

Package: homomorpheR (via r-universe)

October 16, 2026

Type Package

Title Homomorphic Computations in R

Version 1.0

VignetteBuilder knitr

URL <https://bnaras.github.io/homomorpheR/>

BugReports <https://github.com/bnaras/homomorpheR/issues>

Depends R (>= 3.5.0)

Suggests knitr, kableExtra, rmarkdown, survival, CVXR, tinytest

Imports S7, cli, gmp, openfhe.R, rlang, sodium

Description Privacy-preserving statistics across sites that never share their data, using fully homomorphic encryption through the 'openfhe.R' interface to OpenFHE (CKKS, BFV, BGV), with n-of-n threshold key generation so that no single party can decrypt. Ships master/worker primitives that let ordinary R modeling code run across sites, and a frozen implementation of the Paillier additive scheme kept for backward compatibility. The protocols are described in Narasimhan (2026) <[doi:10.48550/arXiv.2610.04163](https://doi.org/10.48550/arXiv.2610.04163)>.

License MIT + file LICENSE

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE)

Encoding UTF-8

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libgmp3-dev make libsodium-dev

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

Date/Publication 2026-10-06 14:05:52 UTC

RemoteUrl <https://github.com/bnaras/homomorpheR>

RemoteRef HEAD

RemoteSha 32ca485121cee3dc2c41e9ffb7ada90a159c6ebf

Contents

actor-encryption	3
add_site	4
CKKSMaster	5
contribute	6
cox_results	7
cox_threshold_dp_results	7
cox_threshold_results	8
cvxr_admm_dp_results	8
cvxr_consensus	9
DLBCL	10
DLBCL_gex	11
encrypt_real	12
get_lambda	13
get_private_key	13
homomorpheR	14
keygen_round	15
LocalSite	16
make_ckks_master	17
make_master	17
make_ncparty	18
make_threshold_master	18
make_worker	19
Master	20
master_aggregate	21
NCParty	22
OpenFHEParams	23
paillier_keypair	23
PaillierCiphertext	24
PaillierEncryptedReal	24
PaillierKeyPair	25
PaillierMaster	26
PaillierParams	27
PaillierPrivateKey	27
PaillierPublicKey	28
partial_decrypt	28
PublicParams	29
random.bigz	30
RemoteSite	30
round_robin_chain	32
run_round_robin	33
set_next_site	34
set_public_key	34
set_public_params	35
set_workers	36
similarity_results	37
Site	37

site_params	38
site_unavailable	39
ThresholdMaster	39

Index	41
--------------	-----------

actor-encryption	<i>Encrypt and decrypt with protocol actors</i>
------------------	---

Description

homomorpheR adds methods to `openfhe.R`'s `encrypt()` and `decrypt()` generics, so the same two verbs serve both layers of the stack and the class of the first argument selects which layer answers. This page describes the actor-level methods; the key-level ones are documented in `openfhe.R`.

Encrypting

Encryption needs only public material, so a party handed that material at setup encrypts entirely on its own, with nothing to consult and no one to ask — which is what makes `contribute()` a purely local computation. Methods are registered on whatever holds the public material:

- a `Site` encrypts with the parameters it was given when it was wired, so `encrypt(site, value)` needs nothing besides the site itself;
- `OpenFHEParams` encrypts under a bundle held directly, which is what a party reads back from a site with `site_params()`;
- the frozen `PaillierParams` and `PaillierPublicKey` encrypt under the Paillier scheme.

For the `openfhe` backends the encoding follows whatever the context was built for, read back from the context itself: packed reals under CKKS, packed integers under BFV and BGV. The exact schemes reject a value they cannot represent rather than round it; see `OpenFHEParams`.

There is deliberately no method on `Master`, and `public_params()` is deliberately not exported. An encryption entry point taking a master would advertise a privilege that does not exist, and would invite site-side code to reach back to a coordinator for something it was already given. A site is autonomous once configured.

Decrypting

Decryption is the asymmetric half of the pair, and that asymmetry is the point: it takes either secret material or the standing to convene every site, while encryption takes neither. Methods are registered on the decrypting party:

- `CKKSMaster` decrypts with the secret key it holds;
- `ThresholdMaster` holds no key material at all and recovers a value by asking each site for a partial decryption through `partial_decrypt()` and fusing the results, so no party — the master included — can decrypt alone;
- the frozen `PaillierMaster` and `PaillierPrivateKey` decrypt under the Paillier scheme.

The master methods take `len`, the number of packed slots to return, defaulting to 1.

Return semantics

The methods return deliberately different types, because the encodings differ:

- [CKKSMaster](#) and [ThresholdMaster](#) return a numeric of length len, decoded for whatever scheme the context was built for.
- [PaillierMaster](#) returns the scalar real its protocol accumulated.
- a [PaillierCiphertext](#) under a [PaillierPrivateKey](#) gives a [gmp::bigz](#) in $[0, n)$. This preserves raw mod-n arithmetic; callers wanting signed integers should re-center themselves (if $m > n/2$) $m - n$).
- a [PaillierEncryptedReal](#) gives a numeric in $(-n/2, n/2)$. The method re-centers the raw mod-n residues so that negative real numbers and running totals that cross zero round-trip correctly. See [PaillierEncryptedReal](#) for the full convention.

Argument names

The generics belong to `openfhe.R`, and their argument names follow the OpenFHE C++ signatures: `encrypt(key, pt, ...)` for `Encrypt(publicKey, plaintext)`, and `decrypt(ct, key, ...)` for `Decrypt(ciphertext, privateKey)`. S7 requires every method to use the generic's names for the arguments it dispatches on, so those are the names here too, and in a call on a protocol actor they read by position: in `encrypt(site, value)`, key is the site and pt the value; in `decrypt(master, ct, len)`, ct is the master and key the encrypted value. Every call in this package and its vignettes is positional, so the names are never written out.

See Also

[openfhe.R::encrypt\(\)](#) and [openfhe.R::decrypt\(\)](#) for the key-level methods; [contribute\(\)](#), which is how a [Site](#) encrypts its own data during a round; [partial_decrypt\(\)](#) for the site side of a threshold decryption.

add_site

Add a site to a non-cooperating party

Description

Part of the frozen Paillier-era legacy surface.

Usage

```
add_site(ncp, ...)
```

Arguments

ncp	an NCParty .
...	method-specific arguments. The NCParty method takes a single Site .

Value

the [NCParty](#) ncp, invisibly. Called for its side effect: the site is appended to the list of sites the party manages, held in its state environment.

CKKSMaster	<i>CKKS-backed master</i>
------------	---------------------------

Description

A [Master](#) that drives the protocol over openfhe.R's CKKS encryption. CKKS handles real-valued arithmetic natively, so no den denominator is needed. Constructed by [make_ckks_master\(\)](#).

Usage

```
CKKSMaster(
  name = character(0),
  state = new.env(parent = emptyenv()),
  crypto_context = openfhe.R::CryptoContext(),
  keypair = openfhe.R::KeyPair()
)
```

Arguments

name	short identifier shown in printed output.
state	an environment for mutable bookkeeping.
crypto_context	an openfhe.R CryptoContext configured for CKKS.
keypair	an openfhe.R KeyPair.

Value

an S7 object of class CKKSMaster, inheriting from [Master](#), with properties name, crypto_context, keypair and state. It holds a single CKKS key pair, so it is the appropriate master when one party is allowed to hold the secret key; when no party may, use [ThresholdMaster](#). Construct with [make_ckks_master\(\)](#).

contribute

*A site's encrypted contribution at a parameter value***Description**

The single call the protocol runner makes on a site. Implementations return the site's contribution **already encrypted**, using the public parameters the site was given when it was configured, so an individual site's cleartext contribution never reaches the aggregator — that is the property the whole protocol rests on, and `master_aggregate()` refuses a reply that is neither an encrypted value under this protocol's key nor NA.

Usage

```
contribute(site, ...)
```

Arguments

<code>site</code>	a LocalSite , or a user-defined subclass of RemoteSite .
<code>...</code>	method-specific arguments; both built-in methods take <code>theta</code> , the parameter value being queried.

Details

The computation is entirely local. A site needs nothing at call time beyond `theta`, its own data, and what it already holds.

The only permitted plaintext reply is NA, signaling that `theta` is non-evaluable at this site; CKKS has no representation for it, so it cannot be encrypted. A site that cannot be *reached* must signal `site_unavailable()` instead of returning NA.

Value

an encrypted contribution, of whatever type the site's own public parameters imply, or NA if `theta` is non-evaluable here.

See Also

[RemoteSite](#) for the contract a remote implementation must honor.

cox_results	<i>Precomputed results for the cox vignette</i>
-------------	---

Description

Precomputed results for the cox vignette

Usage

cox_results

Format

A list with coef (estimate and standard-error matrix of the encrypted single-decrypter stats4::mle() fit), loglik (its log-likelihood), and counts (function and gradient evaluation counts).

Source

data-raw/cox_results.R, from vignettes/cox.Rmd.

cox_threshold_dp_results	<i>Precomputed results for the cox-threshold-dp vignette</i>
--------------------------	--

Description

Precomputed results for the cox-threshold-dp vignette

Usage

cox_threshold_dp_results

Format

A list of the vignette's tables: clean_check (the fit at zero noise against coxph()), bfgs_table and nm_table (BFGS and Nelder-Mead fits over the noise grid), and budget (the zCDP privacy budget of the fits at the first three noise scales).

Source

data-raw/cox_threshold_dp_results.R, from vignettes/cox-threshold-dp.Rmd.

`cox_threshold_results` *Precomputed results for the cox-threshold vignette*

Description

Precomputed results for the cox-threshold vignette

Usage

`cox_threshold_results`

Format

A list with `coef`, `loglik`, and counts for the threshold-encrypted `stats4::mle()` fit, as in [cox_results](#), and `share_check`, a logical vector recording that the master holds no key share and that a site holds its own.

Source

`data-raw/cox_threshold_results.R`, from `vignettes/cox-threshold.Rmd`.

`cvxr_admm_dp_results` *Precomputed results for the cvxr-consensus-admm-dp vignette*

Description

Precomputed results for the cvxr-consensus-admm-dp vignette

Usage

`cvxr_admm_dp_results`

Format

A list with `tol`, `rho_sweep` (convergence on the surrogate cohort), `rho_chosen` and `T_fixed` (the pre-committed constants), `sigma_grid` (the noise scales), `clean_dev` (deviation from the centralized fit at zero noise), and `summary_table` (coefficients at each noise scale).

Source

`data-raw/cvxr_admm_dp_results.R`, from `vignettes/cvxr-consensus-admm-dp.Rmd`.

cvxr_consensus

Precomputed encrypted Cox-lasso consensus-ADMM results

Description

Result objects from the encrypted stratified Cox-lasso consensus-ADMM demonstration on the [DLBCL / DLBCL_gex](#) cohort: a centralized [CVXR](#) ground-truth fit, the same fit recovered by consensus ADMM in the clear, and the encrypted threshold-FHE fit, whose standardization, screening, and consensus rounds all run under encryption. The iterated ADMM runs are expensive, so they are computed once and shipped here; the manuscript and the `cvxr-cox-lasso-dlbcl` vignette load this object instead of recomputing (see Details).

Usage

```
cvxr_consensus
```

Format

A named list with components

params list of the run constants: K (screened probes, 100), LAMBDA (L1 penalty, 5), RHO (ADMM penalty, 50), MAX_ITER (200), TOL (5e-3).

top_idx integer vector of length K; column indices into `DLBCL_gex` of the top-K univariate-screened probes.

sigma_K numeric vector of length K; pooled standard deviations of the screened probes, for the back-transform to the original scale.

agg_beta numeric vector of length K; centralized CVXR Cox-lasso coefficients (the ground truth), on the standardized scale.

z_ref numeric vector of length K; consensus-ADMM coefficients computed in the clear (cleartext reference).

z_enc numeric vector of length K; consensus-ADMM coefficients under threshold FHE.

trajectory list of numeric vectors of length K; the encrypted consensus iterate z^t at each ADMM iteration.

n_iter_ref, n_iter_enc iterations to convergence for the plaintext and encrypted runs.

pool_agree list μ , σ : max absolute disagreement between the encrypted and plaintext pooled standardization moments.

screen_match logical; whether the encrypted screen selected the same probes, in the same order, as the plaintext screen.

Details

The `cvxr-cox-lasso-dlbcl` vignette is the single source of truth. `data-raw/cvxr_consensus.R` extracts its code chunks with `knitr::purl()` into `inst/scripts/cvxr-consensus.R`, runs that script, and saves the result. The `openfhe-jss` manuscript reads the labeled chunks of the generated script with `knitr::read_chunk()`, so the code displayed there is exactly the code that produced these results. Find the installed copy with `system.file("scripts", "cvxr-consensus.R", package = "homomorpher")`.

See Also

[DLBCL](#), [DLBCL_gex](#)

DLBCL	<i>Diffuse Large B-cell Lymphoma Cohort (Rosenwald et al. 2002)</i>
-------	---

Description

Patient-level survival data and gene-expression signature scores from the diffuse large-B-cell lymphoma (DLBCL) cohort of Rosenwald et al. (2002). Used in the Cox-regression vignettes to demonstrate distributed Cox estimation under threshold FHE with sites partitioned by molecular subgroup.

Usage

`data(DLBCL)`

Format

- A data frame with 235 observations on the following 12 variables:
- ID LYM patient identifier (integer).
 - Set Original analysis set assignment, either "Training" or "Validation".
 - Subgroup Molecular subgroup, a factor with levels "GCB" (germinal-center B-cell-like), "ABC" (activated B-cell-like), and "Type III" (unclassified).
 - IPI International Prognostic Index group ("Low", "Medium", "High", or NA).
 - time Follow-up time in years.
 - status Vital status at last follow-up coded as 1 for death and 0 for alive at follow-up.
 - GCB_sig Germinal-center B-cell signature score.
 - LN_sig Lymph-node signature score.
 - Prolif_sig Proliferation signature score.
 - BMP6 BMP6 expression score.
 - MHC2_sig MHC class II signature score.
 - Score Outcome predictor score combining the four signatures and BMP6, as published.

Details

Each row represents one patient. The four signature columns and BMP6 are carried over as published, without further scaling. GCB_sig, LN_sig, Prolif_sig and MHC2_sig are averages of median-centered log-ratio expression values over the genes of each signature; BMP6 is the median-centered log ratio of the single gene BMP6 (Rosenwald et al. 2002, Supplementary Appendix 1). Following Bayle, Fan and Lou (2025), the five patients with zero follow-up time are excluded, so the cohort spans 235 patients with 133 deaths (event rate 56.6%) over a median follow-up of 2.8 years. The

molecular-subgroup partition gives three sites of unequal size: GCB (n=115, 54 deaths), ABC (n=71, 49 deaths), and Type III (n=49, 30 deaths).

The vignettes use Subgroup as the site boundary for distributed Cox estimation; the partition is a choice made for the demonstration. Stratified Cox regression with `strata(Subgroup)` factors the partial log-likelihood additively across the three subgroups, which is exactly the decomposition the master/worker protocol exploits.

Source

The Lymphoma/Leukemia Molecular Profiling Project release of the Rosenwald et al. (2002) study, file 'DLBCL_patient_data_NEW.txt' at <https://llmpp.ccr.cancer.gov/DLBCL/>; processed by 'data-raw/DLBCL.R'.

References

Rosenwald, A., Wright, G., Chan, W. C., et al. (2002). The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphoma. *New England Journal of Medicine* **346**(25), 1937–1947. doi:10.1056/NEJMoa012914

Bayle, P., Fan, J., and Lou, Z. (2025). Communication-Efficient Distributed Estimation and Inference for Cox's Model. *Journal of the American Statistical Association*. doi:10.1080/01621459.2025.2516820

See Also

[DLBCL_gex](#) for the full Lymphochip gene-expression matrix (235 x 6416) on the same cohort.

Examples

```
data(DLBCL)
table(DLBCL$Subgroup, DLBCL$status)

## Stratified Cox fit on the four signatures and BMP6
if (requireNamespace("survival", quietly = TRUE)) {
  fit <- survival::coxph(
    survival::Surv(time, status) ~ GCB_sig + LN_sig + Prolif_sig +
      BMP6 + MHC2_sig + survival::strata(Subgroup),
    data = DLBCL)
  print(fit)
}
```

DLBCL_gex

DLBCL Lymphochip gene-expression matrix

Description

Gene-expression profiles for the 235-patient DLBCL cohort of Rosenwald et al. (2002), used as the high-dimensional benchmark in the encrypted distributed Cox-lasso demonstration.

Usage

```
DLBCL_gex
```

Format

A numeric matrix with 235 rows (patients) and 6416 columns (Lymphochip microarray features). Row names are the patient LYM identifiers, matching DLBCL\$ID; column names are the microarray UNIQIDs. Values are log-ratios on the original scale.

Details

Derived from the public Lymphoma/Leukemia Molecular Profiling Project release (<https://llmpp.ccr.cancer.gov/DLBCL/>; files DLBCL_patient_data_NEW.txt and NEJM_Web_Fig1data). Patients are matched to expression columns by LYM number. Of the 7399 Lymphochip features, the 6416 observed in at least 75% of the 240 patients are retained, and their remaining missing values are imputed by the per-feature mean (Bayle, Fan and Lou, 2025, impute by the median); the five patients with zero follow-up time are excluded, leaving 235 patients. Standardization is deliberately *not* baked into the stored matrix — it is performed inside the encrypted pipeline so that the demonstration exercises encrypted standardization. The full processing script is data-raw/DLBCL.R.

Source

Rosenwald A, Wright G, Chan WC, et al. (2002). The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphoma. *New England Journal of Medicine* 346(25):1937–1947. Data: <https://llmpp.ccr.cancer.gov/DLBCL/>.

See Also

[DLBCL](#)

```
encrypt_real
```

Encrypt a real number under a Paillier public key

Description

Splits x into integer and fractional parts, encrypts each part as a separate [PaillierCiphertext](#), and packages the result as a [PaillierEncryptedReal](#) so that later additions and subtractions can be performed via R's arithmetic operators.

Usage

```
encrypt_real(public_key, x, den)
```

Arguments

public_key	a PaillierPublicKey .
x	a real number.
den	the denominator used to scale the fractional part. The same denominator must be used at encryption and decryption.

Value

a [PaillierEncryptedReal](#).

get_lambda	<i>Return the secret lambda from a private key</i>
------------	--

Description

Return the secret lambda from a private key

Usage

```
get_lambda(private_key, ...)
```

Arguments

private_key	a PaillierPrivateKey .
...	unused.

Value

a [gmp::bigz](#) value.

get_private_key	<i>Return the private key from a key pair</i>
-----------------	---

Description

Return the private key from a key pair

Usage

```
get_private_key(keypair, ...)
```

Arguments

keypair	a PaillierKeyPair .
...	unused.

Value

a [PaillierPrivateKey](#).

homomorpheR

homomorpheR: privacy-preserving statistics across sites

Description

homomorpheR runs statistical computations across sites that never share their data, using fully homomorphic encryption through the `openfhe.R` interface to OpenFHE: CKKS for real-valued arithmetic, BFV and BGV for exact integers, with n -of- n threshold key generation so that no single party can decrypt.

Details

The protocol actors are a [Master](#) and its [Sites](#). A [LocalSite](#), built with `make_worker()`, holds its data and a `contribution_fn`; `master_aggregate()` runs one round, in which each site returns its contribution already encrypted and only the aggregate is decrypted. Build the master with `make_ckks_master()` when one party may hold the secret key and with `make_threshold_master()` when none may. Ordinary R modeling code – `stats4::mle()`, `stratified survival::coxph()`, convex programs via CVXR – then runs unchanged with the encrypted round as its objective. The package vignettes develop each protocol in full.

A frozen implementation of the Paillier additive scheme is kept for backward compatibility; see `paillier_keypair()`.

Author(s)

Maintainer: Balasubramanian Narasimhan <naras@stat.Stanford.EDU> [copyright holder]

Authors:

- Balasubramanian Narasimhan <naras@stat.Stanford.EDU> [copyright holder]

See Also

Useful links:

- <https://bnaras.github.io/homomorpheR/>
- Report bugs at <https://github.com/bnaras/homomorpheR/issues>

Examples

```
## A Poisson rate estimated across three sites: each site encrypts
## its negative log-likelihood, and only the sum is decrypted.
local_nll <- function(data, lambda)
  -sum(stats::dpois(data, lambda, log = TRUE))
y <- c(9, 12, 7, 11, 10, 8, 13, 9, 10, 12)

cc <- openfhe.R::fhe_context("CKKS", multiplicative_depth = 1L,
                             scaling_mod_size = 50L, batch_size = 8L)
keys <- openfhe.R::key_gen(cc)
master <- make_ckks_master("Master", crypto_context = cc, keypair = keys)
```

```

set_workers(master, list(
  make_worker("S1", y[1:3], local_n11),
  make_worker("S2", y[4:6], local_n11),
  make_worker("S3", y[7:10], local_n11)))

fit <- stats4::mle(function(lambda) master_aggregate(master, lambda),
  start = list(lambda = 5))
c(encrypted = stats4::coef(fit)[["lambda"]], cleartext = mean(y))

```

keygen_round

One site's step in the threshold key-generation chain

Description

The site derives its own secret share from its predecessor's cumulative public key, **keeps the share**, and returns only the new cumulative public key. The share is generated at the site and is never a return value, so no other party can hold it.

Usage

```
keygen_round(site, ...)
```

Arguments

site	a LocalSite , or a user-defined subclass of RemoteSite .
...	method-specific arguments; the built-in method takes cc, the crypto context, and prev_pk, the cumulative public key from the previous site in the chain (NULL for the lead site, which starts the chain with a fresh keypair).

Details

Called by [make_threshold_master\(\)](#), once per site, in order. A [RemoteSite](#) implementation must do the same thing at the far end: receive a public key, generate and retain a share locally, send a public key back. Nothing secret crosses the wire in either direction.

Value

the cumulative public key including this site's contribution. Never a secret key.

See Also

[make_threshold_master\(\)](#), [partial_decrypt\(\)](#).

LocalSite

A site whose data lives in this R session

Description

The ordinary case: the records are here, and `contribution_fn` is evaluated in-process. `contribute()` computes the contribution and **encrypts it** with the public parameters the site was given when it was configured, so what leaves is already a ciphertext.

Usage

```
LocalSite(
  name = character(0),
  state = new.env(parent = emptyenv()),
  data = NULL,
  contribution_fn = function() NULL
)
```

Arguments

<code>name</code>	short identifier shown in printed output. A single non-empty string; it names the site in every error message, so an empty or vectorized name is rejected at construction.
<code>state</code>	an environment for mutable bookkeeping — the public parameters the site was given when it was configured, its own key share under threshold keys, and, on the frozen legacy path, the next link in the round-robin chain. Default: a fresh empty env.
<code>data</code>	whatever <code>contribution_fn</code> needs in order to answer — a dataset, a database connection, a cohort identifier.
<code>contribution_fn</code>	a function with signature <code>function(data, theta)</code> returning this site's contribution at <code>theta</code> as a plain numeric value. It does not encrypt; <code>contribute()</code> does that. May return NA to signal that <code>theta</code> is non-evaluable here.

Details

A `LocalSite` demonstrates the protocol's roles inside one R session. It is not a deployment boundary: its data, and under threshold keys its key share, are objects in this process, and anything else in this process can reach them. Separating the parties for real means separately controlled processes, which is what `RemoteSite` is for.

Value

an S7 object of class `LocalSite`. Construct with `make_worker()`.

make_ckks_master	<i>Construct a CKKS-backed master</i>
------------------	---------------------------------------

Description

The context must be a CKKS one. A [CKKSMaster](#) built over BFV or BGV would work arithmetically but every sentence of its documentation, and the class name a user reads in printed output, would be wrong about which scheme is in use; exact-integer work goes through [make_threshold_master\(\)](#), which is scheme-agnostic by design and says so.

Usage

```
make_ckks_master(name, crypto_context, keypair)
```

Arguments

name	short identifier shown in printed output.
crypto_context	an <code>openfhe.R</code> <code>CryptoContext</code> configured for CKKS.
keypair	an <code>openfhe.R</code> <code>KeyPair</code> .

Value

a [CKKSMaster](#).

make_master	<i>Construct a Paillier-backed master</i>
-------------	---

Description

Part of the frozen Paillier-era legacy surface.

Usage

```
make_master(name, keypair, den = gmp::as.bigq(2)^256)
```

Arguments

name	short identifier shown in printed output.
keypair	a PaillierKeyPair .
den	a gmp::bigq denominator used to scale fractional parts when encrypting real numbers via encrypt_real() .

Value

a [PaillierMaster](#).

make_ncparty	Construct an NCParty
--------------	--------------------------------------

Description

Part of the frozen Paillier-era legacy surface.

Usage

```
make_ncparty(name, number)
```

Arguments

name	short identifier shown in printed output.
number	which share this NCP receives, 1 or 2.

Value

an [NCParty](#).

make_threshold_master	Run threshold key generation across sites and construct the master
-----------------------	--

Description

Drives the chained key-generation ceremony *through the sites* and returns a master wired to them. The lead site generates a fresh keypair (pk_1, sk_1) ; each subsequent site i derives $(pk_{\{1..i\}}, sk_i)$ from its predecessor's cumulative public key. The final $pk_{\{1..n\}}$ is the joint public key under which everything is encrypted.

Usage

```
make_threshold_master(name, crypto_context, sites)
```

Arguments

name	short identifier.
crypto_context	an openfhe.R CryptoContext (CKKS, BFV, or BGV) <i>with</i> the MULTIPARTY feature enabled. Pass <code>features = c(Feature\$MULTIPARTY)</code> to <code>fhe_context()</code> . The scheme is read back from the context, so the same master drives the protocol over real-valued (CKKS) or exact-integer (BFV/BGV) arithmetic without further configuration.
sites	a list of at least two distinct, unconfigured Sites , built with make_worker() . The first is the lead site. Each ends up holding its own secret share and the joint public key. Listing one site twice, or reusing a site that already holds a share or public parameters, is an error: the repeat would discard what the first round left behind, and under BFV or BGV nothing afterwards detects the loss.

Details

Each step runs at the site, through `keygen_round()`: the site keeps `sk_i` in its own state and returns only the cumulative *public* key. No share is ever generated centrally, and none is returned to this function, so the master cannot hold one even by accident. Only public keys travel between parties, which is exactly what can be sent over a wire to an untrusted peer.

Decryption is n-of-n: `decrypt()` asks each site for a partial decryption via `partial_decrypt()` and fuses the results with `multiparty_decrypt_fusion`. There is no path by which the master decrypts alone.

The returned master is already wired, so `set_workers()` is neither needed nor permitted afterwards — the site order fixed by the key-generation chain is the order partial decryptions must be fused in, and re-wiring would break it.

A ceremony that fails part-way — an unimplemented `RemoteSite`, an unreachable endpoint, a context without MULTIPARTY — leaves no trace on the sites it had already visited: their shares and parameters are cleared before the error propagates, so the same sites can be used again once the cause is fixed. For a `RemoteSite` that undo reaches the local proxy only, so a remote implementation should tolerate a repeated ceremony.

Value

a `ThresholdMaster`, wired to sites.

What this does not defend against

The construction assumes participants follow the protocol (honest-but-curious). A site that deviates can (a) return a well-formed ciphertext that is not its honest contribution, (b) return a malformed partial decryption, which corrupts the fused plaintext *silently* — nothing in the scheme detects it — or (c) contribute a degenerate share during key generation, weakening the threshold. The chain is sequential, so each site also sees its predecessors' cumulative public key; OpenFHE's multiparty key generation carries no proofs of knowledge or commitments, so rogue-key behavior is not prevented here. Defending against any of this needs verifiable decryption and committed key generation, neither of which this package provides.

See Also

`keygen_round()`, `partial_decrypt()`, `actor-encryption`.

make_worker

Construct a worker

Description

Builds the `Site` one party contributes to a multi-party protocol. A Site becomes a *worker* once it has been wired and given its public parameters; from that point it is autonomous, computing and encrypting on its own.

Usage

```
make_worker(name, data, contribution_fn)
```

Arguments

name	short identifier shown in printed output. A single non-empty string; it names the site in every error message, so an empty or vectorized name is rejected at construction.
data	whatever <code>contribution_fn</code> needs in order to answer — a dataset, a database connection, a cohort identifier.
contribution_fn	a function with signature <code>function(data, theta)</code> returning this site's contribution at <code>theta</code> as a plain numeric value. It does not encrypt; <code>contribute()</code> does that. May return NA to signal that <code>theta</code> is non-evaluable here.

Value

a [LocalSite](#).

See Also

[RemoteSite](#) for a site whose contribution is produced outside this R session.

Master

Abstract master class

Description

Common base for [CKKSMaster](#) and [ThresholdMaster](#) (and the frozen legacy [PaillierMaster](#)). Concrete masters carry whatever context and public keys their cryptographic backend needs; the protocol body in `master_aggregate()` reaches sites through `contribute()` and recovers the total through the `decrypt()` generic, which dispatches on the concrete master class, so the same protocol runs over any backend.

Usage

```
Master(name = character(0), state = new.env(parent = emptyenv()))
```

Arguments

name	short identifier shown in printed output.
state	an environment for mutable bookkeeping.

Details

A master never encrypts site data, and has no encryption entry point at all. Each party encrypts its own values with `encrypt()`, which for a `Site` takes nothing but the site itself: it encrypts with the public parameters it was handed when it was wired. The asymmetry is deliberate and worth reading off the API: decryption is privileged — it needs secret material, or the standing to convene every site — while encryption needs only public material and is available to anyone.

Value

nothing — this class is abstract, so calling it raises an error instead of returning an object. It exists so that `decrypt()` and `master_aggregate()` dispatch on a common parent. Construct a concrete master with `make_ckks_master()` or `make_threshold_master()`.

master_aggregate	<i>Run one round of the master/worker protocol</i>
------------------	--

Description

Backend-agnostic: sites are reached through `contribute()` and the total is recovered through the `decrypt()` generic, so the same body works over `CKKSMaster` and `ThresholdMaster`.

Usage

```
master_aggregate(master, theta)
```

Arguments

master	a <code>Master</code> , wired to its workers — with <code>set_workers()</code> for a <code>CKKSMaster</code> , or by <code>make_threshold_master()</code> , which returns one already wired.
theta	the current parameter value (passed through to each worker).

Details

The master broadcasts `theta` to each worker — and only `theta`; each worker supplies its own data. Each worker returns `contribution_fn(data, theta)` and the result is encrypted under the master's public key. The master sums the encrypted contributions homomorphically and decrypts the total.

This is the topology that mirrors how distcomp, DataSHIELD, and similar federated-analysis frameworks actually deploy: a flat fan-out / fan-in. With a single-decrypter master, the master could in principle decrypt individual contributions; the cryptographic guarantee strengthens when paired with threshold key generation (no single party holds the secret key).

Each worker returns an *already encrypted* contribution (see `contribute()`), so no individual site's cleartext value reaches the master. Only the aggregate is decrypted.

Two failure modes, deliberately distinct. If a worker returns NA, `theta` is non-evaluable there and this function returns `NA_real_`, which optimizers read as "back off and try elsewhere". If a worker signals `site_unavailable()`, it could not be reached at all; that condition propagates and aborts

the round, because continuing would sum over a different set of sites and silently change the objective between optimizer iterations.

NA is the one value that travels in the clear, since CKKS cannot represent it. A master that chooses theta adaptively therefore learns which parameter values break which site — a residual side channel that no amount of encryption here removes.

Value

the aggregated value, or NA_real_ if some site found theta non-evaluable.

NCParty	<i>A non-cooperating party</i>
---------	--------------------------------

Description

Sits between the master and the sites in the non-cooperating-parties topology. Two NCPs receive *additive shares* of each site's contribution; each NCP sums its share across sites and ships the result to the master, who combines the two NCP totals and decrypts. No single party — neither master nor an NCP — sees an individual site's contribution. Use [make_ncparty\(\)](#) to construct.

Usage

```
NCParty(
  name = character(0),
  number = integer(0),
  state = new.env(parent = emptyenv())
)
```

Arguments

name	short identifier shown in printed output.
number	which share this NCP receives, 1 or 2.
state	an environment for mutable bookkeeping (the list of sites it manages, public key). Default: a fresh empty env.

Details

Part of the frozen Paillier-era legacy surface: the NCP masking construction compensated for Paillier's single decryption key, a role that threshold key generation ([make_threshold_master\(\)](#)) now fills without extra parties.

Value

an S7 object of class NCParty with properties name, number and state. number (1 or 2) records which of the two additive shares this party receives; state holds the sites it manages and the public key. Construct with [make_ncparty\(\)](#).

OpenFHEParams	<i>Public parameters for the openfhe.R backends</i>
---------------	---

Description

The crypto context and the public key to encrypt under — the joint public key when the protocol uses threshold keys. Both are public. The scheme is read back from the context, so one class serves CKKS, BFV, and BGV.

Usage

```
OpenFHEParams(cc = openfhe.R::CryptoContext(), pk = openfhe.R::PublicKey())
```

Arguments

cc	an openfhe.R CryptoContext.
pk	an openfhe.R PublicKey.

Value

an S7 object of class OpenFHEParams, inheriting from [PublicParams](#), with properties cc and pk.

See Also

[actor-encryption](#), [site_params\(\)](#)

paillier_keypair	<i>Generate a new Paillier key pair</i>
------------------	---

Description

Generates two random primes of modulus_bits / 2 bits each, forms the modulus, and returns a [PaillierKeyPair](#) containing the matching public and private keys.

Usage

```
paillier_keypair(modulus_bits)
```

Arguments

modulus_bits	modulus length in bits (e.g. 1024 or 2048).
--------------	---

Value

a [PaillierKeyPair](#).

Examples

```

keys <- paillier_keypair(1024)
ct <- encrypt(keys@pubkey, gmp::as.bigz(42))
ct

## Only the private key recovers the cleartext:
decrypt(get_private_key(keys), ct)

```

PaillierCiphertext	<i>A Paillier ciphertext</i>
--------------------	------------------------------

Description

Wraps the encrypted big-integer value together with the public key it was encrypted under. Two ciphertexts encrypted under the same public key can be combined with + and -; a ciphertext can be multiplied by a cleartext integer with *.

Usage

```
PaillierCiphertext(value = NULL, pubkey = PaillierPublicKey())
```

Arguments

value	the encrypted big-integer value.
pubkey	the PaillierPublicKey under which it was encrypted.

Value

an S7 object of class PaillierCiphertext with properties value (the encrypted big integer, which lives modulo n^2) and pubkey (the [PaillierPublicKey](#) it was encrypted under). Ciphertexts under the same key add and subtract with + and -, and multiply by a cleartext integer with *; [decrypt\(\)](#) recovers the cleartext.

PaillierEncryptedReal	<i>A Paillier-encrypted real number</i>
-----------------------	---

Description

A pair of Paillier ciphertexts representing the integer and fractional parts of a real number, together with the denominator used to scale the fractional part. Two PaillierEncryptedReal values encrypted under the same key with the same denominator combine via the standard arithmetic operators.

Usage

```
PaillierEncryptedReal(
    int = PaillierCiphertext(),
    frac = PaillierCiphertext(),
    den = NULL
)
```

Arguments

`int` the [PaillierCiphertext](#) holding the integer part.

`frac` the [PaillierCiphertext](#) holding the scaled fractional part.

`den` the denominator used to scale the fractional part (a [gmp::bigq](#)).

Value

an S7 object of class `PaillierEncryptedReal` with properties `int`, `frac` and `den`: the [PaillierCiphertext](#) carrying the integer part, the [PaillierCiphertext](#) carrying the fractional part scaled by `den`, and the denominator itself. It adds and subtracts with `+` and `-`; [decrypt\(\)](#) recombines the two parts and re-centers the result into $(-n/2, n/2)$ so that signed values round-trip. Produced by [encrypt_real\(\)](#).

Signed-arithmetic convention

Paillier’s plaintext space is $\mathbb{Z}_n$ (a residue class modulo n , the modulus carried by the public key). Negative real numbers and running totals that cross zero are stored in their mod- n representation, which lives in the upper half of $[0, n)$. The [decrypt\(\)](#) method for `PaillierEncryptedReal` re-centers the raw decrypted residues into the interval $(-n/2, n/2)$ so that signed values round-trip correctly. This means a `PaillierEncryptedReal` is *correct for signed real arithmetic* as long as the true cleartext stays in $(-n/2, n/2)$ — for default 1024-bit keys, that is $> 10^{307}$, well beyond any plausible statistical workload.

This convention applies only to `PaillierEncryptedReal`. The integer-only [PaillierCiphertext](#) decrypt method preserves raw mod- n semantics and does not center.

PaillierKeyPair	<i>Paillier key pair</i>
-----------------	--------------------------

Description

A matched pair of public and private keys. Use [paillier_keypair\(\)](#) to generate one.

Usage

```
PaillierKeyPair(pubkey = PaillierPublicKey(), privkey = PaillierPrivateKey())
```

Arguments

pubkey a [PaillierPublicKey](#).
 privkey a [PaillierPrivateKey](#).

Value

an S7 object of class `PaillierKeyPair` with properties `pubkey` (a [PaillierPublicKey](#)) and `privkey` (a [PaillierPrivateKey](#)) — the two halves of one generated key. Returned by `paillier_keypair()`.

PaillierMaster	<i>Paillier-backed master</i>
----------------	-------------------------------

Description

A [Master](#) that drives the protocol over Paillier additive encryption. Constructed by `make_master()`. Part of the frozen Paillier-era legacy surface; new work should use `make_ckks_master()` or `make_threshold_master()`.

Usage

```
PaillierMaster(  
  name = character(0),  
  state = new.env(parent = emptyenv()),  
  keypair = PaillierKeyPair(),  
  den = NULL  
)
```

Arguments

name short identifier shown in printed output.
 state an environment for mutable bookkeeping.
 keypair a [PaillierKeyPair](#).
 den a `gmp::bigq` denominator used to scale fractional parts when encrypting real numbers via `encrypt_real()`.

Value

an S7 object of class `PaillierMaster`, inheriting from [Master](#), with properties `name`, `keypair`, `den` and `state`: the [PaillierKeyPair](#) the protocol encrypts under, and the denominator used to scale fractional parts when encrypting reals. Construct with `make_master()`.

PaillierParams	<i>Public parameters for the frozen Paillier backend</i>
----------------	--

Description

Part of the frozen Paillier-era legacy surface. Paillier's public setup is a public key plus the fixed-point denominator its real-valued encoding needs; both are public, so a Paillier site encrypts its own contribution in `contribute()` exactly like an OpenFHE one. The supported backends use [OpenFHEParams](#).

Usage

```
PaillierParams(pk = PaillierPublicKey(), den = NULL)
```

Arguments

pk	a PaillierPublicKey .
den	a gmp::bigq denominator used to scale fractional parts.

Value

an S7 object of class `PaillierParams`, inheriting from [PublicParams](#), with properties `pk` and `den`.

PaillierPrivateKey	<i>Paillier private key</i>
--------------------	-----------------------------

Description

Holds the secret `lambda` and a cached value `x` used during decryption, together with a reference to the matching public key.

Usage

```
PaillierPrivateKey(lambda, pubkey)
```

Arguments

lambda	the secret <code>lambda</code> .
pubkey	the matching PaillierPublicKey .

Value

an S7 object of class `PaillierPrivateKey` with properties `pubkey`, `lambda` and `x`: the matching [PaillierPublicKey](#), the secret `lambda`, and a value cached from it so that decryption does not recompute a modular inverse each time. Obtain one with `get_private_key()` on the pair returned by `paillier_keypair()`.

PaillierPublicKey	<i>Paillier public key</i>
-------------------	----------------------------

Description

Holds the modulus and precomputed values used during encryption. Construct via `paillier_keypair()` rather than directly.

Usage

```
PaillierPublicKey(bits, n)
```

Arguments

bits	modulus length in bits.
n	the modulus.

Value

an S7 object of class PaillierPublicKey with properties bits, n, n_squared and n_plus_one: the modulus length, the modulus itself, and the two values precomputed from it that encryption needs. Obtain one as the pubkey component of the pair returned by `paillier_keypair()` rather than constructing it directly.

partial_decrypt	<i>One site's partial decryption of a ciphertext</i>
-----------------	--

Description

Under threshold keys no party can decrypt alone. A ciphertext is sent to each site; each site applies **its own** secret share and returns a partial decryption, and the partials are fused (see `decrypt()`). The share never leaves the site, so no other party ends up holding anything that would let it decrypt.

Usage

```
partial_decrypt(site, ...)
```

Arguments

site	a LocalSite , or a user-defined subclass of RemoteSite .
...	method-specific arguments; the built-in method takes ciphertext and lead, a flag marking the first site in the chain.

Details

Whether a site plays the lead role is fixed by its position in the key-generation chain, so it arrives with the request; the site does not choose and does not need to know who is asking.

A site that cannot be reached must signal `site_unavailable()`. Because decryption is n-of-n, this loses the entire round rather than one summand — see the availability note in [RemoteSite](#).

Value

a partial decryption, to be fused by `decrypt()`.

See Also

`make_threshold_master()`, `keygen_round()`.

PublicParams

The public parameters a party encrypts under

Description

Abstract base for the setup bundle a party is handed once, when it is configured, and holds from then on. It is public in full: it is exactly the message a coordinator would put on a wire to an untrusted peer, and it is all anyone needs in order to encrypt.

Usage

`PublicParams()`

Details

The class carries **no secret property**, which is what makes the claim structural rather than a promise in prose — there is nowhere for a secret key or a key share to travel in this object. The concrete kinds are [OpenFHEParams](#) and, on the frozen legacy path, `PaillierParams`.

Obtain the bundle a site holds with `site_params()`; encrypt with `encrypt()`.

Value

nothing — this class is abstract. Its concrete subclasses are constructed for you when a party is configured.

See Also

[OpenFHEParams](#), `site_params()`, [actor-encryption](#)

random.bigz	<i>Random big integer</i>
-------------	---------------------------

Description

Returns a random big integer using the cryptographically secure generator from the sodium package.

Usage

```
random.bigz(nBits)
```

Arguments

nBits	number of bits, which must be a multiple of 8 (not checked, for efficiency).
-------	--

Value

a `gmp::bigz` value.

RemoteSite	<i>A site whose contribution is produced outside this R session</i>
------------	---

Description

Abstract. [homomorpheR](#) deliberately ships **no** implementation: transports differ too much, and a crypto package has no business carrying an HTTP client. Subclass it, add whatever properties your transport needs, and register methods:

Usage

```
RemoteSite(name = character(0), state = new.env(parent = emptyenv()))
```

Arguments

name	short identifier shown in printed output. A single non-empty string; it names the site in every error message, so an empty or vectorized name is rejected at construction.
state	an environment for mutable bookkeeping — the public parameters the site was given when it was configured, its own key share under threshold keys, and, on the frozen legacy path, the next link in the round-robin chain. Default: a fresh empty env.

Details

```

HttpSite <- S7::new_class("HttpSite", parent = RemoteSite,
  properties = list(url = S7::class_character))
S7::method(set_public_params, HttpSite) <- function(site, params) {
  ## ... POST the public context and key to site@url; the far end
  ##   stores them. Nothing secret travels.
}
S7::method(contribute, HttpSite) <- function(site, theta) {
  ## ... call site@url with theta; the far end encrypts ...
}

```

Value

nothing — this class is abstract. Subclass it as shown above.

What this class is, and is not

A RemoteSite is an **architectural seam with a documented contract**, not a trust boundary the package establishes. Three cases are worth keeping apart:

A LocalSite demonstration. Data, key shares, sites, and the aggregating party are all objects in one R process. The classes model the protocol's *roles*; they do not create a process or trust boundary, and nothing prevents one object from reaching another. This is the right scope for a vignette.

A single-decrypter deployment. Each site returns a ciphertext, but a [CKKSMaster](#) holds the secret key and could decrypt an individual contribution. "Only the aggregate is decrypted" describes what [master_aggregate\(\)](#) does, not something the cryptography enforces.

A remote threshold deployment. Separately controlled endpoints keep their own shares and return ciphertexts or partial decryptions. Here the party boundary is real — provided *your* transport, authentication, endpoint code, and key storage implement it. [homomorpheR](#) supplies none of those, and detects no deliberately dishonest reply.

What the package does enforce: a site cannot be configured except through [set_public_params\(\)](#), which the base RemoteSite method refuses, so an endpoint that was never provisioned fails closed rather than looking wired; the base class likewise refuses [contribute\(\)](#), [keygen_round\(\)](#) and [partial_decrypt\(\)](#) rather than evaluating a remote party's data or performing its secret-key operation in this process; and [master_aggregate\(\)](#) checks that a reply is an encrypted value under this protocol's key before adding it to a total.

The contract an implementation must honor

Provision the far end at setup. Implement [set_public_params\(\)](#) to send the public context and key to the endpoint and have it retain them. This is one of only two moments anything passes between the parties — the other being a round. Only public material travels.

Return a ciphertext, never a plain number. The remote end was given the public parameters when it was wired, so it encrypts *before* the value crosses the wire. A RemoteSite that returns cleartext hands the aggregator an individual per-site contribution, which is precisely what the

protocol exists to prevent — `master_aggregate()` now refuses such a reply, but an honest implementation should not produce one. NA is the one permitted plaintext reply, because CKKS has no representation for it; the aggregator consequently learns which theta a site could not evaluate, and that residual side channel is documented in `master_aggregate()`.

Distinguish "non-evaluable" from "unreachable". NA means *this theta broke my solver* — the optimizer will back off and try a different parameter, which is the right response. A network, authentication, or timeout failure is a different event, and backing off to another theta does nothing about it. Signal `site_unavailable()` instead. Never return NA for an unreachable service.

Do not drop out silently. A round sums over all sites. A site that quietly returns nothing changes the objective function between optimizer iterations, so the fit converges to something that is not the estimand, with no error raised anywhere. Aborting the round is always preferable.

Be deterministic in theta. The same theta must give the same contribution. Optimizers estimate gradients by finite differences, so a service that re-samples or jitters its answer turns the gradient into noise — with `optim()`'s default `ndeps = 1e-3` the amplification is roughly 700-fold. Determinism also makes retries safe.

Budget timeouts against call count. A single `mle()` fit may query every site hundreds of times. A per-call timeout that looks reasonable in isolation is not.

With a `ThresholdMaster`, availability is not optional. Decryption is n-of-n, so an unreachable site withholds a partial decryption and the round cannot be decrypted at all. Under a `CKKS-Master` an unavailable site costs you a summand; under threshold keys it costs you the entire result.

What the package leaves to you

Transport, identity, authentication, attestation, remote key storage, serialization of the parameter bundle, retry and timeout policy — and any defense against a party that deviates from the protocol rather than merely observing it. The trust model throughout is honest-but-curious.

See Also

`set_public_params()`, `contribute()`, `keygen_round()`, `partial_decrypt()`, `site_unavailable()`, `LocalSite`

round_robin_chain

Wire a master and a list of sites into a round-robin chain

Description

Sets `master -> sites[[1]] -> sites[[2]] -> ... -> sites[[n]] -> master` and broadcasts the master's public key to every site. After this call, `run_round_robin()` can drive an iteration of the protocol.

Usage

```
round_robin_chain(master, sites)
```

Arguments

master a [Master](#).
 sites a list of [Sites](#).

Details

Part of the frozen Paillier-era legacy surface; the supported topology is [set_workers\(\)](#) + [master_aggregate\(\)](#).

Value

the master, invisibly.

run_round_robin	<i>Run one round of the round-robin protocol</i>
-----------------	--

Description

Backend-agnostic via the [decrypt\(\)](#) generic, but part of the frozen Paillier-era legacy surface: the random-offset chain idiom compensated for Paillier-era trust assumptions, and it encrypts each site's value *at the master*, which the supported topology deliberately does not. The supported pattern is [master_aggregate\(\)](#).

Usage

```
run_round_robin(master, theta)
```

Arguments

master a [Master](#), wired to a chain via [round_robin_chain\(\)](#).
 theta the current parameter value (passed through to each worker's `contribution_fn`).

Details

The master generates a random real offset, encrypts it under its public key, and sends it around the chain. Each worker site adds its encrypted local summary to the running total and forwards. On return, the master decrypts the running total, subtracts the offset in the clear, and returns the resulting scalar.

If any worker's `contribution_fn` returns NA, the chain stops and this function returns `NA_real_`.

Value

the aggregated value, or `NA_real_` on failure.

set_next_site	<i>Wire one site's next_site to another</i>
---------------	---

Description

Part of the frozen Paillier-era legacy surface (round-robin chain wiring); the supported topology is [set_workers\(\)](#) + [master_aggregate\(\)](#).

Usage

```
set_next_site(obj, ...)
```

Arguments

obj	a Site or Master .
...	method-specific arguments. The Site/Master methods take a single next_site.

Value

the object obj, invisibly. Called for its side effect: next_site is recorded in obj's state environment, forming one link of the round-robin chain.

set_public_key	<i>Distribute the public key from the master to a downstream actor</i>
----------------	--

Description

Part of the frozen Paillier-era legacy surface, used by [round_robin_chain\(\)](#). The supported setup seam is [set_public_params\(\)](#), which carries the whole public bundle and which a [RemoteSite](#) can implement.

Usage

```
set_public_key(obj, ...)
```

Arguments

obj	a Site (or legacy NCParty) to receive the key.
...	method-specific arguments. The methods take a single public key pubkey of the master's backend type.

Value

the object obj, invisibly. Called for its side effect: the master's public key is stored in the receiving actor's state environment, and in the [NCParty](#) method is forwarded on to every [Site](#) that party manages, so each site can encrypt under it.

set_public_params	<i>Give a party the public parameters it will encrypt under</i>
-------------------	---

Description

The setup step of the protocol, and one of only two moments at which anything passes between a coordinating party and a site — the other being a round itself, which carries a query out and a ciphertext back. A party receives its [PublicParams](#) once, here, and from then on computes and encrypts with what it holds.

Usage

```
set_public_params(site, ...)
```

Arguments

site	a Site , or a user-defined subclass of RemoteSite .
...	method-specific arguments; the built-in method takes params, a PublicParams object.

Details

Called for you by [set_workers\(\)](#) and [make_threshold_master\(\)](#). You would call it directly only when writing a [RemoteSite](#) method.

Value

the site, invisibly. Called for its side effect.

Why this is a generic

Setup is a *message*. For a co-located site, delivering it is an assignment; for a remote one it is a network call that must provision the far endpoint, and nothing in this process can do that on the endpoint's behalf. Writing the parameters straight into a remote proxy's state would leave the proxy looking configured while the far end had never been told anything — a setup failure that surfaces only much later, as a wrong answer. So the base [RemoteSite](#) method **refuses**, and a subclass must implement the provisioning it alone knows how to do. Missing remote setup fails closed.

What crosses is public in full: a crypto context and a public key. There is no secret material in a [PublicParams](#) object and no property for one to occupy.

Reconfiguring

Receiving the same parameters again is harmless and allowed. Receiving *different* ones is refused. A site that silently switched keys would keep answering its first coordinator, in a key that coordinator cannot read — under CKKS that surfaces as an approximation-error abort, and under BFV or BGV as a plausible wrong integer with nothing raised. Build a fresh site instead; they are cheap.

See Also

[site_params\(\)](#) to read them back, [actor-encryption](#) for using them, [RemoteSite](#) for the full remote contract.

set_workers

Wire a master to a flat list of workers

Description

Stashes the workers in the master's state and publishes its public parameters to each one. That bundle is public: it is the setup broadcast a coordinator would send over the wire, and it is all a site needs in order to encrypt. After this call, [master_aggregate\(\)](#) can drive an iteration of the protocol.

Usage

```
set_workers(master, workers)
```

Arguments

master	a CKKSMaster , or the frozen legacy PaillierMaster .
workers	a list of worker Sites .

Details

Use this for the realistic master/worker (star) topology that distcomp- and DataSHIELD-style federated analyses follow. For the legacy Paillier round-robin idiom, use [round_robin_chain\(\)](#) instead.

A [ThresholdMaster](#) does **not** use this function: its joint public key does not exist until key generation has run through every site, so [make_threshold_master\(\)](#) takes the sites and returns a master already wired to them, in the order the chain fixed.

Value

the master, invisibly.

similarity_results	<i>Precomputed results for the similarity vignette</i>
--------------------	--

Description

Precomputed results for the similarity vignette

Usage

```
similarity_results
```

Format

A list of the encrypted walk-through's outputs: the printed public parameters (`pub_print`), the smoke-test errors (`rot_err`, `matvec_err`, `ip_err`, `fold_err`, `slots_same`, `slots_dev`), the site-1 and full-query timings (`site1_n`, `site1_elapsed`, `query_elapsed`), the encrypted top-k table (`top_result`), and its agreement with the cleartext reference (`score_err`, `set_match`).

Source

`data-raw/similarity_results.R`, from `vignettes/similarity.Rmd`.

Site	<i>A site in a multi-party protocol</i>
------	---

Description

Abstract base for the two kinds of participating party: a [LocalSite](#), whose data is in this R session, and a [RemoteSite](#), whose contribution is produced elsewhere. Both answer the same generic, `contribute()`, and are indistinguishable to whoever asks: each returns an *encrypted* contribution at the requested parameter.

Usage

```
Site(name = character(0), state = new.env(parent = emptyenv()))
```

Arguments

<code>name</code>	short identifier shown in printed output. A single non-empty string; it names the site in every error message, so an empty or vectorized name is rejected at construction.
<code>state</code>	an environment for mutable bookkeeping — the public parameters the site was given when it was configured, its own key share under threshold keys, and, on the frozen legacy path, the next link in the round-robin chain. Default: a fresh empty env.

Details

A site is autonomous once constructed. It is given public parameters once, when it is wired, and from then on it computes and encrypts entirely on its own — it holds no reference to the party that aggregates its answers, and needs none.

Value

nothing — this class is abstract, so calling it raises an error instead of returning an object. It is the common parent of [LocalSite](#) and [RemoteSite](#), and the dispatch target for [contribute\(\)](#). Construct a co-located site with [make_worker\(\)](#).

See Also

[LocalSite](#), [RemoteSite](#), [contribute\(\)](#)

 site_params

The public parameters a party holds

Description

Reads back what [set_public_params\(\)](#) delivered. Encryption needs only this, so a party that has it is self-sufficient, and any other party that will encrypt under the same key — a querier that is not itself a site, say — can be handed a copy.

Usage

```
site_params(site, ...)
```

Arguments

site	a Site , or a user-defined subclass of RemoteSite .
...	method-specific arguments; the built-in method takes none.

Details

Aborts if the site was never configured, rather than returning NULL for a caller to encrypt with.

Value

a [PublicParams](#) object.

See Also

[set_public_params\(\)](#), [actor-encryption](#)

site_unavailable	<i>Signal that a site could not be reached</i>
------------------	--

Description

The condition a [RemoteSite](#) implementation raises when a transport, authentication, or timeout failure stops it from answering. This is **not** the same event as returning NA, which means the requested theta is non-evaluable at a site that answered perfectly well; see the contract in [RemoteSite](#). Raising it aborts the round rather than silently changing the set of sites being summed over.

Usage

```
site_unavailable(message, site = NULL, parent = NULL)
```

Arguments

message	what went wrong, for the caller.
site	optionally, the Site that was unreachable; its name is added to the message by master_aggregate() .
parent	optionally, the underlying condition (an http2 error, say) to chain for debugging.

Value

nothing — called for its side effect of signaling a condition of class `homomorpheR_site_unavailable`.

What the re-raised condition carries

When [master_aggregate\(\)](#) or [decrypt\(\)](#) re-raise this, the condition they signal carries a `site_name` field and **not** the site object. A [LocalSite](#) would drag its data, and under threshold keys its key share, into anything that logs or serializes the condition. Catch on the class and read `cond$site_name`.

ThresholdMaster	<i>Threshold-CKKS master (n-of-n key generation)</i>
-----------------	--

Description

A [Master](#) that drives the protocol over `openfhe.R` with threshold key generation, under whichever scheme the supplied crypto context was built for (CKKS for real-valued work, BFV or BGV for exact integer work). There is no single secret key: each site generates and keeps its own share `sk_i`, and the joint public key `pk_{1..n}` is built by chaining [keygen_round\(\)](#) across the sites. Encryption goes under `joint_pubkey`. Decryption requires all `n` sites to return partial decryptions, which the master then fuses.

Usage

```
ThresholdMaster(
  name = character(0),
  state = new.env(parent = emptyenv()),
  crypto_context = openfhe.R::CryptoContext(),
  joint_pubkey = openfhe.R::PublicKey()
)
```

Arguments

<code>name</code>	short identifier.
<code>state</code>	an environment for mutable bookkeeping (the wired sites, in the order the key-generation chain visited them).
<code>crypto_context</code>	an openfhe.R <code>CryptoContext</code> with the MULTIPARTY feature enabled.
<code>joint_pubkey</code>	the joint public key produced by chaining keygen_round() across the sites.

Details

The master has no secret-key or secret-share property, and its methods use no secret material. Its properties are the crypto context and the joint public key, both public; the shares live at the sites that generated them and never travel. That is what makes the n-of-n claim true of the objects and not merely of the prose — see [partial_decrypt\(\)](#) for the decryption seam.

Read that at the right scope. In a [LocalSite](#) demonstration every role still inhabits one R process, and the master holds the site objects in order to query them, so the shares are reachable from the master’s object graph even though no property of the master contains one. A boundary between the parties requires separately controlled processes behind [RemoteSite](#).

Value

an S7 object of class `ThresholdMaster`, inheriting from [Master](#), with properties `name`, `crypto_context`, `joint_pubkey` and `state`. It carries no secret key and no secret shares: decryption is driven by asking each site for a partial decryption and fusing the results, so no party — the master included — can decrypt alone. Construct with [make_threshold_master\(\)](#).

Exact-integer contexts

Under BFV or BGV a site cannot contribute a value the scheme cannot carry: [contribute\(\)](#) refuses a non-integer, a non-finite value, or one outside the plaintext modulus rather than rounding it. What no party can check is the *total*: a sum that exceeds the modulus wraps, and the wrapped value decrypts as an ordinary integer with nothing to mark it. Choose `plaintext_modulus` for the largest total the protocol can produce, not the largest summand.

Constructed by [make_threshold_master\(\)](#).

Index

* datasets

- cox_results, 7
- cox_threshold_dp_results, 7
- cox_threshold_results, 8
- cvxr_admm_dp_results, 8
- cvxr_consensus, 9
- DLBCL, 10
- DLBCL_gex, 11
- similarity_results, 37

actor-decryption (actor-encryption), 3

actor-encryption, 3, 19, 23, 29, 36, 38

add_site, 4

CKKSMaster, 3, 4, 5, 17, 20, 21, 31, 32, 36

contribute, 6

contribute(), 3, 4, 16, 20, 21, 27, 31, 32, 37, 38, 40

cox_results, 7, 8

cox_threshold_dp_results, 7

cox_threshold_results, 8

CVXR, 9

cvxr_admm_dp_results, 8

cvxr_consensus, 9

decrypt(), 3, 19–21, 24, 25, 28, 29, 33, 39

DLBCL, 9, 10, 10, 12

DLBCL_gex, 9, 10, 11, 11

encrypt(), 3, 21, 29

encrypt_real, 12

encrypt_real(), 17, 25, 26

get_lambda, 13

get_private_key, 13

get_private_key(), 27

gmp::bigq, 17, 25–27

gmp::bigz, 4, 13, 30

homomorpheR, 14, 30, 31

homomorpheR-package (homomorpheR), 14

keygen_round, 15

keygen_round(), 19, 29, 31, 32, 39, 40

knitr::purl(), 9

knitr::read_chunk(), 9

LocalSite, 6, 14, 15, 16, 20, 28, 31, 32, 37–40

make_ckks_master, 17

make_ckks_master(), 5, 14, 21, 26

make_master, 17

make_master(), 26

make_ncparty, 18

make_ncparty(), 22

make_threshold_master, 18

make_threshold_master(), 14, 15, 17, 21, 22, 26, 29, 35, 36, 40

make_worker, 19

make_worker(), 14, 16, 18, 38

Master, 3, 5, 14, 20, 21, 26, 33, 34, 39, 40

master_aggregate, 21

master_aggregate(), 6, 14, 20, 21, 31–34, 36, 39

NCParty, 4, 5, 18, 22, 34

openfhe.R::decrypt(), 4

openfhe.R::encrypt(), 4

OpenFHEParams, 3, 23, 27, 29

paillier_keypair, 23

paillier_keypair(), 14, 25–28

PaillierCiphertext, 4, 12, 24, 25

PaillierEncryptedReal, 4, 12, 13, 24

PaillierKeyPair, 13, 17, 23, 25, 26

PaillierMaster, 3, 4, 17, 20, 26, 36

PaillierParams, 3, 27

PaillierPrivateKey, 3, 4, 13, 26, 27

PaillierPublicKey, 3, 12, 24, 26, 27, 28

partial_decrypt, 28

partial_decrypt(), 3, 4, 15, 19, 31, 32, 40

PublicParams, 23, 27, 29, 35, 38

random.bigz, [30](#)
RemoteSite, [6](#), [15](#), [16](#), [19](#), [20](#), [28](#), [29](#), [30](#),
 [34–40](#)
round_robin_chain, [32](#)
round_robin_chain(), [33](#), [34](#), [36](#)
run_round_robin, [33](#)
run_round_robin(), [32](#)

set_next_site, [34](#)
set_public_key, [34](#)
set_public_params, [35](#)
set_public_params(), [31](#), [32](#), [34](#), [38](#)
set_workers, [36](#)
set_workers(), [19](#), [21](#), [33–35](#)
similarity_results, [37](#)
Site, [3](#), [4](#), [14](#), [18](#), [19](#), [21](#), [33–36](#), [37](#), [38](#), [39](#)
site_params, [38](#)
site_params(), [3](#), [23](#), [29](#), [36](#)
site_unavailable, [39](#)
site_unavailable(), [6](#), [21](#), [29](#), [32](#)

ThresholdMaster, [3–5](#), [19–21](#), [32](#), [36](#), [39](#)