptoContext</code> .
sk	A <code>PrivateKey</code> 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	<i>Evaluate sign on an encrypted value (functional bootstrapping)</i>
-----------	---

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

<code>ctx</code>	A BinFHE context built with <code>arb_func = TRUE</code>
<code>ct</code>	An LWE Ciphertext encrypted via <code>bin_encrypt(..., mod = Q)</code>
<code>scheme_switch</code>	Logical; when <code>TRUE</code> , the output ciphertext is encoded compatibly with the CKKS<->FHEW scheme-switching pipeline (the <code>schemeSwitch</code> flag at <code>binfhecontext.h</code> line 367). Default <code>FALSE</code> 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 LWE Ciphertext encrypting 0 if the input was negative (i.e. lay in the upper half of $[0, Q)$), 1 otherwise

eval_sin	<i>Evaluate sine on a ciphertext (Chebyshev approximation)</i>
----------	--

Description

Evaluate sine on a ciphertext (Chebyshev approximation)

Usage

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

Arguments

<code>ct</code>	A Ciphertext
<code>a</code>	Lower bound of the approximation interval
<code>b</code>	Upper bound of the approximation interval
<code>degree</code>	Chebyshev polynomial degree

Value

A Ciphertext

eval_square	<i>Homomorphic squaring</i>
-------------	-----------------------------

Description

Homomorphic squaring

Usage

eval_square(x, ...)

Arguments

- x A Ciphertext
- ... Method-specific arguments

Value

A Ciphertext

eval_square_mutable	<i>Homomorphic square, mutable variant</i>
---------------------	--

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	<i>Homomorphic subtraction</i>
----------	--------------------------------

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	<i>Homomorphic subtraction in place</i>
-------------------	---

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	<i>Homomorphic subtract, mutable variant</i>
------------------	--

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	<i>Sum all slots in a ciphertext</i>
----------	--------------------------------------

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	<i>Generate sum keys for a secret key</i>
------------------	---

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

Value

NULL, invisibly.

EvalKey	<i>EvalKey class for multi-party key operations</i>
---------	---

Description

EvalKey class for multi-party key operations

Usage

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

Arguments

ptr	External pointer (internal use)
-----	---------------------------------

Value

An S7 object of class EvalKey, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ EvalKey<DCRTPoly>. An evaluation key is the public material that lets the computing party carry out one operation — a relinearization, a rotation, or a re-encryption — without the secret key. Obtain one from [multi_key_switch_gen\(\)](#) or the rest of the multi_*() family rather than by calling this constructor directly.

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.

Value

An S7 object of class EvalKeyMap, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ `std::map<uint32_t, EvalKey<DCRTPoly>>`. It carries one evaluation key per rotation or automorphism index and acts as the transport format for the multi-party key protocols; it is produced by the `multi_eval_*_key_gen()` family rather than by calling this constructor directly.

ExecutionMode	<i>Execution Mode</i>
---------------	-----------------------

Description

Mirrors the C++ enum ExecutionMode in pke/constants-defs.h.

Usage

ExecutionMode

Value

A named list of 2 integer scalars, each the value of the OpenFHE C++ enumerator of the same name: EXEC_EVALUATION runs the computation normally, while EXEC_NOISE_ESTIMATION runs it only to measure the noise growth, which is the first of the two passes needed when decryption uses noise flooding. Pass an element as the execution_mode argument of [CKKSParams\(\)](#) or [fhe_context\(\)](#).

FastRotationPrecomputation	<i>Precomputed digit decomposition for hoisted rotations</i>
----------------------------	--

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)

Value

An S7 object of class FastRotationPrecomputation, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ digit decomposition. It caches the work that is common to every rotation of one source ciphertext; obtain one from [eval_fast_rotation_precompute\(\)](#) and pass it to [eval_fast_rotation\(\)](#) rather than calling this constructor directly.

Feature	<i>PKE Scheme Features (bitmask)</i>
---------	--------------------------------------

Description

Mirrors the C++ enum `PKESchemeFeature` in `pke/constants-defs.h`. The values are bit flags, so several may be combined with `bitwOr()` to describe a set of capabilities.

Usage

Feature

Value

A named list of 8 integer scalars. Each element carries the integer value the OpenFHE C++ enumerator of the same name has, and names a capability of a crypto context: `PKE` (encryption and decryption), `KEYSWITCH`, `PRE` (proxy re-encryption), `LEVELED SHE` (leveled homomorphic arithmetic), `ADVANCED SHE`, `MULTIPARTY`, `FHE` (bootstrapping) and `SCHEMESWITCH`. Pass an element to `enable_feature()` to turn that capability on for a context.

<code>fhe_ckks_tolerance</code>	<i>Per-test CKKS precision tolerance</i>
---------------------------------	--

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

- | | |
|------------------|---|
| <code>x</code> | 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. |
| <code>...</code> | Method-specific arguments. The numeric method accepts <code>scaling_factor_bits</code> (integer bit size of the scaling modulus, typical values 50/59/78), <code>scaling_technique</code> (a character string like "FLEXIBLEAUTO" or an integer from the <code>ScalingTechnique</code> enum), and <code>k</code> (conservative multiplicative factor, default 8). The Ciphertext method accepts only <code>k</code> . |

Details

Two dispatch forms:

- **Stage 1 (numeric)**: pass parameters as direct arguments. Useful when the ciphertext isn't yet constructed — e.g. when you need 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 the parameters directly:
fhe_ckks_tolerance(4L, 50L, "FLEXIBLEAUTO")

## Stage 2 -- pass a ciphertext and let it read the parameters
## off the associated context:
cc <- fhe_context("CKKS", multiplicative_depth = 4L,
                  scaling_mod_size = 50L, batch_size = 8L)
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 = cc)
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 <code>CCParams</code> setter arguments. Forwarded to <code>BFVParams()</code> , <code>BGVParams()</code> , or <code>CKKSParams()</code> . Example: <code>fhe_context("BFV", plaintext_modulus = 65537, multiplicative_depth = 2)</code> or <code>fhe_context("CKKS", multiplicative_depth = 4, scaling_mod_size = 50, scaling_technique = ScalingTechnique\$FLEXIBLEAUTO)</code> .

features	Additional Feature values to enable on the context beyond the default PKE KEYSWITCH LEVELED SHE 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. 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\(\)](#)

Examples

```
## An exact context over the integers modulo 65537 (BFV). The ring
## dimension is chosen for you from the security level.
bfv <- fhe_context("BFV", plaintext_modulus = 65537L,
                  multiplicative_depth = 2L)
ring_dimension(bfv)

## An approximate context over the reals (CKKS), which is what the
## statistical applications use.
ckks <- fhe_context("CKKS", multiplicative_depth = 2L,
                   scaling_mod_size = 50L, batch_size = 8L)
ring_dimension(ckks)
```

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	<i>Serialize an OpenFHE object to file</i>
---------------	--

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	<i>Compute the automorphism index for a single slot index</i>
-------------------------	---

Description

Maps a CKKS slot index to the corresponding automorphism index in the cyclotomic ring $\mathbb{Z}[X]/(X^N + 1)$. The automorphism group of the ring is isomorphic to the multiplicative group $(\mathbb{Z}/2N)^*$; 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).

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.

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

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

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_key_map\(\)](#), both share backing storage on the C++ side, but the two accessors are exposed separately so callers 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	<i>Get the required multiplicative depth for CKKS bootstrapping</i>
---------------------	---

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	<i>Get the CKKS bootstrap correction factor</i>
---------------------------------	---

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	<i>Associated CryptoContext of a Ciphertext</i>
--------------------	---

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

<code>ct</code>	A <code>Ciphertext</code> .
<code>...</code>	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	<i>Crypto parameters of a CryptoContext</i>
-----------------------	---

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

<code>cc</code>	A <code>CryptoContext</code> .
<code>...</code>	Reserved for future method-specific arguments.

Value

A CryptoParameters S7 object.

get_cyclotomic_order	<i>Cyclotomic order of a CryptoContext</i>
----------------------	--

Description

Integer m such that the underlying polynomial ring is $\mathbb{Z}[x]/(x^n + 1)$ with $n = m/2$ (the ring dimension). Always $2 * \text{ring_dimension}(cc)$ 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	<i>Element parameters of a CryptoContext</i>
--------------------	--

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	<i>Encoding parameters of a CryptoContext</i>
---------------------	---

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	<i>Retrieve the automorphism-key map for a given key tag</i>
-------------------------------	--

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`.

<code>get_key_gen_level</code>	<i>Key-generation level of a <code>CryptoContext</code></i>
--------------------------------	---

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

<code>cc</code>	A <code>CryptoContext</code> .
<code>...</code>	Reserved for future method-specific arguments.

Value

Integer.

<code>get_max_plaintext_space</code>	<i>Maximum supported plaintext space for functional bootstrapping</i>
--------------------------------------	---

Description

Maximum supported plaintext space for functional bootstrapping

Usage

```
get_max_plaintext_space(ctx)
```

Arguments

<code>ctx</code>	A BinFHE context built with <code>arb_func = TRUE</code>
------------------	--

Value

A numeric scalar ($q / (2 * \text{beta})$)

`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_num_threads`*Report the number of OpenMP threads available to OpenFHE*

Description

Returns the cap OpenFHE currently applies to its parallel regions: the number of hardware threads at load, or the value most recently passed to `set_num_threads()`. Returns 1 when the package was built without OpenMP.

Usage

```
get_num_threads()
```

Value

integer; the current thread cap.

See Also

[set_num_threads\(\)](#)

Examples

```
get_num_threads()
```

get_packed_value	<i>Get packed integer values from a plaintext</i>
------------------	---

Description

Get packed integer values from a plaintext

Usage

```
get_packed_value(pt)
```

Arguments

pt	A Plaintext
----	-------------

Value

Integer vector

get_real_packed_value	<i>Get real values from a CKKS plaintext</i>
-----------------------	--

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

<code>cc</code>	A <code>CryptoContext</code> .
<code>...</code>	Method-specific arguments. The <code>CryptoContext</code> method accepts <code>level</code> (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 $\log_2$ 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	<i>Insert a joined sum-key map into the cc registry</i>
---------------------	---

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 <code>multi_add_eval_sum_keys()</code> 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	<i>Combine encrypted and unencrypted masked decryptions</i>
--------------	---

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 <code>int_boot_decrypt()</code> and <code>int_boot_encrypt()</code> .
----------	---

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	<i>Client-side re-encryption for interactive bootstrap</i>
------------------	--

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	<i>Aggregate multi-party shares pairs</i>
-----------------	---

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 <code>int_mp_boot_add()</code> .
a	The common random element Ciphertext used in the per-party <code>int_mp_boot_decrypt()</code> 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	<i>Is a KeyPair valid?</i>
---------	----------------------------

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	<i>Generate key pair</i>
---------	--------------------------

Description

Generate key pair

Usage

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

Arguments

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

Value

A KeyPair

Examples

```
cc <- fhe_context("CKKS", multiplicative_depth = 2L,
                  scaling_mod_size = 50L, batch_size = 8L)

## Encryption keys only:
kp <- key_gen(cc)
kp

## Also generate the relinearization key homomorphic multiplication
## needs and the rotation keys for shifts by one slot either way.
## These are stored inside the context, not in the returned pair.
kp <- key_gen(cc, eval_mult = TRUE, rotations = c(1L, -1L))
kp@public
kp@secret
```

key_switch_down	<i>Scale a ciphertext down from extended CRT basis to Q</i>
-----------------	--

Description

Brings a ciphertext that lives in the extended PQ 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.

Value

A Ciphertext in the Q basis.

See Also

[eval_fast_rotation_ext\(\)](#)

key_tag	<i>Key tag accessors</i>
---------	--------------------------

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	<i>Key Generation Mode</i>
------------	----------------------------

Description

Mirrors the C++ enum KEYGEN_MODE in binfhe/binfhe-constants.h.

Usage

```
KeygenMode
```

Value

A named list of 2 integer scalars, each the value of the OpenFHE C++ enumerator of the same name: SYM_ENCRYPT generates only the material needed for symmetric-key encryption, while PUB_ENCRYPT additionally generates a public key. Pass an element as the keygen_mode argument of [bin_bt_key_gen\(\)](#).

KeyPair	<i>Key Pair</i>
---------	-----------------

Description

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

Usage

KeyPair(public = NULL, secret = NULL)

Arguments

public	A PublicKey
secret	A PrivateKey

Value

An S7 object of class KeyPair with two properties, public (a [PublicKey](#)) and secret (a [PrivateKey](#)), which are the two halves of one freshly generated key. Obtain one from [key_gen\(\)](#) rather than by calling this constructor directly.

KeySwitchTechnique	<i>Key Switching Techniques</i>
--------------------	---------------------------------

Description

Mirrors the C++ enum KeySwitchTechnique in pke/constants-defs.h.

Usage

KeySwitchTechnique

Value

A named list of 3 integer scalars, each the value of the OpenFHE C++ enumerator of the same name. BV selects the Brakerski-Vaikuntanathan digit-decomposition key switch and HYBRID the hybrid variant, which is the usual choice; INVALID_KS_TECH marks an unset value. Pass an element as the key_switch_technique argument of [CKKSParams\(\)](#), [BFVParams\(\)](#), [BGVParams\(\)](#) or [fhe_context\(\)](#).

level_reduce	<i>Reduce the modulus chain by multiple levels</i>
--------------	--

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

<code>x</code>	A Ciphertext.
<code>...</code>	Method-specific arguments: <code>eval_key</code> (an <code>EvalKey</code> from <code>key_gen</code>), <code>levels</code> (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	<i>Reduce the modulus chain by multiple levels, in place</i>
-----------------------	--

Description

Reduce the modulus chain by multiple levels, in place

Usage

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

Arguments

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

Value

`x` invisibly.

LWECiphertext	<i>LWE Ciphertext (Binary FHE)</i>
---------------	------------------------------------

Description

LWE Ciphertext (Binary FHE)

Usage

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

Arguments

ptr	External pointer (internal use)
-----	---------------------------------

Value

An S7 object of class LWECiphertext, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ LWE ciphertext. This is the encrypted-bit type the boolean-circuit (BinFHE) schemes operate on; obtain one from [bin_encrypt\(\)](#) or from a gate evaluation rather than by calling this constructor directly.

LWEPrivateKey	<i>LWE Private Key (Binary FHE)</i>
---------------	-------------------------------------

Description

LWE Private Key (Binary FHE)

Usage

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

Arguments

ptr	External pointer (internal use)
-----	---------------------------------

Value

An S7 object of class LWEPrivateKey, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ LWE secret key. It both encrypts and decrypts in the boolean-circuit (BinFHE) schemes; obtain one from [bin_key_gen\(\)](#) rather than by calling this constructor directly.

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

<code>cc</code>	A <code>CryptoContext</code> built with <code>scheme = "CKKS"</code> .
<code>values</code>	A numeric vector to pack. Length must not exceed <code>slots</code> (or <code>batch_size / 2</code> when <code>slots</code> is left at <code>0L</code>).
<code>noise_scale_deg</code>	Integer degree of the initial scaling factor applied to the encoded plaintext, expressed as a power of the scheme's scaling factor. Defaults to <code>1L</code> , meaning "scale by the base scaling factor once", which is the value every current vignette implicitly uses. Setting <code>noise_scale_deg = 2L</code> 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 <code>FLEXIBLEAUTO</code> scaling the scheme's auto-rescale logic overrides this argument at context-generation time, so this parameter is only meaningful under <code>FIXEDMANUAL</code> . Couples tightly to <code>level</code> : a plaintext at (<code>noise_scale_deg = k</code> , <code>level = L</code>) may only interact with ciphertexts at the same (<code>k</code> , <code>L</code>).
<code>level</code>	Integer target encryption level of the encoded plaintext. Defaults to <code>0L</code> , 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 <code>L</code> times, the plaintext must be encoded at <code>level = L</code> 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 <code>noise_scale_deg</code> 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 <code>get_element_params(cc)</code>), so in practice the argument is accepted for compatibility with the C++ API 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 <code>eval_sum</code> . 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.

Examples

```
cc <- fhe_context("CKKS", multiplicative_depth = 2L,
                  scaling_mod_size = 50L, batch_size = 8L)

## Encode four reals into the CKKS slots. The remaining slots are
## zero-padded out to `batch_size`.
pt <- make_ckks_packed_plaintext(cc, c(0.25, 0.5, 0.75, 1))
pt
get_real_packed_value(set_length(pt, 4L))
```

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). 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	<i>Reduce the modulus chain by one level</i>
------------	--

Description

Synonym for [rescale\(\)](#). Both names dispatch to the same C++ operation (`CryptoContextImpl::Rescale` delegates to `ModReduce` internally); the R binding keeps both so callers 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	<i>Reduce the modulus chain by one level, in place</i>
---------------------	--

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	<i>Combine public keys from multiple parties</i>
--------------------	--

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	<i>Generate a joint rotation-at-index key share</i>
-----------------------------	---

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.

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 <code>key_gen(cc, rotations = ...)</code> .
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	<i>Multi-party key-switch eval-key generation</i>
----------------------	---

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 ($n \geq 2$). 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

<code>cc</code>	A <code>CryptoContext</code>
<code>...</code>	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

<code>cc</code>	A <code>CryptoContext</code>
<code>sk</code>	This party's <code>PrivateKey</code>
<code>ct</code>	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	<i>Generate a key pair for a secondary party in threshold FHE</i>
--------------------	---

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

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

Value

A `KeyPair` for this party

<code>MultipartyMode</code>	<i>Multiparty Mode</i>
-----------------------------	------------------------

Description

Mirrors the C++ enum `MultipartyMode` in `pke/constants-defs.h`.

Usage

`MultipartyMode`

Value

A named list of 3 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, selecting how a threshold context masks the secret shares: `FIXED_NOISE_MULTIPARTY` adds a fixed amount of noise, while `NOISE_FLOODING_MULTIPARTY` adds enough to give provable circuit privacy at a higher cost. `INVALID_MULTIPARTY_MODE` marks an unset value. Pass an element as the `multiparty_mode` argument of `BFVParams()`, `BGVParams()` or `fhe_context()`.

MultiplicationTechnique

Multiplication Technique (BFV)

Description

Mirrors the C++ enum MultiplicationTechnique in pke/constants-defs.h.

Usage

MultiplicationTechnique

Value

A named list of 4 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, selecting the algorithm BFV uses for homomorphic multiplication: BEHZ (Bajard-Eynard-Hasan-Zucca) or the Halevi-Polyakov-Shoup variants HPS, HPSPOVERQ and HPSPOVERQLEVELED. Pass an element as the multiplication_technique argument of [BFVParams\(\)](#) or [fhe_context\(\)](#).

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)

Value

An S7 object of class OpenFHEObject, the base class every object in this package inherits from. It has a single property, ptr, holding an external pointer to the underlying C++ shared_ptr. Constructing one directly is of no use; the class exists so that methods can dispatch on the common parent and so that the pointer is managed in one place.

Plaintext	<i>Plaintext</i>
-----------	------------------

Description

Plaintext

Usage

Plaintext(ptr = NULL)

Arguments

ptr External pointer (internal use)

Value

An S7 object of class Plaintext, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ plaintext. A plaintext holds an encoded but unencrypted vector together with its encoding type; obtain one from a `make_*_plaintext()` factory or from `decrypt()` rather than by calling this constructor directly, and read its contents with `get_packed_value()`, `get_real_value()` and the other accessors.

plaintext_accessors	<i>Plaintext accessors</i>
---------------------	----------------------------

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

<code>x</code>	A Plaintext.
<code>...</code>	Reserved for future method-specific arguments. Setters accept a value argument here; <code>get_formatted_values</code> 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. Incremented by each multiplication before a rescale.
- `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.
- `get_scaling_factor`: Numeric scaling factor for CKKS-based plaintexts — the multiplier the real vector is scaled by before encoding. Equals $2^{\text{scaling_mod_size}}$ 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 $\log_2$ 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`: integer encoding type (see `PlaintextEncodings` for the enum values).
- `get_scaling_factor_int`: BGV integer scaling factor. Returned as a double carrying a losslessly rounded 53-bit integer.
- `get_scheme_id`: scheme identifier (see `SchemeId` enum).
- `is_encoded`: 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 uninitialized form.
- `low_bound`: integer lower bound that can be encoded with the current plaintext modulus: $-\text{floor}(t / 2)$.
- `high_bound`: integer upper bound that can be encoded with the current plaintext modulus: $\text{floor}(t / 2)$.
- `get_slots`: integer CKKS slot count. For `Plaintext` dispatch this is the `GetSlots()` value set at construction.
- `get_log_error`: numeric $\log_2$ of the error estimate. Only meaningful for CKKS plaintexts; throws for BFV/BGV.

- `get_coef_packed_value`: integer vector of the underlying coef-packed encoding. Throws for plaintexts whose subclass is not coef-packed.
- `get_string_value`: string value of a string-encoded plaintext. Throws for plaintexts whose subclass is not string-encoded.
- `get_element_ring_dimension`: 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`: set the CKKS plaintext scaling factor.
- `set_scaling_factor_int`: set the BGV plaintext integer scaling factor.
- `set_noise_scale_deg`: 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`: set the plaintext level. As with `set_noise_scale_deg`, most users should not call this directly.
- `set_slots`: set the CKKS slot count. As with `set_length`, this is typically managed by the factory methods.
- `set_string_value`: set the string value of a string-encoded plaintext. Throws for plaintexts whose subclass is not string-encoded.
- `set_int_vector_value`: 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

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	<i>Plaintext Encoding Types</i>
--------------------	---------------------------------

Description

Mirrors the C++ enum PlaintextEncodings in pke/constants-defs.h.

Usage

PlaintextEncodings

Value

A named list of 5 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, naming how values are laid out inside a plaintext polynomial: COEF_PACKED_ENCODING, PACKED_ENCODING (integer SIMD slots), STRING_ENCODING and CKKS_PACKED_ENCODING (approximate real or complex slots). INVALID_ENCODING marks an unset value. This is the value reported by get_encoding_type() on a Plaintext.

PREMode	<i>Proxy Re-encryption Mode</i>
---------	---------------------------------

Description

Mirrors the C++ enum ProxyReEncryptionMode in pke/constants-defs.h. The R-side name PREMode is a shortened form, the same pattern as Feature for PKESchemeFeature.

Usage

PREMode

Value

A named list of 4 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, naming the security notion the re-encryption key should satisfy: INDCPA, FIXED_NOISE_HRA or NOISE_FLOODING_HRA, the last two being honest-re-encryption-attack secure. NOT_SET disables proxy re-encryption. Pass an element as the pre_mode argument of [CKKSParams\(\)](#), [BFVParams\(\)](#), [BGVParams\(\)](#) or [fhe_context\(\)](#).

PrivateKey	<i>Private Key</i>
------------	--------------------

Description

Private Key

Usage

PrivateKey(ptr = NULL)

Arguments

ptr External pointer (internal use)

Value

An S7 object of class PrivateKey, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ secret key. This is the key [decrypt\(\)](#) takes; obtain one as the secret element of the [KeyPair](#) returned by [key_gen\(\)](#) rather than by calling this constructor directly.

PublicKey	<i>Public Key</i>
-----------	-------------------

Description

Public Key

Usage

PublicKey(ptr = NULL)

Arguments

ptr External pointer (internal use)

Value

An S7 object of class PublicKey, inheriting from [OpenFHEObject](#), whose ptr property holds an external pointer to the C++ public key. This is the key [encrypt\(\)](#) takes; obtain one as the public element of the [KeyPair](#) returned by [key_gen\(\)](#) rather than by calling this constructor directly.

recover_shared_key	<i>Recover a secret key from distributed shares</i>
--------------------	---

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	<i>Relinearize a higher-degree ciphertext</i>
-------------	---

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

<code>x</code>	A Ciphertext (may have more than 2 components).
<code>...</code>	Reserved for future method-specific arguments.

Value

A Ciphertext with exactly 2 components.

rescale	<i>Rescale a CKKS ciphertext (alias for ModReduce)</i>
---------	--

Description

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

Usage

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

Arguments

<code>ct</code>	A Ciphertext
<code>...</code>	Reserved for future method-specific arguments

Value

A Ciphertext at one lower level

ring_dimension	<i>Ring dimension of a CryptoContext</i>
----------------	--

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	<i>Scaling Techniques (CKKS)</i>
------------------	----------------------------------

Description

Mirrors the C++ enum ScalingTechnique in pke/constants-defs.h.

Usage

```
ScalingTechnique
```

Value

A named list of 8 integer scalars, each the value of the OpenFHE C++ enumerator of the same name. The element chosen tells a CKKS context how to manage the scaling factor between multiplications: FIXEDMANUAL leaves rescaling to the caller, FIXEDAUTO and the FLEXIBLE* variants rescale automatically with increasing precision, the COMPOSITESCALING* variants split the scaling factor across several moduli, and NORESCALE disables rescaling. INVALID_RS_TECHNIQUE marks an unset value. Pass an element as the scaling_technique argument of [CKKSParams\(\)](#), [BGVParams\(\)](#) or [fhe_context\(\)](#).

SchemeId	<i>Scheme Identifier</i>
----------	--------------------------

Description

Mirrors the C++ enum SCHEME in pke/scheme/scheme-id.h. The R-side name SchemeId matches the upstream header filename and avoids colliding with a potential future Scheme S7 class.

Usage

SchemeId

Value

A named list of 4 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, identifying the encryption scheme a context implements: CKKSRNS_SCHEME, BFVRNS_SCHEME or BGVRNS_SCHEME, with INVALID_SCHEME for an unset value. This is the value reported by get_scheme() on any CCParams object.

SecretKeyDist	<i>Secret Key Distribution</i>
---------------	--------------------------------

Description

Mirrors the C++ enum SecretKeyDist in core/lattice/constants-lattice.h.

Usage

SecretKeyDist

Value

A named list of 4 integer scalars, each the value of the OpenFHE C++ enumerator of the same name, naming the distribution the secret key polynomial is drawn from: GAUSSIAN, UNIFORM_TERNARY (the default, coefficients uniform on -1, 0, 1), SPARSE_TERNARY (ternary with a fixed small Hamming weight) and SPARSE_ENCAPSULATED. Pass an element as the secret_key_dist argument of [CKKSParams\(\)](#), [BFVParams\(\)](#), [BGVParams\(\)](#) or [fhe_context\(\)](#).

SecretShareMap	<i>Map of secret-key shares for threshold-FHE abort recovery</i>
----------------	--

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.

Value

An S7 object of class `SecretShareMap`, inheriting from `OpenFHEObject`, whose `ptr` property holds an external pointer to the C++ `std::unordered_map<uint32_t, DCRTPoly>`. It carries one share per party index and acts as the transport format for the threshold-FHE abort-recovery protocol; it is produced by `share_keys()` rather than by calling this constructor directly.

SecurityLevel	<i>Security Levels</i>
---------------	------------------------

Description

Mirrors the C++ enum `SecurityLevel` in `core/lattice/stdlatticeparms.h`.

Usage

`SecurityLevel`

Value

A named list of 7 integer scalars, each the value of the OpenFHE C++ enumerator of the same name. An element names the number of bits of security the ring dimension is chosen to provide against a classical (HEStd*_classic) or quantum (HEStd*_quantum) adversary, following the HomomorphicEncryption.org standard tables. HEStd_NotSet skips the table lookup, in which case the ring dimension must be set explicitly. Pass an element as the security_level argument of CKKSParams(), BFVParams(), BGVPParams() or fhe_context().

serialize_eval_keys	<i>Serialize evaluation keys to file</i>
---------------------	--

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

<code>cc</code>	A <code>CryptoContext</code> .
<code>cf</code>	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

<code>cc</code>	A <code>CryptoContext</code> .
<code>...</code>	Method-specific arguments; the <code>level</code> integer is passed here.

Value

`cc` invisibly.

set_length	<i>Set the effective length of a plaintext</i>
------------	--

Description

Set the effective length of a plaintext

Usage

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

Arguments

pt	A Plaintext
len	Integer length

Value

Invisibly, the pt plaintext that was passed in. Called for its side effect: the plaintext is truncated in place so that the accessors report only the first len slots, which is how the padding introduced by encoding is trimmed after decryption.

set_num_threads	<i>Set the number of OpenMP threads OpenFHE may use</i>
-----------------	---

Description

OpenFHE parallelizes its core arithmetic with OpenMP and, by default, uses every hardware thread the machine reports. `set_num_threads()` sets the cap through OpenFHE's own thread controls, which every parallel region in the library consults when it starts, so the change takes effect immediately, on every platform, and regardless of which thread the work runs on. It is a no-op when the package was built without OpenMP.

Usage

```
set_num_threads(n)
```

Arguments

n	integer; the requested maximum number of threads (at least 1).
---	--

Details

Requests above the number of threads available when the library was loaded (the hardware count, or OMP_NUM_THREADS if that was set before R started) are reduced to that number. The value in effect is returned, so a reduced request is visible to the caller; `get_num_threads()` reports the same value later.

The package default is uncapped, so interactive users get full parallelism. Under R CMD check, which sets `_R_CHECK_LIMIT_CORES_`, the package caps itself at two threads when it is loaded, in line with CRAN's two-thread policy. This also covers packages that depend on 'openfhe.R', which therefore need no thread handling of their own.

Value

integer; the cap now in effect, invisibly.

See Also

`get_num_threads()`

Examples

```
old <- get_num_threads()
set_num_threads(2L)
set_num_threads(old)
```

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 $\text{floor}(n_parties/2) + 1$.
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" — $\text{floor}(N/2) + 1$ threshold; the secret can be reconstructed from any majority subset of the distributed shares. Robust against up to $\text{floor}((N-1)/2)$ parties dropping out.

Value

A SecretShareMap suitable for passing to [recover_shared_key\(\)](#).

See Also

[recover_shared_key\(\)](#)

threshold_decrypt	<i>Threshold decryption convenience: lead + main + fusion in one call</i>
-------------------	---

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	<i>Execute code with automatic cleanup of FHE state</i>
------------------	---

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

BFVParams, 7
BFVParams(), 34, 65, 89, 103, 104, 109, 114, 116
BGVParams, 10
BGVParams(), 65, 89, 103, 109, 113, 114, 116
bin_bt_key_gen, 12
bin_bt_key_gen(), 88
bin_decrypt, 12
bin_encrypt, 13
bin_encrypt(), 16, 91
bin_fhe_context, 13
bin_fhe_context(), 15, 16
bin_key_gen, 14
bin_key_gen(), 91
BinFHEContext, 14, 15
BinFHEMethod, 15
BinFHEOutput, 16
BinFHEParamSet, 16
BinGate, 17, 38
bitwOr(), 63

ccparams_getters, 17
Ciphertext, 21, 44, 48
ckks_scaling_factor_bits, 21
ckks_scaling_factor_bits(), 64
CKKSDataType, 22
CKKSParams, 22
CKKSParams(), 27, 30, 62, 65, 89, 109, 113, 114, 116
clear_eval_automorphism_keys, 24
clear_eval_automorphism_keys(), 25
clear_eval_mult_keys, 25
clear_eval_mult_keys(), 24, 25, 69
clear_fhe_state, 26
clear_fhe_state(), 24, 25
clear_static_maps_and_vectors, 26
compress, 27
CompressionLevel, 27
CryptoContext, 15, 28
CryptoParameters, 28

decrypt, 29
decrypt(), 105, 110
DecryptionNoiseMode, 30
deserialize_eval_keys, 30
DistributionType, 31

ElementParams, 31
enable_feature, 32
enable_feature(), 63
EncodingParams, 32
encrypt, 33
encrypt(), 21, 29, 110
EncryptionTechnique, 34
eval_add, 34
eval_add_in_place, 35
eval_add_mutable, 35
eval_at_index_key_gen, 36
eval_automorphism, 36
eval_automorphism(), 37
eval_automorphism_key_gen, 37
eval_automorphism_key_gen(), 36, 37
eval_bin_gate, 38
eval_bin_gate(), 12, 17
eval_bootstrap, 38
eval_bootstrap(), 39
eval_bootstrap_key_gen, 39
eval_bootstrap_key_gen(), 40
eval_bootstrap_setup, 39
eval_bootstrap_setup(), 71, 117
eval_chebyshev, 40
eval_chebyshev(), 41–43
eval_chebyshev_coefficients, 41
eval_chebyshev_coefficients(), 41, 42
eval_chebyshev_function, 42
eval_chebyshev_function(), 41
eval_chebyshev_linear, 42
eval_chebyshev_linear(), 41, 43
eval_chebyshev_ps, 43
eval_chebyshev_ps(), 41, 43
eval_cos, 44

eval_cos(), [42](#)
 eval_divide, [44](#)
 eval_divide(), [42](#)
 eval_fast_rotation, [45](#)
 eval_fast_rotation(), [45](#), [46](#), [62](#)
 eval_fast_rotation_ext, [45](#)
 eval_fast_rotation_ext(), [87](#)
 eval_fast_rotation_precompute, [46](#)
 eval_fast_rotation_precompute(), [45](#), [62](#)
 eval_floor, [46](#)
 eval_func, [47](#)
 eval_func(), [47](#)
 eval_logistic, [48](#)
 eval_logistic(), [42](#)
 eval_mult, [48](#)
 eval_mult(), [49](#)
 eval_mult_and_relinearize, [49](#)
 eval_mult_in_place, [49](#)
 eval_mult_key_gen, [50](#)
 eval_mult_mutable, [50](#)
 eval_mult_no_relin, [51](#)
 eval_negate, [51](#)
 eval_negate_in_place, [52](#)
 eval_not, [52](#)
 eval_poly, [53](#)
 eval_poly(), [54](#)
 eval_poly_linear, [53](#)
 eval_poly_linear(), [53](#), [54](#)
 eval_poly_ps, [54](#)
 eval_poly_ps(), [53](#), [54](#)
 eval_rotate, [54](#)
 eval_rotate(), [37](#), [55](#)
 eval_rotate_key_gen, [55](#)
 eval_rotate_key_gen(), [36](#)
 eval_sign, [56](#)
 eval_sign(), [47](#)
 eval_sin, [56](#)
 eval_sin(), [42](#)
 eval_square, [57](#)
 eval_square_mutable, [57](#)
 eval_sub, [58](#)
 eval_sub_in_place, [58](#)
 eval_sub_mutable, [59](#)
 eval_sum, [59](#)
 eval_sum(), [60](#)
 eval_sum_key_gen, [60](#)
 EvalKey, [60](#)
 EvalKeyMap, [61](#)
 ExecutionMode, [62](#)
 FastRotationPrecomputation, [62](#)
 Feature, [63](#)
 fhe_ckks_tolerance, [63](#)
 fhe_ckks_tolerance(), [21](#)
 fhe_context, [64](#)
 fhe_context(), [27](#), [28](#), [30](#), [34](#), [62](#), [89](#), [103](#),
 [104](#), [109](#), [113](#), [114](#), [116](#)
 fhe_deserialize, [65](#)
 fhe_serialize, [66](#)
 find_automorphism_index, [66](#)
 find_automorphism_index(), [67](#)
 find_automorphism_indices, [67](#)
 find_automorphism_indices(), [37](#), [67](#)
 generate_lut_via_function, [68](#)
 get_all_eval_automorphism_keys, [68](#)
 get_all_eval_automorphism_keys(), [69](#)
 get_all_eval_mult_keys, [69](#)
 get_all_eval_mult_keys(), [75](#)
 get_all_eval_sum_keys, [69](#)
 get_batch_size(ccparams_getters), [17](#)
 get_bootstrap_depth, [70](#)
 get_ckks_boot_correction_factor, [70](#)
 get_ckks_boot_correction_factor(), [117](#)
 get_ckks_data_type(ccparams_getters),
 [17](#)
 get_coef_packed_value
 (plaintext_accessors), [105](#)
 get_complex_packed_value, [71](#)
 get_complex_packed_value(), [78](#)
 get_composite_degree
 (ccparams_getters), [17](#)
 get_crypto_context, [72](#)
 get_crypto_context(), [64](#)
 get_crypto_parameters, [72](#)
 get_cyclotomic_order, [73](#)
 get_decryption_noise_mode
 (ccparams_getters), [17](#)
 get_desired_precision
 (ccparams_getters), [17](#)
 get_digit_size(ccparams_getters), [17](#)
 get_element_params, [73](#)
 get_element_ring_dimension
 (plaintext_accessors), [105](#)
 get_encoding_params, [74](#)
 get_encoding_type
 (plaintext_accessors), [105](#)

get_encryption_technique
 (ccparams_getters), 17
 get_eval_add_count(ccparams_getters),
 17
 get_eval_automorphism_key_map, 74
 get_eval_automorphism_key_map(), 61, 69,
 99
 get_eval_mult_key_vector, 75
 get_eval_mult_key_vector(), 69
 get_eval_sum_key_map, 75
 get_eval_sum_key_map(), 61, 70, 100
 get_execution_mode(ccparams_getters),
 17
 get_first_mod_size(ccparams_getters),
 17
 get_formatted_values
 (plaintext_accessors), 105
 get_interactive_boot_compression_level
 (ccparams_getters), 17
 get_key_gen_level, 76
 get_key_switch_count
 (ccparams_getters), 17
 get_key_switch_technique
 (ccparams_getters), 17
 get_key_tag(key_tag), 88
 get_key_tag(), 98
 get_length(plaintext_accessors), 105
 get_level(plaintext_accessors), 105
 get_log_error(plaintext_accessors), 105
 get_log_precision
 (plaintext_accessors), 105
 get_max_plaintext_space, 76
 get_max_relin_sk_deg
 (ccparams_getters), 17
 get_multipart_mode(ccparams_getters),
 17
 get_multiplication_technique
 (ccparams_getters), 17
 get_multiplicative_depth
 (ccparams_getters), 17
 get_native_int, 77
 get_noise_estimate(ccparams_getters),
 17
 get_noise_scale_deg
 (plaintext_accessors), 105
 get_num_adversarial_queries
 (ccparams_getters), 17
 get_num_large_digits
 (ccparams_getters), 17
 get_num_threads, 77
 get_num_threads(), 119
 get_packed_value, 78
 get_packed_value(), 105
 get_plaintext_modulus
 (ccparams_getters), 17
 get_pre_mode(ccparams_getters), 17
 get_pre_num_hops(ccparams_getters), 17
 get_real_packed_value, 78
 get_real_packed_value(), 71
 get_register_word_size
 (ccparams_getters), 17
 get_ring_dim(ccparams_getters), 17
 get_scaling_factor
 (plaintext_accessors), 105
 get_scaling_factor_int
 (plaintext_accessors), 105
 get_scaling_factor_real, 79
 get_scaling_factor_real(), 21
 get_scaling_mod_size
 (ccparams_getters), 17
 get_scaling_technique
 (ccparams_getters), 17
 get_scheme(ccparams_getters), 17
 get_scheme_id(plaintext_accessors), 105
 get_secret_key_dist(ccparams_getters),
 17
 get_security_level(ccparams_getters),
 17
 get_slots(plaintext_accessors), 105
 get_standard_deviation
 (ccparams_getters), 17
 get_statistical_security
 (ccparams_getters), 17
 get_string_value(plaintext_accessors),
 105
 get_threshold_num_of_parties
 (ccparams_getters), 17
 high_bound(plaintext_accessors), 105
 insert_eval_automorphism_key, 79
 insert_eval_automorphism_key(), 61, 69,
 80, 96
 insert_eval_mult_key, 80
 insert_eval_mult_key(), 69, 75
 insert_eval_sum_key, 81
 insert_eval_sum_key(), 61, 70, 80, 98

- int_boot_add, 81
- int_boot_add(), 82
- int_boot_adjust_scale, 82
- int_boot_adjust_scale(), 82, 84
- int_boot_decrypt, 82
- int_boot_decrypt(), 81–83
- int_boot_encrypt, 83
- int_boot_encrypt(), 81, 82
- int_mp_boot_add, 83
- int_mp_boot_add(), 84, 85
- int_mp_boot_adjust_scale, 84
- int_mp_boot_decrypt, 84
- int_mp_boot_decrypt(), 83, 85
- int_mp_boot_encrypt, 85
- int_mp_boot_encrypt(), 83
- int_mp_boot_random_element_gen, 85
- int_mp_boot_random_element_gen(), 84
- is_encoded (plaintext_accessors), 105
- is_good, 86

- key_gen, 86
- key_gen(), 37, 50, 55, 60, 89, 103, 110
- key_switch_down, 87
- key_tag, 88
- KeygenMode, 12, 88
- KeyPair, 89, 110
- KeySwitchTechnique, 89

- level_reduce, 90
- level_reduce_in_place, 90
- low_bound (plaintext_accessors), 105
- LWECiphertext, 91
- LWEPrivateKey, 91

- make_ckks_packed_plaintext, 92
- make_ckks_packed_plaintext(), 71
- make_coef_packed_plaintext, 93
- make_packed_plaintext, 94
- make_packed_plaintext(), 94
- mod_reduce, 95
- mod_reduce_in_place, 95
- mod_reduce_in_place(), 95
- multi_add_eval_automorphism_keys, 96
- multi_add_eval_automorphism_keys(), 61, 79, 99
- multi_add_eval_keys, 96
- multi_add_eval_keys(), 97, 100
- multi_add_eval_mult_keys, 97
- multi_add_eval_mult_keys(), 80, 96

- multi_add_eval_sum_keys, 97
- multi_add_eval_sum_keys(), 61, 81
- multi_add_pub_keys, 98
- multi_eval_at_index_key_gen, 98
- multi_eval_at_index_key_gen(), 55
- multi_eval_automorphism_key_gen, 99
- multi_eval_automorphism_key_gen(), 98
- multi_eval_sum_key_gen, 99
- multi_key_switch_gen, 100
- multi_key_switch_gen(), 61, 80, 97
- multiparty_decrypt_fusion, 101
- multiparty_decrypt_fusion(), 120
- multiparty_decrypt_lead, 101
- multiparty_decrypt_lead(), 101, 102, 120
- multiparty_decrypt_main, 102
- multiparty_decrypt_main(), 101, 120
- multiparty_key_gen, 103
- MultipartyMode, 103
- MultiplicationTechnique, 104

- openfhe.R (openfhe.R-package), 6
- openfhe.R-package, 6
- OpenFHEObject, 15, 21, 28, 31, 33, 61, 62, 91, 104, 105, 110, 115

- Plaintext, 105
- plaintext_accessors, 105
- plaintext_params_hash, 108
- PlaintextEncodings, 109
- PREMode, 109
- PrivateKey, 89, 110
- PublicKey, 89, 110

- recover_shared_key, 111
- recover_shared_key(), 115, 120
- relinearize, 112
- rescale, 112
- rescale(), 95
- ring_dimension, 113

- ScalingTechnique, 113
- SchemeId, 114
- SecretKeyDist, 114
- SecretShareMap, 115
- SecurityLevel, 115
- serialize_eval_keys, 116
- set_ckks_boot_correction_factor, 117
- set_ckks_boot_correction_factor(), 70, 71

set_ckks_data_type
 (plaintext_accessors), [105](#)
set_int_vector_value
 (plaintext_accessors), [105](#)
set_key_gen_level, [117](#)
set_key_tag(key_tag), [88](#)
set_length, [118](#)
set_level(plaintext_accessors), [105](#)
set_noise_scale_deg
 (plaintext_accessors), [105](#)
set_num_threads, [118](#)
set_num_threads(), [77](#)
set_scaling_factor
 (plaintext_accessors), [105](#)
set_scaling_factor_int
 (plaintext_accessors), [105](#)
set_slots(plaintext_accessors), [105](#)
set_string_value(plaintext_accessors),
 [105](#)
share_keys, [119](#)
share_keys(), [111](#), [115](#)

threshold_decrypt, [120](#)

with_fhe_context, [121](#)