

HElib

Implementing Homomorphic Encryption

Generated by Doxygen 1.8.18



<b>1 HElib v1.0.2 Documentation</b>	<b>1</b>
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List . . . . .	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy . . . . .	5
<b>4 Class Index</b>	<b>9</b>
4.1 Class List . . . . .	9
<b>5 File Index</b>	<b>15</b>
5.1 File List . . . . .	15
<b>6 Namespace Documentation</b>	<b>19</b>
6.1 Fft Namespace Reference . . . . .	19
6.1.1 Function Documentation . . . . .	19
6.1.1.1 convolve() . . . . .	19
6.1.1.2 inverseTransform() . . . . .	19
6.1.1.3 transform() . . . . .	20
6.1.1.4 transformBluestein() . . . . .	20
6.1.1.5 transformRadix2() . . . . .	20
6.2 helib Namespace Reference . . . . .	20
6.2.1 Typedef Documentation . . . . .	41
6.2.1.1 aligned_vector . . . . .	42
6.2.1.2 cmplx_t . . . . .	42
6.2.1.3 CPtrMat . . . . .	42
6.2.1.4 CPtrMat_ptVecCt . . . . .	42
6.2.1.5 CPtrMat_ptvectorCt . . . . .	42
6.2.1.6 CPtrMat_VecCt . . . . .	42
6.2.1.7 CPtrMat_vectorCt . . . . .	42
6.2.1.8 CPtrs . . . . .	43
6.2.1.9 CPtrs_slice . . . . .	43
6.2.1.10 CPtrs_VecCt . . . . .	43
6.2.1.11 CPtrs_VecPt . . . . .	43
6.2.1.12 CPtrs_vectorCt . . . . .	43
6.2.1.13 CPtrs_vectorPt . . . . .	43
6.2.1.14 cx_double . . . . .	43
6.2.1.15 DCRTptr . . . . .	43
6.2.1.16 FlowGraph . . . . .	44
6.2.1.17 FNeighborList . . . . .	44
6.2.1.18 ldbl . . . . .	44
6.2.1.19 LNeighborList . . . . .	44
6.2.1.20 LONG . . . . .	44

6.2.1.21 NodeIdx	44
6.2.1.22 OneGeneratorTree	44
6.2.1.23 Permut	45
6.2.1.24 zzX	45
6.2.1.25 ZZXptr	45
6.2.2 Enumeration Type Documentation	45
6.2.2.1 PA_tag	45
6.2.3 Function Documentation	45
6.2.3.1 add() [1/3]	45
6.2.3.2 add() [2/3]	46
6.2.3.3 add() [3/3]	46
6.2.3.4 add1DMatrices()	46
6.2.3.5 addAllMatrices()	46
6.2.3.6 addBSGS1DMatrices()	46
6.2.3.7 addBSGSFrbMatrices()	47
6.2.3.8 addCtxtPrimes()	47
6.2.3.9 addFewMatrices()	47
6.2.3.10 addFrbMatrices()	47
6.2.3.11 addManyNumbers()	47
6.2.3.12 addMatrices4Network()	48
6.2.3.13 addMinimal1DMatrices()	48
6.2.3.14 addMinimalFrbMatrices()	49
6.2.3.15 addOffset()	49
6.2.3.16 addSmallPrimes()	49
6.2.3.17 addSome1DMatrices()	49
6.2.3.18 addSomeFrbMatrices()	49
6.2.3.19 addSomePrimes()	50
6.2.3.20 addTheseMatrices()	50
6.2.3.21 addTwoNumbers()	50
6.2.3.22 applyLinPoly() [1/2]	51
6.2.3.23 applyLinPoly() [2/2]	51
6.2.3.24 applyLinPoly1()	52
6.2.3.25 applyLinPolyLL() [1/4]	52
6.2.3.26 applyLinPolyLL() [2/4]	52
6.2.3.27 applyLinPolyLL() [3/4]	52
6.2.3.28 applyLinPolyLL() [4/4]	52
6.2.3.29 applyLinPolyMany()	53
6.2.3.30 applyPerm()	53
6.2.3.31 applyPermsToVec() [1/2]	53
6.2.3.32 applyPermsToVec() [2/2]	53
6.2.3.33 applyPermsToVec< long >() [1/2]	53
6.2.3.34 applyPermsToVec< long >() [2/2]	54

6.2.3.35 applyPermToVec() [1/2]	54
6.2.3.36 applyPermToVec() [2/2]	54
6.2.3.37 applyPermToVec< long >() [1/2]	54
6.2.3.38 applyPermToVec< long >() [2/2]	54
6.2.3.39 applyPermToVec< NTL::ZZX >()	55
6.2.3.40 areTimersOn()	55
6.2.3.41 argmax() [1/2]	55
6.2.3.42 argmax() [2/2]	55
6.2.3.43 argmin()	55
6.2.3.44 argminmax()	55
6.2.3.45 assertEq()	56
6.2.3.46 assertFalse()	57
6.2.3.47 assertInRange()	58
6.2.3.48 assertNeq()	59
6.2.3.49 assertNotNull()	60
6.2.3.50 assertTrue()	61
6.2.3.51 atoVec()	62
6.2.3.52 atovector()	62
6.2.3.53 balanced_MulMod()	62
6.2.3.54 balanced_zzX() [1/2]	63
6.2.3.55 balanced_zzX() [2/2]	63
6.2.3.56 balRem()	63
6.2.3.57 binaryCond()	63
6.2.3.58 binaryMask()	64
6.2.3.59 bitSetToLong()	65
6.2.3.60 bitwiseAnd() [1/2]	66
6.2.3.61 bitwiseAnd() [2/2]	66
6.2.3.62 bitwiseNot()	68
6.2.3.63 bitwiseOr()	69
6.2.3.64 bitwiseRotate()	69
6.2.3.65 bitwiseXOR()	70
6.2.3.66 BluesteinFFT() [1/2]	71
6.2.3.67 BluesteinFFT() [2/2]	71
6.2.3.68 BluesteinInit()	71
6.2.3.69 boundFreshNoise()	71
6.2.3.70 boundRoundingNoise() [1/2]	72
6.2.3.71 boundRoundingNoise() [2/2]	72
6.2.3.72 breakPermByDim()	72
6.2.3.73 breakPermTo3()	72
6.2.3.74 build_ConstMultiplier() [1/2]	72
6.2.3.75 build_ConstMultiplier() [2/2]	73
6.2.3.76 buildBenesCostTable()	73

6.2.3.77 BuildContext()	73
6.2.3.78 buildContextFromAscii()	73
6.2.3.79 buildContextFromBinary()	73
6.2.3.80 buildEncryptedArray()	73
6.2.3.81 buildGeneralAutomorphPrecon()	74
6.2.3.82 buildLinPolyCoeffs() [1/2]	74
6.2.3.83 buildLinPolyCoeffs() [2/2]	74
6.2.3.84 buildLinPolyMatrix() [1/2]	74
6.2.3.85 buildLinPolyMatrix() [2/2]	74
6.2.3.86 buildLookupTable()	75
6.2.3.87 buildModChain()	75
6.2.3.88 buildPAlgebraMod()	75
6.2.3.89 buildRandomBlockMatrix()	76
6.2.3.90 buildRandomFullBlockMatrix()	76
6.2.3.91 buildRandomFullMatrix()	76
6.2.3.92 buildRandomMatrix()	76
6.2.3.93 buildRandomMultiBlockMatrix()	76
6.2.3.94 buildRandomMultiMatrix()	76
6.2.3.95 buildUnpackSlotEncoding()	77
6.2.3.96 calcPolyNormBnd()	77
6.2.3.97 card()	77
6.2.3.98 CheckCtxt()	77
6.2.3.99 checkNoise()	77
6.2.3.100 CKKS_canonicalEmbedding() [1/3]	77
6.2.3.101 CKKS_canonicalEmbedding() [2/3]	78
6.2.3.102 CKKS_canonicalEmbedding() [3/3]	78
6.2.3.103 CKKS_embedInSlots()	78
6.2.3.104 cleanupDebugGlobals()	78
6.2.3.105 clear()	78
6.2.3.106 closeToOne()	78
6.2.3.107 coeffsL2Norm() [1/3]	79
6.2.3.108 coeffsL2Norm() [2/3]	79
6.2.3.109 coeffsL2Norm() [3/3]	79
6.2.3.110 coeffsL2NormSquared() [1/3]	79
6.2.3.111 coeffsL2NormSquared() [2/3]	79
6.2.3.112 coeffsL2NormSquared() [3/3]	79
6.2.3.113 comparePAlgebra() [1/2]	80
6.2.3.114 comparePAlgebra() [2/2]	80
6.2.3.115 compareTwoNumbers() [1/2]	80
6.2.3.116 compareTwoNumbers() [2/2]	82
6.2.3.117 compareTwoNumbersImplementation()	83
6.2.3.118 computeAllProducts()	83

6.2.3.119 computeDivVec()	83
6.2.3.120 computeIntervalForMul()	84
6.2.3.121 computeIntervalForSqr()	84
6.2.3.122 computeInvVec()	84
6.2.3.123 ComputeOneGenMapping()	84
6.2.3.124 computeProd() $[1/2]$	84
6.2.3.125 computeProd() $[2/2]$	85
6.2.3.126 concatBinaryNums()	85
6.2.3.127 conv() $[1/2]$	85
6.2.3.128 conv() $[2/2]$	86
6.2.3.129 convert() $[1/18]$	86
6.2.3.130 convert() $[2/18]$	86
6.2.3.131 convert() $[3/18]$	86
6.2.3.132 convert() $[4/18]$	86
6.2.3.133 convert() $[5/18]$	86
6.2.3.134 convert() $[6/18]$	87
6.2.3.135 convert() $[7/18]$	87
6.2.3.136 convert() $[8/18]$	87
6.2.3.137 convert() $[9/18]$	87
6.2.3.138 convert() $[10/18]$	87
6.2.3.139 convert() $[11/18]$	87
6.2.3.140 convert() $[12/18]$	88
6.2.3.141 convert() $[13/18]$	88
6.2.3.142 convert() $[14/18]$	88
6.2.3.143 convert() $[15/18]$	88
6.2.3.144 convert() $[16/18]$	88
6.2.3.145 convert() $[17/18]$	88
6.2.3.146 convert() $[18/18]$	89
6.2.3.147 convertDataToSlotVector()	89
6.2.3.148 CRTcoeff()	89
6.2.3.149 ctxtPrimeSize()	90
6.2.3.150 Cyclotomic()	90
6.2.3.151 decode() $[1/2]$	90
6.2.3.152 decode() $[2/2]$	90
6.2.3.153 decryptAndCompare()	90
6.2.3.154 decryptAndPrint()	90
6.2.3.155 decryptBinaryNums()	91
6.2.3.156 defaultPmiddle()	92
6.2.3.157 defaultQmiddle()	92
6.2.3.158 DestMulAdd()	93
6.2.3.159 disjoint()	93
6.2.3.160 div() $[1/2]$	93

6.2.3.161 div() [2/2]	93
6.2.3.162 divc()	93
6.2.3.163 DoubleCRT::Op< DoubleCRT::AddFun >() [1/3]	94
6.2.3.164 DoubleCRT::Op< DoubleCRT::AddFun >() [2/3]	94
6.2.3.165 DoubleCRT::Op< DoubleCRT::AddFun >() [3/3]	94
6.2.3.166 DoubleCRT::Op< DoubleCRT::MulFun >() [1/2]	94
6.2.3.167 DoubleCRT::Op< DoubleCRT::MulFun >() [2/2]	94
6.2.3.168 DoubleCRT::Op< DoubleCRT::SubFun >() [1/3]	94
6.2.3.169 DoubleCRT::Op< DoubleCRT::SubFun >() [2/3]	95
6.2.3.170 DoubleCRT::Op< DoubleCRT::SubFun >() [3/3]	95
6.2.3.171 EDF()	95
6.2.3.172 embeddingLargestCoeff() [1/4]	95
6.2.3.173 embeddingLargestCoeff() [2/4]	95
6.2.3.174 embeddingLargestCoeff() [3/4]	95
6.2.3.175 embeddingLargestCoeff() [4/4]	96
6.2.3.176 embeddingLargestCoeff_x2()	96
6.2.3.177 empty()	96
6.2.3.178 encode() [1/4]	96
6.2.3.179 encode() [2/4]	96
6.2.3.180 encode() [3/4]	96
6.2.3.181 encode() [4/4]	97
6.2.3.182 endBuildModChain()	97
6.2.3.183 equals() [1/3]	97
6.2.3.184 equals() [2/3]	97
6.2.3.185 equals() [3/3]	97
6.2.3.186 extendExtractDigits()	97
6.2.3.187 extractDigits() [1/2]	98
6.2.3.188 extractDigits() [2/2]	98
6.2.3.189 extractDigitsPacked()	98
6.2.3.190 extractDigitsThin()	98
6.2.3.191 factorize() [1/3]	99
6.2.3.192 factorize() [2/3]	99
6.2.3.193 factorize() [3/3]	99
6.2.3.194 fastPower()	99
6.2.3.195 fetch_saved_values()	99
6.2.3.196 fifteenOrLess4Four()	99
6.2.3.197 findGenerators()	100
6.2.3.198 FindM()	100
6.2.3.199 findMinBitCapacity() [1/3]	102
6.2.3.200 findMinBitCapacity() [2/3]	102
6.2.3.201 findMinBitCapacity() [3/3]	102
6.2.3.202 FindPrimitiveRoot() [1/2]	103



6.2.3.203 FindPrimitiveRoot() [2/2]	103
6.2.3.204 FindPrimRootT()	103
6.2.3.205 frobeniusAutomorph() [1/2]	103
6.2.3.206 frobeniusAutomorph() [2/2]	103
6.2.3.207 FrobeniusMap()	103
6.2.3.208 fsquare()	104
6.2.3.209 GenBabySteps()	104
6.2.3.210 getG()	104
6.2.3.211 getHyperColumn() [1/3]	104
6.2.3.212 getHyperColumn() [2/3]	104
6.2.3.213 getHyperColumn() [3/3]	105
6.2.3.214 getPhimXMod()	105
6.2.3.215 getTimerByName()	105
6.2.3.216 GetTimerClock()	105
6.2.3.217 incrementalProduct()	105
6.2.3.218 incrementalZeroTest()	105
6.2.3.219 innerProduct() [1/9]	106
6.2.3.220 innerProduct() [2/9]	106
6.2.3.221 innerProduct() [3/9]	106
6.2.3.222 innerProduct() [4/9]	106
6.2.3.223 innerProduct() [5/9]	106
6.2.3.224 innerProduct() [6/9]	106
6.2.3.225 innerProduct() [7/9]	107
6.2.3.226 innerProduct() [8/9]	107
6.2.3.227 innerProduct() [9/9]	107
6.2.3.228 interpolateMod()	108
6.2.3.229 intVecCRT() [1/4]	108
6.2.3.230 intVecCRT() [2/4]	108
6.2.3.231 intVecCRT() [3/4]	108
6.2.3.232 intVecCRT() [4/4]	109
6.2.3.233 InvModpr()	109
6.2.3.234 is2power()	109
6.2.3.235 is_in()	109
6.2.3.236 isDryRun()	110
6.2.3.237 isSetAutomorphVals()	110
6.2.3.238 isSetAutomorphVals2()	110
6.2.3.239 IsZero()	110
6.2.3.240 killVec() [1/2]	110
6.2.3.241 killVec() [2/2]	110
6.2.3.242 KSGiantStepSize()	110
6.2.3.243 largestCoeff() [1/5]	111
6.2.3.244 largestCoeff() [2/5]	111

6.2.3.245 largestCoeff() [3/5]	111
6.2.3.246 largestCoeff() [4/5]	111
6.2.3.247 largestCoeff() [5/5]	111
6.2.3.248 leftBitwiseShift()	111
6.2.3.249 length()	112
6.2.3.250 less_than() [1/8]	112
6.2.3.251 less_than() [2/8]	112
6.2.3.252 less_than() [3/8]	113
6.2.3.253 less_than() [4/8]	113
6.2.3.254 less_than() [5/8]	113
6.2.3.255 less_than() [6/8]	113
6.2.3.256 less_than() [7/8]	113
6.2.3.257 less_than() [8/8]	113
6.2.3.258 log2()	114
6.2.3.259 log2_realToEstimatedNoise()	114
6.2.3.260 longToBitVector()	114
6.2.3.261 lsize() [1/4]	114
6.2.3.262 lsize() [2/4]	115
6.2.3.263 lsize() [3/4]	115
6.2.3.264 lsize() [4/4]	115
6.2.3.265 make_lazy()	115
6.2.3.266 make_lazy_with_fun()	115
6.2.3.267 makeIrredPoly()	116
6.2.3.268 mapTo01() [1/4]	116
6.2.3.269 mapTo01() [2/4]	116
6.2.3.270 mapTo01() [3/4]	116
6.2.3.271 mapTo01() [4/4]	116
6.2.3.272 maximum_flow()	117
6.2.3.273 mcDiv()	117
6.2.3.274 mcMod()	117
6.2.3.275 mobius()	117
6.2.3.276 ModComp()	117
6.2.3.277 mul() [1/7]	118
6.2.3.278 mul() [2/7]	118
6.2.3.279 mul() [3/7]	118
6.2.3.280 mul() [4/7]	118
6.2.3.281 mul() [5/7]	118
6.2.3.282 mul() [6/7]	118
6.2.3.283 mul() [7/7]	119
6.2.3.284 MulAdd()	119
6.2.3.285 MulMod() [1/6]	119
6.2.3.286 MulMod() [2/6]	119

6.2.3.287 MulMod() [3/6]	119
6.2.3.288 MulMod() [4/6]	120
6.2.3.289 MulMod() [5/6]	120
6.2.3.290 MulMod() [6/6]	120
6.2.3.291 multOrd()	120
6.2.3.292 multTwoNumbers()	120
6.2.3.293 negate()	122
6.2.3.294 negateBinary()	122
6.2.3.295 normalize()	122
6.2.3.296 operator"!=()	122
6.2.3.297 operator&()	123
6.2.3.298 operator*()	123
6.2.3.299 operator*=(())	123
6.2.3.300 operator+()	123
6.2.3.301 operator+=(())	123
6.2.3.302 operator/() [1/2]	123
6.2.3.303 operator/() [2/2]	124
6.2.3.304 operator/=(())	124
6.2.3.305 operator<()	124
6.2.3.306 operator<<() [1/23]	124
6.2.3.307 operator<<() [2/23]	125
6.2.3.308 operator<<() [3/23]	125
6.2.3.309 operator<<() [4/23]	125
6.2.3.310 operator<<() [5/23]	125
6.2.3.311 operator<<() [6/23]	126
6.2.3.312 operator<<() [7/23]	126
6.2.3.313 operator<<() [8/23]	126
6.2.3.314 operator<<() [9/23]	126
6.2.3.315 operator<<() [10/23]	126
6.2.3.316 operator<<() [11/23]	126
6.2.3.317 operator<<() [12/23]	127
6.2.3.318 operator<<() [13/23]	127
6.2.3.319 operator<<() [14/23]	127
6.2.3.320 operator<<() [15/23]	127
6.2.3.321 operator<<() [16/23]	127
6.2.3.322 operator<<() [17/23]	127
6.2.3.323 operator<<() [18/23]	128
6.2.3.324 operator<<() [19/23]	128
6.2.3.325 operator<<() [20/23]	128
6.2.3.326 operator<<() [21/23]	128
6.2.3.327 operator<<() [22/23]	128
6.2.3.328 operator<<() [23/23]	128

6.2.3.329 operator<=()	129
6.2.3.330 operator==( )	129
6.2.3.331 operator>() [1/2]	129
6.2.3.332 operator>() [2/2]	129
6.2.3.333 operator>=()	129
6.2.3.334 operator>>() [1/12]	129
6.2.3.335 operator>>() [2/12]	130
6.2.3.336 operator>>() [3/12]	130
6.2.3.337 operator>>() [4/12]	130
6.2.3.338 operator>>() [5/12]	130
6.2.3.339 operator>>() [6/12]	131
6.2.3.340 operator>>() [7/12]	131
6.2.3.341 operator>>() [8/12]	131
6.2.3.342 operator>>() [9/12]	131
6.2.3.343 operator>>() [10/12]	131
6.2.3.344 operator>>() [11/12]	131
6.2.3.345 operator>>() [12/12]	132
6.2.3.346 operator^()	132
6.2.3.347 operator"   ()	132
6.2.3.348 optimalBenes()	132
6.2.3.349 optimalBenesAux()	132
6.2.3.350 optimalLower()	133
6.2.3.351 optimalUpperAux()	133
6.2.3.352 ord()	133
6.2.3.353 packConstant()	133
6.2.3.354 packConstants()	133
6.2.3.355 packedRecrypt() [1/4]	134
6.2.3.356 packedRecrypt() [2/4]	134
6.2.3.357 packedRecrypt() [3/4]	134
6.2.3.358 packedRecrypt() [4/4]	135
6.2.3.359 PAlgebraLift() [1/2]	135
6.2.3.360 PAlgebraLift() [2/2]	135
6.2.3.361 phi_N()	135
6.2.3.362 phiN() [1/2]	135
6.2.3.363 phiN() [2/2]	136
6.2.3.364 plaintextAutomorph() [1/2]	136
6.2.3.365 plaintextAutomorph() [2/2]	136
6.2.3.366 poly_comp()	136
6.2.3.367 polyEval() [1/2]	136
6.2.3.368 polyEval() [2/2]	137
6.2.3.369 polyEvalMod()	138
6.2.3.370 PolyRed() [1/4]	138

6.2.3.371 PolyRed() [2/4]	138
6.2.3.372 PolyRed() [3/4]	139
6.2.3.373 PolyRed() [4/4]	139
6.2.3.374 power()	139
6.2.3.375 pp_factorize()	139
6.2.3.376 pplInvert() [1/4]	139
6.2.3.377 pplInvert() [2/4]	140
6.2.3.378 pplInvert() [3/4]	140
6.2.3.379 pplInvert() [4/4]	140
6.2.3.380 ppsolve() [1/2]	140
6.2.3.381 ppsolve() [2/2]	141
6.2.3.382 primroot()	141
6.2.3.383 print() [1/2]	141
6.2.3.384 print() [2/2]	141
6.2.3.385 print3D() [1/3]	141
6.2.3.386 print3D() [2/3]	142
6.2.3.387 print3D() [3/3]	142
6.2.3.388 print_stats()	142
6.2.3.389 printAllTimers()	142
6.2.3.390 printFlow()	142
6.2.3.391 printNamedTimer()	142
6.2.3.392 printVec()	143
6.2.3.393 printZZX()	143
6.2.3.394 ptr2nonNull()	143
6.2.3.395 random()	143
6.2.3.396 randomPerm()	143
6.2.3.397 randomSlot()	143
6.2.3.398 randomSlot< BGV >()	144
6.2.3.399 randomSlot< CKKS >()	144
6.2.3.400 RandPoly()	144
6.2.3.401 range() [1/2]	144
6.2.3.402 range() [2/2]	144
6.2.3.403 rationalApprox() [1/2]	144
6.2.3.404 rationalApprox() [2/2]	145
6.2.3.405 rawDecrypt()	145
6.2.3.406 read()	145
6.2.3.407 read_ntl_vec_long()	145
6.2.3.408 read_raw_double()	145
6.2.3.409 read_raw_int()	145
6.2.3.410 read_raw_int32()	146
6.2.3.411 read_raw_vector() [1/3]	146
6.2.3.412 read_raw_vector() [2/3]	146

6.2.3.413 read_raw_vector() [3/3]	146
6.2.3.414 read_raw_vector< double >()	146
6.2.3.415 read_raw_vector< long >()	146
6.2.3.416 read_raw_xdouble()	147
6.2.3.417 read_raw_ZZ()	147
6.2.3.418 readContextBase()	147
6.2.3.419 readContextBaseBinary()	147
6.2.3.420 readContextBinary()	147
6.2.3.421 readEyeCatcher()	148
6.2.3.422 readPubKeyBinary()	148
6.2.3.423 readSecKeyBinary()	148
6.2.3.424 realToEstimatedNoise()	148
6.2.3.425 recordAutomorphVal()	148
6.2.3.426 recordAutomorphVal2()	148
6.2.3.427 reducedCount()	149
6.2.3.428 reduceModPhimX()	149
6.2.3.429 registerTimer()	149
6.2.3.430 RelaxedInv() [1/2]	149
6.2.3.431 RelaxedInv() [2/2]	149
6.2.3.432 rem()	149
6.2.3.433 removeDups()	150
6.2.3.434 repack() [1/2]	150
6.2.3.435 repack() [2/2]	150
6.2.3.436 replicate() [1/3]	150
6.2.3.437 replicate() [2/3]	151
6.2.3.438 replicate() [3/3]	151
6.2.3.439 replicate0()	151
6.2.3.440 replicateAll() [1/3]	152
6.2.3.441 replicateAll() [2/3]	152
6.2.3.442 replicateAll() [3/3]	152
6.2.3.443 replicateAllOrig()	153
6.2.3.444 resetAllTimers()	153
6.2.3.445 resize() [1/7]	153
6.2.3.446 resize() [2/7]	153
6.2.3.447 resize() [3/7]	154
6.2.3.448 resize() [4/7]	154
6.2.3.449 resize() [5/7]	154
6.2.3.450 resize() [6/7]	154
6.2.3.451 resize() [7/7]	154
6.2.3.452 reverse()	155
6.2.3.453 RLWE()	155
6.2.3.454 RLWE1()	155

6.2.3.455 rotate() [1/2]	155
6.2.3.456 rotate() [2/2]	155
6.2.3.457 runningSums() [1/2]	156
6.2.3.458 runningSums() [2/2]	156
6.2.3.459 sameObject()	156
6.2.3.460 sampleGaussian() [1/4]	156
6.2.3.461 sampleGaussian() [2/4]	156
6.2.3.462 sampleGaussian() [3/4]	157
6.2.3.463 sampleGaussian() [4/4]	157
6.2.3.464 sampleGaussianBounded()	157
6.2.3.465 sampleHWt() [1/3]	157
6.2.3.466 sampleHWt() [2/3]	157
6.2.3.467 sampleHWt() [3/3]	158
6.2.3.468 sampleHWtBounded()	158
6.2.3.469 sampleHWtBoundedEffectiveBound()	158
6.2.3.470 sampleSmall() [1/3]	158
6.2.3.471 sampleSmall() [2/3]	158
6.2.3.472 sampleSmall() [3/3]	159
6.2.3.473 sampleSmallBounded()	159
6.2.3.474 sampleUniform() [1/4]	159
6.2.3.475 sampleUniform() [2/4]	159
6.2.3.476 sampleUniform() [3/4]	159
6.2.3.477 sampleUniform() [4/4]	160
6.2.3.478 seekPastChar()	160
6.2.3.479 setAutomorphVals()	160
6.2.3.480 setAutomorphVals2()	160
6.2.3.481 setDryRun()	160
6.2.3.482 setHyperColumn() [1/6]	160
6.2.3.483 setHyperColumn() [2/6]	161
6.2.3.484 setHyperColumn() [3/6]	161
6.2.3.485 setHyperColumn() [4/6]	161
6.2.3.486 setHyperColumn() [5/6]	161
6.2.3.487 setHyperColumn() [6/6]	161
6.2.3.488 setLengthZero() [1/4]	162
6.2.3.489 setLengthZero() [2/4]	162
6.2.3.490 setLengthZero() [3/4]	162
6.2.3.491 setLengthZero() [4/4]	162
6.2.3.492 setTimersOff()	162
6.2.3.493 setTimersOn()	162
6.2.3.494 setupDebugGlobals()	163
6.2.3.495 shift()	163
6.2.3.496 splitBinaryNums()	163

6.2.3.497 sub()	164
6.2.3.498 subtractBinary()	164
6.2.3.499 sumOfCoeffs() [1/3]	165
6.2.3.500 sumOfCoeffs() [2/3]	165
6.2.3.501 sumOfCoeffs() [3/3]	165
6.2.3.502 swap() [1/2]	165
6.2.3.503 swap() [2/2]	165
6.2.3.504 tableLookup()	166
6.2.3.505 tableWriteIn()	166
6.2.3.506 timer_compare()	166
6.2.3.507 to_ZZX()	166
6.2.3.508 TofftRep_trunc() [1/2]	166
6.2.3.509 TofftRep_trunc() [2/2]	167
6.2.3.510 totalProduct()	167
6.2.3.511 totalSums()	167
6.2.3.512 traceMap()	167
6.2.3.513 TraceMap()	167
6.2.3.514 unpack() [1/2]	168
6.2.3.515 unpack() [2/2]	168
6.2.3.516 unpackSlots() [1/2]	168
6.2.3.517 unpackSlots() [2/2]	168
6.2.3.518 Vec_replicate()	168
6.2.3.519 vecCopy() [1/4]	169
6.2.3.520 vecCopy() [2/4]	169
6.2.3.521 vecCopy() [3/4]	169
6.2.3.522 vecCopy() [4/4]	169
6.2.3.523 vecRed() [1/2]	169
6.2.3.524 vecRed() [2/2]	170
6.2.3.525 vector_replicate()	170
6.2.3.526 vecToStr()	170
6.2.3.527 Warning() [1/2]	170
6.2.3.528 Warning() [2/2]	170
6.2.3.529 write()	170
6.2.3.530 write_ntl_vec_long()	171
6.2.3.531 write_raw_double()	171
6.2.3.532 write_raw_int()	171
6.2.3.533 write_raw_int32()	171
6.2.3.534 write_raw_vector()	171
6.2.3.535 write_raw_vector< double >()	171
6.2.3.536 write_raw_vector< long >()	172
6.2.3.537 write_raw_xdouble()	172
6.2.3.538 write_raw_ZZ()	172



6.2.3.539 writeContextBase()	172
6.2.3.540 writeContextBaseBinary()	172
6.2.3.541 writeContextBinary()	172
6.2.3.542 writeEyeCatcher()	173
6.2.3.543 writePubKeyBinary()	173
6.2.3.544 writeSecKeyBinary()	173
6.2.4 Variable Documentation	173
6.2.4.1 activeContext	173
6.2.4.2 CLOCK_SCALE	173
6.2.4.3 dbg_ptxt	173
6.2.4.4 dbgEa	174
6.2.4.5 dbgKey	174
6.2.4.6 erfc_inverse	174
6.2.4.7 fhe_force_chen_han	174
6.2.4.8 fhe_stats	174
6.2.4.9 fhe_test_force_bsgs	174
6.2.4.10 fhe_test_force_hoist	175
6.2.4.11 fhe_watcher	175
6.2.4.12 PI	175
6.2.4.13 printFlag	175
6.2.4.14 replicateVerboseFlag	175
6.2.4.15 thinRecrypt_initial_level	175
6.3 helib::FHEglobals Namespace Reference	175
6.3.1 Variable Documentation	176
6.3.1.1 automorphVals	176
6.3.1.2 automorphVals2	176
6.3.1.3 dryRun	176
6.4 NTL Namespace Reference	176
6.5 std Namespace Reference	176
<b>7 Class Documentation</b>	<b>177</b>
7.1 helib::add_pa_impl< type > Class Template Reference	177
7.1.1 Member Function Documentation	177
7.1.1.1 apply()	177
7.2 helib::AddDAG Class Reference	177
7.2.1 Detailed Description	178
7.2.2 Constructor & Destructor Documentation	178
7.2.2.1 AddDAG()	178
7.2.3 Member Function Documentation	178
7.2.3.1 apply()	178
7.2.3.2 findP()	179
7.2.3.3 findQ()	179

7.2.3.4 init()	179
7.2.3.5 lowLvl()	179
7.3 helib::PGFFT::aligned_allocator< T > Class Template Reference	179
7.3.1 Member Typedef Documentation	180
7.3.1.1 value_type	180
7.3.2 Constructor & Destructor Documentation	180
7.3.2.1 aligned_allocator() [1/2]	180
7.3.2.2 aligned_allocator() [2/2]	180
7.3.3 Member Function Documentation	180
7.3.3.1 allocate()	181
7.3.3.2 deallocate()	181
7.3.3.3 operator!=(())	181
7.3.3.4 operator==(())	181
7.4 helib::applyPerm_pa_impl< type > Class Template Reference	181
7.4.1 Member Function Documentation	181
7.4.1.1 apply()	182
7.5 helib::ArgMap Class Reference	182
7.5.1 Detailed Description	183
7.5.2 Member Enumeration Documentation	183
7.5.2.1 Separator	183
7.5.3 Member Function Documentation	184
7.5.3.1 arg() [1/3]	184
7.5.3.2 arg() [2/3]	185
7.5.3.3 arg() [3/3]	185
7.5.3.4 diagnostics()	187
7.5.3.5 doc()	187
7.5.3.6 dots()	187
7.5.3.7 helpArgs() [1/2]	188
7.5.3.8 helpArgs() [2/2]	188
7.5.3.9 named()	188
7.5.3.10 note()	189
7.5.3.11 optional()	189
7.5.3.12 parse() [1/2]	189
7.5.3.13 parse() [2/2]	190
7.5.3.14 positional()	190
7.5.3.15 required()	190
7.5.3.16 separator()	190
7.5.3.17 toggle()	191
7.5.3.18 usage()	191
7.6 helib::ArgMap::ArgProcessor Struct Reference	192
7.6.1 Constructor & Destructor Documentation	192
7.6.1.1 ~ArgProcessor()	192

7.6.2 Member Function Documentation . . . . .	192
7.6.2.1 getArgType() . . . . .	192
7.6.2.2 process() . . . . .	193
7.7 helib::BasicAutomorphPrecon Class Reference . . . . .	193
7.7.1 Detailed Description . . . . .	193
7.7.2 Constructor & Destructor Documentation . . . . .	193
7.7.2.1 BasicAutomorphPrecon() . . . . .	193
7.7.3 Member Function Documentation . . . . .	193
7.7.3.1 automorph() . . . . .	194
7.8 helib::BGV Struct Reference . . . . .	194
7.8.1 Detailed Description . . . . .	194
7.8.2 Member Typedef Documentation . . . . .	194
7.8.2.1 SlotType . . . . .	194
7.9 helib::BipartiteGraph Class Reference . . . . .	194
7.9.1 Detailed Description . . . . .	195
7.9.2 Member Function Documentation . . . . .	195
7.9.2.1 addEdge() . . . . .	195
7.9.2.2 partitionToMatchings() . . . . .	195
7.9.2.3 printout() . . . . .	195
7.9.3 Member Data Documentation . . . . .	195
7.9.3.1 left . . . . .	196
7.10 helib::BlockMatMul1D Class Reference . . . . .	196
7.10.1 Member Typedef Documentation . . . . .	196
7.10.1.1 ExecType . . . . .	196
7.10.2 Constructor & Destructor Documentation . . . . .	196
7.10.2.1 ~BlockMatMul1D() . . . . .	197
7.10.3 Member Function Documentation . . . . .	197
7.10.3.1 getDim() . . . . .	197
7.10.3.2 getEA() . . . . .	197
7.11 helib::BlockMatMul1D_derived< type > Class Template Reference . . . . .	197
7.11.1 Member Function Documentation . . . . .	198
7.11.1.1 get() . . . . .	198
7.11.1.2 multipleTransforms() . . . . .	198
7.11.1.3 processDiagonal() . . . . .	198
7.12 helib::BlockMatMul1D_derived_impl< type > Struct Template Reference . . . . .	198
7.12.1 Member Function Documentation . . . . .	199
7.12.1.1 processDiagonal() . . . . .	199
7.12.1.2 processDiagonal1() . . . . .	199
7.12.1.3 processDiagonal2() . . . . .	199
7.13 helib::BlockMatMul1D_partial< type > Class Template Reference . . . . .	199
7.13.1 Member Function Documentation . . . . .	200
7.13.1.1 processDiagonal() . . . . .	200

7.14 helib::BlockMatMul1DExec Class Reference	200
7.14.1 Constructor & Destructor Documentation	201
7.14.1.1 BlockMatMul1DExec()	201
7.14.2 Member Function Documentation	201
7.14.2.1 getEA()	201
7.14.2.2 mul()	201
7.14.2.3 upgrade()	201
7.14.3 Member Data Documentation	202
7.14.3.1 cache	202
7.14.3.2 cache1	202
7.14.3.3 D	202
7.14.3.4 d	202
7.14.3.5 dim	202
7.14.3.6 ea	202
7.14.3.7 native	202
7.14.3.8 strategy	203
7.15 helib::BlockMatMul1DExec_construct< type > Struct Template Reference	203
7.15.1 Member Function Documentation	203
7.15.1.1 apply()	203
7.16 helib::BlockMatMulFullExec_construct< type >::BlockMatMulDimComp Struct Reference	203
7.16.1 Constructor & Destructor Documentation	204
7.16.1.1 BlockMatMulDimComp()	204
7.16.2 Member Function Documentation	204
7.16.2.1 operator>()	204
7.16.3 Member Data Documentation	204
7.16.3.1 ea	204
7.17 helib::BlockMatMulFull Class Reference	205
7.17.1 Member Typedef Documentation	205
7.17.1.1 ExecType	205
7.17.2 Constructor & Destructor Documentation	205
7.17.2.1 ~BlockMatMulFull()	205
7.17.3 Member Function Documentation	205
7.17.3.1 getEA()	206
7.18 helib::BlockMatMulFull_derived< type > Class Template Reference	206
7.18.1 Member Function Documentation	206
7.18.1.1 get()	206
7.19 helib::BlockMatMulFullExec Class Reference	207
7.19.1 Constructor & Destructor Documentation	207
7.19.1.1 BlockMatMulFullExec()	207
7.19.2 Member Function Documentation	207
7.19.2.1 getEA()	207
7.19.2.2 mul()	208

7.19.2.3 rec_mul()	208
7.19.2.4 upgrade()	208
7.19.3 Member Data Documentation	208
7.19.3.1 dims	208
7.19.3.2 ea	208
7.19.3.3 minimal	208
7.19.3.4 transforms	209
7.20 helib::BlockMatMulFullExec_construct< type > Struct Template Reference	209
7.20.1 Member Function Documentation	209
7.20.1.1 apply()	209
7.20.1.2 rec_mul()	210
7.21 helib::BlockMatMulFullHelper< type > Class Template Reference	210
7.21.1 Constructor & Destructor Documentation	210
7.21.1.1 BlockMatMulFullHelper()	211
7.21.2 Member Function Documentation	211
7.21.2.1 getDim()	211
7.21.2.2 getEA()	211
7.21.2.3 processDiagonal()	211
7.21.3 Member Data Documentation	211
7.21.3.1 dim	212
7.21.3.2 ea_basetype	212
7.21.3.3 init_idxes	212
7.21.3.4 mat	212
7.22 helib::CKKS Struct Reference	212
7.22.1 Detailed Description	212
7.22.2 Member Typedef Documentation	213
7.22.2.1 SlotType	213
7.23 helib::Cmodulus Class Reference	213
7.23.1 Detailed Description	214
7.23.2 Constructor & Destructor Documentation	214
7.23.2.1 Cmodulus() [1/3]	214
7.23.2.2 Cmodulus() [2/3]	214
7.23.2.3 Cmodulus() [3/3]	214
7.23.3 Member Function Documentation	214
7.23.3.1 FFT() [1/2]	215
7.23.3.2 FFT() [2/2]	215
7.23.3.3 FFT_aux()	215
7.23.3.4 getM()	215
7.23.3.5 getPhiM()	215
7.23.3.6 getPhimX()	215
7.23.3.7 getQ()	215
7.23.3.8 getQInv()	216

7.23.3.9	<a href="#">getRoot()</a>	216
7.23.3.10	<a href="#">getScratch_fftRep()</a>	216
7.23.3.11	<a href="#">getScratch_vec_long()</a>	216
7.23.3.12	<a href="#">getScratch_zz_pX()</a>	216
7.23.3.13	<a href="#">getZMStar()</a>	216
7.23.3.14	<a href="#">iFFT()</a>	216
7.23.3.15	<a href="#">operator=()</a>	217
7.23.3.16	<a href="#">restoreModulus()</a>	217
7.24	<a href="#">helib::ColPerm Class Reference</a>	217
7.24.1	<a href="#">Detailed Description</a>	218
7.24.2	<a href="#">Constructor &amp; Destructor Documentation</a>	218
7.24.2.1	<a href="#">ColPerm()</a>	218
7.24.3	<a href="#">Member Function Documentation</a>	218
7.24.3.1	<a href="#">getBenesShiftAmounts()</a>	218
7.24.3.2	<a href="#">getPermDim()</a>	219
7.24.3.3	<a href="#">getShiftAmounts()</a>	219
7.24.3.4	<a href="#">makeExplicit()</a>	219
7.24.3.5	<a href="#">printout()</a>	219
7.24.3.6	<a href="#">setPermDim()</a>	219
7.25	<a href="#">helib::ConstCubeSlice&lt; T &gt; Class Template Reference</a>	219
7.25.1	<a href="#">Detailed Description</a>	220
7.25.2	<a href="#">Constructor &amp; Destructor Documentation</a>	220
7.25.2.1	<a href="#">ConstCubeSlice() [1/4]</a>	221
7.25.2.2	<a href="#">ConstCubeSlice() [2/4]</a>	221
7.25.2.3	<a href="#">ConstCubeSlice() [3/4]</a>	221
7.25.2.4	<a href="#">ConstCubeSlice() [4/4]</a>	221
7.25.3	<a href="#">Member Function Documentation</a>	221
7.25.3.1	<a href="#">addCoord()</a>	221
7.25.3.2	<a href="#">at()</a>	222
7.25.3.3	<a href="#">getCoord()</a>	222
7.25.3.4	<a href="#">getDim()</a>	222
7.25.3.5	<a href="#">getNumDims()</a>	222
7.25.3.6	<a href="#">getProd() [1/2]</a>	222
7.25.3.7	<a href="#">getProd() [2/2]</a>	223
7.25.3.8	<a href="#">getSig()</a>	223
7.25.3.9	<a href="#">getSize()</a>	223
7.25.3.10	<a href="#">numCols()</a>	223
7.25.3.11	<a href="#">numSlices()</a>	223
7.25.3.12	<a href="#">operator[]()</a>	224
7.25.3.13	<a href="#">sliceSize()</a>	224
7.26	<a href="#">helib::ConstMultiplier Struct Reference</a>	224
7.26.1	<a href="#">Constructor &amp; Destructor Documentation</a>	224

7.26.1.1 <code>~ConstMultiplier()</code>	224
7.26.2 Member Function Documentation	225
7.26.2.1 <code>mul()</code>	225
7.26.2.2 <code>upgrade()</code>	225
7.27 <code>helib::ConstMultiplier_DoubleCRT</code> Struct Reference	225
7.27.1 Constructor & Destructor Documentation	226
7.27.1.1 <code>ConstMultiplier_DoubleCRT()</code>	226
7.27.2 Member Function Documentation	226
7.27.2.1 <code>mul()</code>	226
7.27.2.2 <code>upgrade()</code>	226
7.27.3 Member Data Documentation	226
7.27.3.1 <code>data</code>	226
7.27.3.2 <code>sz</code>	226
7.28 <code>helib::ConstMultiplier_zzX</code> Struct Reference	227
7.28.1 Constructor & Destructor Documentation	227
7.28.1.1 <code>ConstMultiplier_zzX()</code>	227
7.28.2 Member Function Documentation	227
7.28.2.1 <code>mul()</code>	227
7.28.2.2 <code>upgrade()</code>	228
7.28.3 Member Data Documentation	228
7.28.3.1 <code>data</code>	228
7.29 <code>helib::ConstMultiplierCache</code> Struct Reference	228
7.29.1 Member Function Documentation	228
7.29.1.1 <code>upgrade()</code>	228
7.29.2 Member Data Documentation	228
7.29.2.1 <code>multiplier</code>	229
7.30 <code>helib::Context</code> Class Reference	229
7.30.1 Detailed Description	231
7.30.2 Constructor & Destructor Documentation	231
7.30.2.1 <code>Context()</code>	231
7.30.3 Member Function Documentation	231
7.30.3.1 <code>AddCtxtPrime()</code>	231
7.30.3.2 <code>AddSmallPrime()</code>	232
7.30.3.3 <code>AddSpecialPrime()</code>	232
7.30.3.4 <code>allPrimes()</code>	232
7.30.3.5 <code>boundForReryption()</code>	232
7.30.3.6 <code>BPL()</code>	232
7.30.3.7 <code>fullPrimes()</code>	232
7.30.3.8 <code>getCtxtPrimes()</code>	232
7.30.3.9 <code>inChain()</code>	233
7.30.3.10 <code>isBootstrappable()</code>	233
7.30.3.11 <code>isZeroDivisor()</code>	233

7.30.3.12	ithModulus()	233
7.30.3.13	ithPrime()	233
7.30.3.14	logOfPrime()	233
7.30.3.15	logOfProduct()	234
7.30.3.16	makeBootstrappable()	234
7.30.3.17	noiseBoundForGaussian()	234
7.30.3.18	noiseBoundForHWt()	234
7.30.3.19	noiseBoundForMod()	234
7.30.3.20	noiseBoundForSmall()	235
7.30.3.21	noiseBoundForUniform() [1/2]	235
7.30.3.22	noiseBoundForUniform() [2/2]	235
7.30.3.23	numPrimes()	235
7.30.3.24	operator"!=()	235
7.30.3.25	operator==(())	236
7.30.3.26	productOfPrimes() [1/2]	236
7.30.3.27	productOfPrimes() [2/2]	236
7.30.3.28	securityLevel()	236
7.30.3.29	setModSizeTable()	236
7.30.3.30	stdDevForRecryption()	236
7.30.4	Friends And Related Function Documentation	237
7.30.4.1	operator<<	237
7.30.4.2	operator>>	237
7.30.4.3	readContextBase	237
7.30.4.4	readContextBinary	237
7.30.4.5	writeContextBase	238
7.30.4.6	writeContextBinary	238
7.30.5	Member Data Documentation	238
7.30.5.1	alMod	238
7.30.5.2	ctxtPrimes	238
7.30.5.3	digits	238
7.30.5.4	ea	239
7.30.5.5	modSizes	239
7.30.5.6	pwfl_converter	239
7.30.5.7	rcData	239
7.30.5.8	scale	239
7.30.5.9	slotRing	239
7.30.5.10	smallPrimes	240
7.30.5.11	specialPrimes	240
7.30.5.12	stdev	240
7.30.5.13	zMStar	240
7.31	helib::Ctxt Class Reference	240
7.31.1	Detailed Description	243



7.31.2 Constructor & Destructor Documentation	244
7.31.2.1 Ctxt() [1/3]	244
7.31.2.2 Ctxt() [2/3]	244
7.31.2.3 Ctxt() [3/3]	244
7.31.3 Member Function Documentation	244
7.31.3.1 addConstant() [1/4]	244
7.31.3.2 addConstant() [2/4]	244
7.31.3.3 addConstant() [3/4]	245
7.31.3.4 addConstant() [4/4]	245
7.31.3.5 addConstantCKKS() [1/7]	245
7.31.3.6 addConstantCKKS() [2/7]	245
7.31.3.7 addConstantCKKS() [3/7]	245
7.31.3.8 addConstantCKKS() [4/7]	246
7.31.3.9 addConstantCKKS() [5/7]	246
7.31.3.10 addConstantCKKS() [6/7]	246
7.31.3.11 addConstantCKKS() [7/7]	246
7.31.3.12 addCtxt()	246
7.31.3.13 automorph()	247
7.31.3.14 bitCapacity()	247
7.31.3.15 blindCtxt()	247
7.31.3.16 bringToSet()	247
7.31.3.17 bumpNoiseBound()	247
7.31.3.18 capacity()	247
7.31.3.19 cleanUp()	248
7.31.3.20 clear()	248
7.31.3.21 complexConj()	248
7.31.3.22 cube()	248
7.31.3.23 divideBy2()	248
7.31.3.24 divideByP()	248
7.31.3.25 dropSmallAndSpecialPrimes()	248
7.31.3.26 DummyEncrypt()	249
7.31.3.27 effectiveR()	249
7.31.3.28 equalizeRationalFactors()	249
7.31.3.29 equalsTo()	249
7.31.3.30 evalPoly()	249
7.31.3.31 extractBits()	249
7.31.3.32 frobeniusAutomorph()	250
7.31.3.33 getContext()	250
7.31.3.34 getKeyID()	250
7.31.3.35 getNoiseBound()	250
7.31.3.36 getPrimeSet()	250
7.31.3.37 getPtxtMag()	250

7.31.3.38 getPtxtSpace()	250
7.31.3.39 getPubKey()	251
7.31.3.40 getRatFactor()	251
7.31.3.41 hackPtxtSpace()	251
7.31.3.42 inCanonicalForm()	251
7.31.3.43 isCKKS()	251
7.31.3.44 isCorrect()	251
7.31.3.45 isEmpty()	251
7.31.3.46 log_of_ratio()	252
7.31.3.47 logOfPrimeSet()	252
7.31.3.48 modDownToSet()	252
7.31.3.49 modSwitchAddedNoiseBound()	252
7.31.3.50 modUpToSet()	252
7.31.3.51 multByConstant() [1/5]	252
7.31.3.52 multByConstant() [2/5]	253
7.31.3.53 multByConstant() [3/5]	253
7.31.3.54 multByConstant() [4/5]	253
7.31.3.55 multByConstant() [5/5]	253
7.31.3.56 multByConstantCKKS() [1/6]	253
7.31.3.57 multByConstantCKKS() [2/6]	254
7.31.3.58 multByConstantCKKS() [3/6]	254
7.31.3.59 multByConstantCKKS() [4/6]	254
7.31.3.60 multByConstantCKKS() [5/6]	254
7.31.3.61 multByConstantCKKS() [6/6]	254
7.31.3.62 multByP()	255
7.31.3.63 multiplyBy()	255
7.31.3.64 multiplyBy2()	255
7.31.3.65 multLowLvl()	255
7.31.3.66 naturalPrimeSet()	255
7.31.3.67 naturalSize()	255
7.31.3.68 negate()	256
7.31.3.69 nxorConstant() [1/2]	256
7.31.3.70 nxorConstant() [2/2]	256
7.31.3.71 operator"!=()	256
7.31.3.72 operator*=( ) [1/5]	256
7.31.3.73 operator*=( ) [2/5]	256
7.31.3.74 operator*=( ) [3/5]	257
7.31.3.75 operator*=( ) [4/5]	257
7.31.3.76 operator*=( ) [5/5]	258
7.31.3.77 operator+=( ) [1/3]	258
7.31.3.78 operator+=( ) [2/3]	258
7.31.3.79 operator+=( ) [3/3]	259

7.31.3.80 operator==( ) [ 1 / 3 ] . . . . .	259
7.31.3.81 operator==( ) [ 2 / 3 ] . . . . .	259
7.31.3.82 operator==( ) [ 3 / 3 ] . . . . .	260
7.31.3.83 operator==( ) . . . . .	260
7.31.3.84 operator==( ) . . . . .	260
7.31.3.85 operator>>==( ) . . . . .	260
7.31.3.86 power( ) . . . . .	261
7.31.3.87 rawModSwitch( ) . . . . .	261
7.31.3.88 read( ) . . . . .	261
7.31.3.89 reducePtxtSpace( ) . . . . .	261
7.31.3.90 reLinearize( ) . . . . .	261
7.31.3.91 setPtxtMag( ) . . . . .	261
7.31.3.92 smartAutomorph( ) . . . . .	262
7.31.3.93 square( ) . . . . .	262
7.31.3.94 write( ) . . . . .	262
7.31.3.95 xorConstant( ) [ 1 / 2 ] . . . . .	262
7.31.3.96 xorConstant( ) [ 2 / 2 ] . . . . .	262
7.31.4 Friends And Related Function Documentation . . . . .	262
7.31.4.1 BasicAutomorphPrecon . . . . .	262
7.31.4.2 operator<< . . . . .	263
7.31.4.3 operator>> . . . . .	263
7.31.4.4 PubKey . . . . .	263
7.31.4.5 SecKey . . . . .	263
7.32 helib::CtxtPart Class Reference . . . . .	263
7.32.1 Detailed Description . . . . .	264
7.32.2 Constructor & Destructor Documentation . . . . .	264
7.32.2.1 CtxtPart( ) [ 1 / 4 ] . . . . .	264
7.32.2.2 CtxtPart( ) [ 2 / 4 ] . . . . .	264
7.32.2.3 CtxtPart( ) [ 3 / 4 ] . . . . .	264
7.32.2.4 CtxtPart( ) [ 4 / 4 ] . . . . .	265
7.32.3 Member Function Documentation . . . . .	265
7.32.3.1 operator!=( ) . . . . .	265
7.32.3.2 operator==( ) . . . . .	265
7.32.3.3 read( ) . . . . .	265
7.32.3.4 write( ) . . . . .	265
7.32.4 Member Data Documentation . . . . .	265
7.32.4.1 skHandle . . . . .	266
7.33 helib::CubeSignature Class Reference . . . . .	266
7.33.1 Detailed Description . . . . .	267
7.33.2 Constructor & Destructor Documentation . . . . .	267
7.33.2.1 CubeSignature( ) [ 1 / 4 ] . . . . .	267
7.33.2.2 CubeSignature( ) [ 2 / 4 ] . . . . .	267

7.33.2.3 CubeSignature() [3/4]	267
7.33.2.4 CubeSignature() [4/4]	267
7.33.3 Member Function Documentation	267
7.33.3.1 addCoord()	268
7.33.3.2 assembleCoords()	268
7.33.3.3 assembleIndexByDim()	268
7.33.3.4 breakIndexByDim()	268
7.33.3.5 getAllCoords()	268
7.33.3.6 getCoord()	269
7.33.3.7 getDim()	269
7.33.3.8 getNumDims()	269
7.33.3.9 getProd() [1/2]	269
7.33.3.10 getProd() [2/2]	269
7.33.3.11 getSize()	270
7.33.3.12 incrementCoords()	270
7.33.3.13 initSignature() [1/2]	270
7.33.3.14 initSignature() [2/2]	270
7.33.3.15 numCols()	270
7.33.3.16 numSlices()	270
7.33.3.17 sliceSize()	271
7.33.4 Friends And Related Function Documentation	271
7.33.4.1 operator<<	271
7.34 helib::CubeSlice< T > Class Template Reference	271
7.34.1 Detailed Description	272
7.34.2 Constructor & Destructor Documentation	272
7.34.2.1 CubeSlice() [1/4]	272
7.34.2.2 CubeSlice() [2/4]	272
7.34.2.3 CubeSlice() [3/4]	272
7.34.2.4 CubeSlice() [4/4]	272
7.34.3 Member Function Documentation	273
7.34.3.1 at()	273
7.34.3.2 copy()	273
7.34.3.3 operator[]()	273
7.35 helib::DAGnode Class Reference	273
7.35.1 Detailed Description	274
7.35.2 Constructor & Destructor Documentation	274
7.35.2.1 DAGnode() [1/2]	274
7.35.2.2 DAGnode() [2/2]	274
7.35.3 Member Function Documentation	274
7.35.3.1 nodeName()	274
7.35.4 Member Data Documentation	275
7.35.4.1 childrenLeft	275

7.35.4.2 ct	275
7.35.4.3 ct_mtx	275
7.35.4.4 idx	275
7.35.4.5 isQ	275
7.35.4.6 level	275
7.35.4.7 parent1	275
7.35.4.8 parent2	276
7.36 helib::decode_pa_impl< type > Class Template Reference	276
7.36.1 Member Function Documentation	276
7.36.1.1 apply()	276
7.37 helib::deep_clone< X > Class Template Reference	276
7.37.1 Detailed Description	276
7.37.2 Member Function Documentation	277
7.37.2.1 apply()	277
7.38 helib::DoubleCRT Class Reference	277
7.38.1 Detailed Description	279
7.38.2 Constructor & Destructor Documentation	279
7.38.2.1 DoubleCRT() [1/4]	280
7.38.2.2 DoubleCRT() [2/4]	280
7.38.2.3 DoubleCRT() [3/4]	281
7.38.2.4 DoubleCRT() [4/4]	281
7.38.3 Member Function Documentation	281
7.38.3.1 Add()	281
7.38.3.2 addPrimes()	281
7.38.3.3 addPrimesAndScale()	281
7.38.3.4 automorph()	282
7.38.3.5 breakIntoDigits()	282
7.38.3.6 complexConj()	282
7.38.3.7 Exp()	282
7.38.3.8 FFT() [1/2]	282
7.38.3.9 FFT() [2/2]	282
7.38.3.10 getContext()	283
7.38.3.11 getIndexSet()	283
7.38.3.12 getMap()	283
7.38.3.13 getOneRow() [1/2]	283
7.38.3.14 getOneRow() [2/2]	283
7.38.3.15 Mul()	283
7.38.3.16 Negate() [1/2]	284
7.38.3.17 Negate() [2/2]	284
7.38.3.18 operator"!=()"	284
7.38.3.19 operator*=( ) [1/4]	284
7.38.3.20 operator*=( ) [2/4]	284

7.38.3.21 operator*=( ) [ 3 / 4 ] . . . . .	284
7.38.3.22 operator*=( ) [ 4 / 4 ] . . . . .	284
7.38.3.23 operator++( ) [ 1 / 2 ] . . . . .	285
7.38.3.24 operator++( ) [ 2 / 2 ] . . . . .	285
7.38.3.25 operator+=( ) [ 1 / 4 ] . . . . .	285
7.38.3.26 operator+=( ) [ 2 / 4 ] . . . . .	285
7.38.3.27 operator+=( ) [ 3 / 4 ] . . . . .	285
7.38.3.28 operator+=( ) [ 4 / 4 ] . . . . .	285
7.38.3.29 operator--( ) [ 1 / 2 ] . . . . .	285
7.38.3.30 operator--( ) [ 2 / 2 ] . . . . .	286
7.38.3.31 operator--=( ) [ 1 / 4 ] . . . . .	286
7.38.3.32 operator--=( ) [ 2 / 4 ] . . . . .	286
7.38.3.33 operator--=( ) [ 3 / 4 ] . . . . .	286
7.38.3.34 operator--=( ) [ 4 / 4 ] . . . . .	286
7.38.3.35 operator/=( ) [ 1 / 2 ] . . . . .	286
7.38.3.36 operator/=( ) [ 2 / 2 ] . . . . .	286
7.38.3.37 operator=( ) [ 1 / 5 ] . . . . .	287
7.38.3.38 operator=( ) [ 2 / 5 ] . . . . .	287
7.38.3.39 operator=( ) [ 3 / 5 ] . . . . .	287
7.38.3.40 operator=( ) [ 4 / 5 ] . . . . .	287
7.38.3.41 operator=( ) [ 5 / 5 ] . . . . .	287
7.38.3.42 operator==( ) . . . . .	287
7.38.3.43 operator>>=( ) . . . . .	287
7.38.3.44 randomize( ) . . . . .	288
7.38.3.45 read( ) . . . . .	288
7.38.3.46 reduce( ) . . . . .	288
7.38.3.47 removePrimes( ) . . . . .	288
7.38.3.48 sampleGaussian( ) . . . . .	288
7.38.3.49 sampleGaussianBounded( ) . . . . .	288
7.38.3.50 sampleHWt( ) . . . . .	289
7.38.3.51 sampleHWtBounded( ) . . . . .	289
7.38.3.52 sampleSmall( ) . . . . .	289
7.38.3.53 sampleSmallBounded( ) . . . . .	289
7.38.3.54 sampleUniform( ) [ 1 / 2 ] . . . . .	289
7.38.3.55 sampleUniform( ) [ 2 / 2 ] . . . . .	289
7.38.3.56 scaleDownToSet( ) . . . . .	290
7.38.3.57 SetOne( ) . . . . .	290
7.38.3.58 setPrimes( ) . . . . .	290
7.38.3.59 SetZero( ) . . . . .	290
7.38.3.60 Sub( ) . . . . .	290
7.38.3.61 toPoly( ) [ 1 / 2 ] . . . . .	290
7.38.3.62 toPoly( ) [ 2 / 2 ] . . . . .	291

7.38.3.63 write()	291
7.38.4 Friends And Related Function Documentation	291
7.38.4.1 operator<<	291
7.38.4.2 operator>>	291
7.39 helib::DoubleCRTHelper Class Reference	292
7.39.1 Detailed Description	292
7.39.2 Constructor & Destructor Documentation	292
7.39.2.1 DoubleCRTHelper()	292
7.39.3 Member Function Documentation	292
7.39.3.1 clone()	292
7.39.3.2 init()	293
7.40 helib::DynamicCtxtPowers Class Reference	293
7.40.1 Detailed Description	293
7.40.2 Constructor & Destructor Documentation	293
7.40.2.1 DynamicCtxtPowers()	293
7.40.3 Member Function Documentation	294
7.40.3.1 at()	294
7.40.3.2 getPower()	294
7.40.3.3 getVector()	294
7.40.3.4 isPowerComputed()	294
7.40.3.5 operator[]()	294
7.40.3.6 size()	294
7.41 helib::encode_pa_impl< type > Class Template Reference	295
7.41.1 Member Function Documentation	295
7.41.1.1 apply() [1/2]	295
7.41.1.2 apply() [2/2]	295
7.42 helib::EncryptedArray Class Reference	295
7.42.1 Detailed Description	296
7.42.2 Constructor & Destructor Documentation	297
7.42.2.1 EncryptedArray() [1/2]	297
7.42.2.2 EncryptedArray() [2/2]	297
7.42.3 Member Function Documentation	297
7.42.3.1 addCoord()	297
7.42.3.2 badDimensionAutomorphCorrection()	297
7.42.3.3 buildLinPolyCoeffs()	298
7.42.3.4 coordinate()	298
7.42.3.5 decode()	298
7.42.3.6 decrypt()	298
7.42.3.7 dimension()	298
7.42.3.8 dispatch()	298
7.42.3.9 encode()	299
7.42.3.10 encodeUnitSelector()	299

7.42.3.11 encrypt()	299
7.42.3.12 getAImod()	299
7.42.3.13 getContext()	299
7.42.3.14 getCx()	299
7.42.3.15 getDegree()	300
7.42.3.16 getDerived()	300
7.42.3.17 getPALgebra()	300
7.42.3.18 getTag()	300
7.42.3.19 nativeDimension()	300
7.42.3.20 operator=()	300
7.42.3.21 random()	300
7.42.3.22 restoreContext()	301
7.42.3.23 restoreContextForG()	301
7.42.3.24 rotate()	301
7.42.3.25 rotate1D() [1/2]	301
7.42.3.26 rotate1D() [2/2]	301
7.42.3.27 shift()	301
7.42.3.28 shift1D()	302
7.42.3.29 size()	302
7.42.3.30 sizeOfDimension()	302
7.43 helib::EncryptedArrayBase Class Reference	302
7.43.1 Detailed Description	304
7.43.2 Constructor & Destructor Documentation	304
7.43.2.1 ~EncryptedArrayBase()	304
7.43.3 Member Function Documentation	305
7.43.3.1 addCoord()	305
7.43.3.2 badDimensionAutomorphCorrection()	305
7.43.3.3 buildLinPolyCoeffs()	305
7.43.3.4 clone()	306
7.43.3.5 coordinate()	306
7.43.3.6 decode() [1/3]	306
7.43.3.7 decode() [2/3]	306
7.43.3.8 decode() [3/3]	307
7.43.3.9 decode1Slot() [1/2]	307
7.43.3.10 decode1Slot() [2/2]	307
7.43.3.11 decrypt() [1/5]	307
7.43.3.12 decrypt() [2/5]	307
7.43.3.13 decrypt() [3/5]	308
7.43.3.14 decrypt() [4/5]	308
7.43.3.15 decrypt() [5/5]	308
7.43.3.16 decrypt1Slot() [1/2]	308
7.43.3.17 decrypt1Slot() [2/2]	308



7.43.3.18 dimension()	309
7.43.3.19 encode() [1/7]	309
7.43.3.20 encode() [2/7]	309
7.43.3.21 encode() [3/7]	309
7.43.3.22 encode() [4/7]	309
7.43.3.23 encode() [5/7]	310
7.43.3.24 encode() [6/7]	310
7.43.3.25 encode() [7/7]	310
7.43.3.26 encodeUnitSelector()	310
7.43.3.27 encrypt()	310
7.43.3.28 getContext()	311
7.43.3.29 getDegree()	311
7.43.3.30 getP2R()	311
7.43.3.31 getPAlgebra()	311
7.43.3.32 getTag()	311
7.43.3.33 nativeDimension()	311
7.43.3.34 random() [1/2]	312
7.43.3.35 random() [2/2]	312
7.43.3.36 restoreContext()	312
7.43.3.37 restoreContextForG()	312
7.43.3.38 rotate()	312
7.43.3.39 rotate1D() [1/2]	312
7.43.3.40 rotate1D() [2/2]	313
7.43.3.41 shift()	314
7.43.3.42 shift1D()	314
7.43.3.43 size()	314
7.43.3.44 sizeOfDimension()	314
7.44 helib::EncryptedArrayCx Class Reference	315
7.44.1 Detailed Description	317
7.44.2 Constructor & Destructor Documentation	317
7.44.2.1 EncryptedArrayCx() [1/2]	317
7.44.2.2 EncryptedArrayCx() [2/2]	317
7.44.3 Member Function Documentation	318
7.44.3.1 badDimensionAutomorphCorrection()	318
7.44.3.2 buildLinPolyCoeffs() [1/3]	318
7.44.3.3 buildLinPolyCoeffs() [2/3]	319
7.44.3.4 buildLinPolyCoeffs() [3/3]	319
7.44.3.5 clone()	319
7.44.3.6 convert() [1/4]	319
7.44.3.7 convert() [2/4]	319
7.44.3.8 convert() [3/4]	320
7.44.3.9 convert() [4/4]	320

7.44.3.10 decode()	[1/7]	320
7.44.3.11 decode()	[2/7]	320
7.44.3.12 decode()	[3/7]	320
7.44.3.13 decode()	[4/7]	320
7.44.3.14 decode()	[5/7]	320
7.44.3.15 decode()	[6/7]	321
7.44.3.16 decode()	[7/7]	321
7.44.3.17 decrypt()	[1/6]	321
7.44.3.18 decrypt()	[2/6]	322
7.44.3.19 decrypt()	[3/6]	322
7.44.3.20 decrypt()	[4/6]	323
7.44.3.21 decrypt()	[5/6]	323
7.44.3.22 decrypt()	[6/6]	323
7.44.3.23 encode()	[1/12]	324
7.44.3.24 encode()	[2/12]	324
7.44.3.25 encode()	[3/12]	324
7.44.3.26 encode()	[4/12]	324
7.44.3.27 encode()	[5/12]	325
7.44.3.28 encode()	[6/12]	325
7.44.3.29 encode()	[7/12]	325
7.44.3.30 encode()	[8/12]	326
7.44.3.31 encode()	[9/12]	326
7.44.3.32 encode()	[10/12]	327
7.44.3.33 encode()	[11/12]	327
7.44.3.34 encode()	[12/12]	327
7.44.3.35 encodei()		327
7.44.3.36 encodeRoundingError()		327
7.44.3.37 encodeScalingFactor()		327
7.44.3.38 encodeUnitSelector()		328
7.44.3.39 encrypt()	[1/2]	328
7.44.3.40 encrypt()	[2/2]	328
7.44.3.41 encryptOneNum()		328
7.44.3.42 extractImPart()	[1/2]	328
7.44.3.43 extractImPart()	[2/2]	329
7.44.3.44 extractRealPart()	[1/2]	330
7.44.3.45 extractRealPart()	[2/2]	330
7.44.3.46 getContext()		330
7.44.3.47 getDegree()		331
7.44.3.48 getiEncoded()		331
7.44.3.49 getP2R()		331
7.44.3.50 getPALgebra()		331
7.44.3.51 getTag()		331

7.44.3.52 random() [1/4]	331
7.44.3.53 random() [2/4]	332
7.44.3.54 random() [3/4]	332
7.44.3.55 random() [4/4]	332
7.44.3.56 rotate()	332
7.44.3.57 rotate1D()	332
7.44.3.58 roundedSize()	333
7.44.3.59 shift()	333
7.44.3.60 shift1D()	334
7.45 helib::EncryptedArrayDerived< type > Class Template Reference	334
7.45.1 Detailed Description	335
7.45.2 Constructor & Destructor Documentation	336
7.45.2.1 EncryptedArrayDerived() [1/2]	336
7.45.2.2 EncryptedArrayDerived() [2/2]	336
7.45.3 Member Function Documentation	336
7.45.3.1 badDimensionAutomorphCorrection()	336
7.45.3.2 buildLinPolyCoeffs() [1/2]	337
7.45.3.3 buildLinPolyCoeffs() [2/2]	337
7.45.3.4 clone()	337
7.45.3.5 decode() [1/7]	337
7.45.3.6 decode() [2/7]	338
7.45.3.7 decode() [3/7]	338
7.45.3.8 decode() [4/7]	338
7.45.3.9 decode() [5/7]	338
7.45.3.10 decode() [6/7]	338
7.45.3.11 decode() [7/7]	339
7.45.3.12 decrypt() [1/6]	339
7.45.3.13 decrypt() [2/6]	339
7.45.3.14 decrypt() [3/6]	339
7.45.3.15 decrypt() [4/6]	339
7.45.3.16 decrypt() [5/6]	340
7.45.3.17 decrypt() [6/6]	340
7.45.3.18 dispatch()	340
7.45.3.19 encode() [1/9]	340
7.45.3.20 encode() [2/9]	341
7.45.3.21 encode() [3/9]	341
7.45.3.22 encode() [4/9]	341
7.45.3.23 encode() [5/9]	341
7.45.3.24 encode() [6/9]	341
7.45.3.25 encode() [7/9]	342
7.45.3.26 encode() [8/9]	342
7.45.3.27 encode() [9/9]	342

7.45.3.28	encodeUnitSelector()	342
7.45.3.29	getContext()	342
7.45.3.30	getDegree()	343
7.45.3.31	getG()	343
7.45.3.32	getNormalBasisMatrix()	343
7.45.3.33	getNormalBasisMatrixInverse()	343
7.45.3.34	getP2R()	343
7.45.3.35	getPAlgebra()	343
7.45.3.36	getTab()	344
7.45.3.37	getTag()	344
7.45.3.38	initNormalBasisMatrix()	344
7.45.3.39	operator=()	344
7.45.3.40	random() [1/3]	344
7.45.3.41	random() [2/3]	344
7.45.3.42	random() [3/3]	345
7.45.3.43	restoreContext()	345
7.45.3.44	restoreContextForG()	345
7.45.3.45	rotate()	345
7.45.3.46	rotate1D() [1/2]	345
7.45.3.47	rotate1D() [2/2]	346
7.45.3.48	shift()	347
7.45.3.49	shift1D()	347
7.46	helib::equals_pa_impl< type > Class Template Reference	347
7.46.1	Member Function Documentation	347
7.46.1.1	apply() [1/3]	348
7.46.1.2	apply() [2/3]	348
7.46.1.3	apply() [3/3]	348
7.47	helib::EvalMap Class Reference	348
7.47.1	Detailed Description	349
7.47.2	Constructor & Destructor Documentation	349
7.47.2.1	EvalMap()	349
7.47.3	Member Function Documentation	349
7.47.3.1	apply()	349
7.47.3.2	upgrade()	350
7.48	helib::Exception Class Reference	350
7.48.1	Detailed Description	350
7.48.2	Constructor & Destructor Documentation	350
7.48.2.1	~Exception()	350
7.48.2.2	Exception()	351
7.48.3	Member Function Documentation	351
7.48.3.1	what()	351
7.49	helib::ExplicitReplicator Class Reference	351

7.49.1 Detailed Description . . . . .	351
7.49.2 Constructor & Destructor Documentation . . . . .	351
7.49.2.1 ExplicitReplicator() . . . . .	352
7.49.3 Member Function Documentation . . . . .	352
7.49.3.1 handle() . . . . .	352
7.50 helib::fhe_stats_record Struct Reference . . . . .	352
7.50.1 Constructor & Destructor Documentation . . . . .	352
7.50.1.1 fhe_stats_record() . . . . .	353
7.50.2 Member Function Documentation . . . . .	353
7.50.2.1 save() . . . . .	353
7.50.2.2 update() . . . . .	353
7.50.3 Member Data Documentation . . . . .	353
7.50.3.1 count . . . . .	353
7.50.3.2 map . . . . .	353
7.50.3.3 max . . . . .	353
7.50.3.4 name . . . . .	354
7.50.3.5 saved_values . . . . .	354
7.50.3.6 sum . . . . .	354
7.51 helib::FHEtimer Class Reference . . . . .	354
7.51.1 Detailed Description . . . . .	354
7.51.2 Constructor & Destructor Documentation . . . . .	355
7.51.2.1 FHEtimer() . . . . .	355
7.51.3 Member Function Documentation . . . . .	355
7.51.3.1 getNumCalls() . . . . .	355
7.51.3.2 getTime() . . . . .	355
7.51.3.3 reset() . . . . .	355
7.51.4 Member Data Documentation . . . . .	355
7.51.4.1 counter . . . . .	355
7.51.4.2 loc . . . . .	355
7.51.4.3 name . . . . .	356
7.51.4.4 numCalls . . . . .	356
7.52 helib::FlowEdge Class Reference . . . . .	356
7.52.1 Detailed Description . . . . .	356
7.52.2 Constructor & Destructor Documentation . . . . .	356
7.52.2.1 FlowEdge() . . . . .	356
7.52.3 Member Data Documentation . . . . .	357
7.52.3.1 capacity . . . . .	357
7.52.3.2 flow . . . . .	357
7.53 helib::frobeniusAutomorph_pa_impl< type > Class Template Reference . . . . .	357
7.53.1 Member Function Documentation . . . . .	357
7.53.1.1 apply() [1/2] . . . . .	357
7.53.1.2 apply() [2/2] . . . . .	358

7.54 helib::FullBinaryTree Class Reference . . . . .	358
7.54.1 Detailed Description . . . . .	358
7.54.2 Constructor & Destructor Documentation . . . . .	359
7.54.2.1 FullBinaryTree() [1/2] . . . . .	359
7.54.2.2 FullBinaryTree() [2/2] . . . . .	359
7.54.3 Member Function Documentation . . . . .	359
7.54.3.1 addChildren() . . . . .	359
7.54.3.2 at() [1/2] . . . . .	359
7.54.3.3 at() [2/2] . . . . .	359
7.54.3.4 collapseToRoot() . . . . .	360
7.54.3.5 DataOfNode() [1/2] . . . . .	360
7.54.3.6 DataOfNode() [2/2] . . . . .	360
7.54.3.7 firstLeaf() . . . . .	360
7.54.3.8 getAuxKey() . . . . .	360
7.54.3.9 getNleaves() . . . . .	360
7.54.3.10 lastLeaf() . . . . .	360
7.54.3.11 leftChildIdx() . . . . .	361
7.54.3.12 nextLeaf() . . . . .	361
7.54.3.13 operator[]() [1/2] . . . . .	361
7.54.3.14 operator[]() [2/2] . . . . .	361
7.54.3.15 parentIdx() . . . . .	361
7.54.3.16 prevLeaf() . . . . .	361
7.54.3.17 printout() . . . . .	361
7.54.3.18 putDataInRoot() . . . . .	362
7.54.3.19 rightChildIdx() . . . . .	362
7.54.3.20 rootIdx() . . . . .	362
7.54.3.21 setAuxKey() . . . . .	362
7.54.3.22 size() . . . . .	362
7.55 helib::GenDescriptor Class Reference . . . . .	362
7.55.1 Detailed Description . . . . .	363
7.55.2 Constructor & Destructor Documentation . . . . .	363
7.55.2.1 GenDescriptor() [1/2] . . . . .	363
7.55.2.2 GenDescriptor() [2/2] . . . . .	363
7.55.3 Member Data Documentation . . . . .	363
7.55.3.1 genIdx . . . . .	363
7.55.3.2 good . . . . .	363
7.55.3.3 order . . . . .	364
7.56 helib::general_range< T > Class Template Reference . . . . .	364
7.56.1 Constructor & Destructor Documentation . . . . .	364
7.56.1.1 general_range() . . . . .	364
7.56.2 Member Function Documentation . . . . .	364
7.56.2.1 begin() . . . . .	364

7.56.2.2 end()	365
7.57 helib::GeneralAutomorphPrecon Class Reference	365
7.57.1 Constructor & Destructor Documentation	365
7.57.1.1 ~GeneralAutomorphPrecon()	365
7.57.2 Member Function Documentation	365
7.57.2.1 automorph()	365
7.58 helib::GeneralAutomorphPrecon_BSGS Class Reference	366
7.58.1 Constructor & Destructor Documentation	366
7.58.1.1 GeneralAutomorphPrecon_BSGS()	366
7.58.2 Member Function Documentation	366
7.58.2.1 automorph()	366
7.59 helib::GeneralAutomorphPrecon_FULL Class Reference	367
7.59.1 Constructor & Destructor Documentation	367
7.59.1.1 GeneralAutomorphPrecon_FULL()	367
7.59.2 Member Function Documentation	367
7.59.2.1 automorph()	367
7.60 helib::GeneralAutomorphPrecon_UNKNOWN Class Reference	368
7.60.1 Constructor & Destructor Documentation	368
7.60.1.1 GeneralAutomorphPrecon_UNKNOWN()	368
7.60.2 Member Function Documentation	368
7.60.2.1 automorph()	368
7.61 helib::GeneralBenesNetwork Class Reference	368
7.61.1 Detailed Description	369
7.61.2 Constructor & Destructor Documentation	369
7.61.2.1 GeneralBenesNetwork()	369
7.61.3 Member Function Documentation	369
7.61.3.1 depth()	369
7.61.3.2 getDepth()	369
7.61.3.3 getLevel()	370
7.61.3.4 getNumLevels()	370
7.61.3.5 getSize()	370
7.61.3.6 levelToDepthMap() [1/2]	370
7.61.3.7 levelToDepthMap() [2/2]	370
7.61.3.8 shamt() [1/2]	370
7.61.3.9 shamt() [2/2]	370
7.61.3.10 testNetwork()	371
7.62 helib::GeneratorTrees Class Reference	371
7.62.1 Detailed Description	371
7.62.2 Constructor & Destructor Documentation	372
7.62.2.1 GeneratorTrees()	372
7.62.3 Member Function Documentation	372
7.62.3.1 at() [1/2]	372

7.62.3.2 at() [2/2]	372
7.62.3.3 buildOptimalTrees()	372
7.62.3.4 ComputeCubeMapping()	372
7.62.3.5 getCubeDims()	373
7.62.3.6 getCubeSubDims()	373
7.62.3.7 getGenTree() [1/2]	373
7.62.3.8 getGenTree() [2/2]	373
7.62.3.9 getSize()	373
7.62.3.10 mapToArray() [1/3]	373
7.62.3.11 mapToArray() [2/3]	373
7.62.3.12 mapToArray() [3/3]	374
7.62.3.13 mapToCube() [1/3]	374
7.62.3.14 mapToCube() [2/3]	374
7.62.3.15 mapToCube() [3/3]	374
7.62.3.16 numLayers()	374
7.62.3.17 numTrees()	374
7.62.3.18 operator[]() [1/2]	374
7.62.3.19 operator[]() [2/2]	375
7.62.4 Friends And Related Function Documentation	375
7.62.4.1 operator<<	375
7.63 helib::half_FFT Struct Reference	375
7.63.1 Constructor & Destructor Documentation	375
7.63.1.1 half_FFT()	375
7.63.2 Member Data Documentation	376
7.63.2.1 fft	376
7.63.2.2 pow	376
7.64 HighLvlTimingData Class Reference	376
7.64.1 Constructor & Destructor Documentation	376
7.64.1.1 HighLvlTimingData()	376
7.64.2 Member Data Documentation	377
7.64.2.1 lvl	377
7.64.2.2 matmul	377
7.64.2.3 permute	377
7.64.2.4 replAll	377
7.64.2.5 replicate	377
7.64.2.6 rotate	377
7.64.2.7 shift	377
7.65 helib::HyperCube< T > Class Template Reference	378
7.65.1 Detailed Description	379
7.65.2 Constructor & Destructor Documentation	379
7.65.2.1 HyperCube() [1/2]	379
7.65.2.2 HyperCube() [2/2]	379



7.65.3 Member Function Documentation	379
7.65.3.1 addCoord()	379
7.65.3.2 at() [1/2]	380
7.65.3.3 at() [2/2]	380
7.65.3.4 getCoord()	380
7.65.3.5 getData() [1/2]	380
7.65.3.6 getData() [2/2]	380
7.65.3.7 getDim()	381
7.65.3.8 getNumDims()	381
7.65.3.9 getProd() [1/2]	381
7.65.3.10 getProd() [2/2]	381
7.65.3.11 getSig()	381
7.65.3.12 getSize()	382
7.65.3.13 numCols()	382
7.65.3.14 numSlices()	382
7.65.3.15 operator"!=()"	382
7.65.3.16 operator=()	382
7.65.3.17 operator==(())	383
7.65.3.18 operator[]() [1/2]	383
7.65.3.19 operator[]() [2/2]	383
7.65.3.20 rotate1D()	383
7.65.3.21 shift1D()	383
7.65.3.22 sliceSize()	384
7.66 helib::IndexMap< T > Class Template Reference	384
7.66.1 Detailed Description	384
7.66.2 Constructor & Destructor Documentation	385
7.66.2.1 IndexMap() [1/2]	385
7.66.2.2 IndexMap() [2/2]	385
7.66.3 Member Function Documentation	385
7.66.3.1 clear()	385
7.66.3.2 getIndexSet()	385
7.66.3.3 insert() [1/2]	386
7.66.3.4 insert() [2/2]	386
7.66.3.5 operator[]() [1/2]	386
7.66.3.6 operator[]() [2/2]	386
7.66.3.7 remove() [1/2]	386
7.66.3.8 remove() [2/2]	387
7.67 helib::IndexMapInit< T > Class Template Reference	387
7.67.1 Detailed Description	387
7.67.2 Constructor & Destructor Documentation	387
7.67.2.1 ~IndexMapInit()	387
7.67.3 Member Function Documentation	387

7.67.3.1 clone()	388
7.67.3.2 init()	388
7.68 helib::IndexSet Class Reference	388
7.68.1 Detailed Description	389
7.68.2 Constructor & Destructor Documentation	389
7.68.2.1 IndexSet() [1/3]	389
7.68.2.2 IndexSet() [2/3]	390
7.68.2.3 IndexSet() [3/3]	390
7.68.3 Member Function Documentation	390
7.68.3.1 begin()	390
7.68.3.2 card()	390
7.68.3.3 clear()	390
7.68.3.4 contains() [1/2]	390
7.68.3.5 contains() [2/2]	391
7.68.3.6 disjointFrom()	391
7.68.3.7 emptySet()	391
7.68.3.8 end()	391
7.68.3.9 first()	391
7.68.3.10 insert() [1/2]	391
7.68.3.11 insert() [2/2]	392
7.68.3.12 isInterval()	392
7.68.3.13 last()	392
7.68.3.14 next()	392
7.68.3.15 operator!=(())	392
7.68.3.16 operator==(())	392
7.68.3.17 prev()	393
7.68.3.18 read()	393
7.68.3.19 remove() [1/2]	393
7.68.3.20 remove() [2/2]	393
7.68.3.21 retain()	393
7.68.3.22 write()	393
7.69 helib::InvalidArgument Class Reference	394
7.69.1 Detailed Description	394
7.69.2 Constructor & Destructor Documentation	394
7.69.2.1 InvalidArgument() [1/2]	394
7.69.2.2 InvalidArgument() [2/2]	394
7.69.2.3 ~InvalidArgument()	395
7.69.3 Member Function Documentation	395
7.69.3.1 what()	395
7.70 helib::IndexSet::iterator Class Reference	395
7.70.1 Constructor & Destructor Documentation	395
7.70.1.1 iterator()	396

7.70.2 Member Function Documentation	396
7.70.2.1 operator"!=()	396
7.70.2.2 operator*()	396
7.70.2.3 operator++()	396
7.70.2.4 operator==()	396
7.70.3 Friends And Related Function Documentation	396
7.70.3.1 IndexSet	396
7.71 helib::general_range< T >::iterator Class Reference	397
7.71.1 Constructor & Destructor Documentation	397
7.71.1.1 iterator()	397
7.71.2 Member Function Documentation	397
7.71.2.1 operator"!=()	397
7.71.2.2 operator*()	398
7.71.2.3 operator++()	398
7.71.2.4 operator==()	398
7.71.3 Friends And Related Function Documentation	398
7.71.3.1 general_range	398
7.72 helib::KeySwitch Class Reference	398
7.72.1 Detailed Description	399
7.72.2 Constructor & Destructor Documentation	400
7.72.2.1 KeySwitch() [1/2]	400
7.72.2.2 KeySwitch() [2/2]	400
7.72.3 Member Function Documentation	400
7.72.3.1 dummy()	400
7.72.3.2 isDummy()	401
7.72.3.3 NumCols()	401
7.72.3.4 operator"!=()	401
7.72.3.5 operator==()	401
7.72.3.6 read()	401
7.72.3.7 readMatrix()	401
7.72.3.8 verify()	402
7.72.3.9 write()	402
7.72.4 Member Data Documentation	402
7.72.4.1 b	402
7.72.4.2 fromKey	402
7.72.4.3 noiseBound	402
7.72.4.4 prgSeed	402
7.72.4.5 ptxtSpace	402
7.72.4.6 toKeyID	403
7.73 helib::LabeledEdge Class Reference	403
7.73.1 Detailed Description	403
7.73.2 Constructor & Destructor Documentation	403

7.73.2.1 LabeledEdge()	403
7.73.3 Member Data Documentation	403
7.73.3.1 color	404
7.73.3.2 from	404
7.73.3.3 label	404
7.73.3.4 to	404
7.74 helib::LabeledVertex Class Reference	404
7.74.1 Detailed Description	405
7.74.2 Constructor & Destructor Documentation	405
7.74.2.1 LabeledVertex()	405
7.74.3 Member Function Documentation	405
7.74.3.1 addEdge()	405
7.74.3.2 addNeighbor()	405
7.74.4 Member Data Documentation	405
7.74.4.1 label	405
7.74.4.2 name	406
7.74.4.3 neighbors	406
7.75 helib::LogicError Class Reference	406
7.75.1 Detailed Description	406
7.75.2 Constructor & Destructor Documentation	406
7.75.2.1 LogicError() [1/2]	407
7.75.2.2 LogicError() [2/2]	407
7.75.2.3 ~LogicError()	407
7.75.3 Member Function Documentation	407
7.75.3.1 what()	407
7.76 LowLvlTimingData Class Reference	407
7.76.1 Constructor & Destructor Documentation	408
7.76.1.1 LowLvlTimingData()	408
7.76.2 Member Data Documentation	408
7.76.2.1 add	408
7.76.2.2 addConst	408
7.76.2.3 autoNative	408
7.76.2.4 autoTypical	408
7.76.2.5 innerProd	408
7.76.2.6 lvl	409
7.76.2.7 mult	409
7.76.2.8 multBy2	409
7.76.2.9 multConst	409
7.77 helib::MappingData< type > Class Template Reference	409
7.77.1 Detailed Description	410
7.77.2 Member Function Documentation	410
7.77.2.1 getDegG()	410

7.77.2.2 getG()	410
7.77.2.3 restoreContextForG()	410
7.77.3 Friends And Related Function Documentation	410
7.77.3.1 PAlgebraModDerived< type >	410
7.78 helib::MatMul1D Class Reference	411
7.78.1 Member Typedef Documentation	411
7.78.1.1 ExecType	411
7.78.2 Constructor & Destructor Documentation	411
7.78.2.1 ~MatMul1D()	411
7.78.3 Member Function Documentation	411
7.78.3.1 getDim()	412
7.78.3.2 getEA()	412
7.79 helib::MatMul1D_derived< type > Class Template Reference	412
7.79.1 Member Function Documentation	412
7.79.1.1 get()	413
7.79.1.2 multipleTransforms()	413
7.79.1.3 processDiagonal()	413
7.80 helib::MatMul1D_derived_impl< type > Struct Template Reference	413
7.80.1 Member Function Documentation	414
7.80.1.1 processDiagonal()	414
7.80.1.2 processDiagonal1()	414
7.80.1.3 processDiagonal2()	414
7.81 helib::MatMul1D_partial< type > Class Template Reference	414
7.81.1 Member Function Documentation	415
7.81.1.1 processDiagonal()	415
7.82 helib::MatMul1DExec Class Reference	415
7.82.1 Constructor & Destructor Documentation	416
7.82.1.1 MatMul1DExec()	416
7.82.2 Member Function Documentation	416
7.82.2.1 getEA()	416
7.82.2.2 mul()	416
7.82.2.3 upgrade()	416
7.82.3 Member Data Documentation	417
7.82.3.1 cache	417
7.82.3.2 cache1	417
7.82.3.3 D	417
7.82.3.4 dim	417
7.82.3.5 ea	417
7.82.3.6 g	417
7.82.3.7 minimal	417
7.82.3.8 native	418
7.83 helib::MatMul1DExec_construct< type > Struct Template Reference	418

7.83.1 Member Function Documentation	418
7.83.1.1 apply()	418
7.84 helib::MatMulFullExec_construct< type >::MatMulDimComp Struct Reference	418
7.84.1 Constructor & Destructor Documentation	419
7.84.1.1 MatMulDimComp()	419
7.84.2 Member Function Documentation	419
7.84.2.1 operator()()	419
7.84.3 Member Data Documentation	419
7.84.3.1 ea	419
7.85 helib::MatMulExecBase Class Reference	419
7.85.1 Constructor & Destructor Documentation	420
7.85.1.1 ~MatMulExecBase()	420
7.85.2 Member Function Documentation	420
7.85.2.1 getEA()	420
7.85.2.2 mul()	420
7.85.2.3 upgrade()	420
7.86 helib::MatMulFull Class Reference	421
7.86.1 Member Typedef Documentation	421
7.86.1.1 ExecType	421
7.86.2 Constructor & Destructor Documentation	421
7.86.2.1 ~MatMulFull()	421
7.86.3 Member Function Documentation	421
7.86.3.1 getEA()	422
7.87 helib::MatMulFull_derived< type > Class Template Reference	422
7.87.1 Member Function Documentation	422
7.87.1.1 get()	422
7.88 helib::MatMulFullExec Class Reference	423
7.88.1 Constructor & Destructor Documentation	423
7.88.1.1 MatMulFullExec()	423
7.88.2 Member Function Documentation	423
7.88.2.1 getEA()	423
7.88.2.2 mul()	424
7.88.2.3 rec_mul()	424
7.88.2.4 upgrade()	424
7.88.3 Member Data Documentation	424
7.88.3.1 dims	424
7.88.3.2 ea	424
7.88.3.3 minimal	424
7.88.3.4 transforms	425
7.89 helib::MatMulFullExec_construct< type > Struct Template Reference	425
7.89.1 Member Function Documentation	425
7.89.1.1 apply()	425

7.89.1.2 rec_mul()	426
7.90 helib::MatMulFullHelper< type > Class Template Reference	426
7.90.1 Constructor & Destructor Documentation	426
7.90.1.1 MatMulFullHelper()	427
7.90.2 Member Function Documentation	427
7.90.2.1 getDim()	427
7.90.2.2 getEA()	427
7.90.2.3 processDiagonal()	427
7.90.3 Member Data Documentation	427
7.90.3.1 dim	428
7.90.3.2 ea_basetype	428
7.90.3.3 init_idxes	428
7.90.3.4 mat	428
7.91 helib::ModuliSizes Class Reference	428
7.91.1 Detailed Description	429
7.91.2 Member Typedef Documentation	429
7.91.2.1 Entry	429
7.91.3 Member Function Documentation	429
7.91.3.1 getSet4Size() [1/2]	429
7.91.3.2 getSet4Size() [2/2]	429
7.91.3.3 init()	430
7.91.3.4 read()	430
7.91.3.5 write()	430
7.91.4 Friends And Related Function Documentation	430
7.91.4.1 operator<<	430
7.91.4.2 operator>>	430
7.92 helib::mul_BlockMatMul1D_impl< type > Struct Template Reference	431
7.92.1 Member Function Documentation	431
7.92.1.1 apply()	431
7.93 helib::mul_BlockMatMulFull_impl< type > Struct Template Reference	431
7.93.1 Member Function Documentation	431
7.93.1.1 apply()	431
7.94 helib::mul_MatMul1D_impl< type > Struct Template Reference	432
7.94.1 Member Function Documentation	432
7.94.1.1 apply()	432
7.95 helib::mul_MatMulFull_impl< type > Struct Template Reference	432
7.95.1 Member Function Documentation	432
7.95.1.1 apply()	432
7.96 helib::mul_pa_impl< type > Class Template Reference	433
7.96.1 Member Function Documentation	433
7.96.1.1 apply()	433
7.97 MyClass Class Reference	433

7.97.1 Constructor & Destructor Documentation	433
7.97.1.1 MyClass()	433
7.97.2 Member Function Documentation	433
7.97.2.1 get()	434
7.97.2.2 set()	434
7.98 helib::negate_pa_impl< type > Class Template Reference	434
7.98.1 Member Function Documentation	434
7.98.1.1 apply()	434
7.99 OtherTimingData Class Reference	434
7.99.1 Member Data Documentation	435
7.99.1.1 decode2	435
7.99.1.2 decode4	435
7.99.1.3 decrypt	435
7.99.1.4 encode2	435
7.99.1.5 encode2d	435
7.99.1.6 encode4	435
7.99.1.7 encode4d	435
7.99.1.8 encrypt	436
7.99.1.9 init2	436
7.99.1.10 init4	436
7.99.1.11 keyGen	436
7.100 helib::OutOfRangeError Class Reference	436
7.100.1 Detailed Description	437
7.100.2 Constructor & Destructor Documentation	437
7.100.2.1 OutOfRangeError() [1/2]	437
7.100.2.2 OutOfRangeError() [2/2]	437
7.100.2.3 ~OutOfRangeError()	437
7.100.3 Member Function Documentation	437
7.100.3.1 what()	437
7.101 helib::PAlgebra Class Reference	438
7.101.1 Detailed Description	439
7.101.2 Constructor & Destructor Documentation	440
7.101.2.1 PAlgebra()	440
7.101.3 Member Function Documentation	440
7.101.3.1 addCoord()	440
7.101.3.2 assembleIndexByDim()	440
7.101.3.3 breakIndexByDim()	440
7.101.3.4 coordinate()	441
7.101.3.5 exponentiate()	441
7.101.3.6 fftSizeNeeded()	441
7.101.3.7 frobeniusPow()	441
7.101.3.8 FrobPerturb()	441



7.101.3.9	genToPow()	441
7.101.3.10	get_cM()	442
7.101.3.11	getFFTInfo()	442
7.101.3.12	getHalfFFTInfo()	442
7.101.3.13	getM()	442
7.101.3.14	getNFactors()	442
7.101.3.15	getNormBnd()	442
7.101.3.16	getNSlots()	442
7.101.3.17	getOrdP()	443
7.101.3.18	getP()	443
7.101.3.19	getPhiM()	443
7.101.3.20	getPhimX()	443
7.101.3.21	getPolyNormBnd()	443
7.101.3.22	getPow2()	443
7.101.3.23	getQuarterFFTInfo()	444
7.101.3.24	getRadM()	444
7.101.3.25	indexInZmstar()	444
7.101.3.26	indexInZmstar_unchecked()	444
7.101.3.27	indexOfRep()	444
7.101.3.28	inZmStar()	444
7.101.3.29	isRep()	445
7.101.3.30	ith_rep()	445
7.101.3.31	nextExpVector()	445
7.101.3.32	numOfGens()	445
7.101.3.33	operator"!=()	445
7.101.3.34	operator==(())	445
7.101.3.35	OrderOf()	446
7.101.3.36	printAll()	446
7.101.3.37	printout()	446
7.101.3.38	ProdOrdsFrom()	446
7.101.3.39	repInZmstar_unchecked()	446
7.101.3.40	SameOrd()	446
7.101.3.41	set_cM()	447
7.101.3.42	ZmStarGen()	447
7.102	helib::PAlgebraMod Class Reference	447
7.102.1	Detailed Description	448
7.102.2	Constructor & Destructor Documentation	448
7.102.2.1	PAlgebraMod()	448
7.102.3	Member Function Documentation	448
7.102.3.1	getCx()	448
7.102.3.2	getDerived()	448
7.102.3.3	getFactorsOverZZ()	449

7.102.3.4 getMask_zzX()	449
7.102.3.5 getPPowR()	449
7.102.3.6 getR()	449
7.102.3.7 getTag()	449
7.102.3.8 getZMStar()	449
7.102.3.9 operator"!=()	450
7.102.3.10 operator==(())	450
7.102.3.11 restoreContext()	450
7.103 helib::PAlgebraModBase Class Reference	450
7.103.1 Detailed Description	451
7.103.2 Constructor & Destructor Documentation	451
7.103.2.1 ~PAlgebraModBase()	451
7.103.3 Member Function Documentation	451
7.103.3.1 clone()	451
7.103.3.2 getFactorsOverZZ()	451
7.103.3.3 getMask_zzX()	451
7.103.3.4 getPPowR()	452
7.103.3.5 getR()	452
7.103.3.6 getTag()	452
7.103.3.7 getZMStar()	452
7.103.3.8 restoreContext()	452
7.104 helib::PAlgebraModCx Class Reference	453
7.104.1 Detailed Description	453
7.104.2 Constructor & Destructor Documentation	453
7.104.2.1 PAlgebraModCx()	453
7.104.3 Member Function Documentation	454
7.104.3.1 clone()	454
7.104.3.2 getFactorsOverZZ()	454
7.104.3.3 getMask_zzX()	454
7.104.3.4 getPPowR()	454
7.104.3.5 getR()	454
7.104.3.6 getTag()	455
7.104.3.7 getZMStar()	455
7.104.3.8 restoreContext()	455
7.105 helib::PAlgebraModDerived< type > Class Template Reference	455
7.105.1 Detailed Description	457
7.105.2 Constructor & Destructor Documentation	457
7.105.2.1 PAlgebraModDerived() [1/2]	457
7.105.2.2 PAlgebraModDerived() [2/2]	457
7.105.3 Member Function Documentation	457
7.105.3.1 buildLinPolyCoeffs()	457
7.105.3.2 clone()	458

7.105.3.3 CRT_decompose()	458
7.105.3.4 CRT_reconstruct()	458
7.105.3.5 decodePlaintext()	458
7.105.3.6 embedInAllSlots()	458
7.105.3.7 embedInSlots()	459
7.105.3.8 getCrtCoeffs()	459
7.105.3.9 getFactors()	459
7.105.3.10 getFactorsOverZZ()	459
7.105.3.11 getMask_zzX()	459
7.105.3.12 getMaskTable()	460
7.105.3.13 getPhimXMod()	460
7.105.3.14 getPPowR()	460
7.105.3.15 getR()	460
7.105.3.16 getTag()	460
7.105.3.17 getZMStar()	461
7.105.3.18 mapToSlots()	461
7.105.3.19 operator=()	461
7.105.3.20 restoreContext()	461
7.106 helib::PermNetLayer Class Reference	461
7.106.1 Detailed Description	462
7.106.2 Member Function Documentation	462
7.106.2.1 getE()	462
7.106.2.2 getGenIdx()	462
7.106.2.3 getShifts()	462
7.106.2.4 isIdentity()	462
7.106.3 Friends And Related Function Documentation	462
7.106.3.1 operator<<	463
7.106.3.2 PermNetwork	463
7.107 helib::PermNetwork Class Reference	463
7.107.1 Detailed Description	463
7.107.2 Constructor & Destructor Documentation	464
7.107.2.1 PermNetwork() [1/2]	464
7.107.2.2 PermNetwork() [2/2]	464
7.107.3 Member Function Documentation	464
7.107.3.1 applyToCtxt()	464
7.107.3.2 applyToCube()	464
7.107.3.3 applyToPtxt()	464
7.107.3.4 buildNetwork()	465
7.107.3.5 depth()	465
7.107.3.6 getLayer()	465
7.107.4 Friends And Related Function Documentation	465
7.107.4.1 operator<<	465

7.108 helib::PGFFT Class Reference	465
7.108.1 Member Typedef Documentation	466
7.108.1.1 aligned_vector	466
7.108.2 Constructor & Destructor Documentation	466
7.108.2.1 PGFFT() [1/3]	466
7.108.2.2 PGFFT() [2/3]	466
7.108.2.3 PGFFT() [3/3]	466
7.108.3 Member Function Documentation	467
7.108.3.1 aligned_allocate()	467
7.108.3.2 aligned_deallocate()	467
7.108.3.3 apply() [1/2]	467
7.108.3.4 apply() [2/2]	467
7.108.3.5 operator=() [1/2]	467
7.108.3.6 operator=() [2/2]	467
7.108.3.7 simd_enabled()	468
7.109 helib::PlaintextArray Class Reference	468
7.109.1 Constructor & Destructor Documentation	468
7.109.1.1 PlaintextArray() [1/2]	468
7.109.1.2 PlaintextArray() [2/2]	468
7.109.2 Member Function Documentation	468
7.109.2.1 getData() [1/2]	469
7.109.2.2 getData() [2/2]	469
7.109.2.3 operator=()	469
7.109.2.4 print()	469
7.110 helib::PlaintextArrayBase Class Reference	469
7.110.1 Constructor & Destructor Documentation	470
7.110.1.1 ~PlaintextArrayBase()	470
7.110.2 Member Function Documentation	470
7.110.2.1 print()	470
7.111 helib::PlaintextArrayDerived< type > Class Template Reference	470
7.111.1 Member Function Documentation	471
7.111.1.1 print()	471
7.111.2 Member Data Documentation	471
7.111.2.1 data	471
7.112 helib::PolyMod Class Reference	471
7.112.1 Detailed Description	473
7.112.2 Constructor & Destructor Documentation	474
7.112.2.1 PolyMod() [1/7]	474
7.112.2.2 PolyMod() [2/7]	474
7.112.2.3 PolyMod() [3/7]	475
7.112.2.4 PolyMod() [4/7]	475
7.112.2.5 PolyMod() [5/7]	476

7.112.2.6 PolyMod() [ 6 / 7 ]	477
7.112.2.7 PolyMod() [ 7 / 7 ]	477
7.112.2.8 ~PolyMod()	478
7.112.3 Member Function Documentation	478
7.112.3.1 getData()	478
7.112.3.2 getG()	478
7.112.3.3 getp2r()	478
7.112.3.4 isValid()	479
7.112.3.5 negate()	479
7.112.3.6 operator long()	479
7.112.3.7 operator NTL::ZZX()	479
7.112.3.8 operator std::vector< long >()	479
7.112.3.9 operator"!=()	479
7.112.3.10 operator*() [ 1 / 3 ]	480
7.112.3.11 operator*() [ 2 / 3 ]	480
7.112.3.12 operator*() [ 3 / 3 ]	481
7.112.3.13 operator*=( ) [ 1 / 3 ]	481
7.112.3.14 operator*=( ) [ 2 / 3 ]	482
7.112.3.15 operator*=( ) [ 3 / 3 ]	482
7.112.3.16 operator+( ) [ 1 / 3 ]	483
7.112.3.17 operator+( ) [ 2 / 3 ]	483
7.112.3.18 operator+( ) [ 3 / 3 ]	483
7.112.3.19 operator+=( ) [ 1 / 3 ]	484
7.112.3.20 operator+=( ) [ 2 / 3 ]	484
7.112.3.21 operator+=( ) [ 3 / 3 ]	485
7.112.3.22 operator-( ) [ 1 / 4 ]	485
7.112.3.23 operator-( ) [ 2 / 4 ]	485
7.112.3.24 operator-( ) [ 3 / 4 ]	486
7.112.3.25 operator-( ) [ 4 / 4 ]	486
7.112.3.26 operator==( ) [ 1 / 3 ]	487
7.112.3.27 operator==( ) [ 2 / 3 ]	487
7.112.3.28 operator==( ) [ 3 / 3 ]	488
7.112.3.29 operator=( ) [ 1 / 6 ]	488
7.112.3.30 operator=( ) [ 2 / 6 ]	488
7.112.3.31 operator=( ) [ 3 / 6 ]	489
7.112.3.32 operator=( ) [ 4 / 6 ]	489
7.112.3.33 operator=( ) [ 5 / 6 ]	490
7.112.3.34 operator=( ) [ 6 / 6 ]	490
7.112.3.35 operator==( ) [ 1 / 4 ]	490
7.112.3.36 operator==( ) [ 2 / 4 ]	491
7.112.3.37 operator==( ) [ 3 / 4 ]	491
7.112.3.38 operator==( ) [ 4 / 4 ]	491

7.112.4 Friends And Related Function Documentation	492
7.112.4.1 operator<<	492
7.112.4.2 operator>>	493
7.113 helib::PolyModRing Struct Reference	493
7.113.1 Detailed Description	494
7.113.2 Constructor & Destructor Documentation	494
7.113.2.1 PolyModRing() [1/4]	494
7.113.2.2 PolyModRing() [2/4]	494
7.113.2.3 PolyModRing() [3/4]	495
7.113.2.4 ~PolyModRing()	495
7.113.2.5 PolyModRing() [4/4]	495
7.113.3 Member Function Documentation	496
7.113.3.1 operator!=(())	496
7.113.3.2 operator=(()) [1/2]	496
7.113.3.3 operator=(()) [2/2]	496
7.113.3.4 operator==(())	497
7.113.4 Friends And Related Function Documentation	497
7.113.4.1 operator<<	497
7.113.5 Member Data Documentation	497
7.113.5.1 G	497
7.113.5.2 p	498
7.113.5.3 p2r	498
7.113.5.4 r	498
7.114 helib::PowerfulConversion Class Reference	498
7.114.1 Detailed Description	499
7.114.2 Constructor & Destructor Documentation	499
7.114.2.1 PowerfulConversion() [1/2]	499
7.114.2.2 PowerfulConversion() [2/2]	499
7.114.3 Member Function Documentation	499
7.114.3.1 getLongSig()	499
7.114.3.2 getShortSig()	500
7.114.3.3 initPConv()	500
7.114.3.4 polyToPowerful()	500
7.114.3.5 powerfulToPoly()	500
7.114.3.6 restoreModulus()	500
7.115 helib::PowerfulDCRT Class Reference	500
7.115.1 Detailed Description	501
7.115.2 Constructor & Destructor Documentation	501
7.115.2.1 PowerfulDCRT()	501
7.115.3 Member Function Documentation	501
7.115.3.1 dcrtToPowerful()	501
7.115.3.2 getIndexTranslation()	501

7.115.3.3 getPConv()	501
7.115.3.4 powerfulToZZX()	502
7.115.3.5 ZZxtoPowerful()	502
7.116 helib::PowerfulTranslationIndexes Class Reference	502
7.116.1 Detailed Description	502
7.116.2 Constructor & Destructor Documentation	503
7.116.2.1 PowerfulTranslationIndexes()	503
7.116.3 Member Data Documentation	503
7.116.3.1 cubeToPolyMap	503
7.116.3.2 cycVec	503
7.116.3.3 divvec	503
7.116.3.4 invvec	503
7.116.3.5 longSig	503
7.116.3.6 m	504
7.116.3.7 mvec	504
7.116.3.8 phim	504
7.116.3.9 phimX	504
7.116.3.10 phivec	504
7.116.3.11 polyToCubeMap	504
7.116.3.12 shortSig	504
7.116.3.13 shortToLongMap	505
7.117 helib::PrimeGenerator Struct Reference	505
7.117.1 Constructor & Destructor Documentation	505
7.117.1.1 PrimeGenerator()	505
7.117.2 Member Function Documentation	505
7.117.2.1 next()	505
7.117.3 Member Data Documentation	505
7.117.3.1 k	506
7.117.3.2 len	506
7.117.3.3 m	506
7.117.3.4 t	506
7.118 helib::print_pa_impl< type > Class Template Reference	506
7.118.1 Member Function Documentation	506
7.118.1.1 apply()	506
7.119 helib::PtrMatrix< T > Struct Template Reference	507
7.119.1 Detailed Description	507
7.119.2 Constructor & Destructor Documentation	507
7.119.2.1 ~PtrMatrix()	507
7.119.3 Member Function Documentation	507
7.119.3.1 operator[]() [1/2]	508
7.119.3.2 operator[]() [2/2]	508
7.119.3.3 ptr2nonNull()	508

7.119.3.4	resize()	508
7.119.3.5	size()	508
7.120	helib::PtrMatrix_PtPtrVector< T > Struct Template Reference	509
7.120.1	Detailed Description	509
7.120.2	Constructor & Destructor Documentation	509
7.120.2.1	PtrMatrix_PtPtrVector()	509
7.120.3	Member Function Documentation	509
7.120.3.1	operator[]() [1/2]	510
7.120.3.2	operator[]() [2/2]	510
7.120.3.3	size()	510
7.120.4	Member Data Documentation	510
7.120.4.1	rows	510
7.121	helib::PtrMatrix_ptVec< T > Struct Template Reference	510
7.121.1	Detailed Description	511
7.121.2	Constructor & Destructor Documentation	511
7.121.2.1	PtrMatrix_ptVec()	511
7.121.3	Member Function Documentation	511
7.121.3.1	operator[]() [1/2]	511
7.121.3.2	operator[]() [2/2]	512
7.121.3.3	size()	512
7.121.4	Member Data Documentation	512
7.121.4.1	buffer	512
7.121.4.2	rows	512
7.122	helib::PtrMatrix_ptvector< T > Struct Template Reference	512
7.122.1	Detailed Description	513
7.122.2	Constructor & Destructor Documentation	513
7.122.2.1	PtrMatrix_ptvector()	513
7.122.3	Member Function Documentation	513
7.122.3.1	operator[]() [1/2]	513
7.122.3.2	operator[]() [2/2]	514
7.122.3.3	size()	514
7.122.4	Member Data Documentation	514
7.122.4.1	buffer	514
7.122.4.2	rows	514
7.123	helib::PtrMatrix_Vec< T > Struct Template Reference	514
7.123.1	Detailed Description	515
7.123.2	Constructor & Destructor Documentation	515
7.123.2.1	PtrMatrix_Vec()	515
7.123.3	Member Function Documentation	515
7.123.3.1	operator[]() [1/2]	515
7.123.3.2	operator[]() [2/2]	516
7.123.3.3	resize()	516



7.123.3.4 size()	516
7.123.4 Member Data Documentation	516
7.123.4.1 buffer	516
7.123.4.2 rows	516
7.124 helib::PtrMatrix_vector< T > Struct Template Reference	517
7.124.1 Detailed Description	517
7.124.2 Constructor & Destructor Documentation	517
7.124.2.1 PtrMatrix_vector()	517
7.124.3 Member Function Documentation	517
7.124.3.1 operator[]() [1/2]	518
7.124.3.2 operator[]() [2/2]	518
7.124.3.3 resize()	518
7.124.3.4 size()	518
7.124.4 Member Data Documentation	518
7.124.4.1 buffer	518
7.124.4.2 rows	519
7.125 helib::PtrVector< T > Struct Template Reference	519
7.125.1 Detailed Description	519
7.125.2 Constructor & Destructor Documentation	519
7.125.2.1 ~PtrVector()	519
7.125.3 Member Function Documentation	520
7.125.3.1 isSet()	520
7.125.3.2 numNonNull()	520
7.125.3.3 operator[]()	520
7.125.3.4 ptr2nonNull()	520
7.125.3.5 resize()	520
7.125.3.6 size()	521
7.126 helib::PtrVector_Singleton< T > Struct Template Reference	521
7.126.1 Detailed Description	521
7.126.2 Constructor & Destructor Documentation	521
7.126.2.1 PtrVector_Singleton()	522
7.126.3 Member Function Documentation	522
7.126.3.1 operator[]()	522
7.126.3.2 size()	522
7.126.4 Member Data Documentation	522
7.126.4.1 v	522
7.127 helib::PtrVector_slice< T > Struct Template Reference	523
7.127.1 Detailed Description	523
7.127.2 Constructor & Destructor Documentation	523
7.127.2.1 PtrVector_slice() [1/2]	523
7.127.2.2 PtrVector_slice() [2/2]	524
7.127.3 Member Function Documentation	524

7.127.3.1 numNonNull()	524
7.127.3.2 operator[]()	524
7.127.3.3 ptr2nonNull()	524
7.127.3.4 size()	524
7.127.4 Member Data Documentation	525
7.127.4.1 orig	525
7.127.4.2 start	525
7.127.4.3 sz	525
7.128 helib::PtrVector_VecPt< T > Struct Template Reference	525
7.128.1 Detailed Description	526
7.128.2 Constructor & Destructor Documentation	526
7.128.2.1 PtrVector_VecPt()	526
7.128.3 Member Function Documentation	526
7.128.3.1 operator[]()	526
7.128.3.2 resize()	526
7.128.3.3 size()	527
7.128.4 Member Data Documentation	527
7.128.4.1 v	527
7.129 helib::PtrVector_VecT< T > Struct Template Reference	527
7.129.1 Detailed Description	528
7.129.2 Constructor & Destructor Documentation	528
7.129.2.1 PtrVector_VecT()	528
7.129.3 Member Function Documentation	528
7.129.3.1 numNonNull()	528
7.129.3.2 operator[]()	528
7.129.3.3 ptr2nonNull()	528
7.129.3.4 resize()	529
7.129.3.5 size()	529
7.129.4 Member Data Documentation	529
7.129.4.1 v	529
7.130 helib::PtrVector_vectorPt< T > Struct Template Reference	529
7.130.1 Detailed Description	530
7.130.2 Constructor & Destructor Documentation	530
7.130.2.1 PtrVector_vectorPt()	530
7.130.3 Member Function Documentation	530
7.130.3.1 operator[]()	530
7.130.3.2 resize()	530
7.130.3.3 size()	531
7.130.4 Member Data Documentation	531
7.130.4.1 v	531
7.131 helib::PtrVector_vectorT< T > Struct Template Reference	531
7.131.1 Detailed Description	532

7.131.2 Constructor & Destructor Documentation	532
7.131.2.1 PtrVector_vectorT()	532
7.131.3 Member Function Documentation	532
7.131.3.1 numNonNull()	532
7.131.3.2 operator[]()	532
7.131.3.3 resize()	532
7.131.3.4 size()	533
7.131.4 Member Data Documentation	533
7.131.4.1 v	533
7.132 helib::Ptxt Class Reference	533
7.132.1 Detailed Description	537
7.132.2 Member Typedef Documentation	538
7.132.2.1 SlotType	538
7.132.3 Constructor & Destructor Documentation	538
7.132.3.1 Ptxt() [1/13]	538
7.132.3.2 Ptxt() [2/13]	538
7.132.3.3 Ptxt() [3/13]	538
7.132.3.4 Ptxt() [4/13]	539
7.132.3.5 Ptxt() [5/13]	539
7.132.3.6 Ptxt() [6/13]	540
7.132.3.7 Ptxt() [7/13]	540
7.132.3.8 Ptxt() [8/13]	541
7.132.3.9 ~Ptxt()	541
7.132.3.10 Ptxt() [9/13]	541
7.132.3.11 Ptxt() [10/13]	541
7.132.3.12 Ptxt() [11/13]	541
7.132.3.13 Ptxt() [12/13]	542
7.132.3.14 Ptxt() [13/13]	542
7.132.4 Member Function Documentation	542
7.132.4.1 addConstant()	542
7.132.4.2 addConstantCKKS()	542
7.132.4.3 at() [1/2]	543
7.132.4.4 at() [2/2]	543
7.132.4.5 automorph() [1/3]	544
7.132.4.6 automorph() [2/3]	544
7.132.4.7 automorph() [3/3]	545
7.132.4.8 clear()	545
7.132.4.9 complexConj() [1/2]	545
7.132.4.10 complexConj() [2/2]	545
7.132.4.11 convertToSlot() [1/3]	545
7.132.4.12 convertToSlot() [2/3]	545
7.132.4.13 convertToSlot() [3/3]	545

7.132.4.14 cube()	546
7.132.4.15 decodeSetData() [1/2]	546
7.132.4.16 decodeSetData() [2/2]	546
7.132.4.17 frobeniusAutomorph() [1/2]	547
7.132.4.18 frobeniusAutomorph() [2/2]	547
7.132.4.19 getPolyRepr() [1/3]	548
7.132.4.20 getPolyRepr() [2/3]	548
7.132.4.21 getPolyRepr() [3/3]	548
7.132.4.22 getSlotRepr()	549
7.132.4.23 imag() [1/2]	549
7.132.4.24 imag() [2/2]	549
7.132.4.25 incrementalProduct()	549
7.132.4.26 isValid()	550
7.132.4.27 lsize()	550
7.132.4.28 mapTo01()	550
7.132.4.29 multiplyBy()	550
7.132.4.30 multiplyBy2()	551
7.132.4.31 negate()	551
7.132.4.32 operator"!=(	552
7.132.4.33 operator*()	552
7.132.4.34 operator*=(	553
7.132.4.35 operator*=(	553
7.132.4.36 operator*=(	553
7.132.4.37 operator*=(	554
7.132.4.38 operator*=(	554
7.132.4.39 operator+()	554
7.132.4.40 operator+=(	555
7.132.4.41 operator+=(	555
7.132.4.42 operator+=(	555
7.132.4.43 operator+=(	556
7.132.4.44 operator+=(	556
7.132.4.45 operator-()	556
7.132.4.46 operator-=(	557
7.132.4.47 operator-=(	557
7.132.4.48 operator-=(	557
7.132.4.49 operator-=(	558
7.132.4.50 operator-=(	558
7.132.4.51 operator=()	558
7.132.4.52 operator=()	559
7.132.4.53 operator==(	559
7.132.4.54 operator[]()	559
7.132.4.55 operator[]()	560

7.132.4.56 power()	560
7.132.4.57 random()	561
7.132.4.58 real() [1/2]	561
7.132.4.59 real() [2/2]	561
7.132.4.60 replicate()	561
7.132.4.61 replicateAll()	562
7.132.4.62 rotate()	562
7.132.4.63 rotate1D()	562
7.132.4.64 runningSums()	563
7.132.4.65 setData() [1/6]	563
7.132.4.66 setData() [2/6]	563
7.132.4.67 setData() [3/6]	564
7.132.4.68 setData() [4/6]	564
7.132.4.69 setData() [5/6]	564
7.132.4.70 setData() [6/6]	565
7.132.4.71 shift()	565
7.132.4.72 shift1D()	565
7.132.4.73 size()	566
7.132.4.74 square()	566
7.132.4.75 totalProduct()	566
7.132.4.76 totalSums()	567
7.132.5 Friends And Related Function Documentation	567
7.132.5.1 operator<<	567
7.132.5.2 operator>>	567
7.133 helib::PubKey Class Reference	568
7.133.1 Detailed Description	570
7.133.2 Constructor & Destructor Documentation	570
7.133.2.1 PubKey() [1/3]	570
7.133.2.2 PubKey() [2/3]	570
7.133.2.3 PubKey() [3/3]	570
7.133.2.4 ~PubKey()	570
7.133.3 Member Function Documentation	571
7.133.3.1 CKKSencrypt() [1/2]	571
7.133.3.2 CKKSencrypt() [2/2]	571
7.133.3.3 clear()	571
7.133.3.4 Encrypt() [1/7]	571
7.133.3.5 Encrypt() [2/7]	572
7.133.3.6 Encrypt() [3/7]	572
7.133.3.7 Encrypt() [4/7]	572
7.133.3.8 Encrypt() [5/7]	572
7.133.3.9 Encrypt() [6/7]	573
7.133.3.10 Encrypt() [7/7]	573

7.133.3.11 ePlusR()	573
7.133.3.12 getAnyKeySWmatrix()	573
7.133.3.13 getContext()	574
7.133.3.14 getKeySWmatrix() [1/2]	574
7.133.3.15 getKeySWmatrix() [2/2]	574
7.133.3.16 getKSStrategy()	574
7.133.3.17 getNextKSWmatrix()	574
7.133.3.18 getPtxtSpace()	574
7.133.3.19 getSKeyBound()	575
7.133.3.20 hackPtxtSpace()	575
7.133.3.21 haveAnyKeySWmatrix()	575
7.133.3.22 haveKeySWmatrix() [1/2]	575
7.133.3.23 haveKeySWmatrix() [2/2]	575
7.133.3.24 isBootstrappable()	575
7.133.3.25 isCKKS()	576
7.133.3.26 isReachable()	576
7.133.3.27 keyExists()	576
7.133.3.28 keySWlist()	576
7.133.3.29 operator"!=()"	576
7.133.3.30 operator==(())	576
7.133.3.31 reCrypt()	576
7.133.3.32 setKeySwitchMap()	577
7.133.3.33 setKSStrategy()	577
7.133.3.34 thinReCrypt()	577
7.133.3.35 void ::helib::readPubKeyBinary()	577
7.133.3.36 void ::helib::writePubKeyBinary()	577
7.133.4 Friends And Related Function Documentation	577
7.133.4.1 operator<<	578
7.133.4.2 operator>>	578
7.133.4.3 SecKey	578
7.134 helib::PubKeyHack Struct Reference	578
7.134.1 Member Data Documentation	578
7.134.1.1 context	578
7.134.1.2 keySwitching	579
7.134.1.3 keySwitchMap	579
7.134.1.4 KS_strategy	579
7.134.1.5 pubEncrKey	579
7.134.1.6 recryptEkey	579
7.134.1.7 recryptKeyID	579
7.134.1.8 skHwts	579
7.135 helib::quarter_FFT Struct Reference	580
7.135.1 Constructor & Destructor Documentation	580

7.135.1.1 quarter_FFT()	580
7.135.2 Member Data Documentation	580
7.135.2.1 fft	580
7.135.2.2 pow1	580
7.135.2.3 pow2	580
7.136 helib::random_pa_impl< type > Class Template Reference	581
7.136.1 Member Function Documentation	581
7.136.1.1 apply()	581
7.137 helib::RandomBlockMatrix< type > Class Template Reference	581
7.137.1 Constructor & Destructor Documentation	582
7.137.1.1 RandomBlockMatrix()	582
7.137.2 Member Function Documentation	582
7.137.2.1 get()	582
7.137.2.2 getDim()	582
7.137.2.3 getEA()	582
7.137.2.4 multipleTransforms()	583
7.138 helib::RandomFullBlockMatrix< type > Class Template Reference	583
7.138.1 Constructor & Destructor Documentation	583
7.138.1.1 RandomFullBlockMatrix()	583
7.138.2 Member Function Documentation	583
7.138.2.1 get()	584
7.138.2.2 getEA()	584
7.139 helib::RandomFullMatrix< type > Class Template Reference	584
7.139.1 Constructor & Destructor Documentation	584
7.139.1.1 RandomFullMatrix()	585
7.139.2 Member Function Documentation	585
7.139.2.1 get()	585
7.139.2.2 getEA()	585
7.140 helib::RandomMatrix< type > Class Template Reference	585
7.140.1 Constructor & Destructor Documentation	586
7.140.1.1 ~RandomMatrix()	586
7.140.1.2 RandomMatrix()	586
7.140.2 Member Function Documentation	586
7.140.2.1 get()	586
7.140.2.2 getDim()	587
7.140.2.3 getEA()	587
7.140.2.4 multipleTransforms()	587
7.141 helib::RandomMultiBlockMatrix< type > Class Template Reference	587
7.141.1 Constructor & Destructor Documentation	588
7.141.1.1 RandomMultiBlockMatrix()	588
7.141.2 Member Function Documentation	588
7.141.2.1 get()	588

7.141.2.2 getDim()	588
7.141.2.3 getEA()	589
7.141.2.4 multipleTransforms()	589
7.142 helib::RandomMultiMatrix< type > Class Template Reference	589
7.142.1 Constructor & Destructor Documentation	590
7.142.1.1 ~RandomMultiMatrix()	590
7.142.1.2 RandomMultiMatrix()	590
7.142.2 Member Function Documentation	590
7.142.2.1 get()	590
7.142.2.2 getDim()	590
7.142.2.3 getEA()	591
7.142.2.4 multipleTransforms()	591
7.143 helib::RandomState Class Reference	591
7.143.1 Detailed Description	591
7.143.2 Constructor & Destructor Documentation	592
7.143.2.1 RandomState()	592
7.143.2.2 ~RandomState()	592
7.143.3 Member Function Documentation	592
7.143.3.1 restore()	592
7.144 helib::RecryptData Class Reference	592
7.144.1 Detailed Description	593
7.144.2 Constructor & Destructor Documentation	593
7.144.2.1 RecryptData()	594
7.144.3 Member Function Documentation	594
7.144.3.1 init()	594
7.144.3.2 operator"!=()"	594
7.144.3.3 operator==(())	594
7.144.3.4 setAE()	594
7.144.4 Member Data Documentation	595
7.144.4.1 alMod	595
7.144.4.2 build_cache	595
7.144.4.3 defSkHwt	595
7.144.4.4 e	595
7.144.4.5 ea	595
7.144.4.6 ePrime	595
7.144.4.7 firstMap	596
7.144.4.8 mvec	596
7.144.4.9 p2dConv	596
7.144.4.10 secondMap	596
7.144.4.11 skHwt	596
7.144.4.12 unpackSlotEncoding	596
7.145 helib::replicate_pa_impl< type > Class Template Reference	597



7.145.1 Member Function Documentation	597
7.145.1.1 apply()	597
7.146 ReplicateDummy Class Reference	597
7.146.1 Constructor & Destructor Documentation	597
7.146.1.1 ReplicateDummy()	597
7.146.2 Member Function Documentation	598
7.146.2.1 handle()	598
7.147 helib::ReplicateHandler Class Reference	598
7.147.1 Detailed Description	598
7.147.2 Constructor & Destructor Documentation	598
7.147.2.1 ~ReplicateHandler()	598
7.147.3 Member Function Documentation	599
7.147.3.1 earlyStop()	599
7.147.3.2 handle()	599
7.148 ReplicateTester Class Reference	599
7.148.1 Constructor & Destructor Documentation	600
7.148.1.1 ReplicateTester()	600
7.148.2 Member Function Documentation	600
7.148.2.1 handle()	600
7.148.3 Member Data Documentation	600
7.148.3.1 B	600
7.148.3.2 ea	600
7.148.3.3 error	600
7.148.3.4 pa	601
7.148.3.5 pos	601
7.148.3.6 sKey	601
7.148.3.7 t_last	601
7.148.3.8 t_total	601
7.149 helib::rotate_pa_impl< type > Class Template Reference	601
7.149.1 Member Function Documentation	601
7.149.1.1 apply()	602
7.150 helib::RuntimeError Class Reference	602
7.150.1 Detailed Description	602
7.150.2 Constructor & Destructor Documentation	602
7.150.2.1 RuntimeError() [1/2]	603
7.150.2.2 RuntimeError() [2/2]	603
7.150.2.3 ~RuntimeError()	603
7.150.3 Member Function Documentation	603
7.150.3.1 what()	603
7.151 helib::ScratchCell Class Reference	603
7.151.1 Detailed Description	604
7.151.2 Constructor & Destructor Documentation	604

7.151.2.1 ScratchCell() [1/2]	604
7.151.2.2 ScratchCell() [2/2]	604
7.151.3 Member Data Documentation	604
7.151.3.1 ct	604
7.151.3.2 used	604
7.152 helib::SecKey Class Reference	605
7.152.1 Detailed Description	606
7.152.2 Constructor & Destructor Documentation	606
7.152.2.1 SecKey() [1/2]	606
7.152.2.2 ~SecKey()	606
7.152.2.3 SecKey() [2/2]	606
7.152.3 Member Function Documentation	606
7.152.3.1 clear()	606
7.152.3.2 Decrypt() [1/5]	607
7.152.3.3 Decrypt() [2/5]	607
7.152.3.4 Decrypt() [3/5]	607
7.152.3.5 Decrypt() [4/5]	607
7.152.3.6 Decrypt() [5/5]	607
7.152.3.7 Encrypt() [1/2]	608
7.152.3.8 Encrypt() [2/2]	608
7.152.3.9 GenKeySWmatrix()	608
7.152.3.10 genRecryptData()	609
7.152.3.11 GenSecKey()	609
7.152.3.12 ImportSecKey()	609
7.152.3.13 operator"!="()	609
7.152.3.14 operator==( )	609
7.152.3.15 skEncrypt() [1/2]	610
7.152.3.16 skEncrypt() [2/2]	610
7.152.3.17 void ::helib::readSecKeyBinary()	610
7.152.3.18 void ::helib::writeSecKeyBinary()	610
7.152.4 Friends And Related Function Documentation	610
7.152.4.1 operator<<	610
7.152.4.2 operator>>	611
7.152.5 Member Data Documentation	611
7.152.5.1 sKeys	611
7.153 helib::shallow_clone< X > Class Template Reference	611
7.153.1 Detailed Description	611
7.153.2 Member Function Documentation	611
7.153.2.1 apply()	612
7.154 helib::shift_pa_impl< type > Class Template Reference	612
7.154.1 Member Function Documentation	612
7.154.1.1 apply()	612

7.155 helib::SKHandle Class Reference	612
7.155.1 Detailed Description	613
7.155.2 Constructor & Destructor Documentation	613
7.155.2.1 SKHandle()	613
7.155.3 Member Function Documentation	613
7.155.3.1 getPowerOfS()	614
7.155.3.2 getPowerOfX()	614
7.155.3.3 getSecretKeyID()	614
7.155.3.4 isBase()	614
7.155.3.5 isOne()	614
7.155.3.6 mul()	614
7.155.3.7 operator"!="()	615
7.155.3.8 operator=="()	615
7.155.3.9 read()	615
7.155.3.10 setBase()	615
7.155.3.11 setOne()	615
7.155.3.12 write()	615
7.155.4 Friends And Related Function Documentation	615
7.155.4.1 Ctxt	616
7.155.4.2 operator>>	616
7.156 StopReplicate Class Reference	616
7.157 helib::sub_pa_impl< type > Class Template Reference	616
7.157.1 Member Function Documentation	616
7.157.1.1 apply()	616
7.158 helib::SubDimension Class Reference	617
7.158.1 Detailed Description	617
7.158.2 Constructor & Destructor Documentation	617
7.158.2.1 SubDimension()	617
7.158.3 Member Function Documentation	617
7.158.3.1 totalLength()	618
7.158.4 Friends And Related Function Documentation	618
7.158.4.1 operator<<	618
7.158.5 Member Data Documentation	618
7.158.5.1 e	618
7.158.5.2 frstBenes	618
7.158.5.3 good	618
7.158.5.4 scndBenes	618
7.158.5.5 size	619
7.159 helib::ThinEvalMap Class Reference	619
7.159.1 Detailed Description	619
7.159.2 Constructor & Destructor Documentation	619
7.159.2.1 ThinEvalMap()	619

7.159.3 Member Function Documentation	619
7.159.3.1 apply()	620
7.159.3.2 upgrade()	620
7.160 helib::ThinRecryptData Class Reference	620
7.160.1 Detailed Description	620
7.160.2 Member Function Documentation	621
7.160.2.1 init()	621
7.160.3 Member Data Documentation	621
7.160.3.1 coeffToSlot	621
7.160.3.2 slotToCoeff	621
7.161 TimingData Class Reference	621
7.161.1 Constructor & Destructor Documentation	622
7.161.1.1 TimingData()	622
7.161.2 Member Data Documentation	622
7.161.2.1 highLvl	622
7.161.2.2 lowLvl	622
7.161.2.3 m	622
7.161.2.4 nSlots	622
7.161.2.5 other	623
7.161.2.6 p	623
7.161.2.7 phim	623
7.162 helib::TreeNode< T > Class Template Reference	623
7.162.1 Detailed Description	624
7.162.2 Constructor & Destructor Documentation	624
7.162.2.1 TreeNode() [1/2]	624
7.162.2.2 TreeNode() [2/2]	624
7.162.3 Member Function Documentation	624
7.162.3.1 getData() [1/2]	624
7.162.3.2 getData() [2/2]	624
7.162.3.3 getLeftChild()	625
7.162.3.4 getNext()	625
7.162.3.5 getParent()	625
7.162.3.6 getPrev()	625
7.162.3.7 getRightChild()	625
7.162.4 Friends And Related Function Documentation	625
7.162.4.1 FullBinaryTree< T >	625
7.163 helib::ZZ_pXModulus1 Class Reference	626
7.163.1 Detailed Description	626
7.163.2 Constructor & Destructor Documentation	626
7.163.2.1 ZZ_pXModulus1()	626
7.163.3 Member Function Documentation	626
7.163.3.1 upcast()	626

7.164 helib::zz_pXModulus1 Class Reference	627
7.164.1 Detailed Description	627
7.164.2 Constructor & Destructor Documentation	627
7.164.2.1 zz_pXModulus1()	627
7.164.3 Member Function Documentation	627
7.164.3.1 upcast()	627
7.164.4 Member Data Documentation	628
7.164.4.1 f	628
7.164.4.2 fm	628
7.164.4.3 k	628
7.164.4.4 k1	628
7.164.4.5 m	628
7.164.4.6 n	628
7.164.4.7 R0	628
7.164.4.8 R1	628
7.164.4.9 specialLogic	628
<b>8 File Documentation</b>	<b>629</b>
8.1 mainpage.dox File Reference	629
8.2 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/apiAttributes.h File Reference	629
8.2.1 Macro Definition Documentation	629
8.2.1.1 UNUSED	629
8.3 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ArgMap.h File Reference	630
8.3.1 Detailed Description	630
8.4 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/assertions.h File Reference	630
8.4.1 Detailed Description	631
8.5 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/binaryArith.h File Reference	631
8.5.1 Detailed Description	632
8.6 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/binaryCompare.h File Reference	633
8.6.1 Detailed Description	633
8.7 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/binio.h File Reference	633
8.7.1 Macro Definition Documentation	634
8.7.1.1 BINIO_32BIT	635
8.7.1.2 BINIO_48BIT	635
8.7.1.3 BINIO_64BIT	635
8.7.1.4 BINIO_EYE_CONTEXT_BEGIN	635
8.7.1.5 BINIO_EYE_CONTEXT_END	635
8.7.1.6 BINIO_EYE_CONTEXTBASE_BEGIN	635

8.7.1.7	<a href="#">BINIO_EYE_CONTEXTBASE_END</a>	635
8.7.1.8	<a href="#">BINIO_EYE_CTXT_BEGIN</a>	635
8.7.1.9	<a href="#">BINIO_EYE_CTXT_END</a>	636
8.7.1.10	<a href="#">BINIO_EYE_PK_BEGIN</a>	636
8.7.1.11	<a href="#">BINIO_EYE_PK_END</a>	636
8.7.1.12	<a href="#">BINIO_EYE_SIZE</a>	636
8.7.1.13	<a href="#">BINIO_EYE_SK_BEGIN</a>	636
8.7.1.14	<a href="#">BINIO_EYE_SK_END</a>	636
8.7.1.15	<a href="#">BINIO_EYE_SKM_BEGIN</a>	636
8.7.1.16	<a href="#">BINIO_EYE_SKM_END</a>	636
8.8	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/bluestein.h</a> File Reference	637
8.8.1	Detailed Description	637
8.9	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/clonedPtr.h</a> File Reference	637
8.9.1	Detailed Description	638
8.9.2	Macro Definition Documentation	638
8.9.2.1	<a href="#">CLONED_PTR_DECLARE</a>	638
8.9.2.2	<a href="#">CLONED_PTR_TEMPLATE_MEMBERS</a>	638
8.10	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/CModulus.h</a> File Reference	639
8.10.1	Detailed Description	639
8.11	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Context.h</a> File Reference	639
8.11.1	Detailed Description	640
8.12	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/CtPtrs.h</a> File Reference	640
8.12.1	Detailed Description	641
8.13	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Ctxt.h</a> File Reference	641
8.13.1	Detailed Description	642
8.14	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/debugging.h</a> File Reference	643
8.14.1	Detailed Description	644
8.14.2	Macro Definition Documentation	644
8.14.2.1	<a href="#">FLAG_PRINT_DVEC</a>	644
8.14.2.2	<a href="#">FLAG_PRINT_POLY</a>	644
8.14.2.3	<a href="#">FLAG_PRINT_VEC</a>	645
8.14.2.4	<a href="#">FLAG_PRINT_XVEC</a>	645
8.14.2.5	<a href="#">FLAG_PRINT_ZZX</a>	645
8.15	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/DoubleC↔ RT.h</a> File Reference	645
8.15.1	Detailed Description	646
8.16	<a href="#">/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Encrypted↔ Array.h</a> File Reference	646

8.16.1 Detailed Description	647
8.16.2 Macro Definition Documentation	648
8.16.2.1 CPA_BOILER	648
8.16.2.2 HELIB_MORE_UNWRAPARGS	648
8.16.2.3 PA_BOILER	648
8.17 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/EvalMap.h File Reference	648
8.17.1 Detailed Description	649
8.18 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/exceptions.h File Reference	649
8.18.1 Detailed Description	650
8.19 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/FHE.h File Reference	650
8.19.1 Detailed Description	650
8.20 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/fhe_stats.h File Reference	650
8.20.1 Macro Definition Documentation	651
8.20.1.1 HELIB_STATS_SAVE	651
8.20.1.2 HELIB_STATS_UPDATE	651
8.21 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/helib.h File Reference	652
8.21.1 Detailed Description	652
8.22 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/hypercube.h File Reference	652
8.22.1 Detailed Description	653
8.23 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/IndexMap.h File Reference	653
8.23.1 Detailed Description	654
8.24 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/IndexSet.h File Reference	654
8.24.1 Detailed Description	655
8.25 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/intraSlot.h File Reference	655
8.25.1 Detailed Description	655
8.26 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/keys.h File Reference	655
8.26.1 Detailed Description	656
8.26.2 Macro Definition Documentation	656
8.26.2.1 HELIB_KSS_BSGS	656
8.26.2.2 HELIB_KSS_FULL	657
8.26.2.3 HELIB_KSS_MIN	657
8.26.2.4 HELIB_KSS_UNKNOWN	657
8.27 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/key↔ Switching.h File Reference	657
8.27.1 Detailed Description	658

8.27.2 Macro Definition Documentation	658
8.27.2.1 HELIB_KEYSWITCH_MIN_THRESH	659
8.27.2.2 HELIB_KEYSWITCH_THRESH	659
8.28 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/matching.h File Reference	659
8.28.1 Detailed Description	660
8.29 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/matmul.h File Reference	660
8.29.1 Detailed Description	661
8.30 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/multicore.h File Reference	661
8.30.1 Detailed Description	661
8.30.2 Macro Definition Documentation	661
8.30.2.1 HELIB_atomic_long	661
8.30.2.2 HELIB_atomic_ulong	662
8.30.2.3 HELIB_MUTEX_GUARD	662
8.30.2.4 HELIB_MUTEX_TYPE	662
8.31 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/norms.h File Reference	662
8.31.1 Detailed Description	663
8.32 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/NumbTh.h File Reference	663
8.32.1 Detailed Description	668
8.32.2 Macro Definition Documentation	669
8.32.2.1 ERFC_INVERSE_SIZE	669
8.33 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PAIgebra.h File Reference	669
8.33.1 Detailed Description	670
8.33.2 Macro Definition Documentation	670
8.33.2.1 PA_INJECT	670
8.34 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/permutations.h File Reference	671
8.34.1 Detailed Description	672
8.35 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PGFFT.h File Reference	672
8.36 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/polyEval.h File Reference	673
8.36.1 Detailed Description	673
8.37 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PolyMod.h File Reference	673
8.37.1 Detailed Description	674
8.38 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PolyMod↔Ring.h File Reference	674
8.38.1 Detailed Description	674



8.39	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/powerful.h File Reference	675
8.39.1	Detailed Description	675
8.40	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/prime↔ Chain.h File Reference	675
8.40.1	Detailed Description	676
8.41	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PtrMatrix.h File Reference	676
8.41.1	Detailed Description	677
8.42	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PtrVector.h File Reference	677
8.42.1	Detailed Description	678
8.43	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Ptxt.h File Reference	678
8.43.1	Detailed Description	679
8.44	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/random↔ Matrices.h File Reference	679
8.44.1	Detailed Description	680
8.45	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/range.h File Reference	680
8.46	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/recryption.h File Reference	680
8.46.1	Detailed Description	681
8.46.2	Macro Definition Documentation	681
8.46.2.1	HELIB_MIN_CAP_FRAC	681
8.47	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/replicate.h File Reference	681
8.47.1	Detailed Description	682
8.48	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/sample.h File Reference	682
8.48.1	Detailed Description	683
8.49	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/table↔ Lookup.h File Reference	683
8.49.1	Detailed Description	684
8.50	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/timing.h File Reference	684
8.50.1	Detailed Description	685
8.50.2	Macro Definition Documentation	685
8.50.2.1	HELIB_AT	685
8.50.2.2	HELIB_NTIMER_START	685
8.50.2.3	HELIB_NTIMER_STOP	686
8.50.2.4	HELIB_stringify	686
8.50.2.5	HELIB_STRINGIFY	686
8.50.2.6	HELIB_stringify_aux	686
8.50.2.7	HELIB_TIMER_START	686

8.50.2.8	HELIB_TIMER_STOP	686
8.50.2.9	HELIB_TOSTRING	686
8.51	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/zzX.h File Reference	687
8.51.1	Detailed Description	687
8.52	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/BenesNetwork.cpp File Reference	688
8.53	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binaryArith.cpp File Reference	688
8.53.1	Detailed Description	690
8.53.2	Macro Definition Documentation	690
8.53.2.1	BPL_ESTIMATE	690
8.54	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binaryCompare.cpp File Reference	690
8.54.1	Detailed Description	691
8.54.2	Macro Definition Documentation	691
8.54.2.1	BPL_ESTIMATE	691
8.55	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binio.cpp File Reference	691
8.56	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/bluestein.cpp File Reference	692
8.56.1	Macro Definition Documentation	693
8.56.1.1	NEW_BLUE	693
8.57	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/CModulus.cpp File Reference	693
8.58	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Context.cpp File Reference	693
8.59	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Ctxt.cpp File Reference	694
8.60	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/debugging.cpp File Reference	695
8.61	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/DoubleCRT.cpp File Reference	696
8.62	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EaCx.cpp File Reference	696
8.63	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EncryptedArray.cpp File Reference	697
8.64	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/eqtesting.cpp File Reference	698
8.64.1	Detailed Description	699
8.65	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EvalMap.cpp File Reference	699
8.66	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/extractDigits.cpp File Reference	699
8.67	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/fhe_stats.cpp File Reference	700
8.68	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/hypercube.cpp File Reference	700

8.69 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/IndexSet.cpp File Reference	701
8.70 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/intraSlot.cpp File Reference	702
8.71 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/keys.cpp File Reference	702
8.71.1 Macro Definition Documentation	703
8.71.1.1 DECRYPT_ON_PWFL_BASIS	703
8.72 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/keySwitching.cpp File Reference	703
8.72.1 Detailed Description	704
8.72.2 Macro Definition Documentation	704
8.72.2.1 computeParams	705
8.73 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/matching.cpp File Reference	705
8.74 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/matmul.cpp File Reference	705
8.74.1 Macro Definition Documentation	707
8.74.1.1 ALT_MATMUL	707
8.74.1.2 HELIB_BSGS_MUL_THRESH	707
8.74.1.3 HELIB_TRACE_THRESH	707
8.75 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/norms.cpp File Reference	707
8.75.1 Detailed Description	708
8.75.2 Macro Definition Documentation	708
8.75.2.1 USE_HALF_FFT	708
8.75.2.2 USE_QUARTER_FFT	708
8.75.2.3 USE_TWO_QUARTERS	708
8.76 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/NumbTh.cpp File Reference	709
8.77 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/OptimizePermutations.cpp File Reference	711
8.78 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PAlgebra.cpp File Reference	712
8.79 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PermNetwork.cpp File Reference	713
8.80 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/permutations.cpp File Reference	713
8.81 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PGFFT.cpp File Reference	714
8.81.1 Macro Definition Documentation	715
8.81.1.1 fwd_butterfly	715
8.81.1.2 fwd_butterfly0	716
8.81.1.3 inv_butterfly	716
8.81.1.4 inv_butterfly0	716
8.81.1.5 PGFFT_BRC_Q	716

8.81.1.6 PGFFT_BRC_THRESH . . . . .	717
8.81.1.7 PGFFT_FFT_RDUP . . . . .	717
8.81.1.8 PGFFT_NEW_FFT_THRESH . . . . .	717
8.81.1.9 PGFFT_STRATEGY_BLUE . . . . .	717
8.81.1.10 PGFFT_STRATEGY_NULL . . . . .	717
8.81.1.11 PGFFT_STRATEGY_POW2 . . . . .	717
8.81.1.12 PGFFT_STRATEGY_TBLUE . . . . .	717
8.81.1.13 PGFFT_USE_EXPLICIT_MUL . . . . .	717
8.81.1.14 PGFFT_USE_TRUNCATED_BLUE . . . . .	718
8.81.1.15 RESTRICT . . . . .	718
8.82 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/polyEval.cpp File Reference . . . . .	718
8.83 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PolyMod.cpp File Reference . . . . .	718
8.84 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PolyModRing.cpp File Reference . . . . .	719
8.85 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/powerful.cpp File Reference . . . . .	719
8.86 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/primeChain.cpp File Reference . . . . .	720
8.86.1 Detailed Description . . . . .	720
8.87 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Ptxt.cpp File Reference . . . . .	721
8.88 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/randomMatrices.cpp File Reference . . . . .	721
8.88.1 Detailed Description . . . . .	721
8.89 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/recryption.cpp File Reference . . . . .	722
8.89.1 Macro Definition Documentation . . . . .	722
8.89.1.1 DROP_BEFORE_THIN_RECRYPT . . . . .	723
8.89.1.2 THIN_RECRYPT_NLEVELS . . . . .	723
8.90 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/replicate.cpp File Reference . . . . .	723
8.91 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/sample.cpp File Reference . . . . .	724
8.92 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/tableLookup.cpp File Reference . . . . .	725
8.92.1 Detailed Description . . . . .	725
8.93 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_approxNums.cpp File Reference . . . . .	725
8.93.1 Function Documentation . . . . .	726
8.93.1.1 add() . . . . .	726
8.93.1.2 calcMaxDiff() . . . . .	726
8.93.1.3 calcMaxRelDiff() . . . . .	727
8.93.1.4 cx_equals() . . . . .	727
8.93.1.5 debugCompare() . . . . .	727

8.93.1.6	main()	727
8.93.1.7	mul()	727
8.93.1.8	negateVec()	727
8.93.1.9	rotate()	728
8.93.1.10	sub()	728
8.93.1.11	testBasicArith()	728
8.93.1.12	testComplexArith()	728
8.93.1.13	testGeneralOps()	728
8.93.1.14	testRotsNShifts()	729
8.93.2	Variable Documentation	729
8.93.2.1	verbose	729
8.94	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Bin_IO.cpp File Reference	729
8.94.1	Function Documentation	729
8.94.1.1	cleanupFiles() [1/2]	730
8.94.1.2	cleanupFiles() [2/2]	730
8.94.1.3	compareFiles()	730
8.94.1.4	isLittleEndian()	730
8.94.1.5	main()	730
8.95	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_binaryArith.cpp File Reference	730
8.95.1	Function Documentation	731
8.95.1.1	main()	731
8.95.1.2	test15for4()	731
8.95.1.3	testAdd()	731
8.95.1.4	testProduct()	731
8.96	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_binaryCompare.cpp File Reference	732
8.96.1	Function Documentation	732
8.96.1.1	main()	732
8.96.1.2	testCompare()	732
8.97	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_bootstrapping.cpp File Reference	733
8.97.1	Macro Definition Documentation	733
8.97.1.1	INNER_REP	733
8.97.1.2	num_mValues	733
8.97.1.3	OUTER_REP	733
8.97.2	Function Documentation	734
8.97.2.1	main()	734
8.97.2.2	TestIt()	734
8.98	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_EaCx.cpp File Reference	734
8.98.1	Function Documentation	734

8.98.1.1	main()	735
8.98.2	Variable Documentation	735
8.98.2.1	noPrint	735
8.99	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_EvalMap.cpp File Reference	735
8.99.1	Function Documentation	735
8.99.1.1	main()	736
8.99.1.2	TestIt()	736
8.100	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_extract←Digits.cpp File Reference	736
8.100.1	Function Documentation	736
8.100.1.1	main()	736
8.101	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_fatboot.cpp File Reference	737
8.101.1	Macro Definition Documentation	737
8.101.1.1	INNER_REP	737
8.101.1.2	OUTER_REP	737
8.101.2	Function Documentation	738
8.101.2.1	main()	738
8.101.2.2	TestIt()	738
8.101.3	Variable Documentation	738
8.101.3.1	printFlag	738
8.102	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_General.cpp File Reference	738
8.102.1	Macro Definition Documentation	739
8.102.1.1	debugCompare	739
8.102.2	Function Documentation	739
8.102.2.1	main()	739
8.102.2.2	TestIt()	739
8.103	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_intraSlot.cpp File Reference	740
8.103.1	Function Documentation	740
8.103.1.1	main()	740
8.103.2	Variable Documentation	740
8.103.2.1	verbose	740
8.104	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_IO.cpp File Reference	740
8.104.1	Macro Definition Documentation	741
8.104.1.1	N_TESTS	741
8.104.2	Function Documentation	741
8.104.2.1	checkCiphertext()	741
8.104.2.2	main()	741
8.105	/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_matmul.cpp File Reference	741

8.105.1 Function Documentation	742
8.105.1.1 DoTest()	742
8.105.1.2 main()	742
8.105.1.3 TestIt()	742
8.105.2 Variable Documentation	742
8.105.2.1 ks_strategy	743
8.106 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PAlgebra.cpp	
File Reference	743
8.106.1 Function Documentation	743
8.106.1.1 main()	743
8.107 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Permutations.cpp	
File Reference	743
8.107.1 Function Documentation	744
8.107.1.1 main()	744
8.107.1.2 testCtxt()	744
8.107.1.3 testCube()	744
8.108 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PGFFT.cpp File	
Reference	744
8.108.1 Typedef Documentation	745
8.108.1.1 cmplx_t	745
8.108.1.2 lcx	745
8.108.1.3 ldbl	745
8.108.2 Function Documentation	745
8.108.2.1 main()	746
8.109 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PolyEval.cpp	
File Reference	746
8.109.1 Function Documentation	746
8.109.1.1 main()	746
8.109.1.2 testEncrypted()	746
8.109.1.3 testIt()	747
8.110 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Powerful.cpp	
File Reference	747
8.110.1 Function Documentation	747
8.110.1.1 main()	747
8.110.1.2 testHighLvConversion()	747
8.110.1.3 testSimpleConversion()	748
8.111 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PtrVector.cpp	
File Reference	748
8.111.1 Typedef Documentation	749
8.111.1.1 MyPtrMat	749
8.111.1.2 MyPtrMat_ptVec	749
8.111.1.3 MyPtrMat_ptvector	749
8.111.1.4 MyPtrMat_Vec	749
8.111.1.5 MyPtrMat_vector	749

8.111.1.6 MyPtrVec	749
8.111.1.7 MyPtrVec_slice	750
8.111.1.8 MyPtrVec_Vec	750
8.111.1.9 MyPtrVec_VecPt	750
8.111.1.10 MyPtrVec_vector	750
8.111.1.11 MyPtrVec_vectorPt	750
8.111.2 Function Documentation	750
8.111.2.1 compPointers()	750
8.111.2.2 main()	750
8.111.2.3 printPtrVector()	751
8.111.2.4 test1()	751
8.111.2.5 test2()	751
8.111.2.6 test3()	751
8.111.2.7 test4()	751
8.111.2.8 test5()	751
8.111.3 Variable Documentation	752
8.111.3.1 verbose	752
8.112 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Replicate.cpp	
File Reference	752
8.112.1 Function Documentation	752
8.112.1.1 main()	752
8.112.1.2 TestIt()	753
8.113 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_tableLookup.cpp	
File Reference	753
8.113.1 Function Documentation	753
8.113.1.1 main()	753
8.113.1.2 testLookup()	753
8.113.1.3 testWritein()	754
8.114 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_thinboot.cpp	
File Reference	754
8.114.1 Function Documentation	754
8.114.1.1 main()	754
8.114.1.2 TestIt()	754
8.115 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_ThinBootstrapping.cpp	
File Reference	755
8.115.1 Macro Definition Documentation	755
8.115.1.1 num_mValues	755
8.115.1.2 OUTER_REP	755
8.115.2 Function Documentation	755
8.115.2.1 main()	755
8.115.2.2 TestIt()	756
8.116 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_ThinEval↔	
Map.cpp File Reference	756



---

8.116.1 Function Documentation . . . . .	756
8.116.1.1 main() . . . . .	756
8.116.1.2 TestIt() . . . . .	756
8.117 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Timing.cpp File Reference . . . . .	757
8.117.1 Macro Definition Documentation . . . . .	757
8.117.1.1 numTests . . . . .	757
8.117.2 Function Documentation . . . . .	758
8.117.2.1 main() . . . . .	758
8.117.2.2 printTimeData() . . . . .	758
8.117.2.3 rotationAmount() . . . . .	758
8.117.2.4 timeHighLvl() . . . . .	758
8.117.2.5 timeInit() . . . . .	758
8.117.2.6 TimeIt() . . . . .	759
8.117.2.7 timeOps() . . . . .	759
8.118 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/timing.cpp File Reference . . . . .	759
8.119 /Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/zzX.cpp File Reference	760
8.119.1 Detailed Description . . . . .	760



# Chapter 1

## HElib v1.0.2 Documentation

HElib is an open-source ( [Apache License v2.0](#)) software library that implements homomorphic encryption (HE). Currently available schemes are the implementations of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme and the Approximate Number scheme of Cheon-Kim-Kim-Song (CKKS), along with many optimizations to make homomorphic evaluation runs faster, focusing mostly on effective use of the Smart-Vercauteren ciphertext packing techniques and the Gentry-Halevi-Smart optimizations.

Articles that describe some aspects of HElib include:

- A (somewhat outdated) [design document](#), Shai Halevi and Victor Shoup, April 2013.
- [Algorithms in HElib](#), Shai Halevi and Victor Shoup, published in [CRYPTO 2014](#).
- [Bootstrapping for HElib](#), Shai Halevi and Victor Shoup, [EUROCRYPT 2015](#).

Since mid-2018 HElib has been under extensive refactoring for *Reliability*, Robustness & Serviceability\*, *Performance*, and most importantly *Usability* for researchers and developers working on HE and its uses.

HElib supports an **"assembly language for HE"**, providing low-level routines (set, add, multiply, shift, etc.), sophisticated automatic noise management, improved BGV bootstrapping, multi-threading, and also support for Ptxt (plaintext) objects which mimics the functionality of Ctxt (ciphertext) objects. See [changes.md](#) for more details.

HElib is written in C++14 and uses the [NTL](#) mathematical library, over GMP.

HElib is distributed under the terms of the [Apache License v2.0](#).

For code downloads and full installation instructions, visit [HElib GitHub Pages](#).



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">Fft</a>	19
<a href="#">helib</a>	20
<a href="#">helib::FHEglobals</a>	175
<a href="#">NTL</a>	176
<a href="#">std</a>	176



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

helib::add_pa_impl< type > . . . . .	177
helib::AddDAG . . . . .	177
helib::PGFFT::aligned_allocator< T > . . . . .	179
helib::applyPerm_pa_impl< type > . . . . .	181
helib::ArgMap . . . . .	182
helib::ArgMap::ArgProcessor . . . . .	192
helib::BasicAutomorphPrecon . . . . .	193
helib::BGV . . . . .	194
helib::BipartiteGraph . . . . .	194
helib::BlockMatMul1D . . . . .	196
helib::BlockMatMul1D_partial< type > . . . . .	199
helib::BlockMatMul1D_derived< type > . . . . .	197
helib::RandomBlockMatrix< type > . . . . .	581
helib::RandomMultiBlockMatrix< type > . . . . .	587
helib::BlockMatMulFullHelper< type > . . . . .	210
helib::BlockMatMul1D_derived_impl< type > . . . . .	198
helib::BlockMatMul1DExec_construct< type > . . . . .	203
helib::BlockMatMulFullExec_construct< type >::BlockMatMulDimComp . . . . .	203
helib::BlockMatMulFull . . . . .	205
helib::BlockMatMulFull_derived< type > . . . . .	206
helib::RandomFullBlockMatrix< type > . . . . .	583
helib::BlockMatMulFullExec_construct< type > . . . . .	209
helib::CKKS . . . . .	212
helib::Cmodulus . . . . .	213
helib::ConstCubeSlice< T > . . . . .	219
helib::CubeSlice< T > . . . . .	271
helib::ConstMultiplier . . . . .	224
helib::ConstMultiplier_DoubleCRT . . . . .	225
helib::ConstMultiplier_zzX . . . . .	227
helib::ConstMultiplierCache . . . . .	228
helib::Context . . . . .	229
helib::Ctxt . . . . .	240
helib::CubeSignature . . . . .	266
helib::DAGnode . . . . .	273

helib::decode_pa_impl< type > . . . . .	276
helib::deep_clone< X > . . . . .	276
helib::DoubleCRT . . . . .	277
helib::CtxtPart . . . . .	263
helib::DynamicCtxtPowers . . . . .	293
helib::encode_pa_impl< type > . . . . .	295
helib::EncryptedArray . . . . .	295
helib::EncryptedArrayBase . . . . .	302
helib::EncryptedArrayCx . . . . .	315
helib::EncryptedArrayDerived< type > . . . . .	334
helib::equals_pa_impl< type > . . . . .	347
helib::EvalMap . . . . .	348
helib::Exception . . . . .	350
helib::InvalidArgument . . . . .	394
helib::LogicError . . . . .	406
helib::OutOfRangeError . . . . .	436
helib::RuntimeError . . . . .	602
helib::fhe_stats_record . . . . .	352
helib::FHEtimer . . . . .	354
helib::FlowEdge . . . . .	356
helib::frobeniusAutomorph_pa_impl< type > . . . . .	357
helib::FullBinaryTree . . . . .	358
helib::GenDescriptor . . . . .	362
helib::general_range< T > . . . . .	364
helib::GeneralAutomorphPrecon . . . . .	365
helib::GeneralAutomorphPrecon_BSGS . . . . .	366
helib::GeneralAutomorphPrecon_FULL . . . . .	367
helib::GeneralAutomorphPrecon_UNKNOWN . . . . .	368
helib::GeneralBenesNetwork . . . . .	368
helib::GeneratorTrees . . . . .	371
helib::half_FFT . . . . .	375
HighLvlTimingData . . . . .	376
helib::HyperCube< T > . . . . .	378
helib::HyperCube< long > . . . . .	378
helib::ColPerm . . . . .	217
helib::IndexMap< T > . . . . .	384
helib::IndexMap< NTL::vec_long > . . . . .	384
helib::IndexMapInit< T > . . . . .	387
helib::IndexMapInit< NTL::vec_long > . . . . .	387
helib::DoubleCRTHelper . . . . .	292
helib::IndexSet . . . . .	388
invalid_argument . . . . .	
helib::InvalidArgument . . . . .	394
helib::IndexSet::iterator . . . . .	395
helib::general_range< T >::iterator . . . . .	397
helib::KeySwitch . . . . .	398
helib::LabeledEdge . . . . .	403
helib::LabeledVertex . . . . .	404
logic_error . . . . .	
helib::LogicError . . . . .	406
LowLvlTimingData . . . . .	407
helib::MappingData< type > . . . . .	409
helib::MatMul1D . . . . .	411
helib::MatMul1D_partial< type > . . . . .	414
helib::MatMul1D_derived< type > . . . . .	412
helib::RandomMatrix< type > . . . . .	585



helib::RandomMultiMatrix< type > . . . . .	589
helib::MatMulFullHelper< type > . . . . .	426
helib::MatMul1D_derived_impl< type > . . . . .	413
helib::MatMul1DExec_construct< type > . . . . .	418
helib::MatMulFullExec_construct< type >::MatMulDimComp . . . . .	418
helib::MatMulExecBase . . . . .	419
helib::BlockMatMul1DExec . . . . .	200
helib::BlockMatMulFullExec . . . . .	207
helib::MatMul1DExec . . . . .	415
helib::MatMulFullExec . . . . .	423
helib::MatMulFull . . . . .	421
helib::MatMulFull_derived< type > . . . . .	422
helib::RandomFullMatrix< type > . . . . .	584
helib::MatMulFullExec_construct< type > . . . . .	425
helib::ModuliSizes . . . . .	428
helib::mul_BlockMatMul1D_impl< type > . . . . .	431
helib::mul_BlockMatMulFull_impl< type > . . . . .	431
helib::mul_MatMul1D_impl< type > . . . . .	432
helib::mul_MatMulFull_impl< type > . . . . .	432
helib::mul_pa_impl< type > . . . . .	433
MyClass . . . . .	433
helib::negate_pa_impl< type > . . . . .	434
OtherTimingData . . . . .	434
out_of_range . . . . .	
helib::OutOfRangeError . . . . .	436
helib::PAlgebra . . . . .	438
helib::PAlgebraMod . . . . .	447
helib::PAlgebraModBase . . . . .	450
helib::PAlgebraModCx . . . . .	453
helib::PAlgebraModDerived< type > . . . . .	455
helib::PermNetLayer . . . . .	461
helib::PermNetwork . . . . .	463
helib::PGFFT . . . . .	465
helib::PlaintextArray . . . . .	468
helib::PlaintextArrayBase . . . . .	469
helib::PlaintextArrayDerived< type > . . . . .	470
helib::PolyMod . . . . .	471
helib::PolyModRing . . . . .	493
helib::PowerfulConversion . . . . .	498
helib::PowerfulDCRT . . . . .	500
helib::PowerfulTranslationIndexes . . . . .	502
helib::PrimeGenerator . . . . .	505
helib::print_pa_impl< type > . . . . .	506
helib::PtrMatrix< T > . . . . .	507
helib::PtrMatrix_PtPtrVector< T > . . . . .	509
helib::PtrMatrix_ptVec< T > . . . . .	510
helib::PtrMatrix_ptvector< T > . . . . .	512
helib::PtrMatrix_Vec< T > . . . . .	514
helib::PtrMatrix_vector< T > . . . . .	517
helib::PtrVector< T > . . . . .	519
helib::PtrVector_Singleton< T > . . . . .	521
helib::PtrVector_slice< T > . . . . .	523
helib::PtrVector_VecPt< T > . . . . .	525
helib::PtrVector_VecT< T > . . . . .	527
helib::PtrVector_vectorPt< T > . . . . .	529
helib::PtrVector_vectorT< T > . . . . .	531

helib::Ptxt . . . . .	533
helib::PubKey . . . . .	568
helib::SecKey . . . . .	605
helib::PubKeyHack . . . . .	578
helib::quarter_FFT . . . . .	580
helib::random_pa_impl< type > . . . . .	581
helib::RandomState . . . . .	591
helib::RecryptData . . . . .	592
helib::ThinRecryptData . . . . .	620
helib::replicate_pa_impl< type > . . . . .	597
helib::ReplicateHandler . . . . .	598
helib::ExplicitReplicator . . . . .	351
ReplicateDummy . . . . .	597
ReplicateTester . . . . .	599
helib::rotate_pa_impl< type > . . . . .	601
runtime_error	
helib::RuntimeError . . . . .	602
helib::ScratchCell . . . . .	603
helib::shallow_clone< X > . . . . .	611
helib::shift_pa_impl< type > . . . . .	612
helib::SKHandle . . . . .	612
StopReplicate . . . . .	616
helib::sub_pa_impl< type > . . . . .	616
helib::SubDimension . . . . .	617
helib::ThinEvalMap . . . . .	619
TimingData . . . . .	621
helib::TreeNode< T > . . . . .	623
ZZ_pXModulus	
helib::ZZ_pXModulus1 . . . . .	626
helib::zz_pXModulus1 . . . . .	627

## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">helib::add_pa_impl&lt; type &gt;</a>	177
<a href="#">helib::AddDAG</a>	
A class representing the logic of the order of bit products when adding two integers	177
<a href="#">helib::PGFFT::aligned_allocator&lt; T &gt;</a>	179
<a href="#">helib::applyPerm_pa_impl&lt; type &gt;</a>	181
<a href="#">helib::ArgMap</a>	
Basic class for arg parsing. Example use:	182
<a href="#">helib::ArgMap::ArgProcessor</a>	192
<a href="#">helib::BasicAutomorphPrecon</a>	
Pre-computation to speed many automorphism on the same ciphertext	193
<a href="#">helib::BGV</a>	
Type for <a href="#">BGV</a> scheme, to be used as template parameter	194
<a href="#">helib::BipartiteGraph</a>	
A bipartite flow graph	194
<a href="#">helib::BlockMatMul1D</a>	196
<a href="#">helib::BlockMatMul1D_derived&lt; type &gt;</a>	197
<a href="#">helib::BlockMatMul1D_derived_impl&lt; type &gt;</a>	198
<a href="#">helib::BlockMatMul1D_partial&lt; type &gt;</a>	199
<a href="#">helib::BlockMatMul1DExec</a>	200
<a href="#">helib::BlockMatMul1DExec_construct&lt; type &gt;</a>	203
<a href="#">helib::BlockMatMulFullExec_construct&lt; type &gt;::BlockMatMulDimComp</a>	203
<a href="#">helib::BlockMatMulFull</a>	205
<a href="#">helib::BlockMatMulFull_derived&lt; type &gt;</a>	206
<a href="#">helib::BlockMatMulFullExec</a>	207
<a href="#">helib::BlockMatMulFullExec_construct&lt; type &gt;</a>	209
<a href="#">helib::BlockMatMulFullHelper&lt; type &gt;</a>	210
<a href="#">helib::CKKS</a>	
Type for <a href="#">CKKS</a> scheme, to be used as template parameter	212
<a href="#">helib::Cmodulus</a>	
Provides FFT and iFFT routines modulo a single-precision prime	213
<a href="#">helib::ColPerm</a>	
Permuting a single dimension (column) of a hypercube	217
<a href="#">helib::ConstCubeSlice&lt; T &gt;</a>	
A constant lower-dimension slice of a hypercube	219
<a href="#">helib::ConstMultiplier</a>	224

<a href="#">helib::ConstMultiplier_DoubleCRT</a>	225
<a href="#">helib::ConstMultiplier_zzX</a>	227
<a href="#">helib::ConstMultiplierCache</a>	228
<a href="#">helib::Context</a>	
Maintaining the parameters	229
<a href="#">helib::Ctxt</a>	
A <a href="#">Ctxt</a> object holds a single ciphertext	240
<a href="#">helib::CtxtPart</a>	
One entry in a ciphertext <code>std::vector</code>	263
<a href="#">helib::CubeSignature</a>	
Holds a vector of dimensions for a hypercube and some additional data	266
<a href="#">helib::CubeSlice&lt; T &gt;</a>	
A lower-dimension slice of a hypercube	271
<a href="#">helib::DAGnode</a>	
A node in an addition-DAG structure	273
<a href="#">helib::decode_pa_impl&lt; type &gt;</a>	276
<a href="#">helib::deep_clone&lt; X &gt;</a>	
Deep copy: initialize with clone	276
<a href="#">helib::DoubleCRT</a>	
Implementing polynomials (elements in the ring $R_Q$ ) in double-CRT form	277
<a href="#">helib::DoubleCRTHelper</a>	
A helper class to enforce consistency within an <a href="#">DoubleCRTHelper</a> object	292
<a href="#">helib::DynamicCtxtPowers</a>	
Store powers of $X$ , compute them dynamically as needed	293
<a href="#">helib::encode_pa_impl&lt; type &gt;</a>	295
<a href="#">helib::EncryptedArray</a>	
A simple wrapper for a smart pointer to an <a href="#">EncryptedArrayBase</a> . This is the interface that higher-level code should use	295
<a href="#">helib::EncryptedArrayBase</a>	
Virtual class for data-movement operations on arrays of slots	302
<a href="#">helib::EncryptedArrayCx</a>	
A different derived class to be used for the approximate-numbers scheme	315
<a href="#">helib::EncryptedArrayDerived&lt; type &gt;</a>	
Derived concrete implementation of <a href="#">EncryptedArrayBase</a>	334
<a href="#">helib::equals_pa_impl&lt; type &gt;</a>	347
<a href="#">helib::EvalMap</a>	
Class that provides the functionality for the linear transforms used in bootstrapping. The constructor is invoked with three arguments:	348
<a href="#">helib::Exception</a>	
Base class that other HELib exception classes inherit from	350
<a href="#">helib::ExplicitReplicator</a>	
An implementation of <a href="#">ReplicateHandler</a> that explicitly returns all the replicated ciphertexts in one big vector	351
<a href="#">helib::fhe_stats_record</a>	352
<a href="#">helib::FHEtimer</a>	
A simple class to accumulate time	354
<a href="#">helib::FlowEdge</a>	
An edge in a flow graph	356
<a href="#">helib::frobeniusAutomorph_pa_impl&lt; type &gt;</a>	357
<a href="#">helib::FullBinaryTree</a>	
A simple implementation of full binary trees (each non-leaf has 2 children)	358
<a href="#">helib::GenDescriptor</a>	
A minimal description of a generator for the purpose of building tree	362
<a href="#">helib::general_range&lt; T &gt;</a>	364
<a href="#">helib::GeneralAutomorphPrecon</a>	365
<a href="#">helib::GeneralAutomorphPrecon_BSGS</a>	366
<a href="#">helib::GeneralAutomorphPrecon_FULL</a>	367
<a href="#">helib::GeneralAutomorphPrecon_UNKNOWN</a>	368

<a href="#">helib::GeneralBenesNetwork</a>	
Implementation of generalized Benes Permutation Network	368
<a href="#">helib::GeneratorTrees</a>	
A std::vector of generator trees, one per generator in $Z_{m^*}(p)$	371
<a href="#">helib::half_FFT</a>	375
<a href="#">HighLvlTimingData</a>	376
<a href="#">helib::HyperCube&lt; T &gt;</a>	
A multi-dimensional cube	378
<a href="#">helib::IndexMap&lt; T &gt;</a>	
IndexMap<T> implements a generic map indexed by a dynamic index set	384
<a href="#">helib::IndexMapInit&lt; T &gt;</a>	
Initializing elements in an <a href="#">IndexMap</a>	387
<a href="#">helib::IndexSet</a>	
A dynamic set of non-negative integers	388
<a href="#">helib::InvalidArgument</a>	
Inherits from <a href="#">Exception</a> and std::invalid_argument	394
<a href="#">helib::IndexSet::iterator</a>	395
<a href="#">helib::general_range&lt; T &gt;::iterator</a>	397
<a href="#">helib::KeySwitch</a>	
Key-switching matrices	398
<a href="#">helib::LabeledEdge</a>	
A generic directed edge in a graph with some labels	403
<a href="#">helib::LabeledVertex</a>	
A generic node in a graph with some labels	404
<a href="#">helib::LogicError</a>	
Inherits from <a href="#">Exception</a> and std::logic_error	406
<a href="#">LowLvlTimingData</a>	407
<a href="#">helib::MappingData&lt; type &gt;</a>	
Auxiliary structure to support encoding/decoding slots	409
<a href="#">helib::MatMul1D</a>	411
<a href="#">helib::MatMul1D_derived&lt; type &gt;</a>	412
<a href="#">helib::MatMul1D_derived_impl&lt; type &gt;</a>	413
<a href="#">helib::MatMul1D_partial&lt; type &gt;</a>	414
<a href="#">helib::MatMul1DExec</a>	415
<a href="#">helib::MatMul1DExec_construct&lt; type &gt;</a>	418
<a href="#">helib::MatMulFullExec_construct&lt; type &gt;::MatMulDimComp</a>	418
<a href="#">helib::MatMulExecBase</a>	419
<a href="#">helib::MatMulFull</a>	421
<a href="#">helib::MatMulFull_derived&lt; type &gt;</a>	422
<a href="#">helib::MatMulFullExec</a>	423
<a href="#">helib::MatMulFullExec_construct&lt; type &gt;</a>	425
<a href="#">helib::MatMulFullHelper&lt; type &gt;</a>	426
<a href="#">helib::ModuliSizes</a>	
A helper class to map required modulo-sizes to primeSets	428
<a href="#">helib::mul_BlockMatMul1D_impl&lt; type &gt;</a>	431
<a href="#">helib::mul_BlockMatMulFull_impl&lt; type &gt;</a>	431
<a href="#">helib::mul_MatMul1D_impl&lt; type &gt;</a>	432
<a href="#">helib::mul_MatMulFull_impl&lt; type &gt;</a>	432
<a href="#">helib::mul_pa_impl&lt; type &gt;</a>	433
<a href="#">MyClass</a>	433
<a href="#">helib::negate_pa_impl&lt; type &gt;</a>	434
<a href="#">OtherTimingData</a>	434
<a href="#">helib::OutOfRangeError</a>	
Inherits from <a href="#">Exception</a> and std::out_of_range	436
<a href="#">helib::PALgebra</a>	
The structure of $(Z/mZ)^* / (p)$	438
<a href="#">helib::PALgebraMod</a>	
The structure of $Z[X]/(\Phi_m(X), p)$	447

<a href="#">helib::PAlgebraModBase</a>	Virtual base class for <a href="#">PAlgebraMod</a> . . . . .	450
<a href="#">helib::PAlgebraModCx</a> . . . . .		453
<a href="#">helib::PAlgebraModDerived&lt; type &gt;</a>	A concrete instantiation of the virtual class . . . . .	455
<a href="#">helib::PermNetLayer</a>	The information needed to apply one layer of a permutation network . . . . .	461
<a href="#">helib::PermNetwork</a>	A full permutation network . . . . .	463
<a href="#">helib::PGFFT</a> . . . . .		465
<a href="#">helib::PlaintextArray</a> . . . . .		468
<a href="#">helib::PlaintextArrayBase</a> . . . . .		469
<a href="#">helib::PlaintextArrayDerived&lt; type &gt;</a> . . . . .		470
<a href="#">helib::PolyMod</a>	An object that contains an <code>NTL::ZZX</code> polynomial along with a coefficient modulus <code>p2r</code> and a polynomial modulus <code>G</code> . . . . .	471
<a href="#">helib::PolyModRing</a>	Lightweight type for describing the structure of a single slot of the plaintext space . . . . .	493
<a href="#">helib::PowerfulConversion</a>	Conversion between powerful representation in <code>R_m/(q)</code> and <code>zz_pX</code> . . . . .	498
<a href="#">helib::PowerfulDCRT</a>	Conversion between powerful representation, <a href="#">DoubleCRT</a> , and <code>ZZX</code> . . . . .	500
<a href="#">helib::PowerfulTranslationIndexes</a>	Holds index tables for translation between powerful and <code>zz_pX</code> . . . . .	502
<a href="#">helib::PrimeGenerator</a> . . . . .		505
<a href="#">helib::print_pa_impl&lt; type &gt;</a> . . . . .		506
<a href="#">helib::PtrMatrix&lt; T &gt;</a>	An abstract class for an array of <code>PtrVectors</code> . . . . .	507
<a href="#">helib::PtrMatrix_PtPtrVector&lt; T &gt;</a>	An implementation of <a href="#">PtrMatrix</a> using <code>vector&lt; PtrVector&lt;T&gt;*&gt;</code> . . . . .	509
<a href="#">helib::PtrMatrix_ptVec&lt; T &gt;</a>	An implementation of <a href="#">PtrMatrix</a> using <code>Vec&lt; Vec&lt;T&gt;*&gt;</code> . . . . .	510
<a href="#">helib::PtrMatrix_ptvector&lt; T &gt;</a>	An implementation of <a href="#">PtrMatrix</a> using <code>vector&lt; vector&lt;T&gt;*&gt;</code> . . . . .	512
<a href="#">helib::PtrMatrix_Vec&lt; T &gt;</a>	An implementation of <a href="#">PtrMatrix</a> using <code>Vec&lt; Vec&lt;T&gt; &gt;</code> . . . . .	514
<a href="#">helib::PtrMatrix_vector&lt; T &gt;</a>	An implementation of <a href="#">PtrMatrix</a> using <code>vector&lt; vector&lt;T&gt; &gt;</code> . . . . .	517
<a href="#">helib::PtrVector&lt; T &gt;</a>	Abstract class for an array of objects . . . . .	519
<a href="#">helib::PtrVector_Singleton&lt; T &gt;</a>	An implementation of <a href="#">PtrVector</a> from a single <code>T</code> object . . . . .	521
<a href="#">helib::PtrVector_slice&lt; T &gt;</a>	An implementation of <a href="#">PtrVector</a> as a slice of another <a href="#">PtrVector</a> . . . . .	523
<a href="#">helib::PtrVector_VecPt&lt; T &gt;</a>	An implementation of <a href="#">PtrVector</a> using <code>Vec&lt;T*&gt;</code> . . . . .	525
<a href="#">helib::PtrVector_VecT&lt; T &gt;</a>	An implementation of <a href="#">PtrVector</a> using <code>Vec&lt;T&gt;</code> . . . . .	527
<a href="#">helib::PtrVector_vectorPt&lt; T &gt;</a>	An implementation of <a href="#">PtrVector</a> using <code>vector&lt;T*&gt;</code> . . . . .	529
<a href="#">helib::PtrVector_vectorT&lt; T &gt;</a>	An implementation of <a href="#">PtrVector</a> using <code>vector&lt;T&gt;</code> . . . . .	531
<a href="#">helib::Ptxt</a>	An object that mimics the functionality of the <code>Ctxt</code> object, and acts as a convenient entry point for inputting/encoding data which is to be encrypted . . . . .	533
<a href="#">helib::PubKey</a>	The public key . . . . .	568
<a href="#">helib::PubKeyHack</a> . . . . .		578

<a href="#">helib::quarter_FFT</a>	580
<a href="#">helib::random_pa_impl&lt; type &gt;</a>	581
<a href="#">helib::RandomBlockMatrix&lt; type &gt;</a>	581
<a href="#">helib::RandomFullBlockMatrix&lt; type &gt;</a>	583
<a href="#">helib::RandomFullMatrix&lt; type &gt;</a>	584
<a href="#">helib::RandomMatrix&lt; type &gt;</a>	585
<a href="#">helib::RandomMultiBlockMatrix&lt; type &gt;</a>	587
<a href="#">helib::RandomMultiMatrix&lt; type &gt;</a>	589
<a href="#">helib::RandomState</a>	
Facility for "restoring" the <a href="#">NTL</a> PRG state	591
<a href="#">helib::RecryptData</a>	
A structure to hold decryption-related data inside the <a href="#">Context</a>	592
<a href="#">helib::replicate_pa_impl&lt; type &gt;</a>	597
<a href="#">ReplicateDummy</a>	597
<a href="#">helib::ReplicateHandler</a>	
An abstract class to handle call-backs to get the output of replicate	598
<a href="#">ReplicateTester</a>	599
<a href="#">helib::rotate_pa_impl&lt; type &gt;</a>	601
<a href="#">helib::RuntimeError</a>	
Inherits from <a href="#">Exception</a> and <code>std::runtime_error</code>	602
<a href="#">helib::ScratchCell</a>	
A class to help manage the allocation of temporary <a href="#">Ctxt</a> objects	603
<a href="#">helib::SecKey</a>	
The secret key	605
<a href="#">helib::shallow_clone&lt; X &gt;</a>	
Shallow copy: initialize with copy constructor	611
<a href="#">helib::shift_pa_impl&lt; type &gt;</a>	612
<a href="#">helib::SKHandle</a>	
A handle, describing the secret-key element that "matches" a part, of the form $s^r(X^t)$	612
<a href="#">StopReplicate</a>	616
<a href="#">helib::sub_pa_impl&lt; type &gt;</a>	616
<a href="#">helib::SubDimension</a>	
A node in a tree relative to some generator	617
<a href="#">helib::ThinEvalMap</a>	
Class that provides the functionality for the linear transforms used in "thin" bootstrapping, where slots are assumed to contain constants. The interface is exactly the same as for <a href="#">EvalMap</a> , except that the constructor does not have a <code>normal_basis</code> parameter	619
<a href="#">helib::ThinRecryptData</a>	
Same as above, but for "thin" bootstrapping, where the slots are assumed to contain constants	620
<a href="#">TimingData</a>	621
<a href="#">helib::TreeNode&lt; T &gt;</a>	
A node in a full binary tree	623
<a href="#">helib::ZZ_pXModulus1</a>	
Placeholder for <code>pXModulus</code> ...no optimizations	626
<a href="#">helib::zz_pXModulus1</a>	
Auxiliary classes to facilitate faster reduction mod $\Phi_m(X)$ when the input has degree less than <code>m</code>	627





## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">apiAttributes.h</a>	629
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">ArgMap.h</a>	
Easier arg parsing	630
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">assertions.h</a>	
Various assertion functions for use within HElib. These are meant as a replacement for C-style asserts, which (undesirably) exit the process without giving the opportunity to clean up and shut down gracefully. Instead, these functions will throw an exception if their conditions are violated	630
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">binaryArith.h</a>	
Implementing integer addition, multiplication in binary representation	631
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">binaryCompare.h</a>	
Implementing integer comparison in binary representation	633
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">binio.h</a>	633
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">bluestein.h</a>	
Declaration of BluesteinFFT(x, n, root, powers, powers_aux, Rb):	637
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">clonedPtr.h</a>	
Implementation of smart pointers with "deep cloning" semantics	637
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">CModulus.h</a>	
Supports forward and backward length-m FFT transformations	639
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">Context.h</a>	
Keeps the parameters of an instance of the cryptosystem	639
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">CtPtrs.h</a>	
Unified interface for vector of pointers to ciphertexts	640
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">Ctxt.h</a>	
Declarations of a BGV-type ciphertext and key-switching matrices	641
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">debugging.h</a>	
Debugging utilities	643
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">DoubleCRT.h</a>	
Integer polynomials (elements in the ring $R_Q$ ) in double-CRT form	645
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">EncryptedArray.h</a>	
Data-movement operations on encrypted arrays of slots	646
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">EvalMap.h</a>	
Implementing the decryption linear transformations	648
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">exceptions.h</a>	
Various HElib-specific exception types	649
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/ <a href="#">FHE.h</a>	
Deprecated entry point header for HElib (legacy alias of <a href="#">helib.h</a> )	650

/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/fhe_stats.h . . .	650
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/helib.h	
Entry point header for HELib . . . . .	652
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/hypercube.h	
Hypercubes and their slices . . . . .	652
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/IndexedMap.h	
Implementation of a map indexed by a dynamic set of integers . . . . .	653
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/IndexedSet.h	
A dynamic set of integers . . . . .	654
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/intraSlot.h	
Packing/unpacking of mod-p integers in $\text{GF}(p^d)$ slots . . . . .	655
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/keys.h	
• Declaration of public key	655
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/keySwitching.h	
• Declaration of key switching matrices	657
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/matching.h	
Classes and functions for max-flow in a generic graph . . . . .	659
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/matmul.h	
Some matrix / linear algebra stuff . . . . .	660
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/multicore.h	
Support for multi-threaded implementations . . . . .	661
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/norms.h . . . .	662
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/NumbTh.h	
Miscellaneous utility functions . . . . .	663
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PAlgebra.h	
Declarations of the classes PAlgebra . . . . .	669
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/permutations.h	
Permutations over Hypercubes and their slices . . . . .	671
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PGFFT.h . . . .	672
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/polyEval.h	
Homomorphic Polynomial Evaluation . . . . .	673
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PolyMod.h	
Underlying slot type structure of BGV ptxts . . . . .	673
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PolyModRing.h	
Definition of the plaintext slot algebraic ring . . . . .	674
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/powerful.h	
The "powerful basis" of a cyclotomic ring . . . . .	675
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/primeChain.h	
Handling the chain of moduli . . . . .	675
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PtrMatrix.h	
Convenience class templates providing a unified interface for a matrix of objects, returning pointers to these objects . . . . .	676
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/PtrVector.h	
Convenience class templates providing a unified interface for a collection of objects, returning pointers to these objects . . . . .	677
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Ptxt.h	
Plaintext object parameterised on CKKS and BGV schemes. Also contains definition of CKKS and BGV structs . . . . .	678
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/randomMatrices.h	
Implementation of random matrices of various forms, used for testing . . . . .	679
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/range.h . . . .	680
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/recryption.h	
Define some data structures to hold recryption data . . . . .	680

/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/replicate.h	
Procedures for replicating a ciphertext slot across a full ciphertext	681
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/sample.h	682
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/tableLookup.h	
Code for homomorphic table lookup and fixed-point functions	683
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/timing.h	
Utility functions for measuring time	684
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/zzX.h	687
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/BenesNetwork.cpp	688
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binaryArith.cpp	
Implementing integer addition, multiplication in binary representation	688
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binaryCompare.cpp	
Implementing integer comparison in binary representation	690
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binio.cpp	691
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/bluestein.cpp	692
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/CModulus.cpp	693
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Context.cpp	693
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Ctxt.cpp	694
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/debugging.cpp	695
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/DoubleCRT.cpp	696
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EaCx.cpp	696
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EncryptedArray.cpp	697
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/eqtesting.cpp	
Useful functions for equality testing..	698
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EvalMap.cpp	699
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/extractDigits.cpp	699
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/fhe_stats.cpp	700
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/hypercube.cpp	700
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/IndexSet.cpp	701
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/intraSlot.cpp	702
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/keys.cpp	702
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/keySwitching.cpp	
A few strategies for generating key-switching matrices	703
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/matching.cpp	705
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/matmul.cpp	705
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/norms.cpp	707
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/NumbTh.cpp	709
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/OptimizePermutations.cpp	711
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PAlgebra.cpp	712
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PermNetwork.cpp	713
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/permutations.cpp	713
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PGFFT.cpp	714
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/polyEval.cpp	718
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PolyMod.cpp	718
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/PolyModRing.cpp	719
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/powerful.cpp	719
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/primeChain.cpp	
Handling the chain of moduli	720
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Ptxt.cpp	721
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/randomMatrices.cpp	
Implementation of random matrices of various forms build functions, used for testing	721
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/recryption.cpp	722
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/replicate.cpp	723
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/sample.cpp	724
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/tableLookup.cpp	
Code for homomorphic table lookup and fixed-point functions	725
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_approxNums.cpp	725
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Bin_IO.cpp	729

/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_binaryArith.cpp . . .	730
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_binaryCompare.cpp .	732
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_bootstrapping.cpp . .	733
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_EaCx.cpp . . . . .	734
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_EvalMap.cpp . . . .	735
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_extractDigits.cpp . . .	736
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_fatboot.cpp . . . . .	737
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_General.cpp . . . . .	738
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_intraSlot.cpp . . . . .	740
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_IO.cpp . . . . .	740
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_matmul.cpp . . . . .	741
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PAlgebra.cpp . . . . .	743
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Permutations.cpp . .	743
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PGFFT.cpp . . . . .	744
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PolyEval.cpp . . . .	746
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Powerful.cpp . . . .	747
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_PtrVector.cpp . . . .	748
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Replicate.cpp . . . .	752
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_tableLookup.cpp . . .	753
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_thinboot.cpp . . . . .	754
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_ThinBootstrapping.cpp	
755	
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_ThinEvalMap.cpp . .	756
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Test_Timing.cpp . . . . .	757
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/timing.cpp . . . . .	759
/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/zzX.cpp . . . . .	760

## Chapter 6

# Namespace Documentation

### 6.1 Fft Namespace Reference

#### Functions

- void `transform` (std::vector< `lcx` > &vec)
- void `inverseTransform` (std::vector< `lcx` > &vec)
- void `transformRadix2` (std::vector< `lcx` > &vec)
- void `transformBluestein` (std::vector< `lcx` > &vec)
- void `convolve` (const std::vector< `lcx` > &vecx, const std::vector< `lcx` > &vecy, std::vector< `lcx` > &vecout)

#### 6.1.1 Function Documentation

##### 6.1.1.1 `convolve()`

```
void Fft::convolve (
    const std::vector< lcx > & vecx,
    const std::vector< lcx > & vecy,
    std::vector< lcx > & vecout )
```

##### 6.1.1.2 `inverseTransform()`

```
void Fft::inverseTransform (
    std::vector< lcx > & vec )
```

### 6.1.1.3 transform()

```
void Fft::transform (
    std::vector< lcx > & vec )
```

### 6.1.1.4 transformBluestein()

```
void Fft::transformBluestein (
    std::vector< lcx > & vec )
```

### 6.1.1.5 transformRadix2()

```
void Fft::transformRadix2 (
    std::vector< lcx > & vec )
```

## 6.2 helib Namespace Reference

### Namespaces

- [FHEglobals](#)

### Classes

- class [add\\_pa\\_impl](#)
- class [AddDAG](#)

*A class representing the logic of the order of bit products when adding two integers.*
- class [applyPerm\\_pa\\_impl](#)
- class [ArgMap](#)

*Basic class for arg parsing. Example use:*
- class [BasicAutomorphPrecon](#)

*Pre-computation to speed many automorphism on the same ciphertext.*
- struct [BGV](#)

*Type for [BGV](#) scheme, to be used as template parameter.*
- class [BipartiteGraph](#)

*A bipartite flow graph.*
- class [BlockMatMul1D](#)
- class [BlockMatMul1D\\_derived](#)
- struct [BlockMatMul1D\\_derived\\_impl](#)
- class [BlockMatMul1D\\_partial](#)
- class [BlockMatMul1DExec](#)
- struct [BlockMatMul1DExec\\_construct](#)
- class [BlockMatMulFull](#)
- class [BlockMatMulFull\\_derived](#)
- class [BlockMatMulFullExec](#)

- struct [BlockMatMulFullExec\\_construct](#)
- class [BlockMatMulFullHelper](#)
- struct [CKKS](#)
  - Type for [CKKS](#) scheme, to be used as template parameter.*
- class [Cmodulus](#)
  - Provides FFT and iFFT routines modulo a single-precision prime.*
- class [ColPerm](#)
  - Permuting a single dimension (column) of a hypercube.*
- class [ConstCubeSlice](#)
  - A constant lower-dimension slice of a hypercube.*
- struct [ConstMultiplier](#)
- struct [ConstMultiplier\\_DoubleCRT](#)
- struct [ConstMultiplier\\_zzX](#)
- struct [ConstMultiplierCache](#)
- class [Context](#)
  - Maintaining the parameters.*
- class [Ctxt](#)
  - A [Ctxt](#) object holds a single ciphertext.*
- class [CtxtPart](#)
  - One entry in a ciphertext `std::vector`.*
- class [CubeSignature](#)
  - Holds a vector of dimensions for a hypercube and some additional data.*
- class [CubeSlice](#)
  - A lower-dimension slice of a hypercube.*
- class [DAGnode](#)
  - A node in an addition-DAG structure.*
- class [decode\\_pa\\_impl](#)
- class [deep\\_clone](#)
  - Deep copy: initialize with clone.*
- class [DoubleCRT](#)
  - Implementing polynomials (elements in the ring  $R_Q$ ) in double-CRT form.*
- class [DoubleCRTHelper](#)
  - A helper class to enforce consistency within an [DoubleCRTHelper](#) object.*
- class [DynamicCtxtPowers](#)
  - Store powers of  $X$ , compute them dynamically as needed.*
- class [encode\\_pa\\_impl](#)
- class [EncryptedArray](#)
  - A simple wrapper for a smart pointer to an [EncryptedArrayBase](#). This is the interface that higher-level code should use.*
- class [EncryptedArrayBase](#)
  - virtual class for data-movement operations on arrays of slots*
- class [EncryptedArrayCx](#)
  - A different derived class to be used for the approximate-numbers scheme.*
- class [EncryptedArrayDerived](#)
  - Derived concrete implementation of [EncryptedArrayBase](#).*
- class [equals\\_pa\\_impl](#)
- class [EvalMap](#)
  - Class that provides the functionality for the linear transforms used in bootstrapping. The constructor is invoked with three arguments:*
- class [Exception](#)
  - Base class that other HELib exception classes inherit from.*
- class [ExplicitReplicator](#)

- An implementation of [ReplicateHandler](#) that explicitly returns all the replicated ciphertexts in one big vector.
- struct [fhe\\_stats\\_record](#)
- class [FHEtimer](#)
  - A simple class to accumulate time.
- class [FlowEdge](#)
  - An edge in a flow graph.
- class [frobeniusAutomorph\\_pa\\_impl](#)
- class [FullBinaryTree](#)
  - A simple implementation of full binary trees (each non-leaf has 2 children)
- class [GenDescriptor](#)
  - A minimal description of a generator for the purpose of building tree.
- class [general\\_range](#)
- class [GeneralAutomorphPrecon](#)
- class [GeneralAutomorphPrecon\\_BSGS](#)
- class [GeneralAutomorphPrecon\\_FULL](#)
- class [GeneralAutomorphPrecon\\_UNKNOWN](#)
- class [GeneralBenesNetwork](#)
  - Implementation of generalized Benes Permutation Network.
- class [GeneratorTrees](#)
  - A `std::vector` of generator trees, one per generator in  $Z_{m^*}/(p)$
- struct [half\\_FFT](#)
- class [HyperCube](#)
  - A multi-dimensional cube.
- class [IndexMap](#)
  - `IndexMap<T>` implements a generic map indexed by a dynamic index set.
- class [IndexMapInit](#)
  - Initializing elements in an [IndexMap](#).
- class [IndexSet](#)
  - A dynamic set of non-negative integers.
- class [InvalidArgument](#)
  - Inherits from [Exception](#) and `std::invalid_argument`.
- class [KeySwitch](#)
  - Key-switching matrices.
- class [LabeledEdge](#)
  - A generic directed edge in a graph with some labels.
- class [LabeledVertex](#)
  - A generic node in a graph with some labels.
- class [LogicError](#)
  - Inherits from [Exception](#) and `std::logic_error`.
- class [MappingData](#)
  - Auxiliary structure to support encoding/decoding slots.
- class [MatMul1D](#)
- class [MatMul1D\\_derived](#)
- struct [MatMul1D\\_derived\\_impl](#)
- class [MatMul1D\\_partial](#)
- class [MatMul1DExec](#)
- struct [MatMul1DExec\\_construct](#)
- class [MatMulExecBase](#)
- class [MatMulFull](#)
- class [MatMulFull\\_derived](#)
- class [MatMulFullExec](#)
- struct [MatMulFullExec\\_construct](#)



- class [MatMulFullHelper](#)
- class [ModuliSizes](#)
  - A helper class to map required modulo-sizes to primeSets.*
- struct [mul\\_BlockMatMul1D\\_impl](#)
- struct [mul\\_BlockMatMulFull\\_impl](#)
- struct [mul\\_MatMul1D\\_impl](#)
- struct [mul\\_MatMulFull\\_impl](#)
- class [mul\\_pa\\_impl](#)
- class [negate\\_pa\\_impl](#)
- class [OutOfRangeError](#)
  - Inherits from [Exception](#) and `std::out_of_range`.*
- class [PAlgebra](#)
  - The structure of  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$*
- class [PAlgebraMod](#)
  - The structure of  $\mathbb{Z}[X]/(\text{Phi}_m(X), p)$*
- class [PAlgebraModBase](#)
  - Virtual base class for [PAlgebraMod](#).*
- class [PAlgebraModCx](#)
- class [PAlgebraModDerived](#)
  - A concrete instantiation of the virtual class.*
- class [PermNetLayer](#)
  - The information needed to apply one layer of a permutation network.*
- class [PermNetwork](#)
  - A full permutation network.*
- class [PGFFT](#)
- class [PlaintextArray](#)
- class [PlaintextArrayBase](#)
- class [PlaintextArrayDerived](#)
- class [PolyMod](#)
  - An object that contains an  $NTL : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$  polynomial along with a coefficient modulus  $p^{2r}$  and a polynomial modulus  $G$ .*
- struct [PolyModRing](#)
  - Lightweight type for describing the structure of a single slot of the plaintext space.*
- class [PowerfulConversion](#)
  - Conversion between powerful representation in  $R_m/(q)$  and  $zz\_pX$ .*
- class [PowerfulDCRT](#)
  - Conversion between powerful representation, [DoubleCRT](#), and  $ZZX$ .*
- class [PowerfulTranslationIndexes](#)
  - Holds index tables for translation between powerful and  $zz\_pX$ .*
- struct [PrimeGenerator](#)
- class [print\\_pa\\_impl](#)
- struct [PtrMatrix](#)
  - An abstract class for an array of [PtrVectors](#).*
- struct [PtrMatrix\\_PtPtrVector](#)
  - An implementation of [PtrMatrix](#) using `vector< PtrVector<T>*>`*
- struct [PtrMatrix\\_ptVec](#)
  - An implementation of [PtrMatrix](#) using `Vec< Vec<T>*>`*
- struct [PtrMatrix\\_ptvector](#)
  - An implementation of [PtrMatrix](#) using `vector< vector<T>*>`*
- struct [PtrMatrix\\_Vec](#)
  - An implementation of [PtrMatrix](#) using `Vec< Vec<T> >`*
- struct [PtrMatrix\\_vector](#)

- An implementation of [PtrMatrix](#) using `vector< vector< T> >`
- struct [PtrVector](#)
  - Abstract class for an array of objects.
- struct [PtrVector\\_Singleton](#)
  - An implementation of [PtrVector](#) from a single `T` object.
- struct [PtrVector\\_slice](#)
  - An implementation of [PtrVector](#) as a slice of another [PtrVector](#).
- struct [PtrVector\\_VecPt](#)
  - An implementation of [PtrVector](#) using `Vec< T*>`
- struct [PtrVector\\_VecT](#)
  - An implementation of [PtrVector](#) using `Vec< T>`
- struct [PtrVector\\_vectorPt](#)
  - An implementation of [PtrVector](#) using `vector< T*>`
- struct [PtrVector\\_vectorT](#)
  - An implementation of [PtrVector](#) using `vector< T>`
- class [Ptxt](#)
  - An object that mimics the functionality of the [Ctxt](#) object, and acts as a convenient entry point for inputting/encoding data which is to be encrypted.
- class [PubKey](#)
  - The public key.
- struct [PubKeyHack](#)
- struct [quarter\\_FFT](#)
- class [random\\_pa\\_impl](#)
- class [RandomBlockMatrix](#)
- class [RandomFullBlockMatrix](#)
- class [RandomFullMatrix](#)
- class [RandomMatrix](#)
- class [RandomMultiBlockMatrix](#)
- class [RandomMultiMatrix](#)
- class [RandomState](#)
  - Facility for "restoring" the [NTL](#) PRG state.
- class [RecryptData](#)
  - A structure to hold recryption-related data inside the [Context](#).
- class [replicate\\_pa\\_impl](#)
- class [ReplicateHandler](#)
  - An abstract class to handle call-backs to get the output of replicate.
- class [rotate\\_pa\\_impl](#)
- class [RuntimeError](#)
  - Inherits from [Exception](#) and `std::runtime_error`.
- class [ScratchCell](#)
  - A class to help manage the allocation of temporary [Ctxt](#) objects.
- class [SecKey](#)
  - The secret key.
- class [shallow\\_clone](#)
  - Shallow copy: initialize with copy constructor.
- class [shift\\_pa\\_impl](#)
- class [SKHandle](#)
  - A handle, describing the secret-key element that "matches" a part, of the form  $s^{\wedge} r(X^{\wedge} t)$ .
- class [sub\\_pa\\_impl](#)
- class [SubDimension](#)
  - A node in a tree relative to some generator.
- class [ThinEvalMap](#)

*Class that provides the functionality for the linear transforms used in "thin" bootstrapping, where slots are assumed to contain constants. The interface is exactly the same as for [EvalMap](#), except that the constructor does not have a `normal_basis` parameter.*

- class [ThinRecryptData](#)

*Same as above, but for "thin" bootstrapping, where the slots are assumed to contain constants.*

- class [TreeNode](#)

*A node in a full binary tree.*

- class [zz\\_pXModulus1](#)

*Auxiliary classes to facilitate faster reduction mod  $\Phi_m(X)$  when the input has degree less than  $m$ .*

- class [ZZ\\_pXModulus1](#)

*placeholder for pXModulus ...no optimizations*

## Typedefs

- typedef [PtrVector](#)< Ctxt > CPtrs
- typedef [PtrVector\\_VecT](#)< Ctxt > CPtrs\_VecCt
- typedef [PtrVector\\_vectorT](#)< Ctxt > CPtrs\_vectorCt
- typedef [PtrVector\\_VecPt](#)< Ctxt > CPtrs\_VecPt
- typedef [PtrVector\\_vectorPt](#)< Ctxt > CPtrs\_vectorPt
- typedef [PtrVector\\_slice](#)< Ctxt > CPtrs\_slice
- typedef [PtrMatrix](#)< Ctxt > CPtrMat
- typedef [PtrMatrix\\_Vec](#)< Ctxt > CPtrMat\_VecCt
- typedef [PtrMatrix\\_vector](#)< Ctxt > CPtrMat\_vectorCt
- typedef [PtrMatrix\\_ptVec](#)< Ctxt > CPtrMat\_ptVecCt
- typedef [PtrMatrix\\_ptvector](#)< Ctxt > CPtrMat\_ptvectorCt
- typedef std::shared\_ptr< [DoubleCRT](#) > DCRTptr
- typedef std::shared\_ptr< NTL::ZZX > ZZxptr
- typedef std::complex< double > cx\_double
- typedef std::unordered\_map< long, [FlowEdge](#) > FNeighborList
- typedef std::vector< [FNeighborList](#) > FlowGraph
- typedef std::unordered\_multimap< long, [LabeledEdge](#) > LNeighborList
- typedef long LONG
- typedef NTL::Vec< long > Permut
- *A simple permutation is just a vector with  $p[i]=\pi_i$ .*
- typedef [FullBinaryTree](#)< [SubDimension](#) > OneGeneratorTree
- typedef NTL::Vec< long > zzX
- typedef std::pair< long, long > NodeIdx
- template<class T >  
using [aligned\\_vector](#) = PGFFT::aligned\_vector< T >
- typedef complex< double > cmplx\_t
- typedef long double ldbl

## Enumerations

- enum [PA\\_tag](#) { [PA\\_GF2\\_tag](#), [PA\\_zz\\_p\\_tag](#), [PA\\_cx\\_tag](#) }

## Functions

- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>`  
`void assertTrue (const T &value, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>`  
`void assertFalse (T value, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>`  
`void assertEq (const T &a, const T &b, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>`  
`void assertNeq (const T &a, const T &b, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>`  
`void assertNotNull (const T &p, const std::string &message)`
- `template<typename ExceptionTy = ::helib::OutOfRangeError, typename T = void>`  
`void assertInRange (const T &elem, const T &min, const T &max, const std::string &message, bool right_inclusive=false)`
- `std::vector< long > longToBitVector (long num, long bitSize)`  
*Returns a number as a vector of bits with LSB on the left.*
- `void binaryCond (CtPtrs &output, const Ctxt &cond, const CtPtrs &trueValue, const CtPtrs &>falseValue)`  
*Implementation of output = cond \* trueValue + (1 - cond) \* falseValue.*
- `void binaryMask (CtPtrs &binaryNums, const Ctxt &mask)`  
*Zeroes the slots of binaryNums where the corresponding slot of mask is 0.*
- `void concatBinaryNums (CtPtrs &output, const CtPtrs &a, const CtPtrs &b)`  
*Concatenates two binary numbers into a single CtPtrs object. E.g. If a=10111, b=00101 then output = 1011100101.*
- `void splitBinaryNums (CtPtrs &leftSplit, CtPtrs &rightSplit, const CtPtrs &input)`  
*Splits a single binary number into two binary numbers leftSplit and rightSplit.*
- `void leftBitwiseShift (CtPtrs &output, const CtPtrs &input, const long shamt)`  
*Left shift input by shamt.*
- `void bitwiseRotate (CtPtrs &output, const CtPtrs &input, long rotamt)`  
*Rotate input by rotamt.*
- `void bitwiseXOR (CtPtrs &output, const CtPtrs &lhs, const CtPtrs &rhs)`  
*Compute a bitwise XOR between lhs and rhs.*
- `void bitwiseOr (CtPtrs &output, const CtPtrs &lhs, const CtPtrs &rhs)`  
*Compute a bitwise OR between lhs and rhs.*
- `void bitwiseAnd (CtPtrs &output, const CtPtrs &lhs, const CtPtrs &rhs)`  
*Compute a bitwise AND between lhs and rhs.*
- `void bitwiseAnd (CtPtrs &output, const CtPtrs &input, const std::vector< long > mask)`  
*Compute a bitwise AND between input and a std::vector<long>.*
- `void bitwiseNot (CtPtrs &output, const CtPtrs &input)`  
*Compute a bitwise NOT of input.*
- `void addTwoNumbers (CtPtrs &sum, const CtPtrs &lhs, const CtPtrs &rhs, long sizeLimit=0, std::vector< zzX > *unpackSlotEncoding=nullptr)`  
*Adds two numbers in binary representation where each ciphertext of the input vector contains a bit.*
- `void negateBinary (CtPtrs &negation, const CtPtrs &input)`  
*Negates a number in binary 2's complement representation.*
- `void subtractBinary (CtPtrs &difference, const CtPtrs &lhs, const CtPtrs &rhs, std::vector< zzX > *unpackSlotEncoding=nullptr)`  
*Subtracts rhs from lhs where lhs, rhs are in 2's complement.*
- `long fifteenOrLess4Four (const CtPtrs &out, const CtPtrs &in, long sizeLimit=4)`  
*Add together up to fifteen {0,1} integers, producing a 4-bit counter.*
- `void addManyNumbers (CtPtrs &sum, CtPtrMat &numbers, long sizeLimit=0, std::vector< zzX > *unpackSlotEncoding=nullptr)`  
*Sum an arbitrary amount of numbers in binary representation.*

- void `multTwoNumbers` (CtPtrs &product, const CtPtrs &lhs, const CtPtrs &rhs, bool rhsTwos←  
Complement=false, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Multiply two numbers in binary representation where each ciphertext of the input vector contains a bit.*
- void `decryptBinaryNums` (std::vector< long > &pNums, const CtPtrs &eNums, const SecKey &sKey, const  
EncryptedArray &ea, bool twosComplement=false, bool allSlots=true)  
*Decrypt the binary numbers that are encrypted in eNums.*
- void `packedRecrypt` (const CtPtrs &a, const CtPtrs &b, std::vector< zzX > \*unpackSlotEncoding)  
*Function for packed recryption to recrypt multiple numbers.*
- void `compareTwoNumbers` (CtPtrs &max, CtPtrs &min, Ctxt &mu, Ctxt &ni, const CtPtrs &a, const CtPtrs &b,  
bool twosComplement=false, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Compares two integers in binary a, b. Returns max(a, b), min(a, b) and indicator bits mu=(a>b) and  
ni=(a<b)*
- void `compareTwoNumbers` (Ctxt &mu, Ctxt &ni, const CtPtrs &a, const CtPtrs &b, bool twos←  
Complement=false, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Compares two integers in binary a, b. Returns only indicator bits mu=(a>b) and ni=(a<b).*
- int `readEyeCatcher` (std::istream &str, const char \*expect)
- void `writeEyeCatcher` (std::ostream &str, const char \*eye)
- void `write_ntl_vec_long` (std::ostream &str, const NTL::vec\_long &vl, long intSize=BINIO\_64BIT)
- void `read_ntl_vec_long` (std::istream &str, NTL::vec\_long &vl)
- long `read_raw_int` (std::istream &str)
- int `read_raw_int32` (std::istream &str)
- void `write_raw_int` (std::ostream &str, long num)
- void `write_raw_int32` (std::ostream &str, int num)
- void `write_raw_double` (std::ostream &str, const double d)
- double `read_raw_double` (std::istream &str)
- void `write_raw_xdouble` (std::ostream &str, const NTL::xdouble xd)
- NTL::xdouble `read_raw_xdouble` (std::istream &str)
- void `write_raw_ZZ` (std::ostream &str, const NTL::ZZ &zz)
- void `read_raw_ZZ` (std::istream &str, NTL::ZZ &zz)
- template<typename T >  
void `write_raw_vector` (std::ostream &str, const std::vector< T > &v)
- template<> void `write_raw_vector< long >` (std::ostream &str, const std::vector< long > &v)
- template<> void `write_raw_vector< double >` (std::ostream &str, const std::vector< double > &v)
- template<typename T >  
void `read_raw_vector` (std::istream &str, std::vector< T > &v, T &init)
- template<typename T >  
void `read_raw_vector` (std::istream &str, std::vector< T > &v)
- template<> void `read_raw_vector< long >` (std::istream &str, std::vector< long > &v)
- template<> void `read_raw_vector< double >` (std::istream &str, std::vector< double > &v)
- template<typename T >  
void `read_raw_vector` (std::istream &str, std::vector< T > &v, const Context &context)
- void `BluesteinInit` (long n, const NTL::zz\_p &root, NTL::zz\_pX &powers, NTL::Vec< NTL::mulmod\_precon\_t  
> &powers\_aux, NTL::fftRep &Rb)  
*initialize bluestein*
- void `BluesteinFFT` (NTL::zz\_pX &x, long n, const NTL::zz\_p &root, const NTL::zz\_pX &powers, const NTL←  
::Vec< NTL::mulmod\_precon\_t > &powers\_aux, const NTL::fftRep &Rb)  
*apply bluestein*
- template<typename X, typename Cloner >  
void `swap` (cloned\_ptr< X, Cloner > &x, cloned\_ptr< X, Cloner > &y)
- template<typename X, typename Cloner >  
void `swap` (copied\_ptr< X, Cloner > &x, copied\_ptr< X, Cloner > &y)
- long `FindM` (long k, long nBits, long c, long p, long d, long s, long chosen\_m, bool verbose=false)  
*Returns smallest parameter m satisfying various constraints:*
- void `writeContextBase` (std::ostream &s, const Context &context)

- write [m p r gens ords] data*
- void [readContextBase](#) (std::istream &s, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)
- read [m p r gens ords] data, needed to construct context*
- std::unique\_ptr< [Context](#) > [buildContextFromAscii](#) (std::istream &str)
  - void [writeContextBaseBinary](#) (std::ostream &str, const [Context](#) &context)
- write [m p r gens ords] data*
- void [writeContextBinary](#) (std::ostream &str, const [Context](#) &context)
  - void [readContextBaseBinary](#) (std::istream &s, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)
- read [m p r gens ords] data, needed to construct context*
- std::unique\_ptr< [Context](#) > [buildContextFromBinary](#) (std::istream &str)
  - void [readContextBinary](#) (std::istream &str, [Context](#) &context)
  - void [buildModChain](#) ([Context](#) &context, long nBits, long nDgts=3, bool willBeBootstrappable=false, long sk←Hwt=0, long resolution=3, long bitsInSpecialPrimes=0)
  - void [endBuildModChain](#) ([Context](#) &context)
  - void [packedRecrypt](#) (const [CtPtrs](#) &cPtrs, const std::vector< [zzX](#) > &unpackConsts, const [EncryptedArray](#) &ea)
  - void [packedRecrypt](#) (const [CtPtrs](#) &array, const std::vector< [zzX](#) > &unpackConsts, const [EncryptedArray](#) &ea, long belowLvl)
  - void [packedRecrypt](#) (const [CtPtrMat](#) &m, const std::vector< [zzX](#) > &unpackConsts, const [EncryptedArray](#) &ea, long belowLvl=LONG\_MAX)
  - long [findMinBitCapacity](#) (const [CtPtrs](#) &v)
  - long [findMinBitCapacity](#) (const [CtPtrMat](#) &m)
  - long [findMinBitCapacity](#) (std::initializer\_list< const [CtPtrs](#) \* > list)
  - void [innerProduct](#) ([Ctxt](#) &result, const [CtPtrs](#) &v1, const [CtPtrs](#) &v2)
  - [Ctxt](#) [innerProduct](#) (const [CtPtrs](#) &v1, const [CtPtrs](#) &v2)
  - std::ostream & [operator<<](#) (std::ostream &s, const [SKHandle](#) &handle)
  - std::istream & [operator>>](#) (std::istream &s, [CtxtPart](#) &p)
  - std::ostream & [operator<<](#) (std::ostream &s, const [CtxtPart](#) &p)
  - void [totalProduct](#) ([Ctxt](#) &out, const std::vector< [Ctxt](#) > &v)
  - void [incrementalProduct](#) (std::vector< [Ctxt](#) > &v)
  - void [innerProduct](#) ([Ctxt](#) &result, const std::vector< [Ctxt](#) > &v1, const std::vector< [Ctxt](#) > &v2)
  - [Ctxt](#) [innerProduct](#) (const std::vector< [Ctxt](#) > &v1, const std::vector< [Ctxt](#) > &v2)
  - void [innerProduct](#) ([Ctxt](#) &result, const std::vector< [Ctxt](#) > &v1, const std::vector< [DoubleCRT](#) > &v2)
- Compute the inner product of a vectors of ciphertexts and a constant vector.*
- [Ctxt](#) [innerProduct](#) (const std::vector< [Ctxt](#) > &v1, const std::vector< [DoubleCRT](#) > &v2)
  - void [innerProduct](#) ([Ctxt](#) &result, const std::vector< [Ctxt](#) > &v1, const std::vector< [NTL::ZZX](#) > &v2)
  - [Ctxt](#) [innerProduct](#) (const std::vector< [Ctxt](#) > &v1, const std::vector< [NTL::ZZX](#) > &v2)
  - void [CheckCtxt](#) (const [Ctxt](#) &c, const char \*label)
- print to cerr some info about ciphertext*
- void [extractDigits](#) (std::vector< [Ctxt](#) > &digits, const [Ctxt](#) &c, long r=0)
- Extract the mod-p digits of a mod-p^r ciphertext.*
- void [extractDigits](#) (std::vector< [Ctxt](#) > &digits, const [Ctxt](#) &c, long r, bool shortCut)
  - void [extendExtractDigits](#) (std::vector< [Ctxt](#) > &digits, const [Ctxt](#) &c, long r, long e)
  - void [setupDebugGlobals](#) ([SecKey](#) \*debug\_key, const std::shared\_ptr< const [EncryptedArray](#) > &debug\_ea, [NTL::ZZX](#) debug\_ptxt=[NTL::ZZX](#){})
- Setup function for setting up the global debug variables.*
- void [cleanupDebugGlobals](#) ()
- Cleanup function for clearing the global debug variables.*
- void [decryptAndPrint](#) (std::ostream &s, const [Ctxt](#) &ctxt, const [SecKey](#) &sk, const [EncryptedArray](#) &ea, long flags=0)
  - [NTL::xdouble](#) [embeddingLargestCoeff](#) (const [Ctxt](#) &ctxt, const [SecKey](#) &sk)
  - double [realToEstimatedNoise](#) (const [Ctxt](#) &ctxt, const [SecKey](#) &sk)

- void [checkNoise](#) (const [Ctxt](#) &ctxt, const [SecKey](#) &sk, const std::string &msg, double thresh=10.0)
- bool [decryptAndCompare](#) (const [Ctxt](#) &ctxt, const [SecKey](#) &sk, const [EncryptedArray](#) &ea, const [PlaintextArray](#) &pa)
- void [rawDecrypt](#) (NTL::ZZX &plaintext, const std::vector< NTL::ZZX > &zzParts, const [DoubleCRT](#) &sKey, long q=0)
- template<typename VEC >  
std::ostream & [printVec](#) (std::ostream &s, const VEC &v, long nCoeffs=40)
- std::ostream & [printZZX](#) (std::ostream &s, const NTL::ZZX &poly, long nCoeffs=40)
- void [conv](#) ([DoubleCRT](#) &d, const NTL::ZZX &p)
- void [conv](#) (NTL::ZZX &p, const [DoubleCRT](#) &d)
- NTL::ZZX [to\\_ZZX](#) (const [DoubleCRT](#) &d)
- template<typename RX , typename RXModulus >  
void [plaintextAutomorph](#) (RX &bb, const RX &a, long k, long m, const RXModulus &PhimX)
- template<typename RX , typename type >  
void [plaintextAutomorph](#) (RX &b, const RX &a, long i, long j, const [EncryptedArrayDerived](#)< type > &ea)
- [EncryptedArrayBase](#) \* [buildEncryptedArray](#) (const [Context](#) &context, const [PAlgebraMod](#) &a1Mod, const NTL::ZZX &G=NTL::ZZX::zero())

A "factory" for building *EncryptedArrays*.

- std::ostream & [operator<<](#) (std::ostream &s, const [PlaintextArray](#) &pa)
- void [rotate](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, long k)
- void [shift](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, long k)
- void [encode](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const std::vector< long > &array)
- void [encode](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const std::vector< NTL::ZZX > &array)
- void [encode](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, long val)
- void [encode](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const NTL::ZZX &val)
- void [random](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa)
- void [decode](#) (const [EncryptedArray](#) &ea, std::vector< long > &array, const [PlaintextArray](#) &pa)
- void [decode](#) (const [EncryptedArray](#) &ea, std::vector< NTL::ZZX > &array, const [PlaintextArray](#) &pa)
- bool [equals](#) (const [EncryptedArray](#) &ea, const [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)
- bool [equals](#) (const [EncryptedArray](#) &ea, const [PlaintextArray](#) &pa, const std::vector< long > &other)
- bool [equals](#) (const [EncryptedArray](#) &ea, const [PlaintextArray](#) &pa, const std::vector< NTL::ZZX > &other)
- void [add](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)
- void [sub](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)
- void [mul](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)
- void [negate](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa)
- void [frobeniusAutomorph](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, long j)
- void [frobeniusAutomorph](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const NTL::Vec< long > &vec)
- void [applyPerm](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, const NTL::Vec< long > &pi)
- void [power](#) (const [EncryptedArray](#) &ea, [PlaintextArray](#) &pa, long e)
- void [runningSums](#) (const [EncryptedArray](#) &ea, [Ctxt](#) &ctxt)

A *ctxt* that encrypts  $(x_1, \dots, x_n)$  is replaced by an encryption of  $(y_1, \dots, y_n)$ , where  $y_i = \sum_{j \leq i} x_j$ .

- void [print\\_stats](#) (std::ostream &s)
- const std::vector< double > \* [fetch\\_saved\\_values](#) (const char \*)
- std::ostream & [operator<<](#) (std::ostream &s, const [CubeSignature](#) &sig)
- template<typename T >  
void [getHyperColumn](#) (NTL::Vec< T > &v, const [ConstCubeSlice](#)< T > &s, long pos)
- template<typename T >  
void [setHyperColumn](#) (const NTL::Vec< T > &v, const [CubeSlice](#)< T > &s, long pos)
- template<typename T >  
void [setHyperColumn](#) (const NTL::Vec< T > &v, const [CubeSlice](#)< T > &s, long pos, const T &val)
- template<typename T >  
void [print3D](#) (const [HyperCube](#)< T > &c)
- template<typename T >  
bool [operator==](#) (const [IndexMap](#)< T > &map1, const [IndexMap](#)< T > &map2)

Comparing maps, by comparing all the elements.



- `template<typename T >`  
`bool operator!= (const IndexMap< T > &map1, const IndexMap< T > &map2)`
- `IndexSet operator| (const IndexSet &s, const IndexSet &t)`  
*union*
- `IndexSet operator& (const IndexSet &s, const IndexSet &t)`  
*intersection*
- `IndexSet operator^ (const IndexSet &s, const IndexSet &t)`  
*exclusive-or*
- `IndexSet operator/ (const IndexSet &s, const IndexSet &t)`  
*set minus*
- `std::ostream & operator<< (std::ostream &str, const IndexSet &set)`
- `std::istream & operator>> (std::istream &str, IndexSet &set)`
- `long card (const IndexSet &s)`  
*Functional cardinality.*
- `bool empty (const IndexSet &s)`
- `bool operator<= (const IndexSet &s1, const IndexSet &s2)`  
*Is s1 subset or equal to s2.*
- `bool operator< (const IndexSet &s1, const IndexSet &s2)`  
*Is s1 strict subset of s2.*
- `bool operator>= (const IndexSet &s1, const IndexSet &s2)`  
*Is s2 subset or equal to s2.*
- `bool operator> (const IndexSet &s1, const IndexSet &s2)`  
*Is s2 strict subset of s1.*
- `bool disjoint (const IndexSet &s1, const IndexSet &s2)`  
*Functional disjoint.*
- `void buildUnpackSlotEncoding (std::vector< zzX > &unpackSlotEncoding, const EncryptedArray &ea)`
- `void unpack (const CtPtrs &unpacked, const Ctxt &packed, const EncryptedArray &ea, const std::vector< zzX > &unpackSlotEncoding)`
- `long unpack (const CtPtrs &unpacked, const CtPtrs &packed, const EncryptedArray &ea, const std::vector< zzX > &unpackSlotEncoding)`
- `void repack (Ctxt &packed, const CtPtrs &unpacked, const EncryptedArray &ea)`
- `long repack (const CtPtrs &packed, const CtPtrs &unpacked, const EncryptedArray &ea)`
- `void unpackSlots (std::vector< unsigned long > &value, PlaintextArray &pa, const EncryptedArray &ea)`
- `void unpackSlots (std::vector< unsigned long > &value, NTL::ZZX &pa, const EncryptedArray &ea)`
- `void packConstant (zzX &result, unsigned long data, long nbits, const EncryptedArray &ea)`
- `void packConstants (zzX &result, const std::vector< unsigned long > &data, long nbits, const EncryptedArray &ea)`
- `void writePubKeyBinary (std::ostream &str, const PubKey &pk)`
- `void readPubKeyBinary (std::istream &str, PubKey &pk)`
- `void writeSecKeyBinary (std::ostream &str, const SecKey &sk)`
- `void readSecKeyBinary (std::istream &str, SecKey &sk)`
- `double RLWE (DoubleCRT &c0, DoubleCRT &c1, const DoubleCRT &s, long p, NTL::ZZ *prgSeed=nullptr)`
- `double RLWE1 (DoubleCRT &c0, const DoubleCRT &c1, const DoubleCRT &s, long p)`  
*Same as RLWE, but assumes that c1 is already chosen by the caller.*
- `std::ostream & operator<< (std::ostream &str, const KeySwitch &matrix)`
- `long maximum_flow (FlowGraph &fg, long src, long sink)`
- `void traceMap (Ctxt &cctx)`
- `void mul (PlaintextArray &pa, const MatMul1D &mat)`
- `void mul (PlaintextArray &pa, const BlockMatMul1D &mat)`
- `void mul (PlaintextArray &pa, const MatMulFull &mat)`
- `void mul (PlaintextArray &pa, const BlockMatMulFull &mat)`
- `long sumOfCoeffs (const zzX &f)`
- `NTL::ZZ sumOfCoeffs (const NTL::ZZX &f)`



- NTL::ZZ [sumOfCoeffs](#) (const [DoubleCRT](#) &f)
- template<typename T >  
double [largestCoeff](#) (const NTL::Vec< T > &f)  
*The L-infinity norm of an element (in coefficient representation)*
- template<typename T >  
double [largestCoeff](#) (const std::vector< T > &f)
- NTL::ZZ [largestCoeff](#) (const NTL::ZZX &f)
- NTL::ZZ [largestCoeff](#) (const NTL::Vec< NTL::ZZ > &f)
- NTL::ZZ [largestCoeff](#) (const [DoubleCRT](#) &f)
- double [coeffsL2NormSquared](#) (const [zzX](#) &f)  
*The L2-norm of an element (in coefficient representation)*
- NTL::xdouble [coeffsL2NormSquared](#) (const NTL::ZZX &f)
- NTL::xdouble [coeffsL2NormSquared](#) (const [DoubleCRT](#) &f)
- double [coeffsL2Norm](#) (const [zzX](#) &f)
- NTL::xdouble [coeffsL2Norm](#) (const NTL::ZZX &f)
- NTL::xdouble [coeffsL2Norm](#) (const [DoubleCRT](#) &f)
- double [embeddingLargestCoeff](#) (const [zzX](#) &f, const [PAlgebra](#) &palg)
- double [embeddingLargestCoeff](#) (const std::vector< double > &f, const [PAlgebra](#) &palg)
- void [embeddingLargestCoeff\\_x2](#) (double &norm1, double &norm2, const std::vector< double > &f1, const std::vector< double > &f2, const [PAlgebra](#) &palg)
- NTL::xdouble [embeddingLargestCoeff](#) (const NTL::ZZX &f, const [PAlgebra](#) &palg)
- void [CKKS\\_canonicalEmbedding](#) (std::vector< [cx\\_double](#) > &v, const [zzX](#) &f, const [PAlgebra](#) &palg)
- void [CKKS\\_canonicalEmbedding](#) (std::vector< [cx\\_double](#) > &v, const NTL::ZZX &f, const [PAlgebra](#) &palg)
- void [CKKS\\_canonicalEmbedding](#) (std::vector< [cx\\_double](#) > &v, const std::vector< double > &f, const [PAlgebra](#) &palg)
- void [CKKS\\_embedInSlots](#) ([zzX](#) &f, const std::vector< [cx\\_double](#) > &v, const [PAlgebra](#) &palg, double scaling)
- bool [setDryRun](#) (bool toWhat=true)
- bool [isDryRun](#) ()
- void [setAutomorphVals](#) (std::set< long > \*aVals)
- bool [isSetAutomorphVals](#) ()
- void [recordAutomorphVal](#) (long k)
- void [setAutomorphVals2](#) (std::set< long > \*aVals)
- bool [isSetAutomorphVals2](#) ()
- void [recordAutomorphVal2](#) (long k)
- long [bitSetToLong](#) (long bits, long bitSize)  
*Considers `bits` as a vector of bits and returns the value it represents when interpreted as a n-bit 2's complement number, where n is given by `bitSize`.*
- long [mcMod](#) (long a, long b)  
*Routines for computing mathematically correct mod and div.*
- long [mcDiv](#) (long a, long b)
- long [balRem](#) (long a, long q)
- double [fsquare](#) (double x)  
*Return the square of a number as a double.*
- long [multOrd](#) (long p, long m)  
*Return multiplicative order of p modulo m, or 0 if GCD(p, m) != 1.*
- void [ppsolve](#) (NTL::vec\_zz\_pE &x, const NTL::mat\_zz\_pE &A, const NTL::vec\_zz\_pE &b, long p, long r)  
*Prime power solver.*
- void [ppsolve](#) (NTL::vec\_GF2E &x, const NTL::mat\_GF2E &A, const NTL::vec\_GF2E &b, long p, long r)  
*A version for GF2: must have p == 2 and r == 1.*
- void [pplInvert](#) (NTL::mat\_zz\_p &X, const NTL::mat\_zz\_p &A, long p, long r)  
*Compute the inverse mod  $p^r$  of an  $n \times n$  matrix.*
- void [pplInvert](#) (NTL::mat\_zz\_pE &X, const NTL::mat\_zz\_pE &A, long p, long r)
- void [pplInvert](#) (NTL::mat\_GF2 &X, const NTL::mat\_GF2 &A, [UNUSED](#) long p, [UNUSED](#) long r)
- void [pplInvert](#) (NTL::mat\_GF2E &X, const NTL::mat\_GF2E &A, [UNUSED](#) long p, [UNUSED](#) long r)

- void [buildLinPolyMatrix](#) (NTL::mat\_zz\_pE &M, long p)
- void [buildLinPolyMatrix](#) (NTL::mat\_GF2E &M, long p)
- void [buildLinPolyCoeffs](#) (NTL::vec\_zz\_pE &C, const NTL::vec\_zz\_pE &L, long p, long r)  
*Combination of buildLinPolyMatrix and ppsolve.*
- void [buildLinPolyCoeffs](#) (NTL::vec\_GF2E &C, const NTL::vec\_GF2E &L, long p, long r)  
*A version for GF2: must be called with p == 2 and r == 1.*
- void [applyLinPoly](#) (NTL::zz\_pE &beta, const NTL::vec\_zz\_pE &C, const NTL::zz\_pE &alpha, long p)  
*Apply a linearized polynomial with coefficient vector C.*
- void [applyLinPoly](#) (NTL::GF2E &beta, const NTL::vec\_GF2E &C, const NTL::GF2E &alpha, long p)  
*A version for GF2: must be called with p == 2 and r == 1.*
- double [log2](#) (const NTL::xdouble &x)  
*Base-2 logarithm.*
- void [factorize](#) (std::vector< long > &factors, long N)  
*Factoring by trial division, only works for  $N < 2^{60}$ , only the primes are recorded, not their multiplicity.*
- void [factorize](#) (std::vector< NTL::ZZ > &factors, const NTL::ZZ &N)
- void [factorize](#) (NTL::Vec< NTL::Pair< long, long >> &factors, long N)  
*Factoring by trial division, only works for  $N < 2^{60}$  primes and multiplicities are recorded.*
- void [pp\\_factorize](#) (std::vector< long > &factors, long N)  
*Prime-power factorization.*
- void [phiN](#) (long &phiN, std::vector< long > &facts, long N)  
*Compute Phi(N) and also factorize N.*
- void [phiN](#) (NTL::ZZ &phiN, std::vector< NTL::ZZ > &facts, const NTL::ZZ &N)
- long [phi\\_N](#) (long N)  
*Compute Phi(N).*
- long [findGenerators](#) (std::vector< long > &gens, std::vector< long > &ords, long m, long p, const std::vector< long > &candidates=std::vector< long >())
- void [FindPrimitiveRoot](#) (NTL::zz\_p &r, unsigned long e)  
*Find e-th root of unity modulo the current modulus.*
- void [FindPrimitiveRoot](#) (NTL::ZZ\_p &r, unsigned long e)
- long [mobius](#) (long n)  
*Compute mobius function (naive method as n is small).*
- NTL::ZZX [Cyclotomic](#) (long N)  
*Compute cyclotomic polynomial.*
- NTL::ZZX [makeIrredPoly](#) (long p, long d)  
*Return a degree-d irreducible polynomial mod p.*
- long [primroot](#) (long N, long [phiN](#))  
*Find a primitive root modulo N.*
- long [ord](#) (long N, long p)  
*Compute the highest power of p that divides N.*
- bool [is2power](#) (long m)
- NTL::ZZX [RandPoly](#) (long n, const NTL::ZZ &p)
- void [MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q, bool abs)
- void [MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q)
- NTL::ZZX [MulMod](#) (const NTL::ZZX &f, long a, long q, bool abs)
- NTL::ZZX [MulMod](#) (const NTL::ZZX &f, long a, long q)
- void [balanced\\_MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q)
- template<typename T1, typename T2>  
void [convert](#) (T1 &x1, const T2 &x2)  
*A generic template that resolves to NTL's conv routine.*
- template<typename T1, typename T2>  
void [convert](#) (std::vector< T1 > &v1, const std::vector< T2 > &v2)  
*generic vector conversion routines*

- `template<typename T1 , typename T2 >`  
`void convert (std::vector< T1 > &v1, const NTL::Vec< T2 > &v2)`
- `template<typename T1 , typename T2 >`  
`void convert (NTL::Vec< T1 > &v1, const std::vector< T2 > &v2)`
- `template<typename T >`  
`void convert (std::vector< T > &v1, const std::vector< T > &v2)`  
*Trivial type conversion, useful for generic code.*
- `template<typename T1 , typename T2 >`  
`T1 convert (const T2 &v2)`
- `template<typename T >`  
`std::vector< T > vector_replicate (const T &a, long n)`
- `template<typename T >`  
`std::vector< T > Vec_replicate (const T &a, long n)`
- `long computeProd (const NTL::Vec< long > &vec)`  
*returns \prod\_d vec[d]*
- `long computeProd (const std::vector< long > &vec)`
- `void mul (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, long b)`
- `void div (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, long b)`
- `void add (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, const std::vector< NTL::ZZX > &b)`
- `long is_in (long x, int *X, long sz)`  
*Finds whether x is an element of the set X of size sz, Returns -1 if not and the location if true.*
- `long CRTcoeff (long p, long q, bool symmetric=false)`  
*Returns a CRT coefficient:  $x = (0 \bmod p, 1 \bmod q)$ . If symmetric is set then  $x \in [-pq/2, pq/2)$ , else  $x \in [0, pq)$*
- `template<class zzvec >`  
`bool intVecCRT (NTL::vec_ZZ &vp, const NTL::ZZ &p, const zzvec &vq, long q)`  
*Incremental integer CRT for vectors.*
- `template<typename T , bool maxFlag>`  
`long argminmax (std::vector< T > &v)`  
*Find the index of the (first) largest/smallest element.*
- `template<typename T >`  
`long argmax (std::vector< T > &v)`
- `template<typename T >`  
`long argmin (std::vector< T > &v)`
- `long argmax (std::vector< long > &v, bool(*moreThan)(long, long))`  
*A variant with a specialized comparison function (\*moreThan)(a,b) returns the comparison a>b.*
- `bool closeToOne (const NTL::xdouble &x, long p)`
- `std::pair< long, long > rationalApprox (double x, long denomBound=0)`
- `std::pair< NTL::ZZ, NTL::ZZ > rationalApprox (NTL::xdouble x, NTL::xdouble denomBound=NTL::xdouble(0.0))`
- `void seekPastChar (std::istream &str, int cc)`  
*Advance the input stream beyond white spaces and a single instance of the char cc.*
- `template<typename T >`  
`void reverse (NTL::Vec< T > &v, long lo, long hi)`  
*Reverse a vector in place.*
- `template<typename T >`  
`void rotate (NTL::Vec< T > &v, long k)`  
*Rotate a vector in place using swaps.*
- `template<typename T >`  
`long lsize (const std::vector< T > &v)`  
*Size of STL vector as a long (rather than unsigned long)*
- `template<typename T >`  
`void killVec (std::vector< T > &vec)`  
*NTL/std compatibility.*

- `template<typename T >`  
`void killVec (NTL::Vec< T > &vec)`
- `template<typename T >`  
`void setLengthZero (std::vector< T > &vec)`
- `template<typename T >`  
`void setLengthZero (NTL::Vec< T > &vec)`
- `template<typename T >`  
`long lsize (const NTL::Vec< T > &v)`
- `template<typename T >`  
`void resize (NTL::Vec< T > &v, long sz, const T &val)`
- `template<typename T >`  
`void resize (std::vector< T > &v, long sz, const T &val)`
- `template<typename T >`  
`void resize (NTL::Vec< T > &v, long sz)`
- `template<typename T >`  
`void resize (std::vector< T > &v, long sz)`
- `template<typename T1 , typename T2 >`  
`bool sameObject (const T1 *p1, const T2 *p2)`  
*Testing if two vectors point to the same object.*
- `void ModComp (NTL::ZZX &res, const NTL::ZZX &g, const NTL::ZZX &h, const NTL::ZZX &f)`  
*Modular composition of polynomials:  $res = g(h) \bmod f$ .*
- `long polyEvalMod (const NTL::ZZX &poly, long x, long p)`  
*Evaluates a modular integer polynomial, returns  $poly(x) \bmod p$ .*
- `void interpolateMod (NTL::ZZX &poly, const NTL::vec_long &x, const NTL::vec_long &y, long p, long e=1)`  
*Interpolate polynomial such that  $poly(x[i] \bmod p) = y[i] \bmod p^e$ . It is assumed that the points  $x[i]$  are all distinct modulo  $p$ .*
- `long divc (long a, long b)`  
*returns ceiling( $a/b$ ); assumes  $a \geq 0$ ,  $b > 0$ ,  $a+b \leq MAX\_LONG$*
- `void rem (NTL::zz_pX &r, const NTL::zz_pX &a, const zz_pXModulus1 &ff)`
- `template<typename T >`  
`std::ostream & operator<< (std::ostream &s, std::vector< T > &v)`
- `template<typename T >`  
`std::istream & operator>> (std::istream &s, std::vector< T > &v)`
- `template<typename T >`  
`std::string vecToStr (const std::vector< T > &v)`
- `template<typename T >`  
`NTL::Vec< T > atovec (const char *a)`
- `template<typename T >`  
`std::vector< T > atovec (const char *a)`
- `void TofftRep_trunc (NTL::fftRep &y, const NTL::zz_pX &x, long k, UNUSED long len, long lo, long hi)`
- `void TofftRep_trunc (NTL::fftRep &y, const NTL::zz_pX &x, long k, long len)`
- `template<typename T , typename P , typename... Args>`  
`void make_lazy (const NTL::Lazy< T, P > &obj, Args &&... args)`
- `template<typename T , typename P , typename F , typename... Args>`  
`void make_lazy_with_fun (const NTL::Lazy< T, P > &obj, F f, Args &&... args)`
- `void Warning (const char *msg)`
- `void Warning (const std::string &msg)`
- `PAAlgebraModBase * buildPAAlgebraMod (const PAAlgebra &zMStar, long r)`  
*Builds a table, of type  $PA\_GF2$  if  $p == 2$  and  $r == 1$ , and  $PA\_zz\_p$  otherwise.*
- `bool comparePAAlgebra (const PAAlgebra &palg, unsigned long m, unsigned long p, unsigned long r, const std::vector< long > &gens, const std::vector< long > &ords)`  
*returns true if the palg parameters match the rest, false otherwise*
- `double calcPolyNormBnd (long m)`
- `template<typename T >`  
`void applyPermToVec (NTL::Vec< T > &out, const NTL::Vec< T > &in, const Permut &p1)`

- Apply a permutation to a `std::vector`, `out[i]=in[p1[i]]` (NOT in-place)*

  - `template<typename T >`  
`void applyPermToVec (std::vector< T > &out, const std::vector< T > &in, const Permut &p1)`
  - `template<typename T >`  
`void applyPermsToVec (NTL::Vec< T > &out, const NTL::Vec< T > &in, const Permut &p2, const Permut &p1)`

*Apply two permutations to a `std::vector` `out[i]=in[p2[p1[i]]]` (NOT in-place)*

  - `template<typename T >`  
`void applyPermsToVec (std::vector< T > &out, const std::vector< T > &in, const Permut &p2, const Permut &p1)`
  - `void randomPerm (Permut &perm, long n)`

*A random size-n permutation.*

  - `std::ostream & operator<< (std::ostream &s, const ColPerm &p)`
  - `void breakPermByDim (std::vector< ColPerm > &out, const Permut &pi, const CubeSignature &sig)`

*Takes a permutation  $\pi$  over  $m$ -dimensional cube  $C=Z_{\{n1\}} \times \dots \times Z_{\{nm\}}$  and expresses  $\pi$  as a product  $\pi = \rho_{i-1} \circ \dots \circ \rho_{i-2} \circ \rho_{i-1}$  where each  $\rho_{i-1}$  is a column permutation along one dimension. Specifically for  $i < m$ , the permutations  $\rho_{i-1}$  and  $\rho_{i-2(m-1)-i}$  permute the  $i$ 'th dimension.*

  - `void polyEval (Ctxt &ret, NTL::ZZX poly, const Ctxt &x, long k=0)`

*Evaluate a cleartext polynomial on an encrypted input.*

  - `void polyEval (Ctxt &ret, const NTL::Vec< Ctxt > &poly, const Ctxt &x)`

*Evaluate an encrypted polynomial on an encrypted input.*

  - `std::ostream & operator<< (std::ostream &s, const ModuliSizes::Entry &e)`
  - `std::istream & operator>> (std::istream &s, ModuliSizes::Entry &e)`
  - `void write (std::ostream &s, const ModuliSizes::Entry &e)`
  - `void read (std::istream &s, ModuliSizes::Entry &e)`
  - `template<typename T >`  
`long lsize (const PtrMatrix< T > &v)`
  - `template<typename T >`  
`void resize (PtrMatrix< T > &v, long newSize)`
  - `template<typename T >`  
`void setLengthZero (PtrMatrix< T > &v)`
  - `template<typename T >`  
`const T * ptr2nonNull (std::initializer_list< const PtrVector< T > * > list)`
  - `template<typename T >`  
`long lsize (const PtrVector< T > &v)`
  - `template<typename T >`  
`void setLengthZero (PtrVector< T > &v)`
  - `template<typename T >`  
`void resize (PtrVector< T > &v, long newSize, const T &val)`
  - `template<typename T >`  
`void resize (PtrVector< T > &v, long newSize, const T *val)`
  - `template<typename V1 , typename V2 >`  
`void vecCopy (V1 &v1, const V2 &v2, long sizeLimit=0)`
  - `template<typename V , typename T >`  
`void vecCopy (V &v1, const PtrVector< T > &v2, long sizeLimit=0)`
  - `template<typename V , typename T >`  
`void vecCopy (PtrVector< T > &v1, const V &v2, long sizeLimit=0)`
  - `template<typename T >`  
`void vecCopy (PtrVector< T > &v1, const PtrVector< T > &v2, long sizeLimit=0)`
  - `template<typename From , typename Scheme >`  
`std::vector< typename Scheme::SlotType > convertDataToSlotVector (const std::vector< From > &data, const Context &context)`

*Converts `std::vector<From>` to `std::vector<Scheme::SlotType>`.*

  - `template<typename Scheme >`  
`void innerProduct (Ptxt< Scheme > &result, const std::vector< Ptxt< Scheme >> &first_vec, const std::vector< Ptxt< Scheme >> &second_vec)`

Free function that computes the inner product of two vectors of *Ptxt*.

- `MatMul1D * buildRandomMatrix` (const `EncryptedArray` &ea, long dim)
- `MatMul1D * buildRandomMultiMatrix` (const `EncryptedArray` &ea, long dim)
- `BlockMatMul1D * buildRandomBlockMatrix` (const `EncryptedArray` &ea, long dim)
- `BlockMatMul1D * buildRandomMultiBlockMatrix` (const `EncryptedArray` &ea, long dim)
- `MatMulFull * buildRandomFullMatrix` (const `EncryptedArray` &ea)
- `BlockMatMulFull * buildRandomFullBlockMatrix` (const `EncryptedArray` &ea)
- `general_range`< long > `range` (long n)
- `general_range`< long > `range` (long m, long n)
- void `replicate` (const `EncryptedArray` &ea, `Ctxt` &ctx, long pos)

The value in slot #pos is replicated in all other slots. On an n-slot ciphertext, this algorithm performs  $O(\log n)$  1D rotations.

- void `replicate0` (const `EncryptedArray` &ea, `Ctxt` &ctx, long pos)  
A lower-level routine. Same as `replicate`, but assumes all slots are zero except slot #pos.
- void `replicateAll` (const `EncryptedArray` &ea, const `Ctxt` &ctx, `ReplicateHandler` \*handler, long recBound=64, `RepAuxDim` \*repAuxPtr=nullptr)
- void `replicateAll` (std::vector< `Ctxt` > &v, const `EncryptedArray` &ea, const `Ctxt` &ctx, long recBound=64, `RepAuxDim` \*repAuxPtr=nullptr)
- template<typename Scheme >  
void `replicateAll` (std::vector< `Ptxt`< Scheme >> &v, const `EncryptedArray` &, const `Ptxt`< Scheme > &ptxt)

Generate a vector of plaintexts with each slot replicated in each plaintext.

- void `replicateAllOrig` (const `EncryptedArray` &ea, const `Ctxt` &ctx, `ReplicateHandler` \*handler, `RepAux` \*repAuxPtr=nullptr)
- void `replicate` (const `EncryptedArray` &ea, `PlaintextArray` &pa, long i)
- template<typename Scheme >  
void `replicate` (const `EncryptedArray` &, `Ptxt`< Scheme > &ptxt, long i)

Replicate single slot of a *Ptxt* object across all of its slots.

- void `sampleSmall` (`zzX` &poly, long n, double prob=0.5)
- void `sampleSmall` (NTL::ZZX &poly, long n, double prob=0.5)
- void `sampleHwt` (`zzX` &poly, long n, long Hwt=100)

Sample a degree-(n-1) poly as above, with only Hwt nonzero coefficients.

- void `sampleHwt` (NTL::ZZX &poly, long n, long Hwt=100)
- void `sampleGaussian` (`zzX` &poly, long n, double stdev)

Sample polynomials with Gaussian coefficients.

- void `sampleGaussian` (NTL::ZZX &poly, long n, double stdev)
- void `sampleUniform` (`zzX` &poly, long n, long B=100)

Sample a degree-(n-1) ZZX, with coefficients uniform in [-B,B].

- void `sampleUniform` (NTL::ZZX &poly, long n, const NTL::ZZ &B=NTL::ZZ(100L))
- void `sampleGaussian` (std::vector< double > &dvec, long n, double stdev)

Choose a vector of continuous Gaussians.

- double `sampleHwt` (`zzX` &poly, const `Context` &context, long Hwt=100)
- double `sampleHwtBounded` (`zzX` &poly, const `Context` &context, long Hwt=100)
- double `sampleHwtBoundedEffectiveBound` (const `Context` &context, long Hwt=100)
- double `sampleSmall` (`zzX` &poly, const `Context` &context)
- double `sampleSmallBounded` (`zzX` &poly, const `Context` &context)
- double `sampleGaussian` (`zzX` &poly, const `Context` &context, double stdev)
- double `sampleGaussianBounded` (`zzX` &poly, const `Context` &context, double stdev)
- double `sampleUniform` (`zzX` &poly, const `Context` &context, long B=100)
- NTL::xdouble `sampleUniform` (NTL::ZZX &poly, const `Context` &context, const NTL::ZZ &B=NTL::ZZ(100L))
- void `reduceModPhimX` (`zzX` &poly, const `PAgebra` &palg)
- const NTL::zz\_pXModulus & `getPhimXMod` (const `PAgebra` &palg)
- void `computeAllProducts` (`CtPtrs` &products, const `CtPtrs` &array, std::vector< `zzX` > \*unpackSlot, Encoding=nullptr)



- void [tableLookup](#) (Ctxt &out, const std::vector< [zzX](#) > &table, const [CtPtrs](#) &idx, std::vector< [zzX](#) > \*unpackSlotEncoding=nullptr)
- void [tableWriteIn](#) (const [CtPtrs](#) &table, const [CtPtrs](#) &idx, std::vector< [zzX](#) > \*unpackSlotEncoding=nullptr)
- void [buildLookupTable](#) (std::vector< [zzX](#) > &T, std::function< double(double)> f, long nbits\_in, long scale\_in, long sign\_in, long nbits\_out, long scale\_out, long sign\_out, const [EncryptedArray](#) &ea)  
*Built a table-lookup for a function in fixed-point representation.*
- void [registerTimer](#) ([FHEtimer](#) \*timer)
- unsigned long [GetTimerClock](#) ()
- void [setTimersOn](#) ()
- void [setTimersOff](#) ()
- bool [areTimersOn](#) ()
- const [FHEtimer](#) \* [getTimerByName](#) (const char \*name)
- void [resetAllTimers](#) ()
- void [printAllTimers](#) (std::ostream &str=std::cerr)  
*Print the value of all timers to stream.*
- bool [printNamedTimer](#) (std::ostream &str, const char \*name)
- bool [IsZero](#) (const [zzX](#) &a)
- void [clear](#) ([zzX](#) &a)
- void [convert](#) (NTL::zz\_pX &x, const [zzX](#) &a)
- void [add](#) ([zzX](#) &res, const [zzX](#) &a, const [zzX](#) &b)
- [zzX](#) [operator+](#) (const [zzX](#) &a, const [zzX](#) &b)
- [zzX](#) & [operator+=](#) ([zzX](#) &a, const [zzX](#) &b)
- void [div](#) ([zzX](#) &res, const [zzX](#) &a, long b)
- [zzX](#) [operator/](#) (const [zzX](#) &a, long b)
- [zzX](#) & [operator/=](#) ([zzX](#) &a, long b)
- void [mul](#) ([zzX](#) &res, const [zzX](#) &a, long b)
- [zzX](#) [operator\\*](#) (const [zzX](#) &a, long b)
- [zzX](#) & [operator\\*=](#) ([zzX](#) &a, long b)
- void [normalize](#) ([zzX](#) &f)
- void [MulMod](#) ([zzX](#) &res, const [zzX](#) &a, const [zzX](#) &b, const [PAlgebra](#) &palg)
- [zzX](#) [MulMod](#) (const [zzX](#) &a, const [zzX](#) &b, const [PAlgebra](#) &palg)
- [zzX](#) [balanced\\_zzX](#) (const NTL::zz\_pX &f)
- [zzX](#) [balanced\\_zzX](#) (const NTL::GF2X &f)
- long [defaultPmiddle](#) (long delta)
- long [defaultQmiddle](#) (long delta)
- void [runningSums](#) ([CtPtrs](#) &v)
- void [compareTwoNumbersImplementation](#) ([CtPtrs](#) &max, [CtPtrs](#) &min, Ctxt &mu, Ctxt &ni, const [CtPtrs](#) &aa, const [CtPtrs](#) &bb, bool twosComplement, std::vector< [zzX](#) > \*unpackSlotEncoding, bool cmp\_only)
- void [BluesteinFFT](#) (NTL::zz\_pX &x, long n, [UNUSED](#) const NTL::zz\_p &root, const NTL::zz\_pX &powers, const NTL::Vec< NTL::mulmod\_precon\_t > &powers\_aux, const NTL::fftRep &Rb)
- NTL::zz\_pContext [BuildContext](#) (long p, long maxroot)
- std::ostream & [operator<<](#) (std::ostream &str, const [Context](#) &context)
- std::istream & [operator>>](#) (std::istream &str, [Context](#) &context)
- NTL::ZZX [getG](#) (const [EncryptedArray](#) &ea)
- void [addSomePrimes](#) (Ctxt &c)
- void [computeIntervalForMul](#) (double &lo, double &hi, const Ctxt &ctxt1, const Ctxt &ctxt2)
- void [computeIntervalForSqr](#) (double &lo, double &hi, const Ctxt &ctxt)
- std::istream & [operator>>](#) (std::istream &str, [SKHandle](#) &handle)
- std::ostream & [operator<<](#) (std::ostream &str, const Ctxt &ctxt)
- std::istream & [operator>>](#) (std::istream &str, Ctxt &ctxt)
- double [log2\\_realToEstimatedNoise](#) (const Ctxt &ctxt, const [SecKey](#) &sk)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::AddFun >](#) (const [DoubleCRT](#) &other, AddFun fun, bool matchIndexSets)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::SubFun >](#) (const [DoubleCRT](#) &other, SubFun fun, bool matchIndexSets)

- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::MulFun >](#) (const NTL::ZZ &num, MulFun fun)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::AddFun >](#) (const NTL::ZZ &num, AddFun fun)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::SubFun >](#) (const NTL::ZZ &num, SubFun fun)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::MulFun >](#) (const NTL::ZZX &poly, MulFun fun)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::AddFun >](#) (const NTL::ZZX &poly, AddFun fun)
- template [DoubleCRT](#) & [DoubleCRT::Op< DoubleCRT::SubFun >](#) (const NTL::ZZX &poly, SubFun fun)
- std::ostream & [operator<<](#) (std::ostream &str, const [DoubleCRT](#) &d)
- std::istream & [operator>>](#) (std::istream &str, [DoubleCRT](#) &d)
- void [totalSums](#) (const [EncryptedArray](#) &ea, Ctxt &ctxt)
- void [applyLinPoly1](#) (const [EncryptedArray](#) &ea, Ctxt &ctxt, const std::vector< NTL::ZZX > &C)
- void [applyLinPolyMany](#) (const [EncryptedArray](#) &ea, Ctxt &ctxt, const std::vector< std::vector< NTL::ZZX >> &Cvec)
- template<typename P >  
void [applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< P > &encodedC, long d)
- template void [applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< [zzX](#) > &encodedC, long d)
- template void [applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< NTL::ZZX > &encodedC, long d)
- template void [applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< [DoubleCRT](#) > &encodedC, long d)
- void [print](#) (const [EncryptedArray](#) &ea, std::ostream &s, const [PlaintextArray](#) &pa)
- void [mapTo01](#) (const [EncryptedArray](#) &ea, Ctxt &ctxt)
- template<typename Scheme >  
void [mapTo01](#) (const [EncryptedArray](#) &, Ptxt< Scheme > &ptxt)
- template void [mapTo01](#) (const [EncryptedArray](#) &, Ptxt< [BGV](#) > &ptxt)
- template void [mapTo01](#) (const [EncryptedArray](#) &, Ptxt< [CKKS](#) > &ptxt)
- void [fastPower](#) (Ctxt &ctxt, long d)
- void [incrementalZeroTest](#) (Ctxt \*res[], const [EncryptedArray](#) &ea, const Ctxt &ctxt, long n)
- void [RelaxedInv](#) (NTL::Mat< NTL::zz\_p > &x, const NTL::Mat< NTL::zz\_p > &a)
- void [RelaxedInv](#) (NTL::Mat< NTL::GF2 > &x, const NTL::Mat< NTL::GF2 > &a)
- void [TraceMap](#) (NTL::GF2X &w, const NTL::GF2X &a, long d, const NTL::GF2XModulus &F, const NTL::GF2X &b)
- template void [getHyperColumn](#) (NTL::Vec< long > &v, const [ConstCubeSlice](#)< long > &s, long pos)
- template void [setHyperColumn](#) (const NTL::Vec< long > &v, const [CubeSlice](#)< long > &s, long pos)
- template void [setHyperColumn](#) (const NTL::Vec< long > &v, const [CubeSlice](#)< long > &s, long pos, const long &val)
- template void [print3D](#) (const [HyperCube](#)< long > &c)
- template void [getHyperColumn](#) (NTL::Vec< NTL::zz\_p > &v, const [ConstCubeSlice](#)< NTL::zz\_p > &s, long pos)
- template void [setHyperColumn](#) (const NTL::Vec< NTL::zz\_p > &v, const [CubeSlice](#)< NTL::zz\_p > &s, long pos)
- template void [setHyperColumn](#) (const NTL::Vec< NTL::zz\_p > &v, const [CubeSlice](#)< NTL::zz\_p > &s, long pos, const NTL::zz\_p &val)
- template void [print3D](#) (const [HyperCube](#)< NTL::zz\_p > &c)
- std::ostream & [operator<<](#) (std::ostream &str, const [PubKey](#) &pk)
- std::istream & [operator>>](#) (std::istream &str, [PubKey](#) &pk)
- std::ostream & [operator<<](#) (std::ostream &str, const [SecKey](#) &sk)
- std::istream & [operator>>](#) (std::istream &str, [SecKey](#) &sk)
- void [printFlow](#) ([FlowGraph](#) &fg)
- std::shared\_ptr< [GeneralAutomorphPrecon](#) > [buildGeneralAutomorphPrecon](#) (const Ctxt &ctxt, long dim, const [EncryptedArray](#) &ea)
- template<typename RX >  
std::shared\_ptr< [ConstMultiplier](#) > [build\\_ConstMultiplier](#) (const RX &poly)
- template<typename RX , typename type >  
std::shared\_ptr< [ConstMultiplier](#) > [build\\_ConstMultiplier](#) (const RX &poly, long dim, long amt, const [EncryptedArrayDerived](#)< type > &ea)
- void [MulAdd](#) (Ctxt &x, const std::shared\_ptr< [ConstMultiplier](#) > &a, const Ctxt &b)
- void [DestMulAdd](#) (Ctxt &x, const std::shared\_ptr< [ConstMultiplier](#) > &a, Ctxt &b)



- void [GenBabySteps](#) (std::vector< std::shared\_ptr< [Ctxt](#) >> &v, const [Ctxt](#) &ctxt, long dim, bool clean)
- template<typename zp , typename zz >  
void [FindPrimRootT](#) (zp &root, unsigned long e)
- template bool [intVecCRT](#) (NTL::vec\_ZZ &, const NTL::ZZ &, const NTL::vec\_ZZ &, long)
- template bool [intVecCRT](#) (NTL::vec\_ZZ &, const NTL::ZZ &, const NTL::vec\_long &, long)
- template bool [intVecCRT](#) (NTL::vec\_ZZ &, const NTL::ZZ &, const NTL::Vec< NTL::zz\_p > &, long)
- void [removeDups](#) (std::list< long > &x, bool \*aux)
- void [addOffset](#) (std::list< long > &x, long offset, long n, bool \*aux, [UNUSED](#) bool good=false)
- long [reducedCount](#) (const std::list< long > &x, long n, bool \*aux)
- void [buildBenesCostTable](#) (long n, long k, bool good, NTL::Vec< NTL::Vec< long >> &tab)
- std::ostream & [operator<<](#) (std::ostream &s, LongNodePtr p)
- BenesMemoEntry [optimalBenesAux](#) (long i, long budget, long nlev, const NTL::Vec< NTL::Vec< long >> &costTab, BenesMemoTable &memoTab)
- void [optimalBenes](#) (long n, long budget, bool good, long &cost, LongNodePtr &solution)
- void [print](#) (std::ostream &s, SplitNodePtr p, bool first)
- std::ostream & [operator<<](#) (std::ostream &s, SplitNodePtr p)
- long [length](#) (GenNodePtr ptr)
- std::ostream & [operator<<](#) (std::ostream &s, GenNodePtr p)
- LowerMemoEntry [optimalLower](#) (long order, bool good, long budget, long mid, LowerMemoTable &lower← MemoTable)
- UpperMemoEntry [optimalUpperAux](#) (const NTL::Vec< [GenDescriptor](#) > &vec, long i, long budget, long mid, UpperMemoTable &upperMemoTable, LowerMemoTable &lowerMemoTable)
- template<typename RX >  
bool [poly\\_comp](#) (const RX &a, const RX &b)
- bool [less\\_than](#) (NTL::GF2 a, NTL::GF2 b)
- bool [less\\_than](#) (NTL::zz\_p a, NTL::zz\_p b)
- bool [less\\_than](#) (const NTL::GF2X &a, const NTL::GF2X &b)
- bool [less\\_than](#) (const NTL::zz\_pX &a, const NTL::zz\_pX &b)
- bool [less\\_than](#) (const NTL::GF2E &a, const NTL::GF2E &b)
- bool [less\\_than](#) (const NTL::zz\_pE &a, const NTL::zz\_pE &b)
- bool [less\\_than](#) (const NTL::GF2EX &a, const NTL::GF2EX &b)
- bool [less\\_than](#) (const NTL::zz\_pEX &a, const NTL::zz\_pEX &b)
- bool [comparePAgebra](#) (const [PAgebra](#) &palg, unsigned long m, unsigned long p, [UNUSED](#) unsigned long r, const std::vector< long > &gens, const std::vector< long > &ords)
- template<typename T >  
void [PAgebraLift](#) (const NTL::ZZX &phimx, const T &lfactors, T &factors, T &crtc, long r)
- void [EDF](#) (NTL::vec\_zz\_pX &v, const NTL::zz\_pX &f, long d)
- NTL::zz\_pEX [FrobeniusMap](#) (const NTL::zz\_pEXModulus &F)
- void [InvModpr](#) (NTL::zz\_pX &S, const NTL::zz\_pX &F, const NTL::zz\_pX &G, long p, long r)
- template<> void [PAgebraLift](#) (const NTL::ZZX &phimx, const NTL::vec\_zz\_pX &lfactors, NTL::vec\_zz\_pX &factors, NTL::vec\_zz\_pX &crtc, long r)
- std::ostream & [operator<<](#) (std::ostream &s, const [PermNetwork](#) &net)
- template void [applyPermToVec< long >](#) (NTL::Vec< long > &out, const NTL::Vec< long > &in, const [Permut](#) &p1)
- template void [applyPermToVec< long >](#) (std::vector< long > &out, const std::vector< long > &in, const [Permut](#) &p1)
- template void [applyPermToVec< NTL::ZZX >](#) (std::vector< NTL::ZZX > &out, const std::vector< NTL::ZZX > &in, const [Permut](#) &p1)
- template void [applyPermsToVec< long >](#) (NTL::Vec< long > &out, const NTL::Vec< long > &in, const [Permut](#) &p2, const [Permut](#) &p1)
- template void [applyPermsToVec< long >](#) (std::vector< long > &out, const std::vector< long > &in, const [Permut](#) &p2, const [Permut](#) &p1)
- void [breakPermTo3](#) (const [HyperCube](#)< long > &pi, long dim, [ColPerm](#) &rho1, [HyperCube](#)< long > &rho2, [ColPerm](#) &rho3)
- void [ComputeOneGenMapping](#) ([Permut](#) &genMap, const [OneGeneratorTree](#) &T)

to a single generator tree

- `std::ostream & operator<< (std::ostream &s, const SubDimension &sd)`
- `std::ostream & operator<< (std::ostream &s, const GeneratorTrees &trees)`
- `std::istream & operator>> (std::istream &is, PolyMod &poly)`
- `std::ostream & operator<< (std::ostream &os, const PolyMod &poly)`
- `std::ostream & operator<< (std::ostream &os, const PolyModRing &ring)`
- `void computeDivVec (NTL::Vec< long > &divVec, long m, const NTL::Vec< long > &powVec)`
- `void computeInvVec (NTL::Vec< long > &invVec, const NTL::Vec< long > &divVec, const NTL::Vec< long > &powVec)`
- `bool operator> (const ModuliSizes::Entry &a, const ModuliSizes::Entry &b)`
- `std::ostream & operator<< (std::ostream &s, const ModuliSizes &szs)`
- `std::istream & operator>> (std::istream &s, ModuliSizes &szs)`
- `void addSmallPrimes (Context &context, long resolution, long cpSize)`

Add small primes to get target resolution.

- `long ctxtPrimeSize (long nBits)`
- `void addCtxtPrimes (Context &context, long nBits, long targetSize)`
- `template<typename Scheme >  
Scheme::SlotType randomSlot (const Context &context)`
- `template<> BGV::SlotType randomSlot< BGV > (const Context &context)`
- `template<> CKKS::SlotType randomSlot< CKKS > (UNUSED const Context &context)`
- `void extractDigitsPacked (Ctxt &ctxt, long botHigh, long r, long ePrime, const std::vector< NTL::ZZX > &unpackSlotEncoding)`
- `void extractDigitsThin (Ctxt &ctxt, long botHigh, long r, long ePrime)`
- `double boundRoundingNoise (UNUSED long m, long phim, long p2r, double epsilon)`
- `bool timer_compare (const FHEtimer *a, const FHEtimer *b)`

## Variables

- `Context * activeContext = nullptr`
- `SecKey * dbgKey = nullptr`
- `std::shared_ptr< const EncryptedArray > dbgEa = nullptr`
- `NTL::ZZX dbg_ptxt`
- `bool fhe_stats = false`
- `int fhe_test_force_bsgs = 0`
- `int fhe_test_force_hoist = 0`
- `const long double PI`
- `const double erfc_inverse []`
- `long thinRecrypt_initial_level`
- `long fhe_force_chen_han = 0`
- `long printFlag`
- `NTL_THREAD_LOCAL bool replicateVerboseFlag = false`
- `int fhe_watcher = 0`
- `const unsigned long CLOCK_SCALE = (unsigned long)CLOCKS_PER_SEC`

## Strategies for generating key-switching matrices

These functions are implemented in KeySwitching.cpp

- `long KSGiantStepSize (long D)`  
*Function that returns number of baby steps. Used to keep this and matmul routines "in sync".*
- `void addAllMatrices (SecKey &sKey, long keyID=0)`

- Maximalistic approach: generate matrices  $s(X^e) \rightarrow s(X)$  for all  $e$  in  $Zm^*$ .*

  - void [addFewMatrices](#) ([SecKey](#) &sKey, long keyID=0)

*Generate matrices so every  $s(X^e)$  can be reLinearized in at most two steps.*

  - void [addSome1DMatrices](#) ([SecKey](#) &sKey, long bound=HELIB\_KEYSWITCH\_THRESH, long keyID=0)

*Generate some matrices of the form  $s(X^{\{g^i\}}) \rightarrow s(X)$ , but not all. For a generator  $g$  whose order is larger than bound, generate only enough matrices for the giant-step/baby-step procedures ( $2 \cdot \sqrt{\text{ord}(g)}$ ) of them.*

  - void [add1DMatrices](#) ([SecKey](#) &sKey, long keyID=0)

*Generate all matrices  $s(X^{\{g^i\}}) \rightarrow s(X)$  for generators  $g$  of  $Zm^* / (p)$  and  $i < \text{ord}(g)$ . If  $g$  has different orders in  $Zm^*$  and  $Zm^* / (p)$  then generate also matrices of the form  $s(X^{\{g^i\} - i}) \rightarrow s(X)$*

  - void [addBSGS1DMatrices](#) ([SecKey](#) &sKey, long keyID=0)
  - void [addSomeFrbMatrices](#) ([SecKey](#) &sKey, long bound=HELIB\_KEYSWITCH\_THRESH, long keyID=0)

*Generate all/some Frobenius matrices of the form  $s(X^{\{p^i\}}) \rightarrow s(X)$*

  - void [addFrbMatrices](#) ([SecKey](#) &sKey, long keyID=0)
  - void [addBSGSFrbMatrices](#) ([SecKey](#) &sKey, long keyID=0)
  - void [addMinimal1DMatrices](#) ([SecKey](#) &sKey, long keyID=0)

*These routines just add a single matrix (or two, for bad dimensions)*

  - void [addMinimalFrbMatrices](#) ([SecKey](#) &sKey, long keyID=0)
  - void [addMatrices4Network](#) ([SecKey](#) &sKey, const [PermNetwork](#) &net, long keyID=0)
  - void [addTheseMatrices](#) ([SecKey](#) &sKey, const std::set< long > &automVals, long keyID=0)

*Generate specific key-switching matrices, described by the given set.*
- void [PolyRed](#) (NTL::ZZX &out, const NTL::ZZX &in, long q, bool abs=false)
 

*Reduce all the coefficients of a polynomial modulo  $q$ .*
- void [PolyRed](#) (NTL::ZZX &out, const NTL::ZZX &in, const NTL::ZZ &q, bool abs=false)
- void [PolyRed](#) (NTL::ZZX &F, long q, bool abs=false)
- void [PolyRed](#) (NTL::ZZX &F, const NTL::ZZ &q, bool abs=false)
- void [vecRed](#) (NTL::Vec< NTL::ZZ > &out, const NTL::Vec< NTL::ZZ > &in, long q, bool abs)
- void [vecRed](#) (NTL::Vec< NTL::ZZ > &out, const NTL::Vec< NTL::ZZ > &in, const NTL::ZZ &q, bool abs)

## Some enhanced conversion routines

- void [convert](#) (long &x1, const NTL::GF2X &x2)
  - void [convert](#) (long &x1, const NTL::zz\_pX &x2)
  - void [convert](#) (NTL::vec\_zz\_pE &X, const std::vector< NTL::ZZX > &A)
  - void [convert](#) (NTL::mat\_zz\_pE &X, const std::vector< std::vector< NTL::ZZX > > &A)
  - void [convert](#) (std::vector< NTL::ZZX > &X, const NTL::vec\_zz\_pE &A)
  - void [convert](#) (std::vector< std::vector< NTL::ZZX > > &X, const NTL::mat\_zz\_pE &A)
  - void [convert](#) (NTL::Vec< long > &out, const NTL::ZZX &in)
  - void [convert](#) (NTL::Vec< long > &out, const NTL::zz\_pX &in, bool symmetric=true)
  - void [convert](#) (NTL::Vec< long > &out, const NTL::GF2X &in)
  - void [convert](#) (NTL::ZZX &out, const NTL::Vec< long > &in)
  - void [convert](#) (NTL::GF2X &out, const NTL::Vec< long > &in)
- 
- double [boundFreshNoise](#) (long m, long phim, double sigma, double epsilon=9e-13)
  - double [boundRoundingNoise](#) (long m, long phim, long p2r, double epsilon=9e-13)

### 6.2.1 Typedef Documentation

#### 6.2.1.1 aligned\_vector

```
template<class T >
using helib::aligned_vector = typedef PGFFT::aligned_vector<T>
```

#### 6.2.1.2 cmplx\_t

```
typedef complex<double> helib::cmplx_t
```

#### 6.2.1.3 CPtrMat

```
typedef PtrMatrix<Ctxt> helib::CPtrMat
```

#### 6.2.1.4 CPtrMat\_ptVecCt

```
typedef PtrMatrix_ptVec<Ctxt> helib::CPtrMat_ptVecCt
```

#### 6.2.1.5 CPtrMat\_ptvectorCt

```
typedef PtrMatrix_ptvector<Ctxt> helib::CPtrMat_ptvectorCt
```

#### 6.2.1.6 CPtrMat\_VecCt

```
typedef PtrMatrix_Vec<Ctxt> helib::CPtrMat_VecCt
```

#### 6.2.1.7 CPtrMat\_vectorCt

```
typedef PtrMatrix_vector<Ctxt> helib::CPtrMat_vectorCt
```

### 6.2.1.8 CPtrs

```
typedef PtrVector<Ctxt> helib::CPtrs
```

### 6.2.1.9 CPtrs\_slice

```
typedef PtrVector_slice<Ctxt> helib::CPtrs_slice
```

### 6.2.1.10 CPtrs\_VecCt

```
typedef PtrVector_VecT<Ctxt> helib::CPtrs_VecCt
```

### 6.2.1.11 CPtrs\_VecPt

```
typedef PtrVector_VecPt<Ctxt> helib::CPtrs_VecPt
```

### 6.2.1.12 CPtrs\_vectorCt

```
typedef PtrVector_vectorT<Ctxt> helib::CPtrs_vectorCt
```

### 6.2.1.13 CPtrs\_vectorPt

```
typedef PtrVector_vectorPt<Ctxt> helib::CPtrs_vectorPt
```

### 6.2.1.14 cx\_double

```
typedef std::complex< double > helib::cx_double
```

### 6.2.1.15 DCRTptr

```
typedef std::shared_ptr<DoubleCRT> helib::DCRTptr
```

#### 6.2.1.16 FlowGraph

```
typedef std::vector<FNeighborList> helib::FlowGraph
```

#### 6.2.1.17 FNeighborList

```
typedef std::unordered_map<long, FlowEdge> helib::FNeighborList
```

#### 6.2.1.18 ldbl

```
typedef long double helib::ldbl
```

#### 6.2.1.19 LNeighborList

```
typedef std::unordered_multimap<long, LabeledEdge> helib::LNeighborList
```

#### 6.2.1.20 LONG

```
typedef long helib::LONG
```

#### 6.2.1.21 NodeIdx

```
typedef std::pair<long, long> helib::NodeIdx
```

#### 6.2.1.22 OneGeneratorTree

```
typedef FullBinaryTree<SubDimension> helib::OneGeneratorTree
```

### 6.2.1.23 Permut

```
typedef NTL::Vec<long> helib::Permut
```

A simple permutation is just a vector with  $p[i]=\pi_i$ .

### 6.2.1.24 zzX

```
typedef NTL::Vec<long> helib::zzX
```

### 6.2.1.25 ZZxptr

```
typedef std::shared_ptr<NTL::ZZX> helib::ZZxptr
```

## 6.2 Enumeration Type Documentation

### 6.2.2.1 PA\_tag

```
enum helib::PA_tag
```

Enumerator

PA_GF2_tag	
PA_zz_p_tag	
PA_cx_tag	

## 6.2.3 Function Documentation

### 6.2.3.1 add() [1/3]

```
void helib::add (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const PlaintextArray & other )
```

**6.2.3.2 add()** [2/3]

```
void helib::add (
    std::vector< NTL::ZZX > & x,
    const std::vector< NTL::ZZX > & a,
    const std::vector< NTL::ZZX > & b )
```

**6.2.3.3 add()** [3/3]

```
void helib::add (
    ZZ & res,
    const ZZ & a,
    const ZZ & b )
```

**6.2.3.4 add1DMatrices()**

```
void helib::add1DMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

Generate all matrices  $s(X^{g^i}) \rightarrow s(X)$  for generators  $g$  of  $Zm^*/(p)$  and  $i < \text{ord}(g)$ . If  $g$  has different orders in  $Zm^*$  and  $Zm^*/(p)$  then generate also matrices of the form  $s(X^{g^{-i}}) \rightarrow s(X)$

**6.2.3.5 addAllMatrices()**

```
void helib::addAllMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

Maximalistic approach: generate matrices  $s(X^e) \rightarrow s(X)$  for all  $e$  in  $Zm^*$ .

**6.2.3.6 addBSGS1DMatrices()**

```
void helib::addBSGS1DMatrices (
    SecKey & sKey,
    long keyID = 0 )
```



### 6.2.3.7 addBSGSFrbMatrices()

```
void helib::addBSGSFrbMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

### 6.2.3.8 addCtxtPrimes()

```
void helib::addCtxtPrimes (
    Context & context,
    long nBits,
    long targetSize )
```

### 6.2.3.9 addFewMatrices()

```
void helib::addFewMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

Generate matrices so every  $s(X^e)$  can be reLinearized in at most two steps.

### 6.2.3.10 addFrbMatrices()

```
void helib::addFrbMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

### 6.2.3.11 addManyNumbers()

```
void helib::addManyNumbers (
    CTPtrs & sum,
    CTPtrMat & numbers,
    long sizeLimit = 0,
    std::vector< ZZ > * unpackSlotEncoding = nullptr )
```

Sum an arbitrary amount of numbers in binary representation.

## Parameters

<i>sum</i>	result of the summation.
<i>numbers</i>	values of which to sum.
<i>sizeLimit</i>	number of bits to compute on, taken from the least significant end.
<i>unpackSlotEncoding</i>	vector of constants for unpacking, as used in bootstrapping.

Calculates the sum of many numbers using the 3-for-2 method.

#### 6.2.3.12 addMatrices4Network()

```
void helib::addMatrices4Network (
    SecKey & sKey,
    const PermNetwork & net,
    long keyID = 0 )
```

#### 6.2.3.13 addMinimal1DMatrices()

```
void helib::addMinimal1DMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

These routines just add a single matrix (or two, for bad dimensions)

**6.2.3.14 addMinimalFrbMatrices()**

```
void helib::addMinimalFrbMatrices (
    SecKey & sKey,
    long keyID = 0 )
```

**6.2.3.15 addOffset()**

```
void helib::addOffset (
    std::list< long > & x,
    long offset,
    long n,
    bool * aux,
    UNUSED bool good = false )
```

**6.2.3.16 addSmallPrimes()**

```
void helib::addSmallPrimes (
    Context & context,
    long resolution,
    long cpSize )
```

Add small primes to get target resolution.

**6.2.3.17 addSome1DMatrices()**

```
void helib::addSome1DMatrices (
    SecKey & sKey,
    long bound = HELIB_KEYSWITCH_THRESH,
    long keyID = 0 )
```

Generate some matrices of the form  $s(X^{\{g^i\}} \rightarrow s(X)$ , but not all. For a generator  $g$  whose order is larger than bound, generate only enough matrices for the giant-step/baby-step procedures ( $2 \cdot \sqrt{\text{ord}(g)}$ ) of them).

**6.2.3.18 addSomeFrbMatrices()**

```
void helib::addSomeFrbMatrices (
    SecKey & sKey,
    long bound = HELIB_KEYSWITCH_THRESH,
    long keyID = 0 )
```

Generate all/some Frobenius matrices of the form  $s(X^{\{p^i\}} \rightarrow s(X)$

**6.2.3.19 addSomePrimes()**

```
void helib::addSomePrimes (
    Ctxt & c )
```

**6.2.3.20 addTheseMatrices()**

```
void helib::addTheseMatrices (
    SecKey & sKey,
    const std::set< long > & automVals,
    long keyID = 0 )
```

Generate specific key-switching matrices, described by the given set.

**6.2.3.21 addTwoNumbers()**

```
void helib::addTwoNumbers (
    CtxtPtrs & sum,
    const CtxtPtrs & lhs,
    const CtxtPtrs & rhs,
    long sizeLimit = 0,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

Adds two numbers in binary representation where each ciphertext of the input vector contains a bit.

Add two integers in binary representation.

**Parameters**

<i>sum</i>	result of the addition operation.
<i>lhs</i>	left hand side of the addition.
<i>rhs</i>	right hand side of the addition.

## Parameters

<i>sizeLimit</i>	number of bits to compute on, taken from the least significant end.
<i>unpackSlotEncoding</i>	vector of constants for unpacking, as used in bootstrapping.

**6.2.3.22 applyLinPoly()** [1/2]

```
void helib::applyLinPoly (
    NTL::GF2E & beta,
    const NTL::vec_GF2E & C,
    const NTL::GF2E & alpha,
    long p )
```

A version for GF2: must be called with  $p == 2$  and  $r == 1$ .

**6.2.3.23 applyLinPoly()** [2/2]

```
void helib::applyLinPoly (
    NTL::zz_pE & beta,
    const NTL::vec_zz_pE & C,
    const NTL::zz_pE & alpha,
    long p )
```

Apply a linearized polynomial with coefficient vector C.

[NTL](#)'s current smallint modulus, `zz_p::modulus()`, is assumed to be  $p^r$ , for  $p$  prime,  $r \geq 1$  integer.

**6.2.3.24 applyLinPoly1()**

```
void helib::applyLinPoly1 (
    const EncryptedArray & ea,
    Ctxt & ctxt,
    const std::vector< NTL::ZZX > & C )
```

**6.2.3.25 applyLinPolyLL() [1/4]**

```
template void helib::applyLinPolyLL (
    Ctxt & ctxt,
    const std::vector< DoubleCRT > & encodedC,
    long d )
```

**6.2.3.26 applyLinPolyLL() [2/4]**

```
template void helib::applyLinPolyLL (
    Ctxt & ctxt,
    const std::vector< NTL::ZZX > & encodedC,
    long d )
```

**6.2.3.27 applyLinPolyLL() [3/4]**

```
template<typename P >
void helib::applyLinPolyLL (
    Ctxt & ctxt,
    const std::vector< P > & encodedC,
    long d )
```

**6.2.3.28 applyLinPolyLL() [4/4]**

```
template void helib::applyLinPolyLL (
    Ctxt & ctxt,
    const std::vector< zzX > & encodedC,
    long d )
```

**6.2.3.29 applyLinPolyMany()**

```
void helib::applyLinPolyMany (
    const EncryptedArray & ea,
    Ctxt & ctxt,
    const std::vector< std::vector< NTL::ZZX >> & Cvec )
```

**6.2.3.30 applyPerm()**

```
void helib::applyPerm (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const NTL::Vec< long > & pi )
```

**6.2.3.31 applyPermsToVec() [1/2]**

```
template<typename T >
void helib::applyPermsToVec (
    NTL::Vec< T > & out,
    const NTL::Vec< T > & in,
    const Permut & p2,
    const Permut & p1 )
```

Apply two permutations to a std::vector out[i]=in[p2[p1[i]]] (NOT in-place)

**6.2.3.32 applyPermsToVec() [2/2]**

```
template<typename T >
void helib::applyPermsToVec (
    std::vector< T > & out,
    const std::vector< T > & in,
    const Permut & p2,
    const Permut & p1 )
```

**6.2.3.33 applyPermsToVec< long >() [1/2]**

```
template void helib::applyPermsToVec< long > (
    NTL::Vec< long > & out,
    const NTL::Vec< long > & in,
    const Permut & p2,
    const Permut & p1 )
```

**6.2.3.34 applyPermsToVec< long >() [2/2]**

```
template void helib::applyPermsToVec< long > (
    std::vector< long > & out,
    const std::vector< long > & in,
    const Permut & p2,
    const Permut & p1 )
```

**6.2.3.35 applyPermToVec() [1/2]**

```
template<typename T >
void helib::applyPermToVec (
    NTL::Vec< T > & out,
    const NTL::Vec< T > & in,
    const Permut & p1 )
```

Apply a permutation to a std::vector, out[i]=in[p1[i]] (NOT in-place)

**6.2.3.36 applyPermToVec() [2/2]**

```
template<typename T >
void helib::applyPermToVec (
    std::vector< T > & out,
    const std::vector< T > & in,
    const Permut & p1 )
```

**6.2.3.37 applyPermToVec< long >() [1/2]**

```
template void helib::applyPermToVec< long > (
    NTL::Vec< long > & out,
    const NTL::Vec< long > & in,
    const Permut & p1 )
```

**6.2.3.38 applyPermToVec< long >() [2/2]**

```
template void helib::applyPermToVec< long > (
    std::vector< long > & out,
    const std::vector< long > & in,
    const Permut & p1 )
```



**6.2.3.39 applyPermToVec< NTL::ZZX >()**

```
template void helib::applyPermToVec< NTL::ZZX > (
    std::vector< NTL::ZZX > & out,
    const std::vector< NTL::ZZX > & in,
    const Permut & p1 )
```

**6.2.3.40 areTimersOn()**

```
bool helib::areTimersOn ( ) [inline]
```

**6.2.3.41 argmax() [1/2]**

```
long helib::argmax (
    std::vector< long > & v,
    bool(*) (long, long) moreThan ) [inline]
```

A variant with a specialized comparison function (\*moreThan)(a,b) returns the comparison a>b.

**6.2.3.42 argmax() [2/2]**

```
template<typename T >
long helib::argmax (
    std::vector< T > & v )
```

**6.2.3.43 argmin()**

```
template<typename T >
long helib::argmin (
    std::vector< T > & v )
```

**6.2.3.44 argminmax()**

```
template<typename T , bool maxFlag>
long helib::argminmax (
    std::vector< T > & v )
```

Find the index of the (first) largest/smallest element.

These procedures are roughly just simpler variants of `std::max_element` and `std::min_element`. `argmin/argmax` are implemented as a template, so the code must be placed in the header file for the compiler to find it. The class `T` must have an implementation of `operator>` and `operator<` for this template to work.

## Template Parameters

<i>maxFlag</i>	A boolean value: true - argmax, false - argmin
----------------	--

## 6.2.3.45 assertEq()

```
template<typename ExceptionTy = ::helib::LogicError, typename T = void>
void helib::assertEq (
    const T & a,
    const T & b,
    const std::string & message ) [inline]
```

Function throwing an exception of type ExceptionTy if the two arguments are not equal.

## Template Parameters

<i>ExceptionTy</i>	type of the exception thrown.
<i>T</i>	type of the elements to be compared.

## Parameters

<i>a</i>	the first element to be compared.
<i>b</i>	the second element to be compared.
<i>message</i>	the message of the exception raised if the two values are not equal.

## Exceptions

<i>ExceptionTy</i>	exception if the two values are not equal.
--------------------	--

**Note**

ExceptionTy first and T defaulted to void so that one can specify only ExceptionTy, letting T be inferred from the argument passed.

**6.2.3.46 assertFalse()**

```
template<typename ExceptionTy = ::helib::LogicError, typename T = void>
void helib::assertFalse (
    T value,
    const std::string & message ) [inline]
```

Function throwing an exception of type ExceptionTy if the condition is true.

**Template Parameters**

<i>ExceptionTy</i>	type of the exception thrown.
<i>T</i>	type of the condition being checked (must be a bool).

**Parameters**

<i>value</i>	the condition being checked.
<i>message</i>	the message of the exception raised if the condition is true.

**Exceptions**

<i>ExceptionTy</i>	exception if condition is true.
--------------------	---------------------------------

**Note**

ExceptionTy first and T defaulted to void so that one can specify only ExceptionTy, letting T be inferred from the argument passed.

### 6.2.3.47 assertInRange()

```
template<typename ExceptionTy = ::helib::OutOfRangeError, typename T = void>
void helib::assertInRange (
    const T & elem,
    const T & min,
    const T & max,
    const std::string & message,
    bool right_inclusive = false ) [inline]
```

Function throwing an exception of type ExceptionTy if the element is in the range [min,max) or [min, max]

#### Template Parameters

<i>ExceptionTy</i>	type of the exception thrown.
<i>T</i>	type of the element (and of the range).

#### Parameters

<i>elem</i>	the element to be tested.
<i>min</i>	the left side of the range (always inclusive).
<i>max</i>	the right side of the range (default exclusive).
<i>message</i>	the message of the exception raised if the element is not in the range.

## Parameters

<i>right_inclusive</i>	flag specifying if the right side is inclusive (default false).
------------------------	---

## Exceptions

<i>ExceptionTy</i>	exception if elem is not in the range
--------------------	---------------------------------------

## Note

ExceptionTy first and T defaulted to void so that one can specify only ExceptionTy, letting T be inferred from the argument passed.

## 6.2.3.48 assertNeq()

```
template<typename ExceptionTy = ::helib::LogicError, typename T = void>
void helib::assertNeq (
    const T & a,
    const T & b,
    const std::string & message ) [inline]
```

Function throwing an exception of type ExceptionTy if the two arguments are equal.

## Template Parameters

<i>ExceptionTy</i>	type of the exception thrown.
<i>T</i>	type of the elements to be compared.

## Parameters

<i>a</i>	the first element to be compared.
----------	-----------------------------------

**Parameters**

<i>b</i>	the second element to be compared.
<i>message</i>	the message of the exception raised if the two values are equal.

**Exceptions**

<i>ExceptionTy</i>	exception if the two values are equal.
--------------------	--

**Note**

ExceptionTy first and T defaulted to void so that one can specify only ExceptionTy, letting T be inferred from the argument passed.

**6.2.3.49 assertNotNull()**

```
template<typename ExceptionTy = ::helib::LogicError, typename T = void>
void helib::assertNotNull (
    const T & p,
    const std::string & message ) [inline]
```

Function throwing an exception of type ExceptionTy if the argument is nullptr.

**Template Parameters**

<i>ExceptionTy</i>	type of the exception thrown.
<i>T</i>	type of the element.

## Parameters

<i>p</i>	the element to be tested.
<i>message</i>	the message of the exception raised if the element is nullptr.

## Exceptions

<i>ExceptionTy</i>	exception if p is nullptr.
--------------------	----------------------------

## Note

ExceptionTy first and T defaulted to void so that one can specify only ExceptionTy, letting T be inferred from the argument passed.

## 6.2.3.50 assertTrue()

```
template<typename ExceptionTy = ::helib::LogicError, typename T = void>
void helib::assertTrue (
    const T & value,
    const std::string & message ) [inline]
```

Function throwing an exception of type ExceptionTy if the condition is false.

## Template Parameters

<i>ExceptionTy</i>	type of the exception thrown.
<i>T</i>	type of the condition being checked (must be a bool).

## Parameters

<i>value</i>	the condition being checked.
--------------	------------------------------

**Parameters**

<i>message</i>	the message of the exception raised if the condition is false.
----------------	--

**Exceptions**

<i>ExceptionTy</i>	exception if condition is false.
--------------------	----------------------------------

**Note**

ExceptionTy first and T defaulted to void so that one can specify only ExceptionTy, letting T be inferred from the argument passed.

**6.2.3.51 atoVec()**

```
template<typename T >
Ntl::Vec<T> helib::atoVec (
    const char * a )
```

**6.2.3.52 atovector()**

```
template<typename T >
std::vector<T> helib::atovector (
    const char * a )
```

**6.2.3.53 balanced\_MulMod()**

```
void helib::balanced_MulMod (
    Ntl::ZZX & out,
    const Ntl::ZZX & f,
    long a,
    long q )
```

Multiply the polynomial f by the integer a modulo q output coefficients are balanced (appropriately randomized for even q)



**6.2.3.54 balanced\_zzX()** [1/2]

```
zzX helib::balanced_zzX (
    const NTL::GF2X & f )
```

**6.2.3.55 balanced\_zzX()** [2/2]

```
zzX helib::balanced_zzX (
    const NTL::zz_pX & f )
```

**6.2.3.56 balRem()**

```
long helib::balRem (
    long a,
    long q ) [inline]
```

Return balanced remainder. Assumes  $a$  in  $[0, q)$  and returns balanced remainder in  $(-q/2, q/2]$

**6.2.3.57 binaryCond()**

```
void helib::binaryCond (
    CtPtrs & output,
    const Ctxt & cond,
    const CtPtrs & trueValue,
    const CtPtrs & falseValue )
```

Implementation of  $output = cond * trueValue + (1 - cond) * falseValue$ .

Implementation of  $output = cond ? trueValue : falseValue$ .

**Parameters**

<i>output</i>	Equal to $trueValue$ in slots where $cond$ is one and $falseValue$ in slots where $cond$ is zero.
---------------	---

## Parameters

<i>cond</i>	The condition, namely a <code>Ctxt</code> containing elements of $\{0,1\}$ in each slot.
<i>trueValue</i>	Value of output wherever <i>cond</i> is one.
<i>falseValue</i>	Value of output wherever <i>cond</i> is zero.

## Note

*trueValue*, *falseValue* and *output* must have the same size.

**6.2.3.58 binaryMask()**

```
void helib::binaryMask (
    CPtrs & binaryNums,
    const Ctxt & mask )
```

Zeroes the slots of *binaryNums* where the corresponding slot of *mask* is 0.

Apply mask across the vector of bits slot-wise.

## Parameters

<i>binaryNums</i>	Input bits on which to mask (this is done in place).
<i>mask</i>	Encrypted mask indicating desired slots.

## 6.2.3.59 bitSetToLong()

```
long helib::bitSetToLong (
    long bits,
    long bitSize )
```

Considers *bits* as a vector of bits and returns the value it represents when interpreted as a n-bit 2's complement number, where n is given by *bitSize*.

## Parameters

<i>bits</i>	The value containing the bits to be reinterpreted.
<i>bitSize</i>	The number of bits to use, taken from the least significant end of <i>bits</i> .

**Returns**

The value of the reinterpreted number as a long.

**6.2.3.60 bitwiseAnd() [1/2]**

```
void helib::bitwiseAnd (
    CtPtrs & output,
    const CtPtrs & input,
    const std::vector< long > mask )
```

Compute a bitwise AND between `input` and a `std::vector<long>`.

Compute a bitwise AND between `input` and `mask`.

**Parameters**

<i>output</i>	Equal to the output of the AND operation.
<i>input</i>	Number to $A \leftrightarrow$ ND.
<i>mask</i>	Number to $A \leftrightarrow$ ND with <code>input</code> . This should be a vector of elements of <code>{0,1}</code> .

**Note**

The size of `output` and `input` must be the same.

**6.2.3.61 bitwiseAnd() [2/2]**

```
void helib::bitwiseAnd (
    CtPtrs & output,
```

```
const CtPtrs & lhs,  
const CtPtrs & rhs )
```

Compute a bitwise AND between lhs and rhs.

## Parameters

<i>output</i>	Result of bit-wise lhs AND rhs.
<i>lhs</i>	Left operand to the AND operation.
<i>rhs</i>	Right operand to the AND operation.

## Note

`output`, `lhs` and `rhs` must all have the same size.

**6.2.3.62 bitwiseNot()**

```
void helib::bitwiseNot (
    CtPtrs & output,
    const CtPtrs & input )
```

Compute a bitwise NOT of `input`.

## Parameters

<i>output</i>	Result of bit-flipping <code>input</code> .
<i>input</i>	Binary number to be bit-flipped.

## Note

The size of `output` and `input` must be the same.

### 6.2.3.63 bitwiseOr()

```
void helib::bitwiseOr (
    CtPtrs & output,
    const CtPtrs & lhs,
    const CtPtrs & rhs )
```

Compute a bitwise OR between `lhs` and `rhs`.

#### Parameters

<i>output</i>	Result of bit-wise OR <code>lhs</code> and <code>rhs</code> .
<i>lhs</i>	Left operand to the OR operation.
<i>rhs</i>	Right operand to the OR operation.

#### Note

`output`, `lhs` and `rhs` must all have the same size.

### 6.2.3.64 bitwiseRotate()

```
void helib::bitwiseRotate (
    CtPtrs & output,
    const CtPtrs & input,
    long rotamt )
```

Rotate `input` by `rotamt`.

Rotate binary numbers by `rotamt`.

#### Parameters

<i>output</i>	Rotated result.
<i>input</i>	The number to be bit-wise-rotated.

## Parameters

<i>rotamt</i>	The amount by which to rotate input. May be negative for opposite-direction rotations.
---------------	--

## Note

For positive `rotamt` arguments, this rotates towards the most-significant end (i.e. the same direction as `leftBitwiseShift`).

The size of `output` and `input` must be the same.

## 6.2.3.65 bitwiseXOR()

```
void helib::bitwiseXOR (
    CtPtrs & output,
    const CtPtrs & lhs,
    const CtPtrs & rhs )
```

Compute a bitwise XOR between `lhs` and `rhs`.

## Parameters

<i>output</i>	Result of bitwise <code>lhs XOR rhs</code> .
<i>lhs</i>	Left operand to the XOR operation.
<i>rhs</i>	Right operand to the XOR operation.



**Note**

output, lhs and rhs must all have the same size.

**6.2.3.66 BluesteinFFT() [1/2]**

```
void helib::BluesteinFFT (
    NTL::zz_pX & x,
    long n,
    const NTL::zz_p & root,
    const NTL::zz_pX & powers,
    const NTL::Vec< NTL::mulmod_precon_t > & powers_aux,
    const NTL::fftRep & Rb )
```

apply bluestein

**6.2.3.67 BluesteinFFT() [2/2]**

```
void helib::BluesteinFFT (
    NTL::zz_pX & x,
    long n,
    UNUSED const NTL::zz_p & root,
    const NTL::zz_pX & powers,
    const NTL::Vec< NTL::mulmod_precon_t > & powers_aux,
    const NTL::fftRep & Rb )
```

**6.2.3.68 BluesteinInit()**

```
void helib::BluesteinInit (
    long n,
    const NTL::zz_p & root,
    NTL::zz_pX & powers,
    NTL::Vec< NTL::mulmod_precon_t > & powers_aux,
    NTL::fftRep & Rb )
```

initialize bluestein

**6.2.3.69 boundFreshNoise()**

```
double helib::boundFreshNoise (
    long m,
    long phim,
    double sigma,
    double epsilon = 9e-13 )
```

Helper functions, return a bound B such that for random noise terms we have  $\Pr[\text{canonicalEmbed}(\text{noise})_{\infty} > B] < \epsilon$ . (The default is  $\epsilon = 2^{-40}$ .)

**6.2.3.70 boundRoundingNoise()** [1/2]

```
double helib::boundRoundingNoise (
    long m,
    long phim,
    long p2r,
    double epsilon = 9e-13 )
```

**6.2.3.71 boundRoundingNoise()** [2/2]

```
double helib::boundRoundingNoise (
    UNUSED long m,
    long phim,
    long p2r,
    double epsilon )
```

**6.2.3.72 breakPermByDim()**

```
void helib::breakPermByDim (
    std::vector< ColPerm > & out,
    const Permut & pi,
    const CubeSignature & sig )
```

Takes a permutation  $\pi$  over  $m$ -dimensional cube  $C = Z_{\{n_1\}} \times \dots \times Z_{\{n_m\}}$  and expresses  $\pi$  as a product  $\pi = \rho_{m-1} \circ \dots \circ \rho_2 \circ \rho_1$  where each  $\rho_i$  is a column permutation along one dimension. Specifically for  $i < m$ , the permutations  $\rho_i$  and  $\rho_{2(m-1)-i}$  permute the  $i$ 'th dimension.

**6.2.3.73 breakPermTo3()**

```
void helib::breakPermTo3 (
    const HyperCube< long > & pi,
    long dim,
    ColPerm & rho1,
    HyperCube< long > & rho2,
    ColPerm & rho3 )
```

**6.2.3.74 build\_ConstMultiplier()** [1/2]

```
template<typename RX >
std::shared_ptr<ConstMultiplier> helib::build_ConstMultiplier (
    const RX & poly )
```

**6.2.3.75 build\_ConstMultiplier() [2/2]**

```
template<typename RX , typename type >
std::shared_ptr<ConstMultiplier> helib::build_ConstMultiplier (
    const RX & poly,
    long dim,
    long amt,
    const EncryptedArrayDerived< type > & ea )
```

**6.2.3.76 buildBenesCostTable()**

```
void helib::buildBenesCostTable (
    long n,
    long k,
    bool good,
    NTL::Vec< NTL::Vec< long >> & tab )
```

**6.2.3.77 BuildContext()**

```
NTL::zz_pContext helib::BuildContext (
    long p,
    long maxroot )
```

**6.2.3.78 buildContextFromAscii()**

```
std::unique_ptr< Context > helib::buildContextFromAscii (
    std::istream & str )
```

**6.2.3.79 buildContextFromBinary()**

```
std::unique_ptr< Context > helib::buildContextFromBinary (
    std::istream & str )
```

**6.2.3.80 buildEncryptedArray()**

```
EncryptedArrayBase * helib::buildEncryptedArray (
    const Context & context,
    const PAlgebraMod & alMod,
    const NTL::ZZX & G = NTL::ZZX::zero() )
```

A "factory" for building EncryptedArrays.

**6.2.3.81 buildGeneralAutomorphPrecon()**

```
std::shared_ptr<GeneralAutomorphPrecon> helib::buildGeneralAutomorphPrecon (
    const Ctxt & ctxt,
    long dim,
    const EncryptedArray & ea )
```

**6.2.3.82 buildLinPolyCoeffs() [1/2]**

```
void helib::buildLinPolyCoeffs (
    NTL::vec_GF2E & C,
    const NTL::vec_GF2E & L,
    long p,
    long r )
```

A version for GF2: must be called with  $p == 2$  and  $r == 1$ .

**6.2.3.83 buildLinPolyCoeffs() [2/2]**

```
void helib::buildLinPolyCoeffs (
    NTL::vec_zz_pE & C,
    const NTL::vec_zz_pE & L,
    long p,
    long r )
```

Combination of buildLinPolyMatrix and ppsolve.

Obtain the linearized polynomial coefficients from a vector L representing the action of a linear map on the standard basis for  $zz\_pE$  over  $zz\_p$ .

NTL's current smallint modulus,  $zz\_p::modulus()$ , is assumed to be  $p^r$ , for  $p$  prime,  $r \geq 1$  integer.

**6.2.3.84 buildLinPolyMatrix() [1/2]**

```
void helib::buildLinPolyMatrix (
    NTL::mat_GF2E & M,
    long p )
```

**6.2.3.85 buildLinPolyMatrix() [2/2]**

```
void helib::buildLinPolyMatrix (
    NTL::mat_zz_pE & M,
    long p )
```

**6.2.3.86 buildLookupTable()**

```
void helib::buildLookupTable (
    std::vector< zzX > & T,
    std::function< double(double)> f,
    long nbits_in,
    long scale_in,
    long sign_in,
    long nbits_out,
    long scale_out,
    long sign_out,
    const EncryptedArray & ea )
```

Built a table-lookup for a function in fixed-point representation.

@function buildLookupTable FIXED-POINT CONVENTIONS: Fixed-point numbers are specified by a triple (nbits,scale,signed). Such a number is represented as an integer  $x$  with  $\text{nbits}$  bits. If  $\text{signed} == 1$ , then  $x$  is treated as a signed integer in 2's complement; otherwise it is as an unsigned integer. The value represented by  $x$  is  $x \cdot 2^{\text{scale}}$ .

The buildLookupTable function builds a lookup table  $T$ , which can be used in conjunction with the tableLookup function above. The size of  $T$  will be  $2^{\text{nbits\_in}}$ . For every signed integer  $x$  with bit-size ' $\text{nbits\_in}$ ', we will have  $T[x] = f(x \cdot 2^{\text{scale\_in}}) \cdot 2^{-\text{scale\_out}}$ , rounded to the nearest integer and truncated to ' $\text{nbits\_out}$ ' bits. The bits are packed inside the slots, so it is assumed that each slot has enough room to fit these many bits. (Otherwise we only keep as many low-order bits as fit in a slot.)

SATURATED ARITHMETIC: Applications of  $f$  that return a result that is too large to represent in the output format will be converted to the maximum representable value. Similarly, Applications of  $f$  that return a result that is too small will be converted to the minimal representable value. (This applies also to applications of  $f$  that return infinities, NaNs will just be mapped to zero.) For this to work correctly, you should be working with standard IEEE arithmetic...which will be the case on almost all platforms.

EXAMPLE:

```
buildLookupTable(T, [](double x){ return 1/x;}, nbits_in, scale_in, nbits_out, scale_out, sign_out, ea)
```

will build a lookup table for inversion.

**6.2.3.87 buildModChain()**

```
void helib::buildModChain (
    Context & context,
    long nBits,
    long nDgts = 3,
    bool willBeBootstrappable = false,
    long skHwt = 0,
    long resolution = 3,
    long bitsInSpecialPrimes = 0 )
```

**6.2.3.88 buildPALgebraMod()**

```
PAlgebraModBase * helib::buildPALgebraMod (
    const PAlgebra & zMStar,
    long r )
```

Builds a table, of type PA\_GF2 if  $p == 2$  and  $r == 1$ , and PA\_zz\_p otherwise.

#### 6.2.3.89 buildRandomBlockMatrix()

```
BlockMatMul1D * helib::buildRandomBlockMatrix (
    const EncryptedArray & ea,
    long dim )
```

#### 6.2.3.90 buildRandomFullBlockMatrix()

```
BlockMatMulFull * helib::buildRandomFullBlockMatrix (
    const EncryptedArray & ea )
```

#### 6.2.3.91 buildRandomFullMatrix()

```
MatMulFull * helib::buildRandomFullMatrix (
    const EncryptedArray & ea )
```

#### 6.2.3.92 buildRandomMatrix()

```
MatMul1D * helib::buildRandomMatrix (
    const EncryptedArray & ea,
    long dim )
```

#### 6.2.3.93 buildRandomMultiBlockMatrix()

```
BlockMatMul1D * helib::buildRandomMultiBlockMatrix (
    const EncryptedArray & ea,
    long dim )
```

#### 6.2.3.94 buildRandomMultiMatrix()

```
MatMul1D * helib::buildRandomMultiMatrix (
    const EncryptedArray & ea,
    long dim )
```

### 6.2.3.95 buildUnpackSlotEncoding()

```
void helib::buildUnpackSlotEncoding (
    std::vector< zzX > & unpackSlotEncoding,
    const EncryptedArray & ea )
```

### 6.2.3.96 calcPolyNormBnd()

```
double helib::calcPolyNormBnd (
    long m )
```

### 6.2.3.97 card()

```
long helib::card (
    const IndexSet & s )
```

Functional cardinality.

### 6.2.3.98 CheckCtxt()

```
void helib::CheckCtxt (
    const Ctxt & c,
    const char * label )
```

print to cerr some info about ciphertext

### 6.2.3.99 checkNoise()

```
void helib::checkNoise (
    const Ctxt & ctxt,
    const SecKey & sk,
    const std::string & msg,
    double thresh = 10.0 )
```

### 6.2.3.100 CKKS\_canonicalEmbedding() [1/3]

```
void helib::CKKS_canonicalEmbedding (
    std::vector< cx_double > & v,
    const NTL::ZZX & f,
    const PAlgebra & palg )
```

**6.2.3.101 CKKS\_canonicalEmbedding() [2/3]**

```
void helib::CKKS_canonicalEmbedding (
    std::vector< cx_double > & v,
    const std::vector< double > & f,
    const PAlgebra & palg )
```

**6.2.3.102 CKKS\_canonicalEmbedding() [3/3]**

```
void helib::CKKS_canonicalEmbedding (
    std::vector< cx_double > & v,
    const zzX & f,
    const PAlgebra & palg )
```

Computes canonical embedding. Requires  $p=-1$  and  $m=2^k$  where  $k \geq 2$  and  $f.length() < m/2$ . Sets  $v[m/4-1-i] = \text{DFT}[palg.ith\_rep(i)]$  for  $i$  in  $\text{range}(m/4)$ , where  $\text{DFT}[j] = f(W^j)$  for  $j$  in  $\text{range}(m)$ , and  $W = \exp(-2\pi i/m)$ .

**6.2.3.103 CKKS\_embedInSlots()**

```
void helib::CKKS_embedInSlots (
    zzX & f,
    const std::vector< cx_double > & v,
    const PAlgebra & palg,
    double scaling )
```

Requires  $p=-1$  and  $m=2^k$  where  $k \geq 2$ . Computes the inverse of canonical embedding, scaled by scaling and then rounded to nearest integer.

**6.2.3.104 cleanupDebugGlobals()**

```
void helib::cleanupDebugGlobals ( ) [inline]
```

Cleanup function for clearing the global debug variables.

**6.2.3.105 clear()**

```
void helib::clear (
    zzX & a ) [inline]
```

**6.2.3.106 closeToOne()**

```
bool helib::closeToOne (
    const NTL::xdouble & x,
    long p ) [inline]
```



**6.2.3.107 coeffsL2Norm()** [1/3]

```
NTL::xdouble helib::coeffsL2Norm (
    const DoubleCRT & f ) [inline]
```

**6.2.3.108 coeffsL2Norm()** [2/3]

```
NTL::xdouble helib::coeffsL2Norm (
    const NTL::ZZX & f ) [inline]
```

**6.2.3.109 coeffsL2Norm()** [3/3]

```
double helib::coeffsL2Norm (
    const ZZ & f ) [inline]
```

**6.2.3.110 coeffsL2NormSquared()** [1/3]

```
NTL::xdouble helib::coeffsL2NormSquared (
    const DoubleCRT & f )
```

**6.2.3.111 coeffsL2NormSquared()** [2/3]

```
NTL::xdouble helib::coeffsL2NormSquared (
    const NTL::ZZX & f )
```

**6.2.3.112 coeffsL2NormSquared()** [3/3]

```
double helib::coeffsL2NormSquared (
    const ZZ & f )
```

The L2-norm of an element (in coefficient representation)

**6.2.3.113 comparePALgebra()** [1/2]

```
bool helib::comparePALgebra (
    const PALgebra & palg,
    unsigned long m,
    unsigned long p,
    unsigned long r,
    const std::vector< long > & gens,
    const std::vector< long > & ords )
```

returns true if the palg parameters match the rest, false otherwise

**6.2.3.114 comparePALgebra()** [2/2]

```
bool helib::comparePALgebra (
    const PALgebra & palg,
    unsigned long m,
    unsigned long p,
    UNUSED unsigned long r,
    const std::vector< long > & gens,
    const std::vector< long > & ords )
```

**6.2.3.115 compareTwoNumbers()** [1/2]

```
void helib::compareTwoNumbers (
    CTPtrs & max,
    CTPtrs & min,
    Ctxt & mu,
    Ctxt & ni,
    const CTPtrs & a,
    const CTPtrs & b,
    bool twosComplement = false,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

Compares two integers in binary a, b. Returns max(a, b), min(a, b) and indicator bits mu=(a>b) and ni=(a<b)

**Parameters**

<i>max</i>	Maximum of a and b.
<i>min</i>	Minimum of a and b.
<i>mu</i>	Indicator bits mu=(a>b).
<i>ni</i>	Indicator bits ni=(a<b).

## Parameters

<i>a</i>	First number to compare.
<i>b</i>	Second number to compare.
<i>twosComplement</i>	When set to <code>true</code> , the inputs are signed integers in 2's complement. If set to <code>false</code> (default), unsigned comparison is performed.
<i>unpackSlotEncoding</i>	Vector of constants for unpacking, as used in bootstrapping.

## Note

If  $a=b$  then  $\mu=n_i=0$

**6.2.3.116 compareTwoNumbers()** [2/2]

```
void helib::compareTwoNumbers (
    Ctxt & mu,
    Ctxt & ni,
    const CPtrs & a,
    const CPtrs & b,
    bool twosComplement = false,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

Compares two integers in binary a, b. Returns only indicator bits mu=(a>b) and ni=(a<b).

**Parameters**

<i>mu</i>	Indicator bits mu=(a>b).
<i>ni</i>	Indicator bits ni=(a<b).
<i>a</i>	First number to compare.
<i>b</i>	Second number to compare.
<i>twosComplement</i>	When set to true, the inputs are signed integers in 2's complement. If set to false (default), unsigned comparison is performed.

## Parameters

<i>unpackSlotEncoding</i>	Vector of constants for unpacking, as used in bootstrapping.
---------------------------	--

## Note

If  $a=b$  then  $\mu=n_i=0$

**6.2.3.117 compareTwoNumbersImplementation()**

```
void helib::compareTwoNumbersImplementation (
    CTPtrs & max,
    CTPtrs & min,
    Ctxt & mu,
    Ctxt & ni,
    const CTPtrs & aa,
    const CTPtrs & bb,
    bool twosComplement,
    std::vector< zzX > * unpackSlotEncoding,
    bool cmp_only )
```

**6.2.3.118 computeAllProducts()**

```
void helib::computeAllProducts (
    CTPtrs & products,
    const CTPtrs & array,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

For an  $n$ -size array, compute the  $2^n$  products  $\text{products}[j] = \prod_{i \text{ s.t. } j_i=1} \text{array}[i] \times \prod_{i \text{ s.t. } j_i=0} (\text{a-array}[i])$

**6.2.3.119 computeDivVec()**

```
void helib::computeDivVec (
    NTL::Vec< long > & divVec,
    long m,
    const NTL::Vec< long > & powVec ) [inline]
```

**6.2.3.120 computeIntervalForMul()**

```
void helib::computeIntervalForMul (
    double & lo,
    double & hi,
    const Ctxt & ctxt1,
    const Ctxt & ctxt2 )
```

**6.2.3.121 computeIntervalForSqr()**

```
void helib::computeIntervalForSqr (
    double & lo,
    double & hi,
    const Ctxt & ctxt )
```

**6.2.3.122 computeInvVec()**

```
void helib::computeInvVec (
    NTL::Vec< long > & invVec,
    const NTL::Vec< long > & divVec,
    const NTL::Vec< long > & powVec ) [inline]
```

**6.2.3.123 ComputeOneGenMapping()**

```
void helib::ComputeOneGenMapping (
    Permut & genMap,
    const OneGeneratorTree & T )
```

to a single generator tree

**6.2.3.124 computeProd() [1/2]**

```
long helib::computeProd (
    const NTL::Vec< long > & vec )
```

returns \prod\_d vec[d]

**6.2.3.125 computeProd()** [2/2]

```
long helib::computeProd (
    const std::vector< long > & vec )
```

**6.2.3.126 concatBinaryNums()**

```
void helib::concatBinaryNums (
    CTPtrs & output,
    const CTPtrs & a,
    const CTPtrs & b )
```

Concatenates two binary numbers into a single CTPtrs object. E.g. If a=10111, b=00101 then output = 1011100101.

Concatenate two binary numbers into a single CTPtrs object.

**Parameters**

<i>output</i>	Equal to the concatenation of a and b.
<i>a</i>	First number to copy into output.
<i>b</i>	Second number to concatenate to a.

**Note**

The size of output must be of size `a.size() + b.size()`.

**6.2.3.127 conv()** [1/2]

```
void helib::conv (
    DoubleCRT & d,
    const NTL::ZZX & p ) [inline]
```

**6.2.3.128 conv()** [2/2]

```
void helib::conv (
    NTL::ZZX & p,
    const DoubleCRT & d ) [inline]
```

**6.2.3.129 convert()** [1/18]

```
template<typename T1 , typename T2 >
T1 helib::convert (
    const T2 & v2 )
```

**6.2.3.130 convert()** [2/18]

```
void helib::convert (
    long & x1,
    const NTL::GF2X & x2 ) [inline]
```

**6.2.3.131 convert()** [3/18]

```
void helib::convert (
    long & x1,
    const NTL::zz_pX & x2 ) [inline]
```

**6.2.3.132 convert()** [4/18]

```
void helib::convert (
    NTL::GF2X & out,
    const NTL::Vec< long > & in )
```

**6.2.3.133 convert()** [5/18]

```
void helib::convert (
    NTL::mat_zz_pE & X,
    const std::vector< std::vector< NTL::ZZX >> & A )
```



**6.2.3.134 convert()** [6/18]

```
void helib::convert (
    NTL::Vec< long > & out,
    const NTL::GF2X & in )
```

**6.2.3.135 convert()** [7/18]

```
void helib::convert (
    NTL::Vec< long > & out,
    const NTL::zz_pX & in,
    bool symmetric = true )
```

**6.2.3.136 convert()** [8/18]

```
void helib::convert (
    NTL::Vec< long > & out,
    const NTL::ZZX & in )
```

**6.2.3.137 convert()** [9/18]

```
template<typename T1 , typename T2 >
void helib::convert (
    NTL::Vec< T1 > & v1,
    const std::vector< T2 > & v2 )
```

**6.2.3.138 convert()** [10/18]

```
void helib::convert (
    NTL::vec_zz_pE & X,
    const std::vector< NTL::ZZX > & A )
```

**6.2.3.139 convert()** [11/18]

```
void helib::convert (
    NTL::zz_pX & x,
    const zzX & a ) [inline]
```

**6.2.3.140 convert()** [12/18]

```
void helib::convert (
    NTL::ZZX & out,
    const NTL::Vec< long > & in )
```

**6.2.3.141 convert()** [13/18]

```
void helib::convert (
    std::vector< NTL::ZZX > & X,
    const NTL::vec_zz_pE & A )
```

**6.2.3.142 convert()** [14/18]

```
void helib::convert (
    std::vector< std::vector< NTL::ZZX >> & X,
    const NTL::mat_zz_pE & A )
```

**6.2.3.143 convert()** [15/18]

```
template<typename T >
void helib::convert (
    std::vector< T > & v1,
    const std::vector< T > & v2 )
```

Trivial type conversion, useful for generic code.

**6.2.3.144 convert()** [16/18]

```
template<typename T1 , typename T2 >
void helib::convert (
    std::vector< T1 > & v1,
    const NTL::Vec< T2 > & v2 )
```

**6.2.3.145 convert()** [17/18]

```
template<typename T1 , typename T2 >
void helib::convert (
    std::vector< T1 > & v1,
    const std::vector< T2 > & v2 )
```

generic vector conversion routines

**6.2.3.146 convert()** [18/18]

```
template<typename T1 , typename T2 >
void helib::convert (
    T1 & x1,
    const T2 & x2 )
```

A generic template that resolves to [NTL](#)'s conv routine.

**6.2.3.147 convertDataToSlotVector()**

```
template<typename From , typename Scheme >
std::vector<typename Scheme::SlotType> helib::convertDataToSlotVector (
    const std::vector< From > & data,
    const Context & context ) [inline]
```

Converts `std::vector<From>` to `std::vector<Scheme::SlotType>`.

**Template Parameters**

<i>From</i>	Type of the element in the input vector.
<i>Scheme</i>	The encryption scheme to be used, must be <a href="#">BGV</a> or <a href="#">CKKS</a> .

**Parameters**

<i>data</i>	Vector to be converted.
-------------	-------------------------

**Returns**

Vector of converted values of type `Scheme::SlotType`.

**Note**

Only exists for [BGV](#) and [CKKS](#).

**6.2.3.148 CRTcoeff()**

```
long helib::CRTcoeff (
    long p,
    long q,
    bool symmetric = false ) [inline]
```

Returns a CRT coefficient:  $x = (0 \bmod p, 1 \bmod q)$ . If `symmetric` is set then  $x \in [-pq/2, pq/2)$ , else  $x \in [0, pq)$

#### 6.2.3.149 ctxtPrimeSize()

```
long helib::ctxtPrimeSize (
    long nBits )
```

#### 6.2.3.150 Cyclotomic()

```
NTL::ZZX helib::Cyclotomic (
    long N )
```

Compute cyclotomic polynomial.

#### 6.2.3.151 decode() [1/2]

```
void helib::decode (
    const EncryptedArray & ea,
    std::vector< long > & array,
    const PlaintextArray & pa )
```

#### 6.2.3.152 decode() [2/2]

```
void helib::decode (
    const EncryptedArray & ea,
    std::vector< NTL::ZZX > & array,
    const PlaintextArray & pa )
```

#### 6.2.3.153 decryptAndCompare()

```
bool helib::decryptAndCompare (
    const Ctxt & ctxt,
    const SecKey & sk,
    const EncryptedArray & ea,
    const PlaintextArray & pa )
```

#### 6.2.3.154 decryptAndPrint()

```
void helib::decryptAndPrint (
    std::ostream & s,
    const Ctxt & ctxt,
    const SecKey & sk,
    const EncryptedArray & ea,
    long flags = 0 )
```

**6.2.3.155 decryptBinaryNums()**

```

void helib::decryptBinaryNums (
    std::vector< long > & pNums,
    const CtPtrs & eNums,
    const SecKey & sKey,
    const EncryptedArray & ea,
    bool twosComplement = false,
    bool allSlots = true )

```

Decrypt the binary numbers that are encrypted in eNums.

**Parameters**

<i>pNums</i>	vector to decrypt the binary numbers into.
<i>eNums</i>	encrypted binary numbers of which to be decrypted.
<i>sKey</i>	secret key used for decryption.
<i>ea</i>	encrypted array that holds necessary information for decryption.

## Parameters

<i>twosComplement</i>	when set to true, the number to decrypt is a signed integer in 2's complement.
<i>allSlots</i>	when set to false, return only the subcube with index=0 in the last dimension within each ciphertext.

The bits are encrypted in a bit-sliced manner. Namely, `encNums[0]` contains the LSB of all the numbers, `encNums[1]` the next bits from all, etc. If `twosComplement==true` then the number is interpreted as a signed integer in 2's-complement representation. If `allSlots==false` then we only return the subcube with index `i=0` in the last dimension within each ciphertext. Namely, the bit for the `j`'th counter is found in slot of index `j*sizeof(lastDim)`.

**6.2.3.156 defaultPmiddle()**

```
long helib::defaultPmiddle (
    long delta ) [inline]
```

**6.2.3.157 defaultQmiddle()**

```
long helib::defaultQmiddle (
    long delta ) [inline]
```

**6.2.3.158 DestMulAdd()**

```
void helib::DestMulAdd (
    Ctxt & x,
    const std::shared_ptr< ConstMultiplier > & a,
    Ctxt & b )
```

**6.2.3.159 disjoint()**

```
bool helib::disjoint (
    const IndexSet & s1,
    const IndexSet & s2 ) [inline]
```

Functional disjoint.

**6.2.3.160 div() [1/2]**

```
void helib::div (
    std::vector< NTL::ZZX > & x,
    const std::vector< NTL::ZZX > & a,
    long b )
```

**6.2.3.161 div() [2/2]**

```
void helib::div (
    ZZx & res,
    const ZZx & a,
    long b )
```

**6.2.3.162 divc()**

```
long helib::divc (
    long a,
    long b ) [inline]
```

returns ceiling(a/b); assumes a >=0, b>0, a+b <= MAX\_LONG

**6.2.3.163 DoubleCRT::Op< DoubleCRT::AddFun >() [1/3]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::AddFun > (
    const DoubleCRT & other,
    AddFun fun,
    bool matchIndexSets )
```

**6.2.3.164 DoubleCRT::Op< DoubleCRT::AddFun >() [2/3]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::AddFun > (
    const NTL::ZZ & num,
    AddFun fun )
```

**6.2.3.165 DoubleCRT::Op< DoubleCRT::AddFun >() [3/3]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::AddFun > (
    const NTL::ZZX & poly,
    AddFun fun )
```

**6.2.3.166 DoubleCRT::Op< DoubleCRT::MulFun >() [1/2]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::MulFun > (
    const NTL::ZZ & num,
    MulFun fun )
```

**6.2.3.167 DoubleCRT::Op< DoubleCRT::MulFun >() [2/2]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::MulFun > (
    const NTL::ZZX & poly,
    MulFun fun )
```

**6.2.3.168 DoubleCRT::Op< DoubleCRT::SubFun >() [1/3]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::SubFun > (
    const DoubleCRT & other,
    SubFun fun,
    bool matchIndexSets )
```



**6.2.3.169 DoubleCRT::Op< DoubleCRT::SubFun >() [2/3]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::SubFun > (
    const NTL::ZZ & num,
    SubFun fun )
```

**6.2.3.170 DoubleCRT::Op< DoubleCRT::SubFun >() [3/3]**

```
template DoubleCRT& helib::DoubleCRT::Op< DoubleCRT::SubFun > (
    const NTL::ZZX & poly,
    SubFun fun )
```

**6.2.3.171 EDF()**

```
void helib::EDF (
    NTL::vec_zz_pX & v,
    const NTL::zz_pX & f,
    long d )
```

**6.2.3.172 embeddingLargestCoeff() [1/4]**

```
NTL::xdouble helib::embeddingLargestCoeff (
    const Ctxt & ctxt,
    const SecKey & sk )
```

**6.2.3.173 embeddingLargestCoeff() [2/4]**

```
NTL::xdouble helib::embeddingLargestCoeff (
    const NTL::ZZX & f,
    const PAlgebra & palg )
```

**6.2.3.174 embeddingLargestCoeff() [3/4]**

```
double helib::embeddingLargestCoeff (
    const std::vector< double > & f,
    const PAlgebra & palg )
```

**6.2.3.175 embeddingLargestCoeff() [4/4]**

```
double helib::embeddingLargestCoeff (
    const ZZ & f,
    const PAlgebra & palg )
```

Computing the L-infinity norm of the canonical embedding Assumed:  $\deg(f) < \phi(m)$ .

**6.2.3.176 embeddingLargestCoeff\_x2()**

```
void helib::embeddingLargestCoeff_x2 (
    double & norm1,
    double & norm2,
    const std::vector< double > & f1,
    const std::vector< double > & f2,
    const PAlgebra & palg )
```

**6.2.3.177 empty()**

```
bool helib::empty (
    const IndexSet & s ) [inline]
```

**6.2.3.178 encode() [1/4]**

```
void helib::encode (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const NTL::ZZX & val )
```

**6.2.3.179 encode() [2/4]**

```
void helib::encode (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const std::vector< long > & array )
```

**6.2.3.180 encode() [3/4]**

```
void helib::encode (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const std::vector< NTL::ZZX > & array )
```

**6.2.3.181 encode() [4/4]**

```
void helib::encode (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    long val )
```

**6.2.3.182 endBuildModChain()**

```
void helib::endBuildModChain (
    Context & context )
```

**6.2.3.183 equals() [1/3]**

```
bool helib::equals (
    const EncryptedArray & ea,
    const PlaintextArray & pa,
    const PlaintextArray & other )
```

**6.2.3.184 equals() [2/3]**

```
bool helib::equals (
    const EncryptedArray & ea,
    const PlaintextArray & pa,
    const std::vector< long > & other )
```

**6.2.3.185 equals() [3/3]**

```
bool helib::equals (
    const EncryptedArray & ea,
    const PlaintextArray & pa,
    const std::vector< NTL::ZZX > & other )
```

**6.2.3.186 extendExtractDigits()**

```
void helib::extendExtractDigits (
    std::vector< Ctxt > & digits,
    const Ctxt & c,
    long r,
    long e )
```

**6.2.3.187 extractDigits() [1/2]**

```
void helib::extractDigits (
    std::vector< Ctxt > & digits,
    const Ctxt & c,
    long r,
    bool shortcut ) [inline]
```

**6.2.3.188 extractDigits() [2/2]**

```
void helib::extractDigits (
    std::vector< Ctxt > & digits,
    const Ctxt & c,
    long r = 0 )
```

Extract the mod- $p$  digits of a mod- $p^r$  ciphertext.

extractDigits returns in the slots of digits[j] the  $j$ 'th-lowest digits from the integers in the slots of the input. Namely, the  $i$ 'th slot of digits[j] contains the  $j$ 'th digit in the  $p$ -base expansion of the integer in the  $i$ 'th slot of the \*this.

If  $r==0$  then it is set to c.effectiveR(). It is assumed that the slots of \*this contains integers mod  $p^r$ , i.e., that only the free terms are nonzero. If that assumptions does not hold then the result will not be a valid ciphertext anymore.

The "shortcut" flag is deprecated, it often leads to catastrophic failure in the noise estimate. Calling the function with shortcut=true has not effect, except printing a warning message to cerr.

The output ciphertext digits[j] contains the  $j$ 'th digit in the base- $p$  expansion of the input, and its plaintext space is modulo  $p^{r-j}$ . All the ciphertexts in the output are at the same level.

**6.2.3.189 extractDigitsPacked()**

```
void helib::extractDigitsPacked (
    Ctxt & ctxt,
    long botHigh,
    long r,
    long ePrime,
    const std::vector< NTL::ZZX > & unpackSlotEncoding )
```

**6.2.3.190 extractDigitsThin()**

```
void helib::extractDigitsThin (
    Ctxt & ctxt,
    long botHigh,
    long r,
    long ePrime )
```

**6.2.3.191 factorize()** [1/3]

```
void helib::factorize (
    NTL::Vec< NTL::Pair< long, long >> & factors,
    long N )
```

Factoring by trial division, only works for  $N < 2^{60}$  primes and multiplicities are recorded.

**6.2.3.192 factorize()** [2/3]

```
void helib::factorize (
    std::vector< long > & factors,
    long N )
```

Factoring by trial division, only works for  $N < 2^{60}$ , only the primes are recorded, not their multiplicity.

**6.2.3.193 factorize()** [3/3]

```
void helib::factorize (
    std::vector< NTL::ZZ > & factors,
    const NTL::ZZ & N )
```

**6.2.3.194 fastPower()**

```
void helib::fastPower (
    Ctxt & ctxt,
    long d )
```

**6.2.3.195 fetch\_saved\_values()**

```
const std::vector< double > * helib::fetch_saved_values (
    const char * name )
```

**6.2.3.196 fifteenOrLess4Four()**

```
long helib::fifteenOrLess4Four (
    const CPtrs & out,
    const CPtrs & in,
    long sizeLimit = 4 )
```

Add together up to fifteen {0,1} integers, producing a 4-bit counter.

**Parameters**

<i>out</i>	4-bit counter to be out-putted.
<i>in</i>	bits to be counted.
<i>sizeLimit</i>	number of bits to compute on, taken from the least significant end.

**Returns**

number of output bits that are not identically zero (i.e. != null).

Adding fifteen input bits, getting a 4-bit counter. Some of the input pointers may be null, but output pointers must point to allocated [Ctxt](#) objects. If sizeLimit<4, only that many bits are computed (taken from the least significant end).

**6.2.3.197 findGenerators()**

```
long helib::findGenerators (
    std::vector< long > & gens,
    std::vector< long > & ords,
    long m,
    long p,
    const std::vector< long > & candidates = std::vector<long>() )
```

Returns in gens a generating set for  $Zm^*$  /

, and in ords the order of these generators. Return value is the order of p in  $Zm^*$ .

**6.2.3.198 FindM()**

```
long helib::FindM (
    long k,
    long nBits,
    long c,
    long p,
    long d,
    long s,
    long chosen_m,
    bool verbose = false )
```

Returns smallest parameter m satisfying various constraints:

## Parameters

$k$	security parameter
$L$	number of levels
$c$	number of columns in key switching matrices
$p$	characteristic of plain-text space
$d$	embedding degree (d==0 or d==1 => no constraint)
$s$	at least that many plain-text slots

## Parameters

<i>chosen_m</i>	<pre> preselected value of m (0 =&gt; not prese- lected) Fails with an error mes- sage if no suit- able m is found prints an infor- mative mes- sage if ver- bose == true </pre>
-----------------	--

**6.2.3.199 findMinBitCapacity()** [1/3]

```

long helib::findMinBitCapacity (
    const CTPtrMat & m ) [inline]

```

**6.2.3.200 findMinBitCapacity()** [2/3]

```

long helib::findMinBitCapacity (
    const CTPtrs & v ) [inline]

```

**6.2.3.201 findMinBitCapacity()** [3/3]

```

long helib::findMinBitCapacity (
    std::initializer_list< const CTPtrs * > list ) [inline]

```



**6.2.3.202 FindPrimitiveRoot()** [1/2]

```
void helib::FindPrimitiveRoot (
    NTL::zz_p & r,
    unsigned long e )
```

Find e-th root of unity modulo the current modulus.

**6.2.3.203 FindPrimitiveRoot()** [2/2]

```
void helib::FindPrimitiveRoot (
    NTL::ZZ_p & r,
    unsigned long e )
```

**6.2.3.204 FindPrimRootT()**

```
template<typename zp , typename zz >
void helib::FindPrimRootT (
    zp & root,
    unsigned long e )
```

**6.2.3.205 frobeniusAutomorph()** [1/2]

```
void helib::frobeniusAutomorph (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const NTL::Vec< long > & vec )
```

**6.2.3.206 frobeniusAutomorph()** [2/2]

```
void helib::frobeniusAutomorph (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    long j )
```

**6.2.3.207 FrobeniusMap()**

```
NTL::zz_pEX helib::FrobeniusMap (
    const NTL::zz_pEXModulus & F )
```

**6.2.3.208 fsquare()**

```
double helib::fsquare (
    double x ) [inline]
```

Return the square of a number as a double.

**6.2.3.209 GenBabySteps()**

```
void helib::GenBabySteps (
    std::vector< std::shared_ptr< Ctxt >> & v,
    const Ctxt & ctxt,
    long dim,
    bool clean )
```

**6.2.3.210 getG()**

```
Ntl::ZZX helib::getG (
    const EncryptedArray & ea )
```

**6.2.3.211 getHyperColumn() [1/3]**

```
template void helib::getHyperColumn (
    Ntl::Vec< long > & v,
    const ConstCubeSlice< long > & s,
    long pos )
```

**6.2.3.212 getHyperColumn() [2/3]**

```
template void helib::getHyperColumn (
    Ntl::Vec< Ntl::zz_p > & v,
    const ConstCubeSlice< Ntl::zz_p > & s,
    long pos )
```

**6.2.3.213 getHyperColumn()** [3/3]

```
template<typename T >
void helib::getHyperColumn (
    NTL::Vec< T > & v,
    const ConstCubeSlice< T > & s,
    long pos )
```

getHyperColumn reads out a (multi-dimensional) column from a slice. The parameter pos specifies the position of the column, which must be in the range  $0 \leq \text{pos} < \text{s.getProd}(1)$ . The vector v is filled with values whose coordinate in the lower dimensional subcube is equal to pos. The length of v will be set to s.getDim(0).

**6.2.3.214 getPhimXMod()**

```
const NTL::zz_pXModulus & helib::getPhimXMod (
    const PAlgebra & palg )
```

**6.2.3.215 getTimerByName()**

```
const FHEtimer * helib::getTimerByName (
    const char * name )
```

**6.2.3.216 GetTimerClock()**

```
unsigned long helib::GetTimerClock ( )
```

**6.2.3.217 incrementalProduct()**

```
void helib::incrementalProduct (
    std::vector< Ctxt > & v )
```

For  $i=n-1 \dots 0$ , set  $v[i]=\text{prod}_{j \leq i} v[j]$  This implementation uses depth  $\log n$  and  $(n \log n)/2$  products

**6.2.3.218 incrementalZeroTest()**

```
void helib::incrementalZeroTest (
    Ctxt * res[],
    const EncryptedArray & ea,
    const Ctxt & ctxt,
    long n )
```

**6.2.3.219 innerProduct()** [1/9]

```
Ctxt helib::innerProduct (
    const CtxtPtrs & v1,
    const CtxtPtrs & v2 ) [inline]
```

**6.2.3.220 innerProduct()** [2/9]

```
Ctxt helib::innerProduct (
    const std::vector< Ctxt > & v1,
    const std::vector< Ctxt > & v2 ) [inline]
```

**6.2.3.221 innerProduct()** [3/9]

```
Ctxt helib::innerProduct (
    const std::vector< Ctxt > & v1,
    const std::vector< DoubleCRT > & v2 ) [inline]
```

**6.2.3.222 innerProduct()** [4/9]

```
Ctxt helib::innerProduct (
    const std::vector< Ctxt > & v1,
    const std::vector< NTL::ZZX > & v2 ) [inline]
```

**6.2.3.223 innerProduct()** [5/9]

```
void helib::innerProduct (
    Ctxt & result,
    const CtxtPtrs & v1,
    const CtxtPtrs & v2 )
```

**6.2.3.224 innerProduct()** [6/9]

```
void helib::innerProduct (
    Ctxt & result,
    const std::vector< Ctxt > & v1,
    const std::vector< Ctxt > & v2 )
```

**6.2.3.225 innerProduct()** [7/9]

```
void helib::innerProduct (
    Ctxt & result,
    const std::vector< Ctxt > & v1,
    const std::vector< DoubleCRT > & v2 )
```

Compute the inner product of a vectors of ciphertexts and a constant vector.

**6.2.3.226 innerProduct()** [8/9]

```
void helib::innerProduct (
    Ctxt & result,
    const std::vector< Ctxt > & v1,
    const std::vector< NTL::ZZX > & v2 )
```

**6.2.3.227 innerProduct()** [9/9]

```
template<typename Scheme >
void helib::innerProduct (
    Ptxt< Scheme > & result,
    const std::vector< Ptxt< Scheme >> & first_vec,
    const std::vector< Ptxt< Scheme >> & second_vec )
```

Free function that computes the inner product of two vectors of `Ptxt`.

**Parameters**

<i>result</i>	The output <code>Ptxt</code> that will hold the result.
<i>first_vec</i>	The first input vector of plain-texts.
<i>second_vec</i>	The second input vector of plain-texts.

**Note**

If the two vector sizes differ, the shorter vector will be padded with zeroes.

**6.2.3.228 interpolateMod()**

```
void helib::interpolateMod (
    NTL::ZZX & poly,
    const NTL::vec_long & x,
    const NTL::vec_long & y,
    long p,
    long e = 1 )
```

Interpolate polynomial such that  $\text{poly}(x[i] \bmod p) = y[i] \bmod p^e$ . It is assumed that the points  $x[i]$  are all distinct modulo  $p$ .

**6.2.3.229 intVecCRT() [1/4]**

```
template bool helib::intVecCRT (
    NTL::vec_ZZ & ,
    const NTL::ZZ & ,
    const NTL::Vec< NTL::zz_p > & ,
    long )
```

**6.2.3.230 intVecCRT() [2/4]**

```
template bool helib::intVecCRT (
    NTL::vec_ZZ & ,
    const NTL::ZZ & ,
    const NTL::vec_long & ,
    long )
```

**6.2.3.231 intVecCRT() [3/4]**

```
template bool helib::intVecCRT (
    NTL::vec_ZZ & ,
    const NTL::ZZ & ,
    const NTL::vec_ZZ & ,
    long )
```

**6.2.3.232 intVecCRT()** [4/4]

```
template<class zzvec >
bool helib::intVecCRT (
    NTL::vec_ZZ & vp,
    const NTL::ZZ & p,
    const zzvec & vq,
    long q )
```

Incremental integer CRT for vectors.

Expects co-primes p,q with q odd, and such that all the entries in v1 are in  $[-p/2, p/2)$ . Returns in v1 the CRT of vp mod p and vq mod q, as integers in  $[-pq/2, pq/2)$ . Uses the formula:

$$CRT(vp, p, vq, q) = vp + [(vq - vp) * p^{-1}]_q * p,$$

where  $[...]_q$  means reduction to the interval  $[-q/2, q/2)$ . Notice that if q is odd then this is the same as reducing to  $[-(q-1)/2, (q-1)/2]$ , which means that  $[...]_q * p$  is in  $[-p(q-1)/2, p(q-1)/2]$ , and since vp is in  $[-p/2, p/2)$  then the sum is indeed in  $[-pq/2, pq/2)$ .

Return true is both vectors are of the same length, false otherwise

**6.2.3.233 InvModpr()**

```
void helib::InvModpr (
    NTL::zz_pX & S,
    const NTL::zz_pX & F,
    const NTL::zz_pX & G,
    long p,
    long r )
```

**6.2.3.234 is2power()**

```
bool helib::is2power (
    long m ) [inline]
```

**6.2.3.235 is\_in()**

```
long helib::is_in (
    long x,
    int * X,
    long sz )
```

Finds whether x is an element of the set X of size sz, Returns -1 if not and the location if true.

**6.2.3.236 isDryRun()**

```
bool helib::isDryRun ( ) [inline]
```

**6.2.3.237 isSetAutomorphVals()**

```
bool helib::isSetAutomorphVals ( ) [inline]
```

**6.2.3.238 isSetAutomorphVals2()**

```
bool helib::isSetAutomorphVals2 ( ) [inline]
```

**6.2.3.239 IsZero()**

```
bool helib::IsZero (
    const zzX & a ) [inline]
```

**6.2.3.240 killVec() [1/2]**

```
template<typename T >
void helib::killVec (
    NTL::Vec< T > & vec )
```

**6.2.3.241 killVec() [2/2]**

```
template<typename T >
void helib::killVec (
    std::vector< T > & vec )
```

NTL/std compatibility.

**6.2.3.242 KSGiantStepSize()**

```
long helib::KSGiantStepSize (
    long D )
```

Function that returns number of baby steps. Used to keep this and matmul routines "in sync".



**6.2.3.243 largestCoeff() [1/5]**

```
NTL::ZZ helib::largestCoeff (
    const DoubleCRT & f )
```

**6.2.3.244 largestCoeff() [2/5]**

```
NTL::ZZ helib::largestCoeff (
    const NTL::Vec< NTL::ZZ > & f )
```

**6.2.3.245 largestCoeff() [3/5]**

```
template<typename T >
double helib::largestCoeff (
    const NTL::Vec< T > & f )
```

The L-infinity norm of an element (in coefficient representation)

**6.2.3.246 largestCoeff() [4/5]**

```
NTL::ZZ helib::largestCoeff (
    const NTL::ZZX & f )
```

**6.2.3.247 largestCoeff() [5/5]**

```
template<typename T >
double helib::largestCoeff (
    const std::vector< T > & f )
```

**6.2.3.248 leftBitwiseShift()**

```
void helib::leftBitwiseShift (
    CtPtrs & output,
    const CtPtrs & input,
    const long shamt )
```

Left shift input by shamt.

Shift binary numbers to the left by shamt

**Parameters**

<i>output</i>	Shifted result.
<i>input</i>	The number to be shifted.
<i>shamt</i>	The number to bits to shift by.

**Note**

This is a left shift only, i.e. the bits are moved to the most-significant end.

*shamt* must be positive.

The size of *output* and *input* must be the same.

**6.2.3.249 length()**

```
long helib::length (
    GenNodePtr ptr )
```

**6.2.3.250 less\_than() [1/8]**

```
bool helib::less_than (
    const NTL::GF2E & a,
    const NTL::GF2E & b )
```

**6.2.3.251 less\_than() [2/8]**

```
bool helib::less_than (
    const NTL::GF2EX & a,
    const NTL::GF2EX & b )
```

**6.2.3.252 less\_than()** [3/8]

```
bool helib::less_than (
    const NTL::GF2X & a,
    const NTL::GF2X & b )
```

**6.2.3.253 less\_than()** [4/8]

```
bool helib::less_than (
    const NTL::zz_pE & a,
    const NTL::zz_pE & b )
```

**6.2.3.254 less\_than()** [5/8]

```
bool helib::less_than (
    const NTL::zz_pEX & a,
    const NTL::zz_pEX & b )
```

**6.2.3.255 less\_than()** [6/8]

```
bool helib::less_than (
    const NTL::zz_pX & a,
    const NTL::zz_pX & b )
```

**6.2.3.256 less\_than()** [7/8]

```
bool helib::less_than (
    NTL::GF2 a,
    NTL::GF2 b )
```

**6.2.3.257 less\_than()** [8/8]

```
bool helib::less_than (
    NTL::zz_p a,
    NTL::zz_p b )
```

**6.2.3.258 log2()**

```
double helib::log2 (
    const NTL::xdouble & x ) [inline]
```

Base-2 logarithm.

**6.2.3.259 log2\_realToEstimatedNoise()**

```
double helib::log2_realToEstimatedNoise (
    const Ctxt & ctxt,
    const SecKey & sk )
```

**6.2.3.260 longToBitVector()**

```
std::vector< long > helib::longToBitVector (
    long num,
    long bitSize )
```

Returns a number as a vector of bits with LSB on the left.

**Parameters**

<i>num</i>	Number to be converted.
<i>bitSize</i>	Number of bits of the input and output.

**Returns**

Bit vector representation of num.

**Note**

`bitSize` must be non-negative.

**6.2.3.261 lsize() [1/4]**

```
template<typename T >
long helib::lsize (
    const NTL::Vec< T > & v ) [inline]
```

**6.2.3.262 lsize()** [2/4]

```
template<typename T >
long helib::lsize (
    const PtrMatrix< T > & v )
```

**6.2.3.263 lsize()** [3/4]

```
template<typename T >
long helib::lsize (
    const PtrVector< T > & v )
```

**6.2.3.264 lsize()** [4/4]

```
template<typename T >
long helib::lsize (
    const std::vector< T > & v ) [inline]
```

Size of STL vector as a long (rather than unsigned long)

**6.2.3.265 make\_lazy()**

```
template<typename T , typename P , typename... Args>
void helib::make_lazy (
    const NTL::Lazy< T, P > & obj,
    Args &&... args )
```

This should go in [NTL](#) some day... Just call as `make_lazy(obj, ...)` to initialize a lazy object via a call to a constructor `T(...)`

**6.2.3.266 make\_lazy\_with\_fun()**

```
template<typename T , typename P , typename F , typename... Args>
void helib::make_lazy_with_fun (
    const NTL::Lazy< T, P > & obj,
    F f,
    Args &&... args )
```

This should go in [NTL](#) some day... Just call as `make_lazy(obj, f, ....)` to initialize a lazy object via a call to `f(*obj, ...)`

**6.2.3.267 makeIrredPoly()**

```
Ntl::ZZX helib::makeIrredPoly (
    long p,
    long d )
```

Return a degree-d irreducible polynomial mod p.

**6.2.3.268 mapTo01() [1/4]**

```
template void helib::mapTo01 (
    const EncryptedArray & ,
    Ptxt< BGV > & ptxt )
```

**6.2.3.269 mapTo01() [2/4]**

```
template void helib::mapTo01 (
    const EncryptedArray & ,
    Ptxt< CKKS > & ptxt )
```

**6.2.3.270 mapTo01() [3/4]**

```
template<typename Scheme >
void helib::mapTo01 (
    const EncryptedArray & ,
    Ptxt< Scheme > & ptxt )
```

**6.2.3.271 mapTo01() [4/4]**

```
void helib::mapTo01 (
    const EncryptedArray & ea,
    Ctxt & ctxt )
```

**6.2.3.272 maximum\_flow()**

```
long helib::maximum_flow (
    FlowGraph & fg,
    long src,
    long sink )
```

Remove from the graph all the flow-zero edges for (long i=0; i<(long)fg.size(); i++) { FNeighborList::iterator it1=fg[i].begin(); do { FNeighborList::iterator it2 = it1; it1++; // increment the iterator before potentially erasing the edge if (it2->second.flow == 0) fg[i].erase(it2); } while (it1 != fg[i].end()); }

Remove from the graph all the flow-zero edges for (long i=0; i<(long)fg.size(); i++) { FNeighborList::iterator it1=fg[i].begin(); do { FNeighborList::iterator it2 = it1; it1++; // increment the iterator before potentially erasing the edge if (it2->second.flow == 0) fg[i].erase(it2); } while (it1 != fg[i].end()); }

**6.2.3.273 mcDiv()**

```
long helib::mcDiv (
    long a,
    long b )
```

**6.2.3.274 mcMod()**

```
long helib::mcMod (
    long a,
    long b )
```

Routines for computing mathematically correct mod and div.

mcDiv(a, b) = floor(a / b), mcMod(a, b) = a - b\*mcDiv(a, b); in particular, mcMod(a, b) is 0 or has the same sign as b

**6.2.3.275 mobius()**

```
long helib::mobius (
    long n )
```

Compute mobius function (naive method as n is small).

**6.2.3.276 ModComp()**

```
void helib::ModComp (
    NTL::ZZX & res,
    const NTL::ZZX & g,
    const NTL::ZZX & h,
    const NTL::ZZX & f )
```

Modular composition of polynomials: res = g(h) mod f.

**6.2.3.277 mul()** [1/7]

```
void helib::mul (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const PlaintextArray & other )
```

**6.2.3.278 mul()** [2/7]

```
void helib::mul (
    PlaintextArray & pa,
    const BlockMatMul1D & mat )
```

**6.2.3.279 mul()** [3/7]

```
void helib::mul (
    PlaintextArray & pa,
    const BlockMatMulFull & mat )
```

**6.2.3.280 mul()** [4/7]

```
void helib::mul (
    PlaintextArray & pa,
    const MatMul1D & mat )
```

**6.2.3.281 mul()** [5/7]

```
void helib::mul (
    PlaintextArray & pa,
    const MatMulFull & mat )
```

**6.2.3.282 mul()** [6/7]

```
void helib::mul (
    std::vector< NTL::ZZX > & x,
    const std::vector< NTL::ZZX > & a,
    long b )
```



**6.2.3.283 mul()** [7/7]

```
void helib::mul (
    zzX & res,
    const zzX & a,
    long b )
```

**6.2.3.284 MulAdd()**

```
void helib::MulAdd (
    Ctxt & x,
    const std::shared_ptr< ConstMultiplier > & a,
    const Ctxt & b )
```

**6.2.3.285 MulMod()** [1/6]

```
NTL::ZZX helib::MulMod (
    const NTL::ZZX & f,
    long a,
    long q ) [inline]
```

**6.2.3.286 MulMod()** [2/6]

```
NTL::ZZX helib::MulMod (
    const NTL::ZZX & f,
    long a,
    long q,
    bool abs ) [inline]
```

**6.2.3.287 MulMod()** [3/6]

```
zzX helib::MulMod (
    const zzX & a,
    const zzX & b,
    const PAlgebra & palg ) [inline]
```

**6.2.3.288 MulMod()** [4/6]

```
void helib::MulMod (
    NTL::ZZX & out,
    const NTL::ZZX & f,
    long a,
    long q ) [inline]
```

**6.2.3.289 MulMod()** [5/6]

```
void helib::MulMod (
    NTL::ZZX & out,
    const NTL::ZZX & f,
    long a,
    long q,
    bool abs )
```

**6.2.3.290 MulMod()** [6/6]

```
void helib::MulMod (
    zzX & res,
    const zzX & a,
    const zzX & b,
    const PAlgebra & palg )
```

**6.2.3.291 multOrd()**

```
long helib::multOrd (
    long p,
    long m )
```

Return multiplicative order of p modulo m, or 0 if GCD(p, m) != 1.

**6.2.3.292 multTwoNumbers()**

```
void helib::multTwoNumbers (
    CtPtrs & product,
    const CtPtrs & lhs,
    const CtPtrs & rhs,
    bool rhsTwosComplement = false,
    long sizeLimit = 0,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

Multiply two numbers in binary representation where each ciphertext of the input vector contains a bit.

## Parameters

<i>product</i>	result of the multiplication operation.
<i>lhs</i>	left hand side of the multiplication.
<i>rhs</i>	right hand side of the multiplication.
<i>rhsTwosComplement</i>	flag to state the multiplier is potentially negative.
<i>sizeLimit</i>	number of bits to compute on, taken from the least significant end.
<i>unpackSlotEncoding</i>	vector of constants for unpacking, as used in bootstrapping.

**6.2.3.293 negate()**

```
void helib::negate (
    const EncryptedArray & ea,
    PlaintextArray & pa )
```

**6.2.3.294 negateBinary()**

```
void helib::negateBinary (
    CtPtrs & negation,
    const CtPtrs & input )
```

Negates a number in binary 2's complement representation.

**Parameters**

<i>negation</i>	Reference to the negated number that will be populated.
<i>input</i>	Number to be negated.

**Note**

*input* will be treated as a number in 2's complement.

*input* must not alias *negation*.

**6.2.3.295 normalize()**

```
void helib::normalize (
    zzX & f )
```

**6.2.3.296 operator"!="()**

```
template<typename T >
bool helib::operator!= (
    const IndexMap< T > & map1,
    const IndexMap< T > & map2 )
```

**6.2.3.297 operator&()**

```
IndexSet helib::operator& (
    const IndexSet & s,
    const IndexSet & t )
```

intersection

**6.2.3.298 operator\*()**

```
zzX helib::operator* (
    const zzX & a,
    long b ) [inline]
```

**6.2.3.299 operator\*=( )**

```
zzX& helib::operator*= (
    zzX & a,
    long b ) [inline]
```

**6.2.3.300 operator+()**

```
zzX helib::operator+ (
    const zzX & a,
    const zzX & b ) [inline]
```

**6.2.3.301 operator+=( )**

```
zzX& helib::operator+= (
    zzX & a,
    const zzX & b ) [inline]
```

**6.2.3.302 operator/() [1/2]**

```
IndexSet helib::operator/ (
    const IndexSet & s,
    const IndexSet & t )
```

set minus

**6.2.3.303 operator/()** [2/2]

```
zzX helib::operator/ (
    const zzX & a,
    long b ) [inline]
```

**6.2.3.304 operator/=(**

```
zzX& helib::operator/=(
    zzX & a,
    long b ) [inline]
```

**6.2.3.305 operator<()**

```
bool helib::operator< (
    const IndexSet & s1,
    const IndexSet & s2 )
```

Is s1 strict subset of s2.

**6.2.3.306 operator<<()** [1/23]

```
std::ostream& helib::operator<< (
    std::ostream & os,
    const PolyMod & poly )
```

**Parameters**

<i>os</i>	Output std:: ::ostream.
<i>poly</i>	PolyMod object to be writ- ten.

**Returns**

Input std::ostream post writing.

**Note**

p2r and G are not serialised, see note of operator>>.

**6.2.3.307 operator<<()** [2/23]

```
std::ostream& helib::operator<< (
    std::ostream & os,
    const PolyModRing & ring )
```

**Parameters**

<i>os</i>	Output std::ostream.
<i>ring</i>	PolyModRing object to be written.

**Returns**

Input std::ostream post writing.

**6.2.3.308 operator<<()** [3/23]

```
std::ostream & helib::operator<< (
    std::ostream & s,
    const ColPerm & p )
```

**6.2.3.309 operator<<()** [4/23]

```
std::ostream & helib::operator<< (
    std::ostream & s,
    const CtxtPart & p )
```

**6.2.3.310 operator<<()** [5/23]

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const CubeSignature & sig ) [inline]
```

**6.2.3.311 operator<<() [6/23]**

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const GeneratorTrees & trees )
```

**6.2.3.312 operator<<() [7/23]**

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const ModuliSizes & szs )
```

**6.2.3.313 operator<<() [8/23]**

```
std::ostream & helib::operator<< (
    std::ostream & s,
    const ModuliSizes::Entry & e )
```

**6.2.3.314 operator<<() [9/23]**

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const PermNetwork & net )
```

**6.2.3.315 operator<<() [10/23]**

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const PlaintextArray & pa ) [inline]
```

**6.2.3.316 operator<<() [11/23]**

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const SKHandle & handle ) [inline]
```



**6.2.3.317 operator<<()** [12/23]

```
std::ostream& helib::operator<< (
    std::ostream & s,
    const SubDimension & sd )
```

**6.2.3.318 operator<<()** [13/23]

```
std::ostream& helib::operator<< (
    std::ostream & s,
    GenNodePtr p )
```

**6.2.3.319 operator<<()** [14/23]

```
std::ostream& helib::operator<< (
    std::ostream & s,
    LongNodePtr p )
```

**6.2.3.320 operator<<()** [15/23]

```
std::ostream& helib::operator<< (
    std::ostream & s,
    SplitNodePtr p )
```

**6.2.3.321 operator<<()** [16/23]

```
template<typename T >
std::ostream& helib::operator<< (
    std::ostream & s,
    std::vector< T > v )
```

**6.2.3.322 operator<<()** [17/23]

```
std::ostream& helib::operator<< (
    std::ostream & str,
    const Context & context )
```

**6.2.3.323 operator<<() [18/23]**

```
std::ostream& helib::operator<< (
    std::ostream & str,
    const Ctxt & ctxt )
```

**6.2.3.324 operator<<() [19/23]**

```
std::ostream& helib::operator<< (
    std::ostream & str,
    const DoubleCRT & d )
```

**6.2.3.325 operator<<() [20/23]**

```
std::ostream & helib::operator<< (
    std::ostream & str,
    const IndexSet & set )
```

**6.2.3.326 operator<<() [21/23]**

```
std::ostream & helib::operator<< (
    std::ostream & str,
    const KeySwitch & matrix )
```

**6.2.3.327 operator<<() [22/23]**

```
std::ostream& helib::operator<< (
    std::ostream & str,
    const PubKey & pk )
```

**6.2.3.328 operator<<() [23/23]**

```
std::ostream& helib::operator<< (
    std::ostream & str,
    const SecKey & sk )
```

**6.2.3.329 operator<=()**

```
bool helib::operator<= (
    const IndexSet & s1,
    const IndexSet & s2 )
```

Is s1 subset or equal to s2.

**6.2.3.330 operator==( )**

```
template<typename T >
bool helib::operator==(
    const IndexMap< T > & map1,
    const IndexMap< T > & map2 )
```

Comparing maps, by comparing all the elements.

**6.2.3.331 operator>() [1/2]**

```
bool helib::operator> (
    const IndexSet & s1,
    const IndexSet & s2 )
```

Is s2 strict subset of s1.

**6.2.3.332 operator>() [2/2]**

```
bool helib::operator> (
    const ModuliSizes::Entry & a,
    const ModuliSizes::Entry & b ) [inline]
```

**6.2.3.333 operator>=()**

```
bool helib::operator>= (
    const IndexSet & s1,
    const IndexSet & s2 )
```

Is s2 subset or equal to s2.

**6.2.3.334 operator>>() [1/12]**

```
std::istream& helib::operator>> (
    std::istream & is,
    PolyMod & poly )
```

**Parameters**

<i>is</i>	Input std::istream.
<i>poly</i>	Destination PolyMod object.

**Returns**

Input std::istream post reading.

**Note**

poly must be constructed with an appropriate p2r and G **BEFORE** calling this function. For example,

```
PolyMod my_poly(p2r, G);
std::cin » my_poly;
```

**6.2.3.335 operator>>() [2/12]**

```
std::istream & helib::operator>> (
    std::istream & s,
    CtxtPart & p )
```

**6.2.3.336 operator>>() [3/12]**

```
std::istream& helib::operator>> (
    std::istream & s,
    ModuliSizes & szs )
```

**6.2.3.337 operator>>() [4/12]**

```
std::istream & helib::operator>> (
    std::istream & s,
    ModuliSizes::Entry & e )
```

**6.2.3.338 operator>>() [5/12]**

```
template<typename T >
std::istream& helib::operator>> (
    std::istream & s,
    std::vector< T > & v )
```

**6.2.3.339 operator>>()** [6/12]

```
std::istream& helib::operator>> (
    std::istream & str,
    Context & context )
```

**6.2.3.340 operator>>()** [7/12]

```
std::istream& helib::operator>> (
    std::istream & str,
    Ctxt & ctxt )
```

**6.2.3.341 operator>>()** [8/12]

```
std::istream& helib::operator>> (
    std::istream & str,
    DoubleCRT & d )
```

**6.2.3.342 operator>>()** [9/12]

```
std::istream & helib::operator>> (
    std::istream & str,
    IndexSet & set )
```

**6.2.3.343 operator>>()** [10/12]

```
std::istream& helib::operator>> (
    std::istream & str,
    PubKey & pk )
```

**6.2.3.344 operator>>()** [11/12]

```
std::istream& helib::operator>> (
    std::istream & str,
    SecKey & sk )
```

**6.2.3.345 operator>>()** [12/12]

```
std::istream& helib::operator>> (
    std::istream & str,
    SKHandle & handle )
```

**6.2.3.346 operator^()**

```
IndexSet helib::operator^ (
    const IndexSet & s,
    const IndexSet & t )
```

exclusive-or

**6.2.3.347 operator" |"()**

```
IndexSet helib::operator| (
    const IndexSet & s,
    const IndexSet & t )
```

union

**6.2.3.348 optimalBenes()**

```
void helib::optimalBenes (
    long n,
    long budget,
    bool good,
    long & cost,
    LongNodePtr & solution )
```

**6.2.3.349 optimalBenesAux()**

```
BenesMemoEntry helib::optimalBenesAux (
    long i,
    long budget,
    long nlev,
    const NTL::Vec< NTL::Vec< long >> & costTab,
    BenesMemoTable & memoTab )
```

**6.2.3.350 optimalLower()**

```
LowerMemoEntry helib::optimalLower (
    long order,
    bool good,
    long budget,
    long mid,
    LowerMemoTable & lowerMemoTable )
```

**6.2.3.351 optimalUpperAux()**

```
UpperMemoEntry helib::optimalUpperAux (
    const NTL::Vec< GenDescriptor > & vec,
    long i,
    long budget,
    long mid,
    UpperMemoTable & upperMemoTable,
    LowerMemoTable & lowerMemoTable )
```

**6.2.3.352 ord()**

```
long helib::ord (
    long N,
    long p )
```

Compute the highest power of p that divides N.

**6.2.3.353 packConstant()**

```
void helib::packConstant (
    zzX & result,
    unsigned long data,
    long nbits,
    const EncryptedArray & ea )
```

**6.2.3.354 packConstants()**

```
void helib::packConstants (
    zzX & result,
    const std::vector< unsigned long > & data,
    long nbits,
    const EncryptedArray & ea )
```

**6.2.3.355 packedRecrypt()** [1/4]

```
void helib::packedRecrypt (
    const CtPtrMat & m,
    const std::vector< zzX > & unpackConsts,
    const EncryptedArray & ea,
    long belowLvl = LONG_MAX )
```

**6.2.3.356 packedRecrypt()** [2/4]

```
void helib::packedRecrypt (
    const CtPtrs & a,
    const CtPtrs & b,
    std::vector< zzX > * unpackSlotEncoding )
```

Function for packed recryption to recrypt multiple numbers.

**Parameters**

<i>a</i>	first input of which to re-crypt.
<i>b</i>	second input of which to re-crypt.
<i>unpackSlotEncoding</i>	vector of constants for unpacking, as used in bootstrapping.

**6.2.3.357 packedRecrypt()** [3/4]

```
void helib::packedRecrypt (
    const CtPtrs & array,
    const std::vector< zzX > & unpackConsts,
    const EncryptedArray & ea,
    long belowLvl )
```



**6.2.3.358 packedRecrypt()** [4/4]

```
void helib::packedRecrypt (
    const CtPtrs & cPtrs,
    const std::vector< zzX > & unpackConsts,
    const EncryptedArray & ea )
```

**6.2.3.359 PALgebraLift()** [1/2]

```
template<>
void helib::PALgebraLift (
    const NTL::ZZX & phimx,
    const NTL::vec_zz_pX & lfactors,
    NTL::vec_zz_pX & factors,
    NTL::vec_zz_pX & crtc,
    long r )
```

**6.2.3.360 PALgebraLift()** [2/2]

```
template<typename T >
void helib::PALgebraLift (
    const NTL::ZZX & phimx,
    const T & lfactors,
    T & factors,
    T & crtc,
    long r )
```

**6.2.3.361 phi\_N()**

```
long helib::phi_N (
    long N )
```

Compute  $\Phi(N)$ .

**6.2.3.362 phiN()** [1/2]

```
void helib::phiN (
    long & phiN,
    std::vector< long > & facts,
    long N )
```

Compute  $\Phi(N)$  and also factorize  $N$ .

**6.2.3.363 phiN()** [2/2]

```
void helib::phiN (
    NTL::ZZ & phiN,
    std::vector< NTL::ZZ > & facts,
    const NTL::ZZ & N )
```

**6.2.3.364 plaintextAutomorph()** [1/2]

```
template<typename RX , typename type >
void helib::plaintextAutomorph (
    RX & b,
    const RX & a,
    long i,
    long j,
    const EncryptedArrayDerived< type > & ea )
```

**6.2.3.365 plaintextAutomorph()** [2/2]

```
template<typename RX , typename RXModulus >
void helib::plaintextAutomorph (
    RX & bb,
    const RX & a,
    long k,
    long m,
    const RXModulus & PhimX )
```

**6.2.3.366 poly\_comp()**

```
template<typename RX >
bool helib::poly_comp (
    const RX & a,
    const RX & b )
```

**6.2.3.367 polyEval()** [1/2]

```
void helib::polyEval (
    Ctxt & ret,
    const NTL::Vec< Ctxt > & poly,
    const Ctxt & x )
```

Evaluate an encrypted polynomial on an encrypted input.

## Parameters

out	<i>res</i>	to hold the return value
in	<i>poly</i>	the degree-d polynomial to evaluate
in	<i>x</i>	the point on which to evaluate

## 6.2.3.368 polyEval() [2/2]

```
void helib::polyEval (
    Ctxt & ret,
    NTL::ZZX poly,
    const Ctxt & x,
    long k = 0 )
```

Evaluate a cleartext polynomial on an encrypted input.

## Parameters

out	<i>res</i>	to hold the return value
in	<i>poly</i>	the degree-d polynomial to evaluate
in	<i>x</i>	the point on which to evaluate

## Parameters

<code>in</code>	<code>k</code>	optional optimization parameter, defaults to $\sqrt{d/2}$ rounded up or down to a power of two
-----------------	----------------	--

**6.2.3.369 polyEvalMod()**

```
long helib::polyEvalMod (
    const NTL::ZZX & poly,
    long x,
    long p )
```

Evaluates a modular integer polynomial, returns  $\text{poly}(x) \bmod p$ .

**6.2.3.370 PolyRed() [1/4]**

```
void helib::PolyRed (
    NTL::ZZX & F,
    const NTL::ZZ & q,
    bool abs = false ) [inline]
```

**6.2.3.371 PolyRed() [2/4]**

```
void helib::PolyRed (
    NTL::ZZX & F,
    long q,
    bool abs = false ) [inline]
```

**6.2.3.372 PolyRed()** [3/4]

```
void helib::PolyRed (
    NTL::ZZX & out,
    const NTL::ZZX & in,
    const NTL::ZZ & q,
    bool abs = false )
```

**6.2.3.373 PolyRed()** [4/4]

```
void helib::PolyRed (
    NTL::ZZX & out,
    const NTL::ZZX & in,
    long q,
    bool abs = false )
```

Reduce all the coefficients of a polynomial modulo  $q$ .

When  $abs=false$  reduce to interval  $(-q/2, \dots, q/2)$ , when  $abs=true$  reduce to  $[0, q)$ . When  $abs=false$  and  $q=2$ , maintains the same sign as the input.

**6.2.3.374 power()**

```
void helib::power (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    long e )
```

**6.2.3.375 pp\_factorize()**

```
void helib::pp_factorize (
    std::vector< long > & factors,
    long N )
```

Prime-power factorization.

**6.2.3.376 ppInvert()** [1/4]

```
void helib::ppInvert (
    NTL::mat_GF2 & X,
    const NTL::mat_GF2 & A,
    UNUSED long p,
    UNUSED long r ) [inline]
```

**6.2.3.377 ppInvert() [2/4]**

```
void helib::ppInvert (
    NTL::mat_GF2E & X,
    const NTL::mat_GF2E & A,
    UNUSED long p,
    UNUSED long r ) [inline]
```

**6.2.3.378 ppInvert() [3/4]**

```
void helib::ppInvert (
    NTL::mat_zz_p & X,
    const NTL::mat_zz_p & A,
    long p,
    long r )
```

Compute the inverse mod  $p^r$  of an  $n \times n$  matrix.

[NTL](#)'s current smallint modulus `zz_p::modulus()` is assumed to be  $p^r$  for  $p$  prime,  $r \geq 1$  integer. For the `zz_pE` variant also `zz_pE::modulus()` must be initialized. An error is raised if  $A$  is not invertible mod  $p$ .

**6.2.3.379 ppInvert() [4/4]**

```
void helib::ppInvert (
    NTL::mat_zz_pE & X,
    const NTL::mat_zz_pE & A,
    long p,
    long r )
```

**6.2.3.380 ppsolve() [1/2]**

```
void helib::ppsolve (
    NTL::vec_GF2E & x,
    const NTL::mat_GF2E & A,
    const NTL::vec_GF2E & b,
    long p,
    long r )
```

A version for GF2: must have  $p == 2$  and  $r == 1$ .

**6.2.3.381 ppsolve() [2/2]**

```
void helib::ppsolve (
    NTL::vec_zz_pE & x,
    const NTL::mat_zz_pE & A,
    const NTL::vec_zz_pE & b,
    long p,
    long r )
```

Prime power solver.

A is an  $n \times n$  matrix, b is a length n (row) vector, this function finds a solution for the matrix-vector equation  $x A = b$ . An error is raised if A is not invertible mod p.

NTL's current smallint modulus, `zz_p::modulus()`, is assumed to be  $p^r$ , for p prime,  $r \geq 1$  integer.

**6.2.3.382 primroot()**

```
long helib::primroot (
    long N,
    long phiN )
```

Find a primitive root modulo N.

**6.2.3.383 print() [1/2]**

```
void helib::print (
    const EncryptedArray & ea,
    std::ostream & s,
    const PlaintextArray & pa )
```

**6.2.3.384 print() [2/2]**

```
void helib::print (
    std::ostream & s,
    SplitNodePtr p,
    bool first )
```

**6.2.3.385 print3D() [1/3]**

```
template void helib::print3D (
    const HyperCube< long > & c )
```

**6.2.3.386 print3D()** [2/3]

```
template void helib::print3D (
    const HyperCube< NTL::zz_p > & c )
```

**6.2.3.387 print3D()** [3/3]

```
template<typename T >
void helib::print3D (
    const HyperCube< T > & c )
```

**6.2.3.388 print\_stats()**

```
void helib::print_stats (
    std::ostream & s )
```

**6.2.3.389 printAllTimers()**

```
void helib::printAllTimers (
    std::ostream & str = std::cerr )
```

Print the value of all timers to stream.

**6.2.3.390 printFlow()**

```
void helib::printFlow (
    FlowGraph & fg )
```

**6.2.3.391 printNamedTimer()**

```
bool helib::printNamedTimer (
    std::ostream & str,
    const char * name )
```



**6.2.3.392 printVec()**

```
template<typename VEC >
std::ostream& helib::printVec (
    std::ostream & s,
    const VEC & v,
    long nCoeffs = 40 )
```

**6.2.3.393 printZZX()**

```
std::ostream& helib::printZZX (
    std::ostream & s,
    const NTL::ZZX & poly,
    long nCoeffs = 40 ) [inline]
```

**6.2.3.394 ptr2nonNull()**

```
template<typename T >
const T* helib::ptr2nonNull (
    std::initializer_list< const PtrVector< T > * > list )
```

**6.2.3.395 random()**

```
void helib::random (
    const EncryptedArray & ea,
    PlaintextArray & pa )
```

**6.2.3.396 randomPerm()**

```
void helib::randomPerm (
    Permut & perm,
    long n )
```

A random size-n permutation.

**6.2.3.397 randomSlot()**

```
template<typename Scheme >
Scheme::SlotType helib::randomSlot (
    const Context & context )
```

**6.2.3.398 randomSlot< BGV >()**

```
template<>
BGV::SlotType helib::randomSlot< BGV > (
    const Context & context )
```

**6.2.3.399 randomSlot< CKKS >()**

```
template<>
CKKS::SlotType helib::randomSlot< CKKS > (
    UNUSED const Context & context )
```

**6.2.3.400 RandPoly()**

```
NTL::ZZX helib::RandPoly (
    long n,
    const NTL::ZZ & p )
```

**6.2.3.401 range() [1/2]**

```
general_range<long> helib::range (
    long m,
    long n ) [inline]
```

**6.2.3.402 range() [2/2]**

```
general_range<long> helib::range (
    long n ) [inline]
```

**6.2.3.403 rationalApprox() [1/2]**

```
std::pair< long, long > helib::rationalApprox (
    double x,
    long denomBound = 0 )
```

**6.2.3.404 rationalApprox() [2/2]**

```
std::pair< NTL::ZZ, NTL::ZZ > helib::rationalApprox (
    NTL::xdouble x,
    NTL::xdouble denomBound = NTL::xdouble(0.0) )
```

**6.2.3.405 rawDecrypt()**

```
void helib::rawDecrypt (
    NTL::ZZX & plaintext,
    const std::vector< NTL::ZZX > & zzParts,
    const DoubleCRT & sKey,
    long q = 0 )
```

**6.2.3.406 read()**

```
void helib::read (
    std::istream & s,
    ModuliSizes::Entry & e )
```

**6.2.3.407 read\_ntl\_vec\_long()**

```
void helib::read_ntl_vec_long (
    std::istream & str,
    NTL::vec_long & vl )
```

**6.2.3.408 read\_raw\_double()**

```
double helib::read_raw_double (
    std::istream & str )
```

**6.2.3.409 read\_raw\_int()**

```
long helib::read_raw_int (
    std::istream & str )
```

#### 6.2.3.410 read\_raw\_int32()

```
int helib::read_raw_int32 (
    std::istream & str )
```

#### 6.2.3.411 read\_raw\_vector() [1/3]

```
template<typename T >
void helib::read_raw_vector (
    std::istream & str,
    std::vector< T > & v )
```

#### 6.2.3.412 read\_raw\_vector() [2/3]

```
template<typename T >
void helib::read_raw_vector (
    std::istream & str,
    std::vector< T > & v,
    const Context & context )
```

#### 6.2.3.413 read\_raw\_vector() [3/3]

```
template<typename T >
void helib::read_raw_vector (
    std::istream & str,
    std::vector< T > & v,
    T & init )
```

#### 6.2.3.414 read\_raw\_vector< double >()

```
template<>
void helib::read_raw_vector< double > (
    std::istream & str,
    std::vector< double > & v )
```

#### 6.2.3.415 read\_raw\_vector< long >()

```
template<>
void helib::read_raw_vector< long > (
    std::istream & str,
    std::vector< long > & v )
```

**6.2.3.416 read\_raw\_xdouble()**

```
Ntl::xdouble helib::read_raw_xdouble (
    std::istream & str )
```

**6.2.3.417 read\_raw\_ZZ()**

```
void helib::read_raw_ZZ (
    std::istream & str,
    Ntl::ZZ & zz )
```

**6.2.3.418 readContextBase()**

```
void helib::readContextBase (
    std::istream & s,
    unsigned long & m,
    unsigned long & p,
    unsigned long & r,
    std::vector< long > & gens,
    std::vector< long > & ords )
```

read [m p r gens ords] data, needed to construct context

read [m p r] data, needed to construct context

**6.2.3.419 readContextBaseBinary()**

```
void helib::readContextBaseBinary (
    std::istream & s,
    unsigned long & m,
    unsigned long & p,
    unsigned long & r,
    std::vector< long > & gens,
    std::vector< long > & ords )
```

read [m p r gens ords] data, needed to construct context

**6.2.3.420 readContextBinary()**

```
void helib::readContextBinary (
    std::istream & str,
    Context & context )
```

**6.2.3.421 readEyeCatcher()**

```
int helib::readEyeCatcher (
    std::istream & str,
    const char * expect )
```

**6.2.3.422 readPubKeyBinary()**

```
void helib::readPubKeyBinary (
    std::istream & str,
    PubKey & pk )
```

**6.2.3.423 readSecKeyBinary()**

```
void helib::readSecKeyBinary (
    std::istream & str,
    SecKey & sk )
```

**6.2.3.424 realToEstimatedNoise()**

```
double helib::realToEstimatedNoise (
    const Ctxt & ctxt,
    const SecKey & sk )
```

**6.2.3.425 recordAutomorphVal()**

```
void helib::recordAutomorphVal (
    long k ) [inline]
```

**6.2.3.426 recordAutomorphVal2()**

```
void helib::recordAutomorphVal2 (
    long k ) [inline]
```

**6.2.3.427 reducedCount()**

```
long helib::reducedCount (
    const std::list< long > & x,
    long n,
    bool * aux )
```

**6.2.3.428 reduceModPhimX()**

```
void helib::reduceModPhimX (
    zzX & poly,
    const PAlgebra & palg )
```

**6.2.3.429 registerTimer()**

```
void helib::registerTimer (
    FHEtimer * timer )
```

**6.2.3.430 RelaxedInv() [1/2]**

```
void helib::RelaxedInv (
    NTL::Mat< NTL::GF2 > & x,
    const NTL::Mat< NTL::GF2 > & a )
```

**6.2.3.431 RelaxedInv() [2/2]**

```
void helib::RelaxedInv (
    NTL::Mat< NTL::zz_p > & x,
    const NTL::Mat< NTL::zz_p > & a )
```

**6.2.3.432 rem()**

```
void helib::rem (
    NTL::zz_pX & r,
    const NTL::zz_pX & a,
    const zz_pXModulus1 & ff )
```

**6.2.3.433 removeDups()**

```
void helib::removeDups (
    std::list< long > & x,
    bool * aux )
```

**6.2.3.434 repack() [1/2]**

```
long helib::repack (
    const CtPtrs & packed,
    const CtPtrs & unpacked,
    const EncryptedArray & ea )
```

**6.2.3.435 repack() [2/2]**

```
void helib::repack (
    Ctxt & packed,
    const CtPtrs & unpacked,
    const EncryptedArray & ea )
```

**6.2.3.436 replicate() [1/3]**

```
template<typename Scheme >
void helib::replicate (
    const EncryptedArray & ,
    Ptxt< Scheme > & ptxt,
    long i )
```

Replicate single slot of a [Ptxt](#) object across all of its slots.

**Template Parameters**

<i>Scheme</i>	Encryption scheme used (must be <a href="#">BGV</a> or <a href="#">CKKS</a> ).
---------------	--

**Parameters**

<i>ptxt</i>	Plaintext on which to do the replication.
-------------	---



## Parameters

<i>i</i>	Position of the slot to replicate.
----------	------------------------------------

## Returns

Reference to `*this` post replication.

**6.2.3.437 replicate()** [2/3]

```
void helib::replicate (
    const EncryptedArray & ea,
    Ctxt & ctx,
    long pos )
```

The value in slot #pos is replicated in all other slots. On an n-slot ciphertext, this algorithm performs  $O(\log n)$  1D rotations.

**6.2.3.438 replicate()** [3/3]

```
void helib::replicate (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    long i )
```

**6.2.3.439 replicate0()**

```
void helib::replicate0 (
    const EncryptedArray & ea,
    Ctxt & ctx,
    long pos )
```

A lower-level routine. Same as replicate, but assumes all slots are zero except slot #pos.

**6.2.3.440 replicateAll() [1/3]**

```
void helib::replicateAll (
    const EncryptedArray & ea,
    const Ctxt & ctxt,
    ReplicateHandler * handler,
    long recBound = 64,
    RepAuxDim * repAuxPtr = nullptr )
```

replicateAll uses a hybrid strategy, combining the  $O(\log n)$  strategy of the replicate method, with an  $O(1)$  strategy, which is faster but introduces more noise. This tradeoff is controlled by the parameter recBound:

- $\text{recBound} < 0$ : recursion to depth  $|\text{recBound}|$  (faster, noisier)
- $\text{recBound} == 0$ : no recursion (slower, less noise)
- $\text{recBound} > 0$ : the recursion depth is chosen heuristically, but is capped at recBound

The default value for recBound is 64, this ensures that the choice is based only on the heuristic, which will introduce noise corresponding to  $O(\log \log n)$  levels of recursion, but still gives an algorithm that theoretically runs in time  $O(n)$ .

**6.2.3.441 replicateAll() [2/3]**

```
void helib::replicateAll (
    std::vector< Ctxt > & v,
    const EncryptedArray & ea,
    const Ctxt & ctxt,
    long recBound = 64,
    RepAuxDim * repAuxPtr = nullptr )
```

return the result as a `std::vector` of ciphertexts, mostly useful for debugging purposes (for real parameters would take a lot of memory)

**6.2.3.442 replicateAll() [3/3]**

```
template<typename Scheme >
void helib::replicateAll (
    std::vector< Ptxt< Scheme >> & v,
    const EncryptedArray & ,
    const Ptxt< Scheme > & ptxt )
```

Generate a vector of plaintexts with each slot replicated in each plaintext.

**Template Parameters**

<i>Scheme</i>	Encryption scheme used (must be <a href="#">BGV</a> or <a href="#">CKKS</a> ).
---------------	--

## Parameters

<i>v</i>	Vector of replicated plain-text slots.
<i>ptxt</i>	Plaintext whose slots will be replicated.

The order of the return vector agrees with the order of the slots. i.e. the *i*th plaintext in the return value is a replication of *\*this[i]*.

**6.2.3.443 replicateAllOrig()**

```
void helib::replicateAllOrig (
    const EncryptedArray & ea,
    const Ctxt & ctxt,
    ReplicateHandler * handler,
    RepAux * repAuxPtr = nullptr )
```

This function is obsolete, and is kept for historical purposes only. It was a first attempt at implementing the  $O(1)$ -amortized algorithm, but is less efficient than the function above.

**6.2.3.444 resetAllTimers()**

```
void helib::resetAllTimers ( )
```

**6.2.3.445 resize()** [1/7]

```
template<typename T >
void helib::resize (
    NTL::Vec< T > & v,
    long sz )
```

**6.2.3.446 resize()** [2/7]

```
template<typename T >
void helib::resize (
    NTL::Vec< T > & v,
    long sz,
    const T & val )
```

**6.2.3.447 resize()** [3/7]

```
template<typename T >
void helib::resize (
    PtrMatrix< T > & v,
    long newSize )
```

**6.2.3.448 resize()** [4/7]

```
template<typename T >
void helib::resize (
    PtrVector< T > & v,
    long newSize,
    const T & val )
```

**6.2.3.449 resize()** [5/7]

```
template<typename T >
void helib::resize (
    PtrVector< T > & v,
    long newSize,
    const T * val )
```

**6.2.3.450 resize()** [6/7]

```
template<typename T >
void helib::resize (
    std::vector< T > & v,
    long sz )
```

**6.2.3.451 resize()** [7/7]

```
template<typename T >
void helib::resize (
    std::vector< T > & v,
    long sz,
    const T & val )
```

**6.2.3.452 reverse()**

```
template<typename T >
void helib::reverse (
    NTL::Vec< T > & v,
    long lo,
    long hi )
```

Reverse a vector in place.

**6.2.3.453 RLWE()**

```
double helib::RLWE (
    DoubleCRT & c0,
    DoubleCRT & c1,
    const DoubleCRT & s,
    long p,
    NTL::ZZ * prgSeed = nullptr )
```

Choose random  $c_0, c_1$  such that  $c_0 + s \cdot c_1 = p \cdot e$  for a short  $e$  Returns a high-probability bound on the L-infty norm of the canonical embedding

**6.2.3.454 RLWE1()**

```
double helib::RLWE1 (
    DoubleCRT & c0,
    const DoubleCRT & c1,
    const DoubleCRT & s,
    long p )
```

Same as RLWE, but assumes that  $c_1$  is already chosen by the caller.

**6.2.3.455 rotate() [1/2]**

```
void helib::rotate (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    long k )
```

**6.2.3.456 rotate() [2/2]**

```
template<typename T >
void helib::rotate (
    NTL::Vec< T > & v,
    long k )
```

Rotate a vector in place using swaps.

**6.2.3.457 runningSums()** [1/2]

```
void helib::runningSums (
    const EncryptedArray & ea,
    Ctxt & ctxt )
```

A ctxt that encrypts  $(x_1, \dots, x_n)$  is replaced by an encryption of  $(y_1, \dots, y_n)$ , where  $y_i = \sum_{j \leq i} x_j$ .

**6.2.3.458 runningSums()** [2/2]

```
void helib::runningSums (
    CtPtrs & v )
```

**6.2.3.459 sameObject()**

```
template<typename T1 , typename T2 >
bool helib::sameObject (
    const T1 * p1,
    const T2 * p2 )
```

Testing if two vectors point to the same object.

**6.2.3.460 sampleGaussian()** [1/4]

```
void helib::sampleGaussian (
    NTL::ZZX & poly,
    long n,
    double stdev )
```

**6.2.3.461 sampleGaussian()** [2/4]

```
void helib::sampleGaussian (
    std::vector< double > & dvec,
    long n,
    double stdev )
```

Choose a vector of continuous Gaussians.

**6.2.3.462 sampleGaussian() [3/4]**

```
double helib::sampleGaussian (
    zzX & poly,
    const Context & context,
    double stdev )
```

**6.2.3.463 sampleGaussian() [4/4]**

```
void helib::sampleGaussian (
    zzX & poly,
    long n,
    double stdev )
```

Sample polynomials with Gaussian coefficients.

**6.2.3.464 sampleGaussianBounded()**

```
double helib::sampleGaussianBounded (
    zzX & poly,
    const Context & context,
    double stdev )
```

**6.2.3.465 sampleHWt() [1/3]**

```
void helib::sampleHWt (
    NTL::ZZX & poly,
    long n,
    long Hwt = 100 )
```

**6.2.3.466 sampleHWt() [2/3]**

```
double helib::sampleHWt (
    zzX & poly,
    const Context & context,
    long Hwt = 100 )
```

**6.2.3.467 sampleHWt()** [3/3]

```
void helib::sampleHWt (
    zzX & poly,
    long n,
    long Hwt = 100 )
```

Sample a degree-(n-1) poly as above, with only Hwt nonzero coefficients.

**6.2.3.468 sampleHWtBounded()**

```
double helib::sampleHWtBounded (
    zzX & poly,
    const Context & context,
    long Hwt = 100 )
```

**6.2.3.469 sampleHWtBoundedEffectiveBound()**

```
double helib::sampleHWtBoundedEffectiveBound (
    const Context & context,
    long Hwt = 100 )
```

**6.2.3.470 sampleSmall()** [1/3]

```
void helib::sampleSmall (
    NTL::ZZX & poly,
    long n,
    double prob = 0.5 )
```

**6.2.3.471 sampleSmall()** [2/3]

```
double helib::sampleSmall (
    zzX & poly,
    const Context & context )
```



**6.2.3.472 sampleSmall()** [3/3]

```
void helib::sampleSmall (
    zzX & poly,
    long n,
    double prob = 0.5 )
```

Sample a degree-(n-1) poly, with -1/0/+1 coefficients. Each coefficients is +-1 with probability prob/2 each, and 0 with probability 1-prob. By default, pr[nonzero]=1/2.

**6.2.3.473 sampleSmallBounded()**

```
double helib::sampleSmallBounded (
    zzX & poly,
    const Context & context )
```

**6.2.3.474 sampleUniform()** [1/4]

```
NTL::xdouble helib::sampleUniform (
    NTL::ZZX & poly,
    const Context & context,
    const NTL::ZZ & B = NTL::ZZ(100L) )
```

**6.2.3.475 sampleUniform()** [2/4]

```
void helib::sampleUniform (
    NTL::ZZX & poly,
    long n,
    const NTL::ZZ & B = NTL::ZZ(100L) )
```

**6.2.3.476 sampleUniform()** [3/4]

```
double helib::sampleUniform (
    zzX & poly,
    const Context & context,
    long B = 100 )
```

**6.2.3.477 sampleUniform()** [4/4]

```
void helib::sampleUniform (
    zzX & poly,
    long n,
    long B = 100 )
```

Sample a degree-(n-1) ZZ<sub>X</sub>, with coefficients uniform in [-B,B].

**6.2.3.478 seekPastChar()**

```
void helib::seekPastChar (
    std::istream & str,
    int cc )
```

Advance the input stream beyond white spaces and a single instance of the char cc.

**6.2.3.479 setAutomorphVals()**

```
void helib::setAutomorphVals (
    std::set< long > * aVals ) [inline]
```

**6.2.3.480 setAutomorphVals2()**

```
void helib::setAutomorphVals2 (
    std::set< long > * aVals ) [inline]
```

**6.2.3.481 setDryRun()**

```
bool helib::setDryRun (
    bool toWhat = true ) [inline]
```

**6.2.3.482 setHyperColumn()** [1/6]

```
template void helib::setHyperColumn (
    const NTL::Vec< long > & v,
    const CubeSlice< long > & s,
    long pos )
```

**6.2.3.483 setHyperColumn() [2/6]**

```
template void helib::setHyperColumn (
    const NTL::Vec< long > & v,
    const CubeSlice< long > & s,
    long pos,
    const long & val )
```

**6.2.3.484 setHyperColumn() [3/6]**

```
template void helib::setHyperColumn (
    const NTL::Vec< NTL::zz_p > & v,
    const CubeSlice< NTL::zz_p > & s,
    long pos )
```

**6.2.3.485 setHyperColumn() [4/6]**

```
template void helib::setHyperColumn (
    const NTL::Vec< NTL::zz_p > & v,
    const CubeSlice< NTL::zz_p > & s,
    long pos,
    const NTL::zz_p & val )
```

**6.2.3.486 setHyperColumn() [5/6]**

```
template<typename T >
void helib::setHyperColumn (
    const NTL::Vec< T > & v,
    const CubeSlice< T > & s,
    long pos )
```

setHyperColumn does the reverse of getHyperColumn, setting the column to the given vector

**6.2.3.487 setHyperColumn() [6/6]**

```
template<typename T >
void helib::setHyperColumn (
    const NTL::Vec< T > & v,
    const CubeSlice< T > & s,
    long pos,
    const T & val )
```

this version of setHyperColumn implicitly pads v with a default value, if v is too short

**6.2.3.488 setLengthZero()** [1/4]

```
template<typename T >
void helib::setLengthZero (
    NTL::Vec< T > & vec )
```

**6.2.3.489 setLengthZero()** [2/4]

```
template<typename T >
void helib::setLengthZero (
    PtrMatrix< T > & v )
```

**6.2.3.490 setLengthZero()** [3/4]

```
template<typename T >
void helib::setLengthZero (
    PtrVector< T > & v )
```

**6.2.3.491 setLengthZero()** [4/4]

```
template<typename T >
void helib::setLengthZero (
    std::vector< T > & vec )
```

**6.2.3.492 setTimersOff()**

```
void helib::setTimersOff ( ) [inline]
```

**6.2.3.493 setTimersOn()**

```
void helib::setTimersOn ( ) [inline]
```

**6.2.3.494 setupDebugGlobals()**

```
void helib::setupDebugGlobals (
    SecKey * debug_key,
    const std::shared_ptr< const EncryptedArray > & debug_ea,
    NTL::ZZX debug_ptxt = NTL::ZZX{} ) [inline]
```

Setup function for setting up the global debug variables.

**Note**

Works only if HELIB\_DEBUG is defined. It does not do anything otherwise

**6.2.3.495 shift()**

```
void helib::shift (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    long k )
```

**6.2.3.496 splitBinaryNums()**

```
void helib::splitBinaryNums (
    CtPtrs & leftSplit,
    CtPtrs & rightSplit,
    const CtPtrs & input )
```

Splits a single binary number into two binary numbers `leftSplit` and `rightSplit`.

Split a binary number into two separate binary numbers.

**Parameters**

<i>leftSplit</i>	Left hand side of the split.
<i>rightSplit</i>	Right hand side of the split.
<i>input</i>	Binary number to be split.

**Note**

The size of `leftSplit` and `rightSplit` must sum to the size of `input`.

The location of the split is defined by the sizes of `leftSplit` and `rightSplit`.

**6.2.3.497 sub()**

```
void helib::sub (
    const EncryptedArray & ea,
    PlaintextArray & pa,
    const PlaintextArray & other )
```

**6.2.3.498 subtractBinary()**

```
void helib::subtractBinary (
    CtPtrs & difference,
    const CtPtrs & lhs,
    const CtPtrs & rhs,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

Subtracts `rhs` from `lhs` where `lhs`, `rhs` are in 2's complement.

**Parameters**

<i>difference</i>	Reference to the difference post subtraction.
<i>lhs</i>	Left hand side of subtraction.
<i>rhs</i>	Right hand side of subtraction.
<i>unpackSlotEncoding</i>	vector of constants for unpacking, as used in
	bootstrapping.

**Note**

`lhs` and `rhs` must have the same size.

**6.2.3.499 sumOfCoeffs() [1/3]**

```
NTL::ZZ helib::sumOfCoeffs (
    const DoubleCRT & f )
```

**6.2.3.500 sumOfCoeffs() [2/3]**

```
NTL::ZZ helib::sumOfCoeffs (
    const NTL::ZZX & f )
```

**6.2.3.501 sumOfCoeffs() [3/3]**

```
long helib::sumOfCoeffs (
    const ZZ & f )
```

**6.2.3.502 swap() [1/2]**

```
template<typename X , typename Cloner >
void helib::swap (
    cloned_ptr< X, Cloner > & x,
    cloned_ptr< X, Cloner > & y )
```

**6.2.3.503 swap() [2/2]**

```
template<typename X , typename Cloner >
void helib::swap (
    copied_ptr< X, Cloner > & x,
    copied_ptr< X, Cloner > & y )
```

**6.2.3.504 tableLookup()**

```
void helib::tableLookup (
    Ctxt & out,
    const std::vector< zzX > & table,
    const CTPtrs & idx,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

The input is a plaintext table  $T[]$  and an array of encrypted bits  $I[]$ , holding the binary representation of an index  $i$  into  $T$ . The output is the encrypted value  $T[i]$ .

**6.2.3.505 tableWriteIn()**

```
void helib::tableWriteIn (
    const CTPtrs & table,
    const CTPtrs & idx,
    std::vector< zzX > * unpackSlotEncoding = nullptr )
```

The input is an encrypted table  $T[]$  and an array of encrypted bits  $I[]$ , holding the binary representation of an index  $i$  into  $T$ . This function increments by one the entry  $T[i]$ .

**6.2.3.506 timer\_compare()**

```
bool helib::timer_compare (
    const FHEtimer * a,
    const FHEtimer * b )
```

**6.2.3.507 to\_ZZX()**

```
NTL::ZZX helib::to_ZZX (
    const DoubleCRT & d ) [inline]
```

**6.2.3.508 TofftRep\_trunc()** [1/2]

```
void helib::TofftRep_trunc (
    NTL::fftRep & y,
    const NTL::zz_pX & x,
    long k,
    long len ) [inline]
```



**6.2.3.509 TofftRep\_trunc()** [2/2]

```
void helib::TofftRep_trunc (
    NTL::fftRep & y,
    const NTL::zz_pX & x,
    long k,
    UNUSED long len,
    long lo,
    long hi ) [inline]
```

**6.2.3.510 totalProduct()**

```
void helib::totalProduct (
    Ctxt & out,
    const std::vector< Ctxt > & v )
```

**6.2.3.511 totalSums()**

```
void helib::totalSums (
    const EncryptedArray & ea,
    Ctxt & ctxt )
```

**6.2.3.512 traceMap()**

```
void helib::traceMap (
    Ctxt & ctxt )
```

**6.2.3.513 TraceMap()**

```
void helib::TraceMap (
    NTL::GF2X & w,
    const NTL::GF2X & a,
    long d,
    const NTL::GF2XModulus & F,
    const NTL::GF2X & b )
```

**6.2.3.514 unpack() [1/2]**

```
long helib::unpack (
    const CtPtrs & unpacked,
    const CtPtrs & packed,
    const EncryptedArray & ea,
    const std::vector< zzX > & unpackSlotEncoding )
```

**6.2.3.515 unpack() [2/2]**

```
void helib::unpack (
    const CtPtrs & unpacked,
    const Ctxt & packed,
    const EncryptedArray & ea,
    const std::vector< zzX > & unpackSlotEncoding )
```

**6.2.3.516 unpackSlots() [1/2]**

```
void helib::unpackSlots (
    std::vector< unsigned long > & value,
    NTL::ZZX & pa,
    const EncryptedArray & ea ) [inline]
```

**6.2.3.517 unpackSlots() [2/2]**

```
void helib::unpackSlots (
    std::vector< unsigned long > & value,
    PlaintextArray & pa,
    const EncryptedArray & ea )
```

**6.2.3.518 Vec\_replicate()**

```
template<typename T >
std::vector<T> helib::Vec_replicate (
    const T & a,
    long n )
```

**6.2.3.519 vecCopy() [1/4]**

```
template<typename T >
void helib::vecCopy (
    PtrVector< T > & v1,
    const PtrVector< T > & v2,
    long sizeLimit = 0 )
```

**6.2.3.520 vecCopy() [2/4]**

```
template<typename V , typename T >
void helib::vecCopy (
    PtrVector< T > & v1,
    const V & v2,
    long sizeLimit = 0 )
```

**6.2.3.521 vecCopy() [3/4]**

```
template<typename V , typename T >
void helib::vecCopy (
    V & v1,
    const PtrVector< T > & v2,
    long sizeLimit = 0 )
```

**6.2.3.522 vecCopy() [4/4]**

```
template<typename V1 , typename V2 >
void helib::vecCopy (
    V1 & v1,
    const V2 & v2,
    long sizeLimit = 0 )
```

**6.2.3.523 vecRed() [1/2]**

```
void helib::vecRed (
    NTL::Vec< NTL::ZZ > & out,
    const NTL::Vec< NTL::ZZ > & in,
    const NTL::ZZ & q,
    bool abs )
```

**6.2.3.524 vecRed()** [2/2]

```
void helib::vecRed (
    NTL::Vec< NTL::ZZ > & out,
    const NTL::Vec< NTL::ZZ > & in,
    long q,
    bool abs )
```

**6.2.3.525 vector\_replicate()**

```
template<typename T >
std::vector<T> helib::vector_replicate (
    const T & a,
    long n )
```

**6.2.3.526 vecToStr()**

```
template<typename T >
std::string helib::vecToStr (
    const std::vector< T > & v )
```

**6.2.3.527 Warning()** [1/2]

```
void helib::Warning (
    const char * msg ) [inline]
```

**6.2.3.528 Warning()** [2/2]

```
void helib::Warning (
    const std::string & msg ) [inline]
```

**6.2.3.529 write()**

```
void helib::write (
    std::ostream & s,
    const ModuliSizes::Entry & e )
```

**6.2.3.530 write\_ntl\_vec\_long()**

```
void helib::write_ntl_vec_long (
    std::ostream & str,
    const NTL::vec_long & vl,
    long intSize = BINIO_64BIT )
```

**6.2.3.531 write\_raw\_double()**

```
void helib::write_raw_double (
    std::ostream & str,
    const double d )
```

**6.2.3.532 write\_raw\_int()**

```
void helib::write_raw_int (
    std::ostream & str,
    long num )
```

**6.2.3.533 write\_raw\_int32()**

```
void helib::write_raw_int32 (
    std::ostream & str,
    int num )
```

**6.2.3.534 write\_raw\_vector()**

```
template<typename T >
void helib::write_raw_vector (
    std::ostream & str,
    const std::vector< T > & v )
```

**6.2.3.535 write\_raw\_vector< double >()**

```
template<>
void helib::write_raw_vector< double > (
    std::ostream & str,
    const std::vector< double > & v )
```

**6.2.3.536 write\_raw\_vector< long >()**

```
template<>
void helib::write_raw_vector< long > (
    std::ostream & str,
    const std::vector< long > & v )
```

**6.2.3.537 write\_raw\_xdouble()**

```
void helib::write_raw_xdouble (
    std::ostream & str,
    const NTL::xdouble xd )
```

**6.2.3.538 write\_raw\_ZZ()**

```
void helib::write_raw_ZZ (
    std::ostream & str,
    const NTL::ZZ & zz )
```

**6.2.3.539 writeContextBase()**

```
void helib::writeContextBase (
    std::ostream & s,
    const Context & context )
```

write [m p r gens ords] data

write [m p r] data

**6.2.3.540 writeContextBaseBinary()**

```
void helib::writeContextBaseBinary (
    std::ostream & str,
    const Context & context )
```

write [m p r gens ords] data

**6.2.3.541 writeContextBinary()**

```
void helib::writeContextBinary (
    std::ostream & str,
    const Context & context )
```

#### 6.2.3.542 writeEyeCatcher()

```
void helib::writeEyeCatcher (
    std::ostream & str,
    const char * eye )
```

#### 6.2.3.543 writePubKeyBinary()

```
void helib::writePubKeyBinary (
    std::ostream & str,
    const PubKey & pk )
```

#### 6.2.3.544 writeSecKeyBinary()

```
void helib::writeSecKeyBinary (
    std::ostream & str,
    const SecKey & sk )
```

### 6.2.4 Variable Documentation

#### 6.2.4.1 activeContext

```
Context * helib::activeContext = nullptr
```

#### 6.2.4.2 CLOCK\_SCALE

```
const unsigned long helib::CLOCK_SCALE = (unsigned long)CLOCKS_PER_SEC
```

#### 6.2.4.3 dbg\_ptxt

```
NTL::ZZX helib::dbg_ptxt
```

#### 6.2.4.4 dbgEa

```
std::shared_ptr< const EncryptedArray > helib::dbgEa = nullptr
```

#### 6.2.4.5 dbgKey

```
SecKey * helib::dbgKey = nullptr
```

#### 6.2.4.6 erfc\_inverse

```
const double helib::erfc_inverse[]
```

**Initial value:**

```
= {0,
    0.6744897501960817432,
    1.1503493803760081782,
    1.5341205443525463117,
    1.8627318674216514554,
    2.1538746940614562129,
    2.4175590162365050618,
    2.6600674686174596585,
    2.8856349124267571473,
    3.0972690781987844623,
    3.2971933456919633418,
    3.4871041041144311068,
    3.6683292851213230192,
    3.8419306855019108708,
    4.0087725941685849622,
    4.1695693233491057549,
    4.3249190408260462571,
    4.4753284246542033544,
    4.6212310014992471565,
    4.7630010342678139569,
    4.9009642079631930118}
```

#### 6.2.4.7 fhe\_force\_chen\_han

```
long helib::fhe_force_chen_han = 0
```

#### 6.2.4.8 fhe\_stats

```
bool helib::fhe_stats = false
```

#### 6.2.4.9 fhe\_test\_force\_bsgs

```
int helib::fhe_test_force_bsgs = 0
```



#### 6.2.4.10 the\_test\_force\_hoist

```
int helib::the_test_force_hoist = 0
```

#### 6.2.4.11 the\_watcher

```
int helib::the_watcher = 0
```

#### 6.2.4.12 PI

```
const long double helib::PI
```

**Initial value:**

```
=
3.1415926535897932384626433832795028841971693993751058209749445923078164L
```

#### 6.2.4.13 printFlag

```
long helib::printFlag
```

#### 6.2.4.14 replicateVerboseFlag

```
NL_THREAD_LOCAL bool helib::replicateVerboseFlag = false
```

#### 6.2.4.15 thinRecrypt\_initial\_level

```
long helib::thinRecrypt_initial_level
```

## 6.3 helib::FHEglobals Namespace Reference

### Variables

- bool `dryRun` = false  
A dry-run flag The dry-run option disables most operations, to save time. This lets us quickly go over the evaluation of a circuit and estimate the resulting noise magnitude, without having to actually compute anything.
- std::set< long > \* `automorphVals` = nullptr  
A list of required automorphisms When non-nullptr, causes Ctxt::smartAutomorphism to just record the requested automorphism rather than actually performing it. This can be used to get a list of needed automorphisms for certain operations and then generate all these key-switching matrices. Should only be used in conjunction with `dryRun=true`.
- std::set< long > \* `automorphVals2` = nullptr

## 6.3.1 Variable Documentation

### 6.3.1.1 automorphVals

```
std::set< long > * helib::FHEglobals::automorphVals = nullptr
```

A list of required automorphisms When non-nullptr, causes Ctxt::smartAutomorphism to just record the requested automorphism rather than actually performing it. This can be used to get a list of needed automorphisms for certain operations and then generate all these key-switching matrices. Should only be used in conjunction with dryRun=true.

### 6.3.1.2 automorphVals2

```
std::set< long > * helib::FHEglobals::automorphVals2 = nullptr
```

### 6.3.1.3 dryRun

```
bool helib::FHEglobals::dryRun = false
```

A dry-run flag The dry-run option disables most operations, to save time. This lets us quickly go over the evaluation of a circuit and estimate the resulting noise magnitude, without having to actually compute anything.

## 6.4 NTL Namespace Reference

## 6.5 std Namespace Reference

## Chapter 7

# Class Documentation

### 7.1 `helib::add_pa_impl< type >` Class Template Reference

#### Static Public Member Functions

- static void `apply` (const `EncryptedArrayDerived< type >` &ea, `PlaintextArray` &pa, const `PlaintextArray` &other)

#### 7.1.1 Member Function Documentation

##### 7.1.1.1 `apply()`

```
template<typename type >
static void helib::add_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const PlaintextArray & other ) [inline], [static]
```

The documentation for this class was generated from the following file:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EncryptedArray.cpp`

### 7.2 `helib::AddDAG` Class Reference

A class representing the logic of the order of bit products when adding two integers.

## Public Member Functions

- void `init` (const `CtPtrs` &a, const `CtPtrs` &b)  
*Build a plan to add a and b.*
- `AddDAG` (const `CtPtrs` &a, const `CtPtrs` &b)
- void `apply` (`CtPtrs` &sum, const `CtPtrs` &a, const `CtPtrs` &b, long sizeLimit=0)  
*Perform the actual addition.*
- long `lowLvl` () const  
*Returns the lowest level in this DAG.*
- `DAGnode` \* `findP` (long i, long j) const  
*Returns a pointer to the a 'p' node of index (i,j)*
- `DAGnode` \* `findQ` (long i, long j) const  
*Returns a pointer to the a 'q' node of index (i,j)*

### 7.2.1 Detailed Description

A class representing the logic of the order of bit products when adding two integers.

Given two input arrays `a[]`, `b[]`, we build a DAG with each node representing a term either of the form  $p_{\{i,j\}} = \prod_{t=j}^i (a[t]+b[t])$ , or of the form  $q_{\{i,j\}} = (a[j]*b[j]) * \prod_{t=j+1}^i (a[t]+b[t])$ . The source nodes are of the forms  $(a[i]*b[i])$  and  $(a[i]+b[i])$ , and each non-source node has exactly two parents, whose product yields that node.

When building the DAG, we keep the level of each node as high as possible. For example we can set  $q_{\{i,j\}} = p_{\{i,k\}} * q_{\{k-1,j\}}$  or  $q_{\{i,j\}} = p_{\{i,k+1\}} * q_{\{k,j\}}$  (among other options), and we choose the option that results in the highest level. In addition, we try to minimize the number of nodes in the DAG that actually need to be computed while adding the two numbers (subject to still consuming as few levels as possible).

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 AddDAG()

```
helib::AddDAG::AddDAG (
    const CtPtrs & a,
    const CtPtrs & b ) [inline]
```

### 7.2.3 Member Function Documentation

#### 7.2.3.1 apply()

```
void helib::AddDAG::apply (
    CtPtrs & sum,
    const CtPtrs & a,
    const CtPtrs & b,
    long sizeLimit = 0 )
```

Perform the actual addition.

Apply the DAG to actually compute the sum.

### 7.2.3.2 findP()

```

DAGNode* helib::AddDAG::findP (
    long i,
    long j ) const [inline]

```

Returns a pointer to the a 'p' node of index (i,j)

### 7.2.3.3 findQ()

```

DAGNode* helib::AddDAG::findQ (
    long i,
    long j ) const [inline]

```

Returns a pointer to the a 'q' node of index (i,j)

### 7.2.3.4 init()

```

void helib::AddDAG::init (
    const CtPtrs & a,
    const CtPtrs & b )

```

Build a plan to add a and b.

### 7.2.3.5 lowLvl()

```

long helib::AddDAG::lowLvl ( ) const [inline]

```

Returns the lowest level in this DAG.

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[binaryArith.cpp](#)

## 7.3 helib::PGFFT::aligned\_allocator< T > Class Template Reference

```

#include <PGFFT.h>

```

### Public Types

- using [value\\_type](#) = T

## Public Member Functions

- [aligned\\_allocator](#) () noexcept
- template<class U >  
  [aligned\\_allocator](#) ([aligned\\_allocator](#)< U > const &) noexcept
- [value\\_type](#) \* [allocate](#) (std::size\_t n)
- void [deallocate](#) ([value\\_type](#) \*p, std::size\_t) noexcept
- template<class U >  
  bool [operator==](#) ([aligned\\_allocator](#)< U > const &) noexcept
- template<class U >  
  bool [operator!=](#) ([aligned\\_allocator](#)< U > const &y) noexcept

## 7.3.1 Member Typedef Documentation

### 7.3.1.1 value\_type

```
template<class T >
using helib::PGFFT::aligned\_allocator< T >::value_type = T
```

## 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 aligned\_allocator() [1/2]

```
template<class T >
helib::PGFFT::aligned\_allocator< T >::aligned_allocator ( ) [inline], [noexcept]
```

### 7.3.2.2 aligned\_allocator() [2/2]

```
template<class T >
template<class U >
helib::PGFFT::aligned\_allocator< T >::aligned_allocator (
    aligned\_allocator< U > const & ) [inline], [noexcept]
```

## 7.3.3 Member Function Documentation

### 7.3.3.1 allocate()

```
template<class T >
value_type* helib::PGFFT::aligned_allocator< T >::allocate (
    std::size_t n ) [inline]
```

### 7.3.3.2 deallocate()

```
template<class T >
void helib::PGFFT::aligned_allocator< T >::deallocate (
    value_type * p,
    std::size_t ) [inline], [noexcept]
```

### 7.3.3.3 operator"!="()

```
template<class T >
template<class U >
bool helib::PGFFT::aligned_allocator< T >::operator!= (
    aligned_allocator< U > const & y ) [inline], [noexcept]
```

### 7.3.3.4 operator==( )

```
template<class T >
template<class U >
bool helib::PGFFT::aligned_allocator< T >::operator== (
    aligned_allocator< U > const & ) [inline], [noexcept]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PGFFT.h](#)

## 7.4 helib::applyPerm\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const NTL::Vec< long > &pi)

### 7.4.1 Member Function Documentation

### 7.4.1.1 apply()

```
template<typename type >
static void helib::applyPerm_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const NTL::Vec< long > & pi ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EncryptedArray.cpp

## 7.5 helib::ArgMap Class Reference

Basic class for arg parsing. Example use:

```
#include <ArgMap.h>
```

### Classes

- struct [ArgProcessor](#)

### Public Types

- enum [Separator](#) { [Separator::COLON](#), [Separator::EQUALS](#), [Separator::WHITESPACE](#) }

### Public Member Functions

- template<typename T >  
[ArgMap](#) & [arg](#) (const std::string &name, T &value)  
*Add a new argument description Adds a new argument description with value of type T. Throws [helib::RuntimeError](#) if the arg key is duplicated or if the storing variable is used more than once.*
- template<typename T >  
[ArgMap](#) & [arg](#) (const std::string &name, T &value, const std::string &doc)  
*Add a new argument with docs Adds a new argument description with value of type T and docs. Throws [helib::RuntimeError](#) if the arg key is duplicated or if the storing variable is used more than once.*
- template<typename T >  
[ArgMap](#) & [arg](#) (const std::string &name, T &value, const std::string &doc, const char \*info)  
*Add a new argument with docs and default description Adds a new argument description with value of type T, with docs and default description. Throws [helib::RuntimeError](#) if the arg key is duplicated or if the storing variable is used more than once.*
- template<typename C >  
[ArgMap](#) & [dots](#) (C &container, const char \*name)  
*Adds variable number of positional arg types after defined arg types are exhausted. These are treated as optional.*
- [ArgMap](#) & [parse](#) (int argc, char \*\*argv)  
*Parse the argv array Parse the argv array If it fails or -h is an argument it prints the usage and exits the program.*
- [ArgMap](#) & [parse](#) (const std::string &filepath)  
*Parse the configuration/parameters file Parsing a configuration file only functions with named arguments Parse the config file Throws [RuntimeError](#) on failure.*



- [ArgMap & optional \(\)](#)  
*Swaps to optional arg mode (default) Swaps to optional arg mode. Following arguments will be considered optional.*
- [ArgMap & required \(\)](#)  
*Swaps to required arg mode Swaps to required arg mode. Following arguments will be considered required.*
- [ArgMap & toggle \(bool t=true\)](#)  
*Swaps to toggle arg type Swaps to required arg mode. Following arguments will be considered of toggle type.*
- [ArgMap & named \(\)](#)  
*Swaps to named arg type (default) Swaps to required arg mode. Following arguments will be considered of named type.*
- [ArgMap & positional \(\)](#)  
*Swaps to positional arg type Swaps to required arg mode. Following arguments will be considered of positional type.*
- [ArgMap & helpArgs \(const std::initializer\\_list< std::string > s\)](#)  
*Provide custom help toggle args. (defaults are "-h", "--help") Overwrite default help toggle args to custom ones for parsing.*
- [ArgMap & helpArgs \(const std::string s\)](#)
- [ArgMap & diagnostics \(std::ostream &ostrm=std::cout\)](#)  
*Turns on diagnostics printout when parsing Swaps to required arg mode. Following arguments will be considered of positional type.*
- [ArgMap & separator \(Separator s\)](#)  
*Sets the key-value separator Sets the named args key-value pair separator character.*
- [ArgMap & note \(const std::string &s\)](#)  
*Adds a note to usage Adds a note to the arg usage description.*
- [void usage \(const std::string &msg=""\) const](#)  
*Print usage and exit Prints the usage and exits the program.*
- [std::string doc \(\) const](#)  
*Return arg docs Returns the argument documentation as a string.*

## 7.5.1 Detailed Description

Basic class for arg parsing. Example use:

```
// Variables to be set by command line.
long p = 2;
long m = 19;
bool t = false;
bool f = true;
std::string k = "Hello World";

ArgMap()
    .required()
    .positional()
    .arg("p", p, "doc for p")
    .arg("m", m, "doc for m", "undefined")
    .optional()
    .named()
    .separator(ArgMap::Separator::WHITESPACE)
    .arg("-k", k, "doc for k", "")
    .note("an extra note")
    .toggle()
    .arg("-t", t, "doc for t", "")
    .toggle(false)
    .arg("-f", f, "doc for f", "")
    .helpArgs({"--myhelp"})
    .parse(argc, argv);

// default values.
// (*) marks default.
// set args to required.
//
// special default info.
// swap to optional args (*).
// named args (*) e.g.k=v.
// change separator to
// whitespace ('=' is (*)).
// no default value info.
// add extra doc/note.
// toggle flag sets bool true.
// toggle flag sets bool false.
//
// changes default help flags
// (*) is {"-h", "--help"}.
// parses and overwrites values
```

## 7.5.2 Member Enumeration Documentation

### 7.5.2.1 Separator

```
enum helib::ArgMap::Separator [strong]
```

## Enumerator

COLON	
EQUALS	
WHITESPACE	

## 7.5.3 Member Function Documentation

### 7.5.3.1 `arg()` [1/3]

```
template<typename T >  
ArgMap & helib::ArgMap::arg (  
    const std::string & name,  
    T & value )
```

Add a new argument description Adds a new argument description with value of type T. Throws [helib::RuntimeError](#) if the arg key is duplicated or if the storing variable is used more than once.

## Template Parameters

<i>T</i>	The type of the argument
----------	--------------------------

## Parameters

<i>name</i>	The argument name (key)
<i>value</i>	a variable where the argument will be stored. Also used as default value

## Returns

A reference to the modified [ArgMap](#) object

### 7.5.3.2 `arg()` [2/3]

```
template<typename T >
ArgMap & helib::ArgMap::arg (
    const std::string & name,
    T & value,
    const std::string & doc )
```

Add a new argument with docs Adds a new argument description with value of type T and docs. Throws [helib::RuntimeError](#) if the arg key is duplicated or if the storing variable is used more than once.

#### Template Parameters

<i>T</i>	The type of the argument
----------	--------------------------

#### Parameters

<i>name</i>	The argument name (key)
<i>value</i>	a variable where the argument will be stored. Also used as default value
<i>doc1</i>	Description of the argument used when displaying usage

#### Returns

A reference to the modified [ArgMap](#) object

### 7.5.3.3 `arg()` [3/3]

```
template<typename T >
ArgMap & helib::ArgMap::arg (
```

```

    const std::string & name,
    T & value,
    const std::string & doc,
    const char * info )

```

Add a new argument with docs and default description Adds a new argument description with value of type T, with docs and default description. Throws [helib::RuntimeError](#) if the arg key is duplicated or if the storing variable is used more than once.

#### Template Parameters

<i>T</i>	The type of the argument
----------	--------------------------

#### Parameters

<i>name</i>	The argument name (key)
<i>value</i>	a variable where the argument will be stored. Also used as default value
<i>doc1</i>	Description of the argument used when displaying usage
<i>info</i>	The default value description (ignored if nullptr or "")

**Returns**

A reference to the modified [ArgMap](#) object

**7.5.3.4 diagnostics()**

```
ArgMap & helib::ArgMap::diagnostics (
    std::ostream & ostrm = std::cout ) [inline]
```

Turns on diagnostics printout when parsing Swaps to required arg mode. Following arguments will be considered of positional type.

**Returns**

A reference to the [ArgMap](#) object

**7.5.3.5 doc()**

```
std::string helib::ArgMap::doc ( ) const [inline]
```

Return arg docs Returns the argument documentation as a string.

**Returns**

the argument documentation string

**7.5.3.6 dots()**

```
template<typename C >
ArgMap & helib::ArgMap::dots (
    C & container,
    const char * name )
```

Adds variable number of positional arg types after defined arg types are exhausted. These are treated as optional.

**Parameters**

<i>container</i>	holds the variable positional args. It must have a <code>push↔_back</code> method for insertion
------------------	---

**Returns**

A reference to the [ArgMap](#) object

**7.5.3.7 helpArgs() [1/2]**

```
ArgMap & helib::ArgMap::helpArgs (
    const std::initializer_list< std::string > s ) [inline]
```

Provide custom help toggle args. (defaults are "-h", "--help") Overwrite default help toggle args to custom ones for parsing.

**Returns**

A reference to the [ArgMap](#) object

**7.5.3.8 helpArgs() [2/2]**

```
ArgMap & helib::ArgMap::helpArgs (
    const std::string s ) [inline]
```

**7.5.3.9 named()**

```
ArgMap & helib::ArgMap::named ( ) [inline]
```

Swaps to named arg type (default) Swaps to required arg mode. Following arguments will be considered of named type.

**Returns**

A reference to the [ArgMap](#) object

### 7.5.3.10 note()

```
ArgMap & helib::ArgMap::note (
    const std::string & s ) [inline]
```

Adds a note to usage Adds a note to the arg usage description.

#### Parameters

s	The note string
---	-----------------------

#### Returns

A reference to the [ArgMap](#) object

### 7.5.3.11 optional()

```
ArgMap & helib::ArgMap::optional ( ) [inline]
```

Swaps to optional arg mode (default) Swaps to optional arg mode. Following arguments will be considered optional.

#### Returns

A reference to the [ArgMap](#) object

### 7.5.3.12 parse() [1/2]

```
ArgMap & helib::ArgMap::parse (
    const std::string & filepath ) [inline]
```

Parse the configuration/parameters file Parsing a configuration file only functions with named arguments Parse the config file Throws [RuntimeError](#) on failure.

#### Parameters

<i>filepath</i>	the config file path
-----------------	-------------------------------

#### Returns

A reference to the [ArgMap](#) object

### 7.5.3.13 parse() [2/2]

```
ArgMap & helib::ArgMap::parse (
    int argc,
    char ** argv ) [inline]
```

Parse the argv array Parse the argv array If it fails or -h is an argument it prints the usage and exits the program.

#### Parameters

<i>argc</i>	number of entries in argv
<i>argv</i>	array containing the arguments

#### Returns

A reference to the [ArgMap](#) object

### 7.5.3.14 positional()

```
ArgMap & helib::ArgMap::positional ( ) [inline]
```

Swaps to positional arg type Swaps to required arg mode. Following arguments will be considered of positional type.

#### Returns

A reference to the [ArgMap](#) object

### 7.5.3.15 required()

```
ArgMap & helib::ArgMap::required ( ) [inline]
```

Swaps to required arg mode Swaps to required arg mode. Following arguments will be considered required.

#### Returns

A reference to the [ArgMap](#) object

### 7.5.3.16 separator()

```
ArgMap & helib::ArgMap::separator (
    Separator s ) [inline]
```

Sets the key-value separator Sets the named args key-value pair separator character.



**Parameters**

<b>s</b>	the separator enum must be set either to C↵ OLON or EQ↵ UAL↵ S(default).
----------	--

**Returns**

A reference to the [ArgMap](#) object

**7.5.3.17 toggle()**

```
ArgMap & helib::ArgMap::toggle (
    bool t = true ) [inline]
```

Swaps to toggle arg type Swaps to required arg mode. Following arguments will be considered of toggle type.

**Returns**

A reference to the [ArgMap](#) object

**7.5.3.18 usage()**

```
void helib::ArgMap::usage (
    const std::string & msg = "" ) const [inline]
```

Print usage and exit Prints the usage and exits the program.

**Parameters**

<i>msg</i>	An additional message to print before showing usage
------------	---

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[ArgMap.h](#)

## 7.6 helib::ArgMap::ArgProcessor Struct Reference

```
#include <ArgMap.h>
```

### Public Member Functions

- virtual [~ArgProcessor](#) ()=default
- virtual ArgType [getArgType](#) ()=0
- virtual bool [process](#) (const std::string &s)=0

### 7.6.1 Constructor & Destructor Documentation

#### 7.6.1.1 ~ArgProcessor()

```
virtual helib::ArgMap::ArgProcessor::~~ArgProcessor ( ) [virtual], [default]
```

### 7.6.2 Member Function Documentation

#### 7.6.2.1 getArgType()

```
virtual ArgType helib::ArgMap::ArgProcessor::getArgType ( ) [pure virtual]
```

### 7.6.2.2 process()

```
virtual bool helib::ArgMap::ArgProcessor::process (
    const std::string & s ) [pure virtual]
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[ArgMap.h](#)

## 7.7 helib::BasicAutomorphPrecon Class Reference

Pre-computation to speed many automorphism on the same ciphertext.

### Public Member Functions

- [BasicAutomorphPrecon](#) (const [Ctxt](#) &\_ctxt)
- std::shared\_ptr< [Ctxt](#) > [automorph](#) (long k) const

### 7.7.1 Detailed Description

Pre-computation to speed many automorphism on the same ciphertext.

The expensive part of homomorphic automorphism is breaking the ciphertext parts into digits. The usual setting is we first rotate the ciphertext parts, then break them into digits. But when we apply many automorphisms it is faster to break the original ciphertext into digits, then rotate the digits (as opposed to first rotate, then break). An [BasicAutomorphPrecon](#) object breaks the original ciphertext and keeps the digits, then when you call automorph is only needs to apply the native automorphism and key switching to the digits, which is fast(er).

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 BasicAutomorphPrecon()

```
helib::BasicAutomorphPrecon::BasicAutomorphPrecon (
    const Ctxt & _ctxt ) [inline]
```

### 7.7.3 Member Function Documentation

### 7.7.3.1 automorph()

```
std::shared_ptr<Ctxt> helib::BasicAutomorphPrecon::automorph (
    long k ) const [inline]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.8 helib::BGV Struct Reference

Type for [BGV](#) scheme, to be used as template parameter.

```
#include <Ptxt.h>
```

### Public Types

- using [SlotType](#) = [PolyMod](#)  
*Slot type used for [BGV](#) plaintexts: [helib::PolyMod](#) i.e. an integer polynomial modulo  $p^r$  and  $G$ .*

### 7.8.1 Detailed Description

Type for [BGV](#) scheme, to be used as template parameter.

### 7.8.2 Member Typedef Documentation

#### 7.8.2.1 SlotType

```
using helib::BGV::SlotType = PolyMod
```

Slot type used for [BGV](#) plaintexts: [helib::PolyMod](#) i.e. an integer polynomial modulo  $p^r$  and  $G$ .

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[Ptxt.h](#)

## 7.9 helib::BipartiteGraph Class Reference

A bipartite flow graph.

```
#include <matching.h>
```

## Public Member Functions

- void [addEdge](#) (long from, long to, long label, long color=0)
- void [partitionToMatchings](#) ()
- void [printout](#) ()

## Public Attributes

- std::vector< [LabeledVertex](#) > [left](#)

### 7.9.1 Detailed Description

A bipartite flow graph.

### 7.9.2 Member Function Documentation

#### 7.9.2.1 addEdge()

```
void helib::BipartiteGraph::addEdge (
    long from,
    long to,
    long label,
    long color = 0 ) [inline]
```

#### 7.9.2.2 partitionToMatchings()

```
void helib::BipartiteGraph::partitionToMatchings ( )
```

#### 7.9.2.3 printout()

```
void helib::BipartiteGraph::printout ( )
```

### 7.9.3 Member Data Documentation

### 7.9.3.1 left

```
std::vector<LabeledVertex> helib::BipartitleGraph::left
```

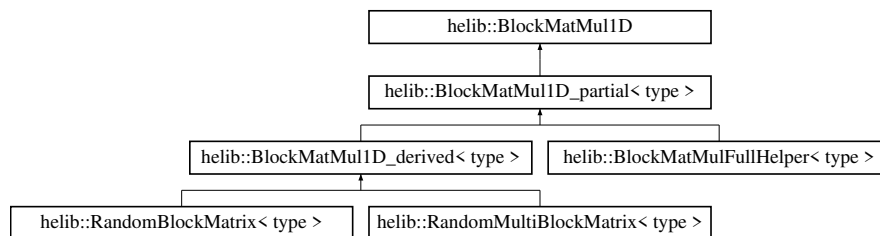
The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matching.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matching.cpp](#)

## 7.10 helib::BlockMatMul1D Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::BlockMatMul1D:



### Public Types

- typedef [BlockMatMul1DExec](#) ExecType

### Public Member Functions

- virtual [~BlockMatMul1D](#) ()
- virtual const [EncryptedArray](#) & [getEA](#) () const =0
- virtual long [getDim](#) () const =0

### 7.10.1 Member Typedef Documentation

#### 7.10.1.1 ExecType

```
typedef BlockMatMul1DExec helib::BlockMatMul1D::ExecType
```

### 7.10.2 Constructor & Destructor Documentation

## 7.10.2.1 ~BlockMatMul1D()

```
virtual helib::BlockMatMul1D::~~BlockMatMul1D ( ) [inline], [virtual]
```

## 7.10.3 Member Function Documentation

## 7.10.3.1 getDim()

```
virtual long helib::BlockMatMul1D::getDim ( ) const [pure virtual]
```

Implemented in [helib::BlockMatMulFullHelper< type >](#), [helib::RandomMultiBlockMatrix< type >](#), and [helib::RandomBlockMatrix< type >](#)

## 7.10.3.2 getEA()

```
virtual const EncryptedArray& helib::BlockMatMul1D::getEA ( ) const [pure virtual]
```

Implemented in [helib::BlockMatMulFullHelper< type >](#), [helib::RandomMultiBlockMatrix< type >](#), and [helib::RandomBlockMatrix< type >](#)

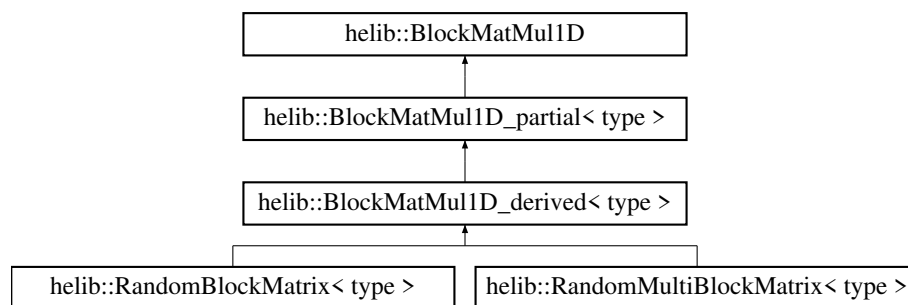
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/matmul.h

## 7.11 helib::BlockMatMul1D\_derived&lt; type &gt; Class Template Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::BlockMatMul1D\_derived< type >:



## Public Member Functions

- virtual bool [multipleTransforms](#) ( ) const =0
- virtual bool [get](#) (mat\_R &out, long i, long j, long k) const =0
- bool [processDiagonal](#) (std::vector< RX > &poly, long i, const [EncryptedArrayDerived](#)< type > &ea) const override

## Additional Inherited Members

### 7.11.1 Member Function Documentation

#### 7.11.1.1 get()

```
template<typename type >
virtual bool helib::BlockMatMul1D_derived< type >::get (
    mat_R & out,
    long i,
    long j,
    long k ) const [pure virtual]
```

Implemented in [helib::RandomMultiBlockMatrix< type >](#).

#### 7.11.1.2 multipleTransforms()

```
template<typename type >
virtual bool helib::BlockMatMul1D_derived< type >::multipleTransforms ( ) const [pure virtual]
```

Implemented in [helib::RandomMultiBlockMatrix< type >](#), and [helib::RandomBlockMatrix< type >](#).

#### 7.11.1.3 processDiagonal()

```
template<typename type >
template bool helib::BlockMatMul1D_derived< type >::processDiagonal (
    std::vector< RX > & poly,
    long i,
    const EncryptedArrayDerived< type > & ea ) const [override], [virtual]
```

Implements [helib::BlockMatMul1D\\_partial< type >](#).

The documentation for this class was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/matmul.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.12 helib::BlockMatMul1D\_derived\_impl< type > Struct Template Reference

### Static Public Member Functions

- static bool [processDiagonal1](#) (std::vector< RX > &poly, long i, const [EncryptedArrayDerived](#)< type > &ea, const [BlockMatMul1D\\_derived](#)< type > &mat)
- static bool [processDiagonal2](#) (std::vector< RX > &poly, long idx, const [EncryptedArrayDerived](#)< type > &ea, const [BlockMatMul1D\\_derived](#)< type > &mat)
- static bool [processDiagonal](#) (std::vector< RX > &poly, long i, const [EncryptedArrayDerived](#)< type > &ea, const [BlockMatMul1D\\_derived](#)< type > &mat)



## 7.12.1 Member Function Documentation

### 7.12.1.1 processDiagonal()

```
template<typename type >
static bool helib::BlockMatMul1D_derived_impl< type >::processDiagonal (
    std::vector< RX > & poly,
    long i,
    const EncryptedArrayDerived< type > & ea,
    const BlockMatMul1D_derived< type > & mat ) [inline], [static]
```

### 7.12.1.2 processDiagonal1()

```
template<typename type >
static bool helib::BlockMatMul1D_derived_impl< type >::processDiagonal1 (
    std::vector< RX > & poly,
    long i,
    const EncryptedArrayDerived< type > & ea,
    const BlockMatMul1D_derived< type > & mat ) [inline], [static]
```

### 7.12.1.3 processDiagonal2()

```
template<typename type >
static bool helib::BlockMatMul1D_derived_impl< type >::processDiagonal2 (
    std::vector< RX > & poly,
    long idx,
    const EncryptedArrayDerived< type > & ea,
    const BlockMatMul1D_derived< type > & mat ) [inline], [static]
```

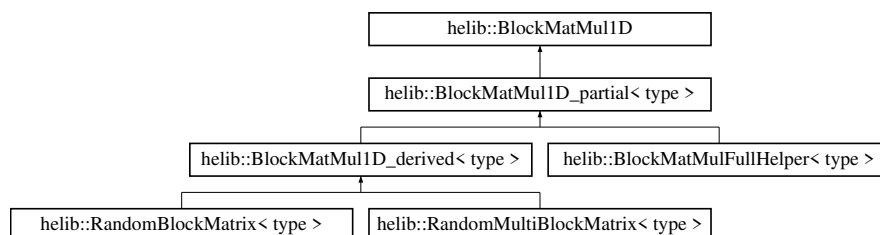
The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.13 helib::BlockMatMul1D\_partial< type > Class Template Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::BlockMatMul1D\_partial< type >:



## Public Member Functions

- virtual bool [processDiagonal](#) (std::vector< RX > &poly, long i, const [EncryptedArrayDerived](#)< type > &ea) const =0

## Additional Inherited Members

### 7.13.1 Member Function Documentation

#### 7.13.1.1 processDiagonal()

```
template<typename type >
virtual bool helib::BlockMatMul1D\_partial< type >::processDiagonal (
    std::vector< RX > & poly,
    long i,
    const EncryptedArrayDerived< type > & ea ) const [pure virtual]
```

Implemented in [helib::BlockMatMulFullHelper](#)< type >, and [helib::BlockMatMul1D\\_derived](#)< type >.

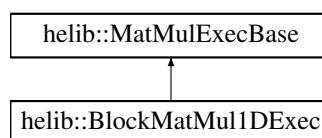
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)

## 7.14 helib::BlockMatMul1DExec Class Reference

```
#include <matmul.h>
```

Inheritance diagram for [helib::BlockMatMul1DExec](#):



## Public Member Functions

- [BlockMatMul1DExec](#) (const [BlockMatMul1D](#) &mat, bool minimal=false)
- void [mul](#) (Ctxt &ctxt) const override
- void [upgrade](#) () override
- const [EncryptedArray](#) & [getEA](#) () const override

## Public Attributes

- const [EncryptedArray](#) & [ea](#)
- long [dim](#)
- long [D](#)
- long [d](#)
- bool [native](#)
- long [strategy](#)
- [ConstMultiplierCache](#) [cache](#)
- [ConstMultiplierCache](#) [cache1](#)

## 7.14.1 Constructor & Destructor Documentation

### 7.14.1.1 BlockMatMul1DExec()

```
helib::BlockMatMul1DExec::BlockMatMul1DExec (  
    const BlockMatMul1D & mat,  
    bool minimal = false ) [explicit]
```

## 7.14.2 Member Function Documentation

### 7.14.2.1 getEA()

```
const EncryptedArray& helib::BlockMatMul1DExec::getEA ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.14.2.2 mul()

```
void helib::BlockMatMul1DExec::mul (  
    Ctxt & ctxt ) const [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.14.2.3 upgrade()

```
void helib::BlockMatMul1DExec::upgrade ( ) [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.14.3 Member Data Documentation

#### 7.14.3.1 cache

`ConstMultiplierCache` helib::BlockMatMul1DExec::cache

#### 7.14.3.2 cache1

`ConstMultiplierCache` helib::BlockMatMul1DExec::cache1

#### 7.14.3.3 D

`long` helib::BlockMatMul1DExec::D

#### 7.14.3.4 d

`long` helib::BlockMatMul1DExec::d

#### 7.14.3.5 dim

`long` helib::BlockMatMul1DExec::dim

#### 7.14.3.6 ea

`const` `EncryptedArray&` helib::BlockMatMul1DExec::ea

#### 7.14.3.7 native

`bool` helib::BlockMatMul1DExec::native

#### 7.14.3.8 strategy

```
long helib::BlockMatMul1DExec::strategy
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/matmul.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.15 helib::BlockMatMul1DExec\_construct< type > Struct Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, const [BlockMatMul1D](#) &mat\_basetype, std::vector< std::shared\_ptr< [ConstMultiplier](#) >> &vec, std::vector< std::shared\_ptr< [ConstMultiplier](#) >> &vec1, long strategy)

### 7.15.1 Member Function Documentation

#### 7.15.1.1 apply()

```
template<typename type >
static void helib::BlockMatMul1DExec_construct< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    const BlockMatMul1D & mat_basetype,
    std::vector< std::shared_ptr< ConstMultiplier >> & vec,
    std::vector< std::shared_ptr< ConstMultiplier >> & vec1,
    long strategy ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.16 helib::BlockMatMulFullExec\_construct< type >::BlockMatMulDimComp Struct Reference

### Public Member Functions

- [BlockMatMulDimComp](#) (const [EncryptedArrayDerived](#)< type > \*\_ea)
- bool [operator\(\)](#) (long i, long j)

## Public Attributes

- const [EncryptedArrayDerived](#)< type > \* [ea](#)

## 7.16.1 Constructor & Destructor Documentation

### 7.16.1.1 BlockMatMulDimComp()

```
template<typename type >
helib::BlockMatMulFullExec\_construct< type >::BlockMatMulDimComp::BlockMatMulDimComp (
    const EncryptedArrayDerived< type > * _ea ) [inline]
```

## 7.16.2 Member Function Documentation

### 7.16.2.1 operator()()

```
template<typename type >
bool helib::BlockMatMulFullExec\_construct< type >::BlockMatMulDimComp::operator() (
    long i,
    long j ) [inline]
```

## 7.16.3 Member Data Documentation

### 7.16.3.1 ea

```
template<typename type >
const EncryptedArrayDerived<type>* helib::BlockMatMulFullExec\_construct< type >::BlockMat↔
MulDimComp::ea
```

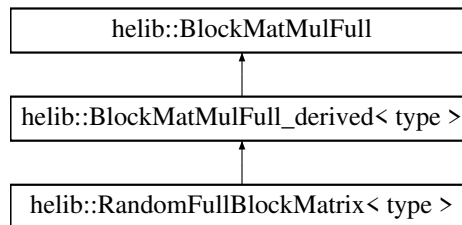
The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.17 helib::BlockMatMulFull Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::BlockMatMulFull:



### Public Types

- typedef [BlockMatMulFullExec](#) ExecType

### Public Member Functions

- virtual [~BlockMatMulFull](#) ()
- virtual const [EncryptedArray](#) & [getEA](#) () const =0

### 7.17.1 Member Typedef Documentation

#### 7.17.1.1 ExecType

```
typedef BlockMatMulFullExec helib::BlockMatMulFull::ExecType
```

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 ~BlockMatMulFull()

```
virtual helib::BlockMatMulFull::~~BlockMatMulFull ( ) [inline], [virtual]
```

### 7.17.3 Member Function Documentation

### 7.17.3.1 getEA()

```
virtual const EncryptedArray& helib::BlockMatMulFull::getEA ( ) const [pure virtual]
```

Implemented in [helib::RandomFullBlockMatrix< type >](#).

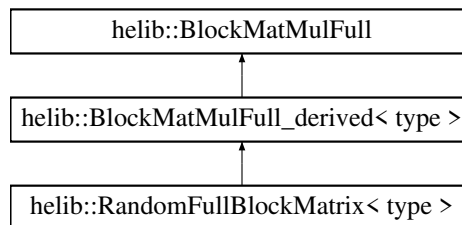
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)

## 7.18 helib::BlockMatMulFull\_derived< type > Class Template Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::BlockMatMulFull\_derived< type >:



### Public Member Functions

- virtual bool [get](#) (mat\_R &out, long i, long j) const =0

### Additional Inherited Members

### 7.18.1 Member Function Documentation

#### 7.18.1.1 get()

```
template<typename type >
virtual bool helib::BlockMatMulFull_derived< type >::get (
    mat_R & out,
    long i,
    long j ) const [pure virtual]
```

Implemented in [helib::RandomFullBlockMatrix< type >](#).

The documentation for this class was generated from the following file:

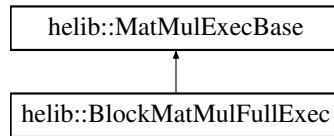
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)



## 7.19 helib::BlockMatMulFullExec Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::BlockMatMulFullExec:



### Public Member Functions

- [BlockMatMulFullExec](#) (const [BlockMatMulFull](#) &mat, bool [minimal](#)=false)
- void [mul](#) (Ctxt &ctxt) const override
- void [upgrade](#) () override
- const [EncryptedArray](#) & [getEA](#) () const override
- long [rec\\_mul](#) (Ctxt &acc, const Ctxt &ctxt, long dim, long idx) const

### Public Attributes

- const [EncryptedArray](#) & [ea](#)
- bool [minimal](#)
- std::vector< long > [dims](#)
- std::vector< [BlockMatMul1DExec](#) > [transforms](#)

## 7.19.1 Constructor & Destructor Documentation

### 7.19.1.1 BlockMatMulFullExec()

```

helib::BlockMatMulFullExec::BlockMatMulFullExec (
    const BlockMatMulFull & mat,
    bool minimal = false ) [explicit]
  
```

## 7.19.2 Member Function Documentation

### 7.19.2.1 getEA()

```
const EncryptedArray& helib::BlockMatMulFullExec::getEA ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.19.2.2 mul()

```
void helib::BlockMatMulFullExec::mul (
    Ctxt & ctxt ) const [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.19.2.3 rec\_mul()

```
long helib::BlockMatMulFullExec::rec_mul (
    Ctxt & acc,
    const Ctxt & ctxt,
    long dim,
    long idx ) const
```

### 7.19.2.4 upgrade()

```
void helib::BlockMatMulFullExec::upgrade ( ) [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

## 7.19.3 Member Data Documentation

### 7.19.3.1 dims

```
std::vector<long> helib::BlockMatMulFullExec::dims
```

### 7.19.3.2 ea

```
const EncryptedArray& helib::BlockMatMulFullExec::ea
```

### 7.19.3.3 minimal

```
bool helib::BlockMatMulFullExec::minimal
```

### 7.19.3.4 transforms

```
std::vector<BlockMatMul1DExec> helib::BlockMatMulFullExec::transforms
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/matmul.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.20 helib::BlockMatMulFullExec\_construct< type > Struct Template Reference

### Classes

- struct [BlockMatMulDimComp](#)

### Static Public Member Functions

- static long [rec\\_mul](#) (long dim, long idx, const std::vector< long > &idxes, std::vector< [BlockMatMul1DExec](#) > &transforms, bool minimal, const std::vector< long > &dims, const [EncryptedArray](#) &ea\_basetype, const [EncryptedArrayDerived](#)< type > &ea, const [BlockMatMulFull\\_derived](#)< type > &mat)
- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, const [EncryptedArray](#) &ea\_basetype, const [BlockMatMulFull](#) &mat\_basetype, std::vector< [BlockMatMul1DExec](#) > &transforms, bool minimal, std::vector< long > &dims)

### 7.20.1 Member Function Documentation

#### 7.20.1.1 apply()

```
template<typename type >
static void helib::BlockMatMulFullExec_construct< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    const EncryptedArray & ea_basetype,
    const BlockMatMulFull & mat_basetype,
    std::vector< BlockMatMul1DExec > & transforms,
    bool minimal,
    std::vector< long > & dims ) [inline], [static]
```

### 7.20.1.2 rec\_mul()

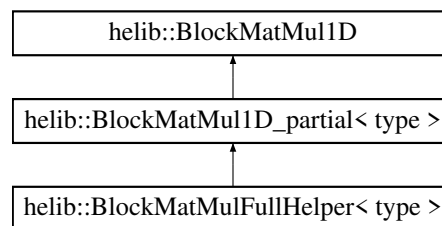
```
template<typename type >
static long helib::BlockMatMulFullExec_construct< type >::rec_mul (
    long dim,
    long idx,
    const std::vector< long > & idxes,
    std::vector< BlockMatMul1DExec > & transforms,
    bool minimal,
    const std::vector< long > & dims,
    const EncryptedArray & ea_basetype,
    const EncryptedArrayDerived< type > & ea,
    const BlockMatMulFull_derived< type > & mat ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.21 helib::BlockMatMulFullHelper< type > Class Template Reference

Inheritance diagram for helib::BlockMatMulFullHelper< type >:



### Public Member Functions

- [BlockMatMulFullHelper](#) (const [EncryptedArray](#) &ea\_basetype, const [BlockMatMulFull\\_derived](#)< type > &↔\_mat, const std::vector< long > &\_init\_idxes, long \_dim)
- bool [processDiagonal](#) (std::vector< RX > &poly, long offset, const [EncryptedArrayDerived](#)< type > &ea) const override
- const [EncryptedArray](#) & [getEA](#) () const override
- long [getDim](#) () const override

### Public Attributes

- const [EncryptedArray](#) &ea\_basetype
- const [BlockMatMulFull\\_derived](#)< type > &mat
- std::vector< long > [init\\_idxes](#)
- long [dim](#)

### Additional Inherited Members

#### 7.21.1 Constructor & Destructor Documentation

### 7.21.1.1 `BlockMatMulFullHelper()`

```
template<typename type >
helib::BlockMatMulFullHelper< type >::BlockMatMulFullHelper (
    const EncryptedArray & _ea_basetype,
    const BlockMatMulFull_derived< type > & _mat,
    const std::vector< long > & _init_idxes,
    long _dim ) [inline]
```

## 7.21.2 Member Function Documentation

### 7.21.2.1 `getDim()`

```
template<typename type >
long helib::BlockMatMulFullHelper< type >::getDim ( ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMul1D](#).

### 7.21.2.2 `getEA()`

```
template<typename type >
const EncryptedArray& helib::BlockMatMulFullHelper< type >::getEA ( ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMul1D](#).

### 7.21.2.3 `processDiagonal()`

```
template<typename type >
bool helib::BlockMatMulFullHelper< type >::processDiagonal (
    std::vector< RX > & poly,
    long offset,
    const EncryptedArrayDerived< type > & ea ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMul1D\\_partial< type >](#).

## 7.21.3 Member Data Documentation

### 7.21.3.1 dim

```
template<typename type >
long helib::BlockMatMulFullHelper< type >::dim
```

### 7.21.3.2 ea\_basetype

```
template<typename type >
const EncryptedArray& helib::BlockMatMulFullHelper< type >::ea_basetype
```

### 7.21.3.3 init\_idxes

```
template<typename type >
std::vector<long> helib::BlockMatMulFullHelper< type >::init_idxes
```

### 7.21.3.4 mat

```
template<typename type >
const BlockMatMulFull_derived<type>& helib::BlockMatMulFullHelper< type >::mat
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.22 helib::CKKS Struct Reference

Type for [CKKS](#) scheme, to be used as template parameter.

```
#include <Ptxt.h>
```

### Public Types

- using [SlotType](#) = std::complex< double >  
*Slot type used for [CKKS](#) plaintexts: `std::complex<double>`.*

### 7.22.1 Detailed Description

Type for [CKKS](#) scheme, to be used as template parameter.

## 7.22.2 Member Typedef Documentation

### 7.22.2.1 SlotType

```
using helib::CKKS::SlotType = std::complex<double>
```

Slot type used for CKKS plaintexts: `std::complex<double>`.

The documentation for this struct was generated from the following file:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Ptxt.h`

## 7.23 helib::Cmodulus Class Reference

Provides FFT and iFFT routines modulo a single-precision prime.

```
#include <CModulus.h>
```

### Public Member Functions

- `Cmodulus ()`  
*Default constructor.*
- `Cmodulus (const Cmodulus &other)`
- `Cmodulus (const PAlgebra &zms, long qq, long rt)`  
*Constructor.*
- `Cmodulus & operator= (const Cmodulus &other)`  
*Copy assignment operator.*
- `const PAlgebra & getZMStar () const`
- `unsigned long getM () const`
- `unsigned long getPhiM () const`
- `long getQ () const`
- `NTL::mulmod_t getQInv () const`
- `long getRoot () const`
- `const zz_pXModulus1 & getPhimX () const`
- `void restoreModulus () const`  
*Restore NTL's current modulus.*
- `void FFT (NTL::vec_long &y, const NTL::ZZX &x) const`
- `void FFT (NTL::vec_long &y, const zzX &x) const`
- `void FFT_aux (NTL::vec_long &y, NTL::zz_pX &tmp) const`
- `void iFFT (NTL::zz_pX &x, const NTL::vec_long &y) const`

### Static Public Member Functions

- `static NTL::zz_pX & getScratch_zz_pX ()`
- `static NTL::Vec< long > & getScratch_vec_long ()`
- `static NTL::fftRep & getScratch_fftRep (long k)`

### 7.23.1 Detailed Description

Provides FFT and iFFT routines modulo a single-precision prime.

On initialization, it initializes NTL's `zz_pContext` for this `q` and computes a  $2m$ -th root of unity  $r \bmod q$  and also  $r^{-1} \bmod q$ . Thereafter this class provides FFT and iFFT routines that converts between time & frequency domains. Some tables are computed the first time that each directions is called, which are then used in subsequent computations.

The "time domain" polynomials are represented as `ZZX`, which are reduced modulo  $\Phi_m(X)$ . The "frequency domain" are just vectors of integers (`vec_long`), that store only the evaluation in primitive  $m$ -th roots of unity.

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 Cmodulus() [1/3]

```
helib::Cmodulus::Cmodulus ( ) [inline]
```

Default constructor.

#### 7.23.2.2 Cmodulus() [2/3]

```
helib::Cmodulus::Cmodulus (
    const Cmodulus & other ) [inline]
```

#### 7.23.2.3 Cmodulus() [3/3]

```
helib::Cmodulus::Cmodulus (
    const PAlgebra & zms,
    long qq,
    long rt )
```

Constructor.

#### Note

Specify  $m$  and  $q$ , and optionally also the root if  $q == 0$ , then the current context is used

### 7.23.3 Member Function Documentation



### 7.23.3.1 FFT() [1/2]

```
void helib::Cmodulus::FFT (
    NTL::vec_long & y,
    const NTL::ZZX & x ) const
```

### 7.23.3.2 FFT() [2/2]

```
void helib::Cmodulus::FFT (
    NTL::vec_long & y,
    const zzX & x ) const
```

### 7.23.3.3 FFT\_aux()

```
void helib::Cmodulus::FFT_aux (
    NTL::vec_long & y,
    NTL::zz_pX & tmp ) const
```

### 7.23.3.4 getM()

```
unsigned long helib::Cmodulus::getM ( ) const [inline]
```

### 7.23.3.5 getPhiM()

```
unsigned long helib::Cmodulus::getPhiM ( ) const [inline]
```

### 7.23.3.6 getPhimX()

```
const zz\_pXModulus1& helib::Cmodulus::getPhimX ( ) const [inline]
```

### 7.23.3.7 getQ()

```
long helib::Cmodulus::getQ ( ) const [inline]
```

### 7.23.3.8 getQInv()

```
NTL::mulmod_t helib::Cmodulus::getQInv ( ) const [inline]
```

### 7.23.3.9 getRoot()

```
long helib::Cmodulus::getRoot ( ) const [inline]
```

### 7.23.3.10 getScratch\_fftRep()

```
NTL::fftRep & helib::Cmodulus::getScratch_fftRep (
    long k ) [static]
```

### 7.23.3.11 getScratch\_vec\_long()

```
NTL::Vec< long > & helib::Cmodulus::getScratch_vec_long ( ) [static]
```

### 7.23.3.12 getScratch\_zz\_pX()

```
NTL::zz_pX & helib::Cmodulus::getScratch_zz_pX ( ) [static]
```

### 7.23.3.13 getZMStar()

```
const PAlgebra& helib::Cmodulus::getZMStar ( ) const [inline]
```

### 7.23.3.14 iFFT()

```
void helib::Cmodulus::iFFT (
    NTL::zz_pX & x,
    const NTL::vec_long & y ) const
```

### 7.23.3.15 operator=()

```
Cmodulus & helib::Cmodulus::operator= (
    const Cmodulus & other )
```

Copy assignment operator.

### 7.23.3.16 restoreModulus()

```
void helib::Cmodulus::restoreModulus ( ) const [inline]
```

Restore [NTL](#)'s current modulus.

The documentation for this class was generated from the following files:

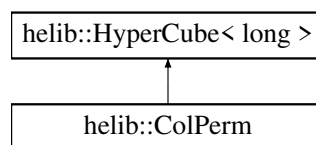
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/CModulus.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/CModulus.cpp](#)

## 7.24 helib::ColPerm Class Reference

Permuting a single dimension (column) of a hypercube.

```
#include <permutations.h>
```

Inheritance diagram for helib::ColPerm:



### Public Member Functions

- [ColPerm](#) (const [CubeSignature](#) &\_sig)
- long [getPermDim](#) () const
- void [setPermDim](#) (long \_dim)
- void [makeExplicit](#) ([Permut](#) &out) const
- long [getShiftAmounts](#) ([Permut](#) &out) const
- void [getBenesShiftAmounts](#) (NTL::Vec< [Permut](#) > &out, NTL::Vec< bool > &idID, const NTL::Vec< long > &benesLvs) const
- void [printout](#) (std::ostream &s)  
*A test/debugging method.*

### 7.24.1 Detailed Description

Permuting a single dimension (column) of a hypercube.

`ColPerm` is derived from a `HyperCube<long>`, and it uses the cube object to store the actual permutation data. The interpretation of this data, however, depends on the data member `dim`.

The cube is partitioned into columns of size  $n = \text{getDim}(\text{dim})$ : a single column consists of the  $n$  entries whose indices  $i$  have the same coordinates in all dimensions other than `dim`. The entries in any such column form a permutation on  $[0..n)$ .

For a given `ColPerm` `perm`, one way to access each column is as follows: for `slice_index = [0..perm.getProd(0, dim))` `CubeSlice` `slice(perm, slice_index, dim)` for `col_index = [0..perm.getProd(dim+1))` `getHyperColumn(column, slice, col_index)`

Another way is to use the `getCoord` and `addCoord` methods.

For example, permuting a  $2 \times 3 \times 2$  cube along `dim=1` (the 2nd dimension), we could have the data `std::vector` as `[ 1 1 0 2 2 0 2 0 1 1 0 2 ]`. This means the four columns are permuted by the permutations `[ 1 0 2 ] [ 1 2 0 ] [ 2 1 0 ] [ 0 1 2 ]`. Written explicitly, we get: `[ 2 3 0 5 4 1 10 7 8 9 6 11 ]`.

Another representation that we provide is by "shift amount": how many slots each element needs to move inside its small permutation. For the example above, this will be: `[ 1 -1 0 ] [ 2 -1 -1 ] [ 2 0 -2 ] [ 0 0 0 ]` so we write the permutation as `[ 1 1 -1 1 0 -2 2 0 0 0 -2 0 ]`.

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 ColPerm()

```
helib::ColPerm::ColPerm (
    const CubeSignature & _sig ) [inline], [explicit]
```

### 7.24.3 Member Function Documentation

#### 7.24.3.1 getBenesShiftAmounts()

```
void helib::ColPerm::getBenesShiftAmounts (
    NTL::Vec< Permut > & out,
    NTL::Vec< bool > & idID,
    const NTL::Vec< long > & benesLvls ) const
```

Get multiple layers of a Benes permutation network. Returns in `out[i][j]` the shift amount to move item  $j$  in the  $i$ 'th layer. Also `isID[i]=true` if the  $i$ 'th layer is the identity (i.e., contains only 0 shift amounts).

### 7.24.3.2 getPermDim()

```
long helib::ColPerm::getPermDim ( ) const [inline]
```

### 7.24.3.3 getShiftAmounts()

```
long helib::ColPerm::getShiftAmounts (
    Permut & out ) const
```

For each position in the data `std::vector`, compute how many slots it should be shifted inside its small permutation. Returns zero if all the shift amounts are zero, nonzero values otherwise.

### 7.24.3.4 makeExplicit()

```
void helib::ColPerm::makeExplicit (
    Permut & out ) const
```

### 7.24.3.5 printout()

```
void helib::ColPerm::printout (
    std::ostream & s ) [inline]
```

A test/debugging method.

### 7.24.3.6 setPermDim()

```
void helib::ColPerm::setPermDim (
    long _dim ) [inline]
```

The documentation for this class was generated from the following files:

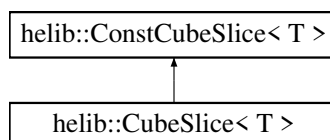
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/permutations.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/permutations.cpp](#)

## 7.25 helib::ConstCubeSlice< T > Class Template Reference

A constant lower-dimension slice of a hypercube.

```
#include <hypercube.h>
```

Inheritance diagram for `helib::ConstCubeSlice< T >`:



## Public Member Functions

- [ConstCubeSlice](#) (const [HyperCube](#)< T > &\_cube)  
*initialize the slice to the full cube*
- [ConstCubeSlice](#) (const NTL::Vec< T > &\_data, const [CubeSignature](#) &\_sig)
- [ConstCubeSlice](#) (const [ConstCubeSlice](#) &bigger, long i, long \_dimOffset=1)
- [ConstCubeSlice](#) (const [HyperCube](#)< T > &\_cube, long i, long \_dimOffset=1)
- const [CubeSignature](#) & [getSig](#) () const  
*const ref to signature*
- long [getSize](#) () const  
*total size*
- long [getNumDims](#) () const  
*number of dimensions*
- long [getDim](#) (long d) const  
*size of dimension d*
- long [getProd](#) (long d) const  
*product of sizes of dimensions d, d+1, ...*
- long [getProd](#) (long from, long to) const  
*product of sizes of dimensions from, from+1, ..., to-1*
- long [getCoord](#) (long i, long d) const  
*get coordinate in dimension d of index i*
- long [addCoord](#) (long i, long d, long offset) const  
*add offset to coordinate in dimension d of index i*
- long [numSlices](#) (long d=1) const  
*number of slices*
- long [sliceSize](#) (long d=1) const  
*size of one slice*
- long [numCols](#) () const  
*number of columns*
- const T & [at](#) (long i) const  
*read-only reference to element at position i, with bounds check*
- const T & [operator\[\]](#) (long i) const  
*read-only reference to element at position i, without bounds check*

### 7.25.1 Detailed Description

```
template<typename T>
class helib::ConstCubeSlice< T >
```

A constant lower-dimension slice of a hypercube.

A [ConstCubeSlice](#) acts like a pointer to a lower dimensional constant subcube of a hypercube. It is initialized using a reference to a hypercube, which must remain alive during the lifetime of the slice, to prevent dangling pointers. The subclass [CubeSlice](#) works also with non-constant cubes and subcubes.

In addition, for greater flexibility, a "slice" may be initialized with a vector and a signature, rather than a cube

### 7.25.2 Constructor & Destructor Documentation

**7.25.2.1 ConstCubeSlice()** [1/4]

```
template<typename T >
helib::ConstCubeSlice< T >::ConstCubeSlice (
    const HyperCube< T > & _cube ) [inline], [explicit]
```

initialize the slice to the full cube

**7.25.2.2 ConstCubeSlice()** [2/4]

```
template<typename T >
helib::ConstCubeSlice< T >::ConstCubeSlice (
    const NTL::Vec< T > & _data,
    const CubeSignature & _sig ) [inline]
```

**7.25.2.3 ConstCubeSlice()** [3/4]

```
template<typename T >
helib::ConstCubeSlice< T >::ConstCubeSlice (
    const ConstCubeSlice< T > & bigger,
    long i,
    long _dimOffset = 1 )
```

initialize the slice to point to the i-th subcube (with some given dimension offset) of the cube pointed to by \_cube or bigger.

**7.25.2.4 ConstCubeSlice()** [4/4]

```
template<typename T >
helib::ConstCubeSlice< T >::ConstCubeSlice (
    const HyperCube< T > & _cube,
    long i,
    long _dimOffset = 1 )
```

**7.25.3 Member Function Documentation****7.25.3.1 addCoord()**

```
template<typename T >
long helib::ConstCubeSlice< T >::addCoord (
    long i,
    long d,
    long offset ) const [inline]
```

add offset to coordinate in dimension d of index i

### 7.25.3.2 at()

```
template<typename T >
const T& helib::ConstCubeSlice< T >::at (
    long i ) const [inline]
```

read-only reference to element at position i, with bounds check

### 7.25.3.3 getCoord()

```
template<typename T >
long helib::ConstCubeSlice< T >::getCoord (
    long i,
    long d ) const [inline]
```

get coordinate in dimension d of index i

### 7.25.3.4 getDim()

```
template<typename T >
long helib::ConstCubeSlice< T >::getDim (
    long d ) const [inline]
```

size of dimension d

### 7.25.3.5 getNumDims()

```
template<typename T >
long helib::ConstCubeSlice< T >::getNumDims ( ) const [inline]
```

number of dimensions

### 7.25.3.6 getProd() [1/2]

```
template<typename T >
long helib::ConstCubeSlice< T >::getProd (
    long d ) const [inline]
```

product of sizes of dimensions d, d+1, ...



### 7.25.3.7 getProd() [2/2]

```
template<typename T >
long helib::ConstCubeSlice< T >::getProd (
    long from,
    long to ) const [inline]
```

product of sizes of dimensions from, from+1, ..., to-1

### 7.25.3.8 getSig()

```
template<typename T >
const CubeSignature& helib::ConstCubeSlice< T >::getSig ( ) const [inline]
```

const ref to signature

### 7.25.3.9 getSize()

```
template<typename T >
long helib::ConstCubeSlice< T >::getSize ( ) const [inline]
```

total size

### 7.25.3.10 numCols()

```
template<typename T >
long helib::ConstCubeSlice< T >::numCols ( ) const [inline]
```

number of columns

### 7.25.3.11 numSlices()

```
template<typename T >
long helib::ConstCubeSlice< T >::numSlices (
    long d = 1 ) const [inline]
```

number of slices

### 7.25.3.12 operator[]()

```
template<typename T >
const T& helib::ConstCubeSlice< T >::operator[] (
    long i ) const [inline]
```

read-only reference to element at position i, without bounds check

### 7.25.3.13 sliceSize()

```
template<typename T >
long helib::ConstCubeSlice< T >::sliceSize (
    long d = 1 ) const [inline]
```

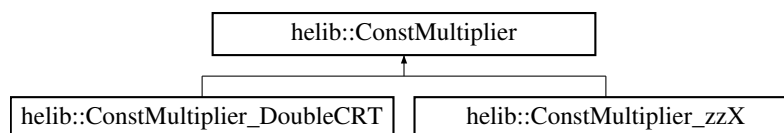
size of one slice

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[hypercube.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[hypercube.cpp](#)

## 7.26 helib::ConstMultiplier Struct Reference

Inheritance diagram for helib::ConstMultiplier:



### Public Member Functions

- virtual [~ConstMultiplier](#) ()
- virtual void [mul](#) (Ctxt &ctxt) const =0
- virtual std::shared\_ptr< [ConstMultiplier](#) > [upgrade](#) (const [Context](#) &context) const =0

### 7.26.1 Constructor & Destructor Documentation

#### 7.26.1.1 ~ConstMultiplier()

```
virtual helib::ConstMultiplier::~~ConstMultiplier ( ) [inline], [virtual]
```

## 7.26.2 Member Function Documentation

### 7.26.2.1 mul()

```
virtual void helib::ConstMultiplier::mul (
    Ctxt & ctxt ) const [pure virtual]
```

Implemented in [helib::ConstMultiplier\\_zzX](#), and [helib::ConstMultiplier\\_DoubleCRT](#).

### 7.26.2.2 upgrade()

```
virtual std::shared_ptr<ConstMultiplier> helib::ConstMultiplier::upgrade (
    const Context & context ) const [pure virtual]
```

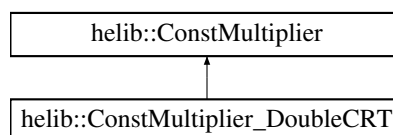
Implemented in [helib::ConstMultiplier\\_zzX](#).

The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.27 helib::ConstMultiplier\_DoubleCRT Struct Reference

Inheritance diagram for `helib::ConstMultiplier_DoubleCRT`:



### Public Member Functions

- [ConstMultiplier\\_DoubleCRT](#) (const [DoubleCRT](#) &\_data, double \_sz)
- void [mul](#) (Ctxt &ctxt) const override
- std::shared\_ptr< [ConstMultiplier](#) > [upgrade](#) ([UNUSED](#) const [Context](#) &context) const override

### Public Attributes

- [DoubleCRT](#) data
- double [sz](#)

## 7.27.1 Constructor & Destructor Documentation

### 7.27.1.1 ConstMultiplier\_DoubleCRT()

```
helib::ConstMultiplier_DoubleCRT::ConstMultiplier_DoubleCRT (
    const DoubleCRT & _data,
    double _sz ) [inline]
```

## 7.27.2 Member Function Documentation

### 7.27.2.1 mul()

```
void helib::ConstMultiplier_DoubleCRT::mul (
    Ctxt & ctxt ) const [inline], [override], [virtual]
```

Implements [helib::ConstMultiplier](#).

### 7.27.2.2 upgrade()

```
std::shared_ptr<ConstMultiplier> helib::ConstMultiplier_DoubleCRT::upgrade (
    UNUSED const Context & context ) const [inline], [override]
```

## 7.27.3 Member Data Documentation

### 7.27.3.1 data

[DoubleCRT](#) helib::ConstMultiplier\_DoubleCRT::data

### 7.27.3.2 sz

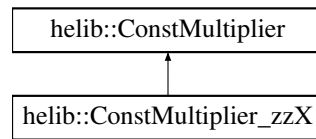
double helib::ConstMultiplier\_DoubleCRT::sz

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.28 helib::ConstMultiplier\_zzX Struct Reference

Inheritance diagram for helib::ConstMultiplier\_zzX:



### Public Member Functions

- [ConstMultiplier\\_zzX](#) (const [zzX](#) &\_data)
- void [mul](#) ([Ctxt](#) &ctxt) const override
- std::shared\_ptr< [ConstMultiplier](#) > [upgrade](#) (const [Context](#) &context) const override

### Public Attributes

- [zzX data](#)

### 7.28.1 Constructor & Destructor Documentation

#### 7.28.1.1 ConstMultiplier\_zzX()

```
helib::ConstMultiplier_zzX::ConstMultiplier_zzX (  
    const zzX &_data ) [inline]
```

### 7.28.2 Member Function Documentation

#### 7.28.2.1 mul()

```
void helib::ConstMultiplier_zzX::mul (  
    Ctxt & ctxt ) const [inline], [override], [virtual]
```

Implements [helib::ConstMultiplier](#).

### 7.28.2.2 upgrade()

```
std::shared_ptr<ConstMultiplier> helib::ConstMultiplier_zzX::upgrade (
    const Context & context ) const [inline], [override], [virtual]
```

Implements [helib::ConstMultiplier](#).

## 7.28.3 Member Data Documentation

### 7.28.3.1 data

```
zzX helib::ConstMultiplier_zzX::data
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.29 helib::ConstMultiplierCache Struct Reference

```
#include <matmul.h>
```

### Public Member Functions

- void [upgrade](#) (const [Context](#) &context)

### Public Attributes

- std::vector< std::shared\_ptr< [ConstMultiplier](#) > > [multiplier](#)

## 7.29.1 Member Function Documentation

### 7.29.1.1 upgrade()

```
void helib::ConstMultiplierCache::upgrade (
    const Context & context )
```

## 7.29.2 Member Data Documentation

### 7.29.2.1 multiplier

```
std::vector<std::shared_ptr<ConstMultiplier> > helib::ConstMultiplierCache::multiplier
```

The documentation for this struct was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/matmul.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.30 helib::Context Class Reference

Maintaining the parameters.

```
#include <Context.h>
```

### Public Member Functions

- double [noiseBoundForUniform](#) (double magBound, long degBound) const
- NTL::xdouble [noiseBoundForUniform](#) (NTL::xdouble magBound, long degBound) const
- double [noiseBoundForMod](#) (long modulus, long degBound) const
- double [noiseBoundForGaussian](#) (double sigma, long degBound) const
- double [noiseBoundForSmall](#) (double prob, long degBound) const
- double [noiseBoundForHwt](#) (long hwt, [UNUSED](#) long degBound) const
- double [stdDevForRecryption](#) (long skHwt=0) const
  - NOTE: this is a bit heuristic. See design document for details.*
- double [boundForRecryption](#) (long skHwt=0) const
- void [setModSizeTable](#) ()
- [Context](#) (unsigned long m, unsigned long p, unsigned long r, const std::vector< long > &gens=std::vector< long >(), const std::vector< long > &ords=std::vector< long >())
- void [makeBootstrappable](#) (const NTL::Vec< long > &mvec, long skWht=0, bool build\_cache=false, bool alsoThick=true)
- bool [isBootstrappable](#) () const
- [IndexSet](#) [fullPrimes](#) () const
- [IndexSet](#) [allPrimes](#) () const
- [IndexSet](#) [getCtxtPrimes](#) (long nprimes) const
- long [BPL](#) () const
- bool [operator==](#) (const [Context](#) &other) const
- bool [operator!=](#) (const [Context](#) &other) const
- long [ithPrime](#) (unsigned long i) const
  - The ith small prime in the modulus chain.*
- const [Cmodulus](#) & [ithModulus](#) (unsigned long i) const
  - Cmodulus object corresponding to ith small prime in the chain.*
- long [numPrimes](#) () const
  - Total number of small prime in the chain.*
- bool [isZeroDivisor](#) (const NTL::ZZ &num) const
  - Is num divisible by any of the primes in the chain?*
- bool [inChain](#) (long p) const
  - Is p already in the chain?*
- double [logOfPrime](#) (unsigned long i) const
  - Returns the natural logarithm of the ith prime.*

- double [logOfProduct](#) (const [IndexSet](#) &s) const  
*Returns the natural logarithm of productOfPrimes(s)*
  - double [securityLevel](#) () const  
*An estimate for the security-level.*
  - void [AddSmallPrime](#) (long q)  
*Just add the given prime to the chain.*
  - void [AddCtxtPrime](#) (long q)
  - void [AddSpecialPrime](#) (long q)
- 
- void [productOfPrimes](#) (NTL::ZZ &p, const [IndexSet](#) &s) const  
*The product of all the primes in the given set.*
  - NTL::ZZ [productOfPrimes](#) (const [IndexSet](#) &s) const

## Public Attributes

- [PAlgebra zMStar](#)  
*The structure of  $Zm^*$ .*
- [PAlgebraMod alMod](#)  
*The structure of  $Z[X]/(\Phi_m(X), p^r)$*
- std::shared\_ptr< const [EncryptedArray](#) > [ea](#)  
*A default [EncryptedArray](#).*
- std::shared\_ptr< const [PowerfulDCRT](#) > [pwfl\\_converter](#)
- std::shared\_ptr< [PolyModRing](#) > [slotRing](#)  
*The structure of a single slot of the plaintext space.*
- NTL::xdouble [stdev](#)  
 *$\sqrt{\text{variance}}$  of the LWE error (default=3.2)*
- double [scale](#)
- [IndexSet](#) [ctxtPrimes](#)
- [IndexSet](#) [specialPrimes](#)
- [IndexSet](#) [smallPrimes](#)
- [ModuliSizes](#) [modSizes](#)  
*A helper table to map required modulo-sizes to primeSets.*
- std::vector< [IndexSet](#) > [digits](#)  
*The set of primes for the digits.*
- [ThinRecryptData](#) [rcData](#)  
*Bootstrapping-related data in the context.*

## Friends

- void [writeContextBinary](#) (std::ostream &str, const [Context](#) &context)
- void [readContextBinary](#) (std::istream &str, [Context](#) &context)



### I/O routines

To write out all the data associated with a context, do the following:

```
writeContextBase(str, context);
str « context;
```

The first function call writes out just  $[m\ p\ r\ gens\ ords]$ , which is the data needed to invoke the context constructor.

The second call writes out all other information, including the stdev field, the prime sequence (including which primes are "special"), and the digits info.

To read in all the data associated with a context, do the following:

```
unsigned long m, p, r;
std::vector<long> gens, ords;
readContextBase(str, m, p, r, gens, ords);
Context context(m, p, r, gens, ords);
str » context;
```

The call to readContextBase just reads the values  $m$ ,  $p$ ,  $r$  and the set of generators in  $Z_{m*}(p)$  and their order. Then, after constructing the context, the  $>>$  operator reads in and attaches all other information.

- void writeContextBase (std::ostream &str, const Context &context)  
write  $[m\ p\ r]$  data
- std::ostream & operator<< (std::ostream &str, const Context &context)  
Write all other data.
- void readContextBase (std::istream &str, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)  
read  $[m\ p\ r]$  data, needed to construct context
- std::istream & operator>> (std::istream &str, Context &context)  
read all other data associated with context

## 7.30.1 Detailed Description

Maintaining the parameters.

## 7.30.2 Constructor & Destructor Documentation

### 7.30.2.1 Context()

```
helib::Context::Context (
    unsigned long m,
    unsigned long p,
    unsigned long r,
    const std::vector< long > & gens = std::vector<long>(),
    const std::vector< long > & ords = std::vector<long>() )
```

## 7.30.3 Member Function Documentation

### 7.30.3.1 AddCtxtPrime()

```
void helib::Context::AddCtxtPrime (
    long q )
```

### 7.30.3.2 AddSmallPrime()

```
void helib::Context::AddSmallPrime (
    long q )
```

Just add the given prime to the chain.

### 7.30.3.3 AddSpecialPrime()

```
void helib::Context::AddSpecialPrime (
    long q )
```

### 7.30.3.4 allPrimes()

```
IndexSet helib::Context::allPrimes ( ) const [inline]
```

### 7.30.3.5 boundForRecryption()

```
double helib::Context::boundForRecryption (
    long skHwt = 0 ) const [inline]
```

### 7.30.3.6 BPL()

```
long helib::Context::BPL ( ) const [inline]
```

### 7.30.3.7 fullPrimes()

```
IndexSet helib::Context::fullPrimes ( ) const [inline]
```

### 7.30.3.8 getCtxtPrimes()

```
IndexSet helib::Context::getCtxtPrimes (
    long nprimes ) const [inline]
```

### 7.30.3.9 inChain()

```
bool helib::Context::inChain (
    long p ) const [inline]
```

Is *p* already in the chain?

### 7.30.3.10 isBootstrappable()

```
bool helib::Context::isBootstrappable ( ) const [inline]
```

### 7.30.3.11 isZeroDivisor()

```
bool helib::Context::isZeroDivisor (
    const NTL::ZZ & num ) const [inline]
```

Is *num* divisible by any of the primes in the chain?

### 7.30.3.12 ithModulus()

```
const Cmodulus& helib::Context::ithModulus (
    unsigned long i ) const [inline]
```

[Cmodulus](#) object corresponding to *ith* small prime in the chain.

### 7.30.3.13 ithPrime()

```
long helib::Context::ithPrime (
    unsigned long i ) const [inline]
```

The *ith* small prime in the modulus chain.

### 7.30.3.14 logOfPrime()

```
double helib::Context::logOfPrime (
    unsigned long i ) const [inline]
```

Returns the natural logarithm of the *ith* prime.

**7.30.3.15 logOfProduct()**

```
double helib::Context::logOfProduct (
    const IndexSet & s ) const [inline]
```

Returns the natural logarithm of productOfPrimes(s)

**7.30.3.16 makeBootstrappable()**

```
void helib::Context::makeBootstrappable (
    const NTL::Vec< long > & mvec,
    long skWht = 0,
    bool build_cache = false,
    bool alsoThick = true ) [inline]
```

**7.30.3.17 noiseBoundForGaussian()**

```
double helib::Context::noiseBoundForGaussian (
    double sigma,
    long degBound ) const [inline]
```

Assume the polynomial  $f(x) = \sum_{i < k} f_i x^i$  is chosen so that each  $f_i$  is chosen uniformly and independently from  $N(0, \sigma^2)$ , and that  $k = \text{degBound}$ . This returns a bound  $B$  such that the  $L_\infty$  norm of the canonical embedding exceeds  $B$  with probability at most  $\epsilon$ .

**7.30.3.18 noiseBoundForHWt()**

```
double helib::Context::noiseBoundForHWt (
    long hwt,
    UNUSED long degBound ) const [inline]
```

Assume the polynomial  $f(x) = \sum_{i < k} f_i x^i$  is chosen so that each  $f_i$  is chosen uniformly and independently from the set of balanced residues modulo the given modulus. This returns a bound  $B$  such that the  $L_\infty$  norm of the canonical embedding exceeds  $B$  with probability at most  $\epsilon$ .

**7.30.3.19 noiseBoundForMod()**

```
double helib::Context::noiseBoundForMod (
    long modulus,
    long degBound ) const [inline]
```

Assume the polynomial  $f(x) = \sum_{i < k} f_i x^i$  is chosen so that each  $f_i$  is chosen uniformly and independently from the set of balanced residues modulo the given modulus. This returns a bound  $B$  such that the  $L_\infty$  norm of the canonical embedding exceeds  $B$  with probability at most  $\epsilon$ .

**7.30.3.20 noiseBoundForSmall()**

```
double helib::Context::noiseBoundForSmall (
    double prob,
    long degBound ) const [inline]
```

Assume the polynomial  $f(x) = \sum_{i < k} f_i x^i$  is chosen so that each  $f_i$  is zero with probability  $1 - \text{prob}$ , 1 with probability  $\text{prob}/2$ , and -1 with probability  $\text{prob}/2$ . This returns a bound  $B$  such that the L-infty norm of the canonical embedding exceeds  $B$  with probability at most  $\epsilon$ .

**7.30.3.21 noiseBoundForUniform() [1/2]**

```
double helib::Context::noiseBoundForUniform (
    double magBound,
    long degBound ) const [inline]
```

$\text{erfc}(\text{scale}/\sqrt{2}) * \phi(m)$  should be less than some negligible parameter  $\epsilon$ . The default value of 10 should be good enough for most applications. NOTE:  $-\log(\text{erfc}(8/\sqrt{2}))/\log(2) = 49.5$   $-\log(\text{erfc}(10/\sqrt{2}))/\log(2) = 75$ .  $-\log(\text{erfc}(11/\sqrt{2}))/\log(2) = 91.1$   $-\log(\text{erfc}(12/\sqrt{2}))/\log(2) = 107.8$  The way this is used is as follows. If we have a normal random variable  $X$  with variance  $\sigma^2$ , then the probability that  $X$  lies outside the interval  $[-\text{scale} * \sigma, \text{scale} * \sigma]$  is  $\delta = \text{erfc}(\text{scale}/\sqrt{2})$ . We will usually apply the union bound to a vector of  $\phi(m)$  such random variables (one for each primitive  $m$ -th root of unity), so that the probability that the L-infty norm exceeds  $\text{scale} * \sigma$  is at most  $\epsilon = \phi(m) * \delta$ . Thus,  $\text{scale} * \sigma$  will be used as a high-probability bound on the L-infty norm of such vectors. Assume the polynomial  $f(x) = \sum_{i < k} f_i x^i$  is chosen so that each  $f_i$  is chosen uniformly and independently from the interval  $[-\text{magBound}, \text{magBound}]$ , and that  $k = \text{degBound}$ . This returns a bound  $B$  such that the L-infty norm of the canonical embedding exceeds  $B$  with probability at most  $\epsilon$ .

**7.30.3.22 noiseBoundForUniform() [2/2]**

```
NTL::xdouble helib::Context::noiseBoundForUniform (
    NTL::xdouble magBound,
    long degBound ) const [inline]
```

**7.30.3.23 numPrimes()**

```
long helib::Context::numPrimes ( ) const [inline]
```

Total number of small prime in the chain.

**7.30.3.24 operator"!="()**

```
bool helib::Context::operator!= (
    const Context & other ) const [inline]
```

**7.30.3.25 operator==()**

```
bool helib::Context::operator== (
    const Context & other ) const
```

**7.30.3.26 productOfPrimes() [1/2]**

```
NTL::ZZ helib::Context::productOfPrimes (
    const IndexSet & s ) const [inline]
```

**7.30.3.27 productOfPrimes() [2/2]**

```
void helib::Context::productOfPrimes (
    NTL::ZZ & p,
    const IndexSet & s ) const
```

The product of all the primes in the given set.

**7.30.3.28 securityLevel()**

```
double helib::Context::securityLevel ( ) const [inline]
```

An estimate for the security-level.

**7.30.3.29 setModSizeTable()**

```
void helib::Context::setModSizeTable ( ) [inline]
```

**7.30.3.30 stdDevForReryption()**

```
double helib::Context::stdDevForReryption (
    long skHwt = 0 ) const [inline]
```

NOTE: this is a bit heuristic. See design document for details.

This computes a high probability bound on the L-infty norm of  $x_0 + s \cdot x_1$  in the pwrfl basis, assuming  $s$  is chosen with coeffs in the pwrfl basis uniformly and independently dist'd over  $[-1/2, 1/2]$ ,  $x_0$  has arbitrary coeffs over  $[-1/2, 1/2]$  in the pwrfl basis, and assuming  $s$  is chosen with  $skHwt$  nonzero coeffs mod  $X^m - 1$  in the power basis (uniformly and independently over  $\{-1, 1\}$ ). The bound should be satisfied with probability epsilon. NOTE: this is still valid even when  $m$  is a power of 2

## 7.30.4 Friends And Related Function Documentation

### 7.30.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & str,
    const Context & context ) [friend]
```

Write all other data.

### 7.30.4.2 operator>>

```
std::istream& operator>> (
    std::istream & str,
    Context & context ) [friend]
```

read all other data associated with context

### 7.30.4.3 readContextBase

```
void readContextBase (
    std::istream & str,
    unsigned long & m,
    unsigned long & p,
    unsigned long & r,
    std::vector< long > & gens,
    std::vector< long > & ords ) [friend]
```

read [m p r] data, needed to construct context

### 7.30.4.4 readContextBinary

```
void readContextBinary (
    std::istream & str,
    Context & context ) [friend]
```

#### 7.30.4.5 writeContextBase

```
void writeContextBase (
    std::ostream & str,
    const Context & context ) [friend]
```

write [m p r] data

#### 7.30.4.6 writeContextBinary

```
void writeContextBinary (
    std::ostream & str,
    const Context & context ) [friend]
```

### 7.30.5 Member Data Documentation

#### 7.30.5.1 alMod

`PAlgebraMod` `helib::Context::alMod`

The structure of  $\mathbb{Z}[X]/(\Phi_m(X), p^r)$

#### 7.30.5.2 ctxtPrimes

`IndexSet` `helib::Context::ctxtPrimes`

The "ciphertext primes" are the "normal" primes that are used to represent the public encryption key and ciphertexts. These are all "large" single-precision primes, or bit-size roughly `NTL_SP_SIZE` bits.

#### 7.30.5.3 digits

`std::vector<IndexSet>` `helib::Context::digits`

The set of primes for the digits.

The different columns in any key-switching matrix contain encryptions of multiplies of the secret key,  $sk$ ,  $B_1*sk$ ,  $B_2*B_1*sk$ ,  $B_3*B_2*B_1*sk$ ,... with each  $B_i$  a product of a few "non-special" primes in the chain. The digits data member indicate which primes correspond to each of the  $B_i$ 's. These are all `IndexSet` objects, whose union is the subset `ctxtPrimes`.

The number of  $B_i$ 's is one less than the number of columns in the key switching matrices (since the 1st column encrypts  $sk$ , without any  $B_i$ 's), but we keep in the digits `std::vector` also an entry for the primes that do not participate in any  $B_i$  (so `digits.size()` is the same as the number of columns in the key switching matrices). See section 3.1.6 in the design document (key-switching).



#### 7.30.5.4 ea

```
std::shared_ptr<const EncryptedArray> helib::Context::ea
```

A default [EncryptedArray](#).

#### 7.30.5.5 modSizes

```
ModuliSizes helib::Context::modSizes
```

A helper table to map required modulo-sizes to primeSets.

#### 7.30.5.6 pwfl\_converter

```
std::shared_ptr<const PowerfulDCRT> helib::Context::pwfl_converter
```

#### 7.30.5.7 rcData

```
ThinReencryptData helib::Context::rcData
```

Bootstrapping-related data in the context.

#### 7.30.5.8 scale

```
double helib::Context::scale
```

#### 7.30.5.9 slotRing

```
std::shared_ptr<PolyModRing> helib::Context::slotRing
```

The structure of a single slot of the plaintext space.

This will be  $\mathbb{Z}[X]/(G(x), p^r)$  for some irreducible factor  $G$  of  $\Phi_m(X)$ .

### 7.30.5.10 `smallPrimes`

`IndexSet` `helib::Context::smallPrimes`

Yet a third set of primes, aimed at allowing modulus-switching with higher resolution. These are somewhat smaller single-precision primes, of size from `NTL_SP_SIZE-20` to `NTL_SP_SIZE-1`.

### 7.30.5.11 `specialPrimes`

`IndexSet` `helib::Context::specialPrimes`

A disjoint set of primes, used for key switching. See section 3.1.6 in the design document (key-switching). These too are "large" single-precision primes, or bit-size close to `NTL_SP_SIZE` bits.

### 7.30.5.12 `stdev`

`NTL::xdouble` `helib::Context::stdev`

`sqrt(variance)` of the LWE error (default=3.2)

### 7.30.5.13 `zMStar`

`PAlgebra` `helib::Context::zMStar`

The structure of  $\mathbb{Z}m^*$ .

The documentation for this class was generated from the following files:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Context.h`
- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Context.cpp`
- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/primeChain.cpp`

## 7.31 `helib::Ctxt` Class Reference

A `Ctxt` object holds a single ciphertext.

```
#include <Ctxt.h>
```

## Public Member Functions

- [Ctxt](#) (const [PubKey](#) &newPubKey, long newPtxtSpace=0)
- [Ctxt](#) (const [Ctxt](#) &other)=default
- [Ctxt](#) (ZeroCtxtLike\_type, const [Ctxt](#) &ctxt)
- void [DummyEncrypt](#) (const NTL::ZZX &ptxt, double size=-1.0)
- [Ctxt](#) & [operator=](#) (const [Ctxt](#) &other)
- bool [operator==](#) (const [Ctxt](#) &other) const
- bool [operator!=](#) (const [Ctxt](#) &other) const
- bool [equalsTo](#) (const [Ctxt](#) &other, bool comparePkeys=true) const
- void [write](#) (std::ostream &str) const
- void [read](#) (std::istream &str)

## Ciphertext arithmetic

- void [negate](#) ()
- [Ctxt](#) & [operator+=](#) (const [Ctxt](#) &other)
- [Ctxt](#) & [operator-=](#) (const [Ctxt](#) &other)
- void [addCtxt](#) (const [Ctxt](#) &other, bool negative=false)
- void [multLowLvl](#) (const [Ctxt](#) &other, bool destructive=false)
- [Ctxt](#) & [operator\\*=](#) (const [Ctxt](#) &other)
- void [automorph](#) (long k)
- [Ctxt](#) & [operator>=>](#) (long k)
- void [complexConj](#) ()
- void [smartAutomorph](#) (long k)  
*automorphism with re-linearization*
- void [frobeniusAutomorph](#) (long j)  
*applies the automorphism  $p^j$  using smartAutomorphism*
- [Ctxt](#) & [operator+=](#) (const [Ptxt](#)< [BGV](#) > &other)  
*Plus equals operator with a [BGV Ptxt](#).*
- [Ctxt](#) & [operator-=](#) (const [Ptxt](#)< [BGV](#) > &other)  
*Minus equals operator with a [BGV Ptxt](#).*
- [Ctxt](#) & [operator\\*=](#) (const [Ptxt](#)< [BGV](#) > &other)  
*Times equals operator with a [BGV Ptxt](#).*
- [Ctxt](#) & [operator+=](#) (const [Ptxt](#)< [CKKS](#) > &other)  
*Plus equals operator with a [CKKS Ptxt](#).*
- [Ctxt](#) & [operator-=](#) (const [Ptxt](#)< [CKKS](#) > &other)  
*Minus equals operator with a [CKKS Ptxt](#).*
- [Ctxt](#) & [operator\\*=](#) (const [Ptxt](#)< [CKKS](#) > &other)  
*Times equals operator with a [CKKS Ptxt](#).*
- [Ctxt](#) & [operator\\*=](#) (const NTL::ZZX &poly)  
*Times equals operator with a [ZZX](#).*
- [Ctxt](#) & [operator\\*=](#) (const long scalar)  
*Times equals operator with a [long](#).*
- void [addConstant](#) (const [DoubleCRT](#) &dcrt, double size=-1.0)
- void [addConstant](#) (const NTL::ZZX &poly, double size=-1.0)
- template<typename Scheme >  
void [addConstant](#) (const [Ptxt](#)< Scheme > &ptxt)  
*Add a [BGV](#) plaintext to this [Ctxt](#).*
- void [addConstant](#) (const NTL::ZZ &c)
- void [addConstantCKKS](#) (std::pair< long, long >)  
*add a rational number in the form a/b, a,b are long*
- void [addConstantCKKS](#) (double x)
- void [addConstantCKKS](#) (const [DoubleCRT](#) &dcrt, NTL::xdouble size=NTL::xdouble(-1.0), NTL::xdouble factor=NTL::xdouble(-1.0))
- void [addConstantCKKS](#) (const NTL::ZZX &poly, NTL::xdouble size=NTL::xdouble(-1.0), NTL::xdouble factor=NTL::xdouble(-1.0))
- void [addConstantCKKS](#) (const std::vector< std::complex< double >> &ptxt)
- void [addConstantCKKS](#) (const [Ptxt](#)< [CKKS](#) > &ptxt)

- *Add a [CKKS](#) plaintext to this [Ctxt](#).*
- void [addConstantCKKS](#) (const NTL::ZZ &c)
- void [multByConstant](#) (const [DoubleCRT](#) &dcrt, double size=-1.0)
- void [multByConstant](#) (const NTL::ZZX &poly, double size=-1.0)
- void [multByConstant](#) (const [zzX](#) &poly, double size=-1.0)
- void [multByConstant](#) (const NTL::ZZ &c)
- template<typename Scheme >  
void [multByConstant](#) (const [Ptxt](#)< Scheme > &ptxt)
- *Multiply a [BGV](#) plaintext to this [Ctxt](#).*
- void [multByConstantCKKS](#) (double x)  
*multiply by a rational number or floating point*
- void [multByConstantCKKS](#) (std::pair< long, long > num)
- void [multByConstantCKKS](#) (const [DoubleCRT](#) &dcrt, NTL::xdouble size=NTL::xdouble(-1.0), NTL::xdouble factor=NTL::xdouble(-1.0), double roundingErr=-1.0)
- void [multByConstantCKKS](#) (const NTL::ZZX &poly, NTL::xdouble size=NTL::xdouble(-1.0), NTL::xdouble factor=NTL::xdouble(-1.0), double roundingErr=-1.0)
- void [multByConstantCKKS](#) (const [Ptxt](#)< [CKKS](#) > &ptxt)
- *Multiply a [CKKS](#) plaintext to this [Ctxt](#).*
- void [multByConstantCKKS](#) (const std::vector< std::complex< double >> &ptxt)
- void [xorConstant](#) (const [DoubleCRT](#) &poly, [UNUSED](#) double size=-1.0)
- void [xorConstant](#) (const NTL::ZZX &poly, double size=-1.0)
- void [nxorConstant](#) (const [DoubleCRT](#) &poly, [UNUSED](#) double size=-1.0)
- void [nxorConstant](#) (const NTL::ZZX &poly, double size=-1.0)
- void [divideByP](#) ()
- void [multByP](#) (long e=1)
- void [divideBy2](#) ()
- void [extractBits](#) (std::vector< [Ctxt](#) > &bits, long nBits2extract=0)
- void [multiplyBy](#) (const [Ctxt](#) &other)
- void [multiplyBy2](#) (const [Ctxt](#) &other1, const [Ctxt](#) &other2)
- void [square](#) ()
- void [cube](#) ()
- void [power](#) (long e)  
*raise ciphertext to some power*

## Ciphertext maintenance

- void [reducePtxtSpace](#) (long newPtxtSpace)  
*Reduce plaintext space to a divisor of the original plaintext space.*
- void [hackPtxtSpace](#) (long newPtxtSpace)
- void [bumpNoiseBound](#) (double factor)
- void [reLinearize](#) (long keyIdx=0)
- void [cleanUp](#) ()
- void [blindCtxt](#) (const NTL::ZZX &poly)  
*Add a high-noise encryption of the given constant.*
- NTL::xdouble [modSwitchAddedNoiseBound](#) () const  
*Estimate the added noise.*
- void [modUpToSet](#) (const [IndexSet](#) &s)  
*Modulus-switching up (to a larger modulus). Must have primeSet <= s, and s must contain either all the special primes or none of them.*
- void [modDownToSet](#) (const [IndexSet](#) &s)  
*Modulus-switching down (to a smaller modulus). mod-switch down to primeSet \intersect s, after this call we have primeSet <= s. s must contain either all special primes or none of them.*
- void [bringToSet](#) (const [IndexSet](#) &s)  
*make the primeSet equal to newPrimeSet, via modUpToSet and modDownToSet*
- double [naturalSize](#) () const
- [IndexSet](#) [naturalPrimeSet](#) () const  
*"natural size" is size before squaring*
- void [dropSmallAndSpecialPrimes](#) ()  
*the corresponding primeSet*
- double [capacity](#) () const  
*returns the "capacity" of a ciphertext, which is the log of the ratio of the modulus to the noise bound*

- long `bitCapacity` () const  
*the capacity in bits, returned as an integer*
- double `logOfPrimeSet` () const  
*returns the log of the prime set*
- double `rawModSwitch` (std::vector< NTL::ZZX > &zzParts, long toModulus) const  
*Special-purpose modulus-switching for bootstrapping.*
- void `evalPoly` (const NTL::ZZX &poly)  
*compute the power  $X, X^2, \dots, X^n$*

### Utility methods

- void `clear` ()
- bool `isEmpty` () const  
*Is this an empty ciphertext without any parts.*
- bool `inCanonicalForm` (long keyID=0) const  
*A canonical ciphertext has (at most) handles pointing to (1,s)*
- bool `isCorrect` () const  
*Would this ciphertext be decrypted without errors?*
- const Context & `getContext` () const
- const PubKey & `getPubKey` () const
- const IndexSet & `getPrimeSet` () const
- long `getPtxtSpace` () const
- const NTL::xdouble & `getNoiseBound` () const
- const NTL::xdouble & `getRatFactor` () const
- const NTL::xdouble & `getPtxtMag` () const
- void `setPtxtMag` (const NTL::xdouble &z)
- long `getKeyID` () const
- bool `isCKKS` () const
- long `effectiveR` () const
- double `log_of_ratio` () const  
*Returns  $\log(\text{noiseBound}) - \log(q)$*

### Static Public Member Functions

- static void `equalizeRationalFactors` (Ctxt &c1, Ctxt &c2)

### Friends

- class PubKey
- class SecKey
- class BasicAutomorphPrecon
- std::istream & `operator>>` (std::istream &str, Ctxt &ctxt)
- std::ostream & `operator<<` (std::ostream &str, const Ctxt &ctxt)

#### 7.31.1 Detailed Description

A `Ctxt` object holds a single ciphertext.

The class `Ctxt` includes a `std::vector<CtxtPart>`: For a `Ctxt` `c`, `c[i]` is the *i*'th ciphertext part, which can be used also as a `DoubleCRT` object (since `CtxtPart` is derived from `DoubleCRT`). By convention, `c[0]`, the first `CtxtPart` object in the `std::vector`, has `skHndl` that points to 1 (i.e., it is just added in upon decryption, without being multiplied by anything). We maintain the invariance that all the parts of a ciphertext are defined relative to the same set of primes.

A ciphertext contains also pointers to the general parameters of this FHE instance and the public key, and a high-probability bound on the noise magnitude (kept in the `noiseBound` data member). The noise bound is a bound on the *l*-infinity norm of the canonical embedding of the noise polynomial, namely its evaluation in roots of the ring polynomial (which are the complex primitive roots of unity). The noise bound is added on addition, multiplied on multiplications, remains unchanged for automorphism, and is roughly scaled down by mod-switching with some added factor, and similarly scaled up by key-switching with some added factor.

## 7.31.2 Constructor & Destructor Documentation

### 7.31.2.1 Ctxt() [1/3]

```
helib::Ctxt::Ctxt (
    const PubKey & newPubKey,
    long newPtxtSpace = 0 ) [explicit]
```

### 7.31.2.2 Ctxt() [2/3]

```
helib::Ctxt::Ctxt (
    const Ctxt & other ) [default]
```

### 7.31.2.3 Ctxt() [3/3]

```
helib::Ctxt::Ctxt (
    ZeroCtxtLike_type ,
    const Ctxt & ctxt )
```

## 7.31.3 Member Function Documentation

### 7.31.3.1 addConstant() [1/4]

```
void helib::Ctxt::addConstant (
    const DoubleCRT & dcrt,
    double size = -1.0 )
```

Add a constant polynomial. If provided, size should be a high-probability bound on the L-infty norm of the canonical embedding. Otherwise, for the `DoubleCRT` variant, a bound based on the assumption that the coefficients are uniformly and independently distributed over  $[-\text{ptxtSpace}/2, \text{ptxtSpace}/2]$ . For the other variants, explicit bounds are computed (if not `CKKS`).

### 7.31.3.2 addConstant() [2/4]

```
void helib::Ctxt::addConstant (
    const NTL::ZZ & c )
```

**7.31.3.3 addConstant()** [3/4]

```
void helib::Ctxt::addConstant (
    const NTL::ZZX & poly,
    double size = -1.0 )
```

**7.31.3.4 addConstant()** [4/4]

```
template<typename Scheme >
void helib::Ctxt::addConstant (
    const Ptxt< Scheme > & ptxt ) [inline]
```

Add a [BGV](#) plaintext to this [Ctxt](#).

**Parameters**

<i>ptxt</i>	Plaintext <a href="#">Ptxt</a> object with which to add.
-------------	---

**7.31.3.5 addConstantCKKS()** [1/7]

```
void helib::Ctxt::addConstantCKKS (
    const DoubleCRT & dCRT,
    NTL::xdouble size = NTL::xdouble(-1.0),
    NTL::xdouble factor = NTL::xdouble(-1.0) )
```

**7.31.3.6 addConstantCKKS()** [2/7]

```
void helib::Ctxt::addConstantCKKS (
    const NTL::ZZ & c )
```

**7.31.3.7 addConstantCKKS()** [3/7]

```
void helib::Ctxt::addConstantCKKS (
    const NTL::ZZX & poly,
    NTL::xdouble size = NTL::xdouble(-1.0),
    NTL::xdouble factor = NTL::xdouble(-1.0) )
```

**7.31.3.8 addConstantCKKS()** [4/7]

```
void helib::Ctxt::addConstantCKKS (
    const Ptxt< CKKS > & ptxt )
```

Add a CKKS plaintext to this Ctxt.

**Parameters**

<i>ptxt</i>	Plaintext Ptxt object with which to add.
-------------	---

**7.31.3.9 addConstantCKKS()** [5/7]

```
void helib::Ctxt::addConstantCKKS (
    const std::vector< std::complex< double >> & ptxt )
```

**7.31.3.10 addConstantCKKS()** [6/7]

```
void helib::Ctxt::addConstantCKKS (
    double x ) [inline]
```

**7.31.3.11 addConstantCKKS()** [7/7]

```
void helib::Ctxt::addConstantCKKS (
    std::pair< long, long > num )
```

add a rational number in the form a/b, a,b are long

**7.31.3.12 addCtxt()**

```
void helib::Ctxt::addCtxt (
    const Ctxt & other,
    bool negative = false )
```



### 7.31.3.13 automorph()

```
void helib::Ctxt::automorph (
    long k )
```

### 7.31.3.14 bitCapacity()

```
long helib::Ctxt::bitCapacity ( ) const [inline]
```

the capacity in bits, returned as an integer

### 7.31.3.15 blindCtxt()

```
void helib::Ctxt::blindCtxt (
    const NTL::ZZX & poly )
```

Add a high-noise encryption of the given constant.

### 7.31.3.16 bringToSet()

```
void helib::Ctxt::bringToSet (
    const IndexSet & s )
```

make the primeSet equal to newPrimeSet, via modUpToSet and modDownToSet

### 7.31.3.17 bumpNoiseBound()

```
void helib::Ctxt::bumpNoiseBound (
    double factor ) [inline]
```

### 7.31.3.18 capacity()

```
double helib::Ctxt::capacity ( ) const [inline]
```

returns the "capacity" of a ciphertext, which is the log of the ratio of the modulus to the noise bound

### 7.31.3.19 `cleanUp()`

```
void helib::Ctxt::cleanUp ( )
```

### 7.31.3.20 `clear()`

```
void helib::Ctxt::clear ( ) [inline]
```

### 7.31.3.21 `complexConj()`

```
void helib::Ctxt::complexConj ( )
```

### 7.31.3.22 `cube()`

```
void helib::Ctxt::cube ( ) [inline]
```

### 7.31.3.23 `divideBy2()`

```
void helib::Ctxt::divideBy2 ( )
```

### 7.31.3.24 `divideByP()`

```
void helib::Ctxt::divideByP ( )
```

Divide a ciphertext by  $p$ , for plaintext space  $p^r$ ,  $r > 1$ . It is assumed that the ciphertext encrypts a polynomial which is zero mod  $p$ . If this is not the case then the result will not be a valid ciphertext anymore. As a side-effect, the plaintext space is reduced from  $p^r$  to  $p^{r-1}$ .

### 7.31.3.25 `dropSmallAndSpecialPrimes()`

```
void helib::Ctxt::dropSmallAndSpecialPrimes ( )
```

the corresponding primeSet

drop all smallPrimes and specialPrimes, adding ctxtPrimes as necessary to ensure that the scaled noise is above the modulus-switching added noise term.

### 7.31.3.26 DummyEncrypt()

```
void helib::Ctxt::DummyEncrypt (
    const NTL::ZZX & ptxt,
    double size = -1.0 )
```

Dummy encryption, just encodes the plaintext in a [Ctxt](#) object If provided, size should be a high-probability bound on the L-infty norm of the canonical embedding

### 7.31.3.27 effectiveR()

```
long helib::Ctxt::effectiveR ( ) const [inline]
```

### 7.31.3.28 equalizeRationalFactors()

```
void helib::Ctxt::equalizeRationalFactors (
    Ctxt & c1,
    Ctxt & c2 ) [static]
```

### 7.31.3.29 equalsTo()

```
bool helib::Ctxt::equalsTo (
    const Ctxt & other,
    bool comparePkeys = true ) const
```

### 7.31.3.30 evalPoly()

```
void helib::Ctxt::evalPoly (
    const NTL::ZZX & poly )
```

compute the power  $X, X^2, \dots, X^n$

Evaluate the cleartext poly on the encrypted ciphertext

### 7.31.3.31 extractBits()

```
void helib::Ctxt::extractBits (
    std::vector< Ctxt > & bits,
    long nBits2extract = 0 ) [inline]
```

### 7.31.3.32 frobeniusAutomorph()

```
void helib::Ctxt::frobeniusAutomorph (
    long j )
```

applies the automorphism  $p^j$  using smartAutomorphism

### 7.31.3.33 getContext()

```
const Context& helib::Ctxt::getContext ( ) const [inline]
```

### 7.31.3.34 getKeyID()

```
long helib::Ctxt::getKeyID ( ) const
```

### 7.31.3.35 getNoiseBound()

```
const NTL::xdouble& helib::Ctxt::getNoiseBound ( ) const [inline]
```

### 7.31.3.36 getPrimeSet()

```
const IndexSet& helib::Ctxt::getPrimeSet ( ) const [inline]
```

### 7.31.3.37 getPtxtMag()

```
const NTL::xdouble& helib::Ctxt::getPtxtMag ( ) const [inline]
```

### 7.31.3.38 getPtxtSpace()

```
long helib::Ctxt::getPtxtSpace ( ) const [inline]
```

### 7.31.3.39 getPubKey()

```
const PubKey& helib::Ctxt::getPubKey ( ) const [inline]
```

### 7.31.3.40 getRatFactor()

```
const NTL::xdouble& helib::Ctxt::getRatFactor ( ) const [inline]
```

### 7.31.3.41 hackPtxtSpace()

```
void helib::Ctxt::hackPtxtSpace (
    long newPtxtSpace ) [inline]
```

### 7.31.3.42 inCanonicalForm()

```
bool helib::Ctxt::inCanonicalForm (
    long keyID = 0 ) const [inline]
```

A canonical ciphertext has (at most) handles pointing to (1,s)

### 7.31.3.43 isCKKS()

```
bool helib::Ctxt::isCKKS ( ) const [inline]
```

### 7.31.3.44 isCorrect()

```
bool helib::Ctxt::isCorrect ( ) const [inline]
```

Would this ciphertext be decrypted without errors?

### 7.31.3.45 isEmpty()

```
bool helib::Ctxt::isEmpty ( ) const [inline]
```

Is this an empty ciphertext without any parts.

**7.31.3.46 log\_of\_ratio()**

```
double helib::Ctxt::log_of_ratio ( ) const [inline]
```

Returns  $\log(\text{noiseBound}) - \log(q)$

**7.31.3.47 logOfPrimeSet()**

```
double helib::Ctxt::logOfPrimeSet ( ) const [inline]
```

returns the log of the prime set

**7.31.3.48 modDownToSet()**

```
void helib::Ctxt::modDownToSet (
    const IndexSet & s )
```

Modulus-switching down (to a smaller modulus). mod-switch down to  $\text{primeSet} \setminus \text{intersect } s$ , after this call we have  $\text{primeSet} \leq s$ .  $s$  must contain either all special primes or none of them.

**7.31.3.49 modSwitchAddedNoiseBound()**

```
NTL::xdouble helib::Ctxt::modSwitchAddedNoiseBound ( ) const
```

Estimate the added noise.

**7.31.3.50 modUpToSet()**

```
void helib::Ctxt::modUpToSet (
    const IndexSet & s )
```

Modulus-switching up (to a larger modulus). Must have  $\text{primeSet} \leq s$ , and  $s$  must contain either all the special primes or none of them.

**7.31.3.51 multByConstant()** [1/5]

```
void helib::Ctxt::multByConstant (
    const DoubleCRT & dcrt,
    double size = -1.0 )
```

Multiply-by-constant. If the size is not given, for the DCRT variant, we use a high probability bound assuming "random" coefficients mod  $\text{ptxtSpace}$ , while for the other variants, we use explicitly computed bounds (if not CKKS).

**7.31.3.52 multByConstant()** [2/5]

```
void helib::Ctxt::multByConstant (
    const NTL::ZZ & c )
```

**7.31.3.53 multByConstant()** [3/5]

```
void helib::Ctxt::multByConstant (
    const NTL::ZZX & poly,
    double size = -1.0 )
```

**7.31.3.54 multByConstant()** [4/5]

```
template<typename Scheme >
void helib::Ctxt::multByConstant (
    const Ptxt< Scheme > & ptxt ) [inline]
```

Multiply a [BGV](#) plaintext to this [Ctxt](#).

**Parameters**

<i>ptxt</i>	Plaintext <a href="#">Ptxt</a> object with which to mul- tiply.
-------------	---

**7.31.3.55 multByConstant()** [5/5]

```
void helib::Ctxt::multByConstant (
    const ZZX & poly,
    double size = -1.0 )
```

**7.31.3.56 multByConstantCKKS()** [1/6]

```
void helib::Ctxt::multByConstantCKKS (
    const DoubleCRT & dCRT,
    NTL::xdouble size = NTL::xdouble(-1.0),
    NTL::xdouble factor = NTL::xdouble(-1.0),
    double roundingErr = -1.0 )
```

**7.31.3.57 multByConstantCKKS()** [2/6]

```
void helib::Ctxt::multByConstantCKKS (
    const NTL::ZZX & poly,
    NTL::xdouble size = NTL::xdouble(-1.0),
    NTL::xdouble factor = NTL::xdouble(-1.0),
    double roundingErr = -1.0 ) [inline]
```

**7.31.3.58 multByConstantCKKS()** [3/6]

```
void helib::Ctxt::multByConstantCKKS (
    const Ptxt< CKKS > & ptxt )
```

Multiply a [CKKS](#) plaintext to this [Ctxt](#).

**Parameters**

<i>ptxt</i>	Plaintext <a href="#">Ptxt</a> object poly- nomial with which to mul- tiply.
-------------	--

**7.31.3.59 multByConstantCKKS()** [4/6]

```
void helib::Ctxt::multByConstantCKKS (
    const std::vector< std::complex< double >> & ptxt )
```

**7.31.3.60 multByConstantCKKS()** [5/6]

```
void helib::Ctxt::multByConstantCKKS (
    double x ) [inline]
```

multiply by a rational number or floating point

**7.31.3.61 multByConstantCKKS()** [6/6]

```
void helib::Ctxt::multByConstantCKKS (
    std::pair< long, long > num ) [inline]
```



### 7.31.3.62 multByP()

```
void helib::Ctxt::multByP (
    long e = 1 ) [inline]
```

Multiply ciphertext by  $p^e$ , for plaintext space  $p^r$ . This also has the side-effect of increasing the plaintext space to  $p^{r+e}$ .

### 7.31.3.63 multiplyBy()

```
void helib::Ctxt::multiplyBy (
    const Ctxt & other )
```

### 7.31.3.64 multiplyBy2()

```
void helib::Ctxt::multiplyBy2 (
    const Ctxt & other1,
    const Ctxt & other2 )
```

### 7.31.3.65 multLowLvl()

```
void helib::Ctxt::multLowLvl (
    const Ctxt & other,
    bool destructive = false )
```

### 7.31.3.66 naturalPrimeSet()

```
IndexSet helib::Ctxt::naturalPrimeSet ( ) const
```

"natural size" is size before squaring

### 7.31.3.67 naturalSize()

```
double helib::Ctxt::naturalSize ( ) const
```

### 7.31.3.68 negate()

```
void helib::Ctxt::negate ( )
```

### 7.31.3.69 nxorConstant() [1/2]

```
void helib::Ctxt::nxorConstant (
    const DoubleCRT & poly,
    UNUSED double size = -1.0 ) [inline]
```

### 7.31.3.70 nxorConstant() [2/2]

```
void helib::Ctxt::nxorConstant (
    const NTL::ZZX & poly,
    double size = -1.0 ) [inline]
```

### 7.31.3.71 operator"!="()

```
bool helib::Ctxt::operator!= (
    const Ctxt & other ) const [inline]
```

### 7.31.3.72 operator\*=( ) [1/5]

```
Ctxt& helib::Ctxt::operator*= (
    const Ctxt & other ) [inline]
```

### 7.31.3.73 operator\*=( ) [2/5]

```
Ctxt & helib::Ctxt::operator*= (
    const long scalar )
```

Times equals operator with a long.

## Parameters

<i>scalar</i>	Constant by which to multiply.
---------------	--------------------------------

## Returns

Reference to `*this` post multiplication.

**7.31.3.74 operator\*=( ) [3/5]**

```
Ctxt & helib::Ctxt::operator*= (
    const NTL::ZZX & poly )
```

Times equals operator with a `ZZX`.

## Parameters

<i>poly</i>	Element by which to multiply.
-------------	-------------------------------

## Returns

Reference to `*this` post multiplication.

**7.31.3.75 operator\*=( ) [4/5]**

```
Ctxt & helib::Ctxt::operator*= (
    const Ptxt< BGV > & other )
```

Times equals operator with a `BGV Ptxt`.

## Parameters

<i>other</i>	Right hand side of multiplication.
--------------	------------------------------------

**Returns**

reference to `*this` post multiplication.

**7.31.3.76 operator\*=( ) [5/5]**

```
Ctxt & helib::Ctxt::operator*= (
    const Ptxt< CKKS > & other )
```

Times equals operator with a [CKKS Ptxt](#).

**Parameters**

<i>other</i>	Right hand side of multiplication.
--------------	------------------------------------

**Returns**

Reference to `*this` post multiplication.

**7.31.3.77 operator+=( ) [1/3]**

```
Ctxt& helib::Ctxt::operator+= (
    const Ctxt & other ) [inline]
```

**7.31.3.78 operator+=( ) [2/3]**

```
Ctxt & helib::Ctxt::operator+= (
    const Ptxt< BGV > & other )
```

Plus equals operator with a [BGV Ptxt](#).

**Parameters**

<i>other</i>	Right hand side of addition.
--------------	------------------------------

**Returns**

Reference to `*this` post addition.

**7.31.3.79 operator+=() [3/3]**

```
Ctxt & helib::Ctxt::operator+= (
    const Ptxt< CKKS > & other )
```

Plus equals operator with a [CKKS Ptxt](#).

**Parameters**

<i>other</i>	Right hand side of addition.
--------------	------------------------------

**Returns**

Reference to `*this` post addition.

**7.31.3.80 operator-=() [1/3]**

```
Ctxt& helib::Ctxt::operator-= (
    const Ctxt & other ) [inline]
```

**7.31.3.81 operator-=() [2/3]**

```
Ctxt & helib::Ctxt::operator-= (
    const Ptxt< BGV > & other )
```

Minus equals operator with a [BGV Ptxt](#).

**Parameters**

<i>other</i>	Right hand side of subtraction.
--------------	---------------------------------

**Returns**

Reference to `*this` post subtraction.

**7.31.3.82 operator-=( ) [3/3]**

```
Ctxt & helib::Ctxt::operator-= (
    const Ptxt< CKKS > & other )
```

Minus equals operator with a `CKKS Ptxt`.

**Parameters**

<i>other</i>	Right hand side of subtraction.
--------------	---------------------------------

**Returns**

Reference to `*this` post subtraction.

**7.31.3.83 operator=( )**

```
Ctxt& helib::Ctxt::operator= (
    const Ctxt & other ) [inline]
```

**7.31.3.84 operator==( )**

```
bool helib::Ctxt::operator== (
    const Ctxt & other ) const [inline]
```

**7.31.3.85 operator>>=( )**

```
Ctxt& helib::Ctxt::operator>>= (
    long k ) [inline]
```

### 7.31.3.86 power()

```
void helib::Ctxt::power (
    long e )
```

raise ciphertext to some power

### 7.31.3.87 rawModSwitch()

```
double helib::Ctxt::rawModSwitch (
    std::vector< NTL::ZZX > & zzParts,
    long toModulus ) const
```

Special-purpose modulus-switching for bootstrapping.

Mod-switch to an externally-supplied modulus. The modulus need not be in the moduli-chain in the context, and does not even need to be a prime. The ciphertext *\*this* is not affected, instead the result is returned in the `zzParts` `std::vector`, as a `std::vector` of `ZZX`'es. Returns an estimate for the scaled noise (not including the additive mod switching noise)

### 7.31.3.88 read()

```
void helib::Ctxt::read (
    std::istream & str )
```

### 7.31.3.89 reducePtxtSpace()

```
void helib::Ctxt::reducePtxtSpace (
    long newPtxtSpace )
```

Reduce plaintext space to a divisor of the original plaintext space.

### 7.31.3.90 reLinearize()

```
void helib::Ctxt::reLinearize (
    long keyIdx = 0 )
```

### 7.31.3.91 setPtxtMag()

```
void helib::Ctxt::setPtxtMag (
    const NTL::xdouble & z ) [inline]
```

**7.31.3.92 smartAutomorph()**

```
void helib::Ctxt::smartAutomorph (
    long k )
```

automorphism with re-linearization

**7.31.3.93 square()**

```
void helib::Ctxt::square ( ) [inline]
```

**7.31.3.94 write()**

```
void helib::Ctxt::write (
    std::ostream & str ) const
```

**7.31.3.95 xorConstant() [1/2]**

```
void helib::Ctxt::xorConstant (
    const DoubleCRT & poly,
    UNUSED double size = -1.0 ) [inline]
```

Convenience method: XOR and nXOR with arbitrary plaintext space:  $a \text{ xor } b = a + b - 2ab = a + (1 - 2a) * b$ ,  $a \text{ nxor } b = 1 - a - b + 2ab = (b - 1)(2a - 1) + a$

**7.31.3.96 xorConstant() [2/2]**

```
void helib::Ctxt::xorConstant (
    const NTL::ZZX & poly,
    double size = -1.0 ) [inline]
```

**7.31.4 Friends And Related Function Documentation****7.31.4.1 BasicAutomorphPrecon**

```
friend class BasicAutomorphPrecon [friend]
```



#### 7.31.4.2 operator<<

```
std::ostream& operator<< (
    std::ostream & str,
    const Ctxt & ctxt ) [friend]
```

#### 7.31.4.3 operator>>

```
std::istream& operator>> (
    std::istream & str,
    Ctxt & ctxt ) [friend]
```

#### 7.31.4.4 PubKey

```
friend class PubKey [friend]
```

#### 7.31.4.5 SecKey

```
friend class SecKey [friend]
```

The documentation for this class was generated from the following files:

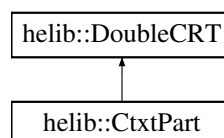
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/Ctxt.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Ctxt.cpp
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/polyEval.cpp

## 7.32 helib::CtxtPart Class Reference

One entry in a ciphertext std::vector.

```
#include <Ctxt.h>
```

Inheritance diagram for helib::CtxtPart:



## Public Member Functions

- bool `operator==` (const `CtxtPart` &other) const
- bool `operator!=` (const `CtxtPart` &other) const
- `CtxtPart` (const `Context` &\_context, const `IndexSet` &s)
- `CtxtPart` (const `Context` &\_context, const `IndexSet` &s, const `SKHandle` &otherHandle)
- `CtxtPart` (const `DoubleCRT` &other)
- `CtxtPart` (const `DoubleCRT` &other, const `SKHandle` &otherHandle)
- void `read` (std::istream &str)
- void `write` (std::ostream &str) const

## Public Attributes

- `SKHandle` `skHandle`

*The handle is a public data member.*

### 7.32.1 Detailed Description

One entry in a ciphertext std::vector.

A ciphertext part consists of a polynomial (element of the ring  $R_Q$ ) and a handle to the corresponding secret-key polynomial.

### 7.32.2 Constructor & Destructor Documentation

#### 7.32.2.1 `CtxtPart()` [1/4]

```
helib::CtxtPart::CtxtPart (
    const Context & _context,
    const IndexSet & s ) [inline]
```

#### 7.32.2.2 `CtxtPart()` [2/4]

```
helib::CtxtPart::CtxtPart (
    const Context & _context,
    const IndexSet & s,
    const SKHandle & otherHandle ) [inline]
```

#### 7.32.2.3 `CtxtPart()` [3/4]

```
helib::CtxtPart::CtxtPart (
    const DoubleCRT & other ) [inline], [explicit]
```

### 7.32.2.4 CtxtPart() [4/4]

```
helib::CtxtPart::CtxtPart (
    const DoubleCRT & other,
    const SKHandle & otherHandle ) [inline]
```

## 7.32.3 Member Function Documentation

### 7.32.3.1 operator!=(())

```
bool helib::CtxtPart::operator!= (
    const CtxtPart & other ) const [inline]
```

### 7.32.3.2 operator==(())

```
bool helib::CtxtPart::operator== (
    const CtxtPart & other ) const
```

### 7.32.3.3 read()

```
void helib::CtxtPart::read (
    std::istream & str )
```

### 7.32.3.4 write()

```
void helib::CtxtPart::write (
    std::ostream & str ) const
```

## 7.32.4 Member Data Documentation

### 7.32.4.1 skHandle

`SKHandle` `helib::CtxtPart::skHandle`

The handle is a public data member.

The documentation for this class was generated from the following files:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/Ctxt.h`
- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/Ctxt.cpp`

## 7.33 helib::CubeSignature Class Reference

Holds a vector of dimensions for a hypercube and some additional data.

```
#include <hypercube.h>
```

### Public Member Functions

- `CubeSignature` ()
- void `initSignature` (const long \_dims[], long \_ndims)
- template<typename VecType >  
void `initSignature` (const VecType &\_dims)
- `CubeSignature` (const long \_dims[], long \_ndims)
- `CubeSignature` (const NTL::Vec< long > &\_dims)
- `CubeSignature` (const std::vector< long > &\_dims)
- long `getNumDims` () const  
*number of dimensions*
- long `getSize` () const  
*total size of cube*
- long `getDim` (long d) const  
*size of dimension d*
- long `getProd` (long d) const  
*product of sizes of dimensions d, d+1, ...*
- long `getProd` (long from, long to) const  
*product of sizes of dimensions from, from+1, ..., to-1*
- long `getCoord` (long i, long d) const  
*get coordinate in dimension d of index i*
- long `addCoord` (long i, long d, long offset) const  
*add offset to coordinate in dimension d of index i*
- template<typename VecType >  
bool `incrementCoords` (VecType &v) const
- template<typename VecType >  
void `getAllCoords` (VecType &v, long i) const
- template<typename VecType >  
long `assembleCoords` (VecType &v) const
- long `numSlices` (long d=1) const  
*number of slices*
- long `sliceSize` (long d=1) const  
*size of one slice*
- long `numCols` () const  
*number of columns*
- std::pair< long, long > `breakIndexByDim` (long idx, long dim) const
- long `assembleIndexByDim` (std::pair< long, long > idx, long dim) const  
*The inverse of breakIndexByDim.*

## Friends

- `std::ostream & operator<< (std::ostream &s, const CubeSignature &sig)`

### 7.33.1 Detailed Description

Holds a vector of dimensions for a hypercube and some additional data.

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 `CubeSignature()` [1/4]

```
helib::CubeSignature::CubeSignature ( ) [inline]
```

#### 7.33.2.2 `CubeSignature()` [2/4]

```
helib::CubeSignature::CubeSignature (
    const long _dims[],
    long _ndims ) [inline]
```

#### 7.33.2.3 `CubeSignature()` [3/4]

```
helib::CubeSignature::CubeSignature (
    const NTL::Vec< long > & _dims ) [inline]
```

#### 7.33.2.4 `CubeSignature()` [4/4]

```
helib::CubeSignature::CubeSignature (
    const std::vector< long > & _dims ) [inline]
```

### 7.33.3 Member Function Documentation

### 7.33.3.1 addCoord()

```
long helib::CubeSignature::addCoord (
    long i,
    long d,
    long offset ) const [inline]
```

add offset to coordinate in dimension d of index i

### 7.33.3.2 assembleCoords()

```
template<typename VecType >
long helib::CubeSignature::assembleCoords (
    VecType & v ) const [inline]
```

reconstruct index from its coordinates VecType is either `std::vector<intType>` or `NTL::Vec<intType>`

### 7.33.3.3 assembleIndexByDim()

```
long helib::CubeSignature::assembleIndexByDim (
    std::pair< long, long > idx,
    long dim ) const
```

The inverse of `breakIndexByDim`.

### 7.33.3.4 breakIndexByDim()

```
std::pair< long, long > helib::CubeSignature::breakIndexByDim (
    long idx,
    long dim ) const
```

Break an index into the hypercube to index of the dimension-dim subcube and index inside that subcube.

### 7.33.3.5 getAllCoords()

```
template<typename VecType >
void helib::CubeSignature::getAllCoords (
    VecType & v,
    long i ) const [inline]
```

get the coordinates of index i in all dimensions. VecType is either `std::vector<intType>` or `NTL::Vec<intType>`

### 7.33.3.6 getCoord()

```
long helib::CubeSignature::getCoord (
    long i,
    long d ) const [inline]
```

get coordinate in dimension d of index i

### 7.33.3.7 getDim()

```
long helib::CubeSignature::getDim (
    long d ) const [inline]
```

size of dimension d

### 7.33.3.8 getNumDims()

```
long helib::CubeSignature::getNumDims ( ) const [inline]
```

number of dimensions

### 7.33.3.9 getProd() [1/2]

```
long helib::CubeSignature::getProd (
    long d ) const [inline]
```

product of sizes of dimensions d, d+1, ...

### 7.33.3.10 getProd() [2/2]

```
long helib::CubeSignature::getProd (
    long from,
    long to ) const [inline]
```

product of sizes of dimensions from, from+1, ..., to-1

#### 7.33.3.11 getSize()

```
long helib::CubeSignature::getSize ( ) const [inline]
```

total size of cube

#### 7.33.3.12 incrementCoords()

```
template<typename VecType >
bool helib::CubeSignature::incrementCoords (
    VecType & v ) const [inline]
```

Increment the coordinates to point to next index, returning false if already at maximum value. VecType is either `std::vector<intType>` or [NTL:Vec<intType>](#)

#### 7.33.3.13 initSignature() [1/2]

```
void helib::CubeSignature::initSignature (
    const long _dims[],
    long _ndims ) [inline]
```

#### 7.33.3.14 initSignature() [2/2]

```
template<typename VecType >
void helib::CubeSignature::initSignature (
    const VecType & _dims ) [inline]
```

#### 7.33.3.15 numCols()

```
long helib::CubeSignature::numCols ( ) const [inline]
```

number of columns

#### 7.33.3.16 numSlices()

```
long helib::CubeSignature::numSlices (
    long d = 1 ) const [inline]
```

number of slices



### 7.33.3.17 sliceSize()

```
long helib::CubeSignature::sliceSize (
    long d = 1 ) const [inline]
```

size of one slice

## 7.33.4 Friends And Related Function Documentation

### 7.33.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const CubeSignature & sig ) [friend]
```

The documentation for this class was generated from the following files:

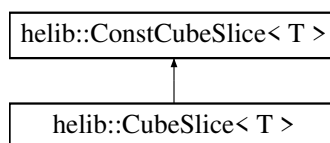
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[hypercube.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[hypercube.cpp](#)

## 7.34 helib::CubeSlice< T > Class Template Reference

A lower-dimension slice of a hypercube.

```
#include <hypercube.h>
```

Inheritance diagram for helib::CubeSlice< T >:



### Public Member Functions

- [CubeSlice](#) ([HyperCube](#)< T > &\_cube)
- [CubeSlice](#) (NTL::Vec< T > &\_data, const [CubeSignature](#) &\_sig)
- [CubeSlice](#) (const [CubeSlice](#)< T > &bigger, long i, long \_dimOffset=1)
- [CubeSlice](#) ([HyperCube](#)< T > &\_cube, long i, long \_dimOffset=1)
- void [copy](#) (const [ConstCubeSlice](#)< T > &other) const
- T & [at](#) (long i) const
- T & [operator\[\]](#) (long i) const

### 7.34.1 Detailed Description

```
template<typename T>
class helib::CubeSlice< T >
```

A lower-dimension slice of a hypercube.

### 7.34.2 Constructor & Destructor Documentation

#### 7.34.2.1 CubeSlice() [1/4]

```
template<typename T >
helib::CubeSlice< T >::CubeSlice (
    HyperCube< T > & _cube ) [inline], [explicit]
```

#### 7.34.2.2 CubeSlice() [2/4]

```
template<typename T >
helib::CubeSlice< T >::CubeSlice (
    NTL::Vec< T > & _data,
    const CubeSignature & _sig ) [inline]
```

#### 7.34.2.3 CubeSlice() [3/4]

```
template<typename T >
helib::CubeSlice< T >::CubeSlice (
    const CubeSlice< T > & bigger,
    long i,
    long _dimOffset = 1 ) [inline]
```

#### 7.34.2.4 CubeSlice() [4/4]

```
template<typename T >
helib::CubeSlice< T >::CubeSlice (
    HyperCube< T > & _cube,
    long i,
    long _dimOffset = 1 ) [inline]
```

### 7.34.3 Member Function Documentation

#### 7.34.3.1 at()

```
template<typename T >
T& helib::CubeSlice< T >::at (
    long i ) const [inline]
```

#### 7.34.3.2 copy()

```
template<typename T >
void helib::CubeSlice< T >::copy (
    const ConstCubeSlice< T > & other ) const
```

#### 7.34.3.3 operator[]()

```
template<typename T >
T& helib::CubeSlice< T >::operator[] (
    long i ) const [inline]
```

The documentation for this class was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/hypercube.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/hypercube.cpp](#)

## 7.35 helib::DAGnode Class Reference

A node in an addition-DAG structure.

### Public Member Functions

- [DAGnode](#) ([NodeIdx](#) ii, bool qq, long lvl, long chl=0, [DAGnode](#) \*pt1=nullptr, [DAGnode](#) \*pt2=nullptr)
- [DAGnode](#) ([DAGnode](#) &&other)
- std::string [nodeName](#) () const

## Public Attributes

- [NodeIdx](#) `idx`
- `bool` `isQ`
- `long` `level`
- `std::atomic_long` `childrenLeft`
- [DAGnode](#) \* `parent1`
- [DAGnode](#) \* `parent2`
- `std::mutex` `ct_mtx`
- [Ctxt](#) \* `ct`

### 7.35.1 Detailed Description

A node in an addition-DAG structure.

### 7.35.2 Constructor & Destructor Documentation

#### 7.35.2.1 DAGnode() [1/2]

```
helib::DAGnode::DAGnode (
    NodeIdx ii,
    bool qq,
    long lvl,
    long chl = 0,
    DAGnode * pt1 = nullptr,
    DAGnode * pt2 = nullptr ) [inline]
```

#### 7.35.2.2 DAGnode() [2/2]

```
helib::DAGnode::DAGnode (
    DAGnode && other ) [inline]
```

### 7.35.3 Member Function Documentation

#### 7.35.3.1 nodeName()

```
std::string helib::DAGnode::nodeName ( ) const [inline]
```

## 7.35.4 Member Data Documentation

### 7.35.4.1 childrenLeft

`std::atomic_long helib::DAGnode::childrenLeft`

### 7.35.4.2 ct

`Ctxt* helib::DAGnode::ct`

### 7.35.4.3 ct\_mtx

`std::mutex helib::DAGnode::ct_mtx`

### 7.35.4.4 idx

`NodeIdx helib::DAGnode::idx`

### 7.35.4.5 isQ

`bool helib::DAGnode::isQ`

### 7.35.4.6 level

`long helib::DAGnode::level`

### 7.35.4.7 parent1

`DAGnode* helib::DAGnode::parent1`

#### 7.35.4.8 parent2

`DAGnode * helib::DAGnode::parent2`

The documentation for this class was generated from the following file:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/binaryArith.cpp`

## 7.36 helib::decode\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- `template<typename T >`  
`static void apply (const EncryptedArrayDerived< type > &ea, std::vector< T > &array, const PlaintextArray &pa)`

#### 7.36.1 Member Function Documentation

##### 7.36.1.1 apply()

```
template<typename type >
template<typename T >
static void helib::decode_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    std::vector< T > & array,
    const PlaintextArray & pa ) [inline], [static]
```

The documentation for this class was generated from the following file:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EncryptedArray.cpp`

## 7.37 helib::deep\_clone< X > Class Template Reference

Deep copy: initialize with clone.

```
#include <clonedPtr.h>
```

### Static Public Member Functions

- `static X * apply (const X *x)`

#### 7.37.1 Detailed Description

```
template<typename X>
class helib::deep_clone< X >
```

Deep copy: initialize with clone.

## Template Parameters

X	The class to which this points
---	--------------------------------

## 7.37.2 Member Function Documentation

### 7.37.2.1 apply()

```
template<typename X >
static X* helib::deep_clone< X >::apply (
    const X * x ) [inline], [static]
```

The documentation for this class was generated from the following file:

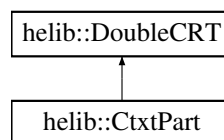
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/clonedPtr.h

## 7.38 helib::DoubleCRT Class Reference

Implementing polynomials (elements in the ring  $R_Q$ ) in double-CRT form.

```
#include <DoubleCRT.h>
```

Inheritance diagram for helib::DoubleCRT:



### Public Member Functions

- [DoubleCRT](#) (const [DoubleCRT](#) &other)=default
- [DoubleCRT](#) (const NTL::ZZX &poly, const [Context](#) &\_context, const [IndexSet](#) &indexSet)  
*Initializing [DoubleCRT](#) from a ZZ $X$  polynomial.*
- [DoubleCRT](#) (const [zzX](#) &poly, const [Context](#) &\_context, const [IndexSet](#) &indexSet)  
*Same as above, but with zz $X$ 's.*
- [DoubleCRT](#) (const [Context](#) &\_context, const [IndexSet](#) &indexSet)  
*Also specify the [IndexSet](#) explicitly.*
- [DoubleCRT](#) & operator= (const [DoubleCRT](#) &other)
- [DoubleCRT](#) & operator= (const [zzX](#) &poly)
- [DoubleCRT](#) & operator= (const NTL::ZZX &poly)
- [DoubleCRT](#) & operator= (const NTL::ZZ &num)
- [DoubleCRT](#) & operator= (const long num)
- long [getOneRow](#) (NTL::Vec< long > &row, long idx, bool positive=false) const

Get one row of a polynomial.

- long [getOneRow](#) (NTL::zz\_pX &row, long idx) const
- void [toPoly](#) (NTL::ZZX &p, const [IndexSet](#) &s, bool positive=false) const
 

*Recovering the polynomial in coefficient representation. This yields an integer polynomial with coefficients in  $[-P/2, P/2]$ , unless the positive flag is set to true, in which case we get coefficients in  $[0, P-1]$  ( $P$  is the product of all moduli used). Using the optional [IndexSet](#) param we compute the polynomial reduced modulo the product of only the primes in that set.*
- void [toPoly](#) (NTL::ZZX &p, bool positive=false) const
- bool [operator==](#) (const [DoubleCRT](#) &other) const
- bool [operator!=](#) (const [DoubleCRT](#) &other) const
- [DoubleCRT](#) & [SetZero](#) ()
- [DoubleCRT](#) & [SetOne](#) ()
- NTL::xdouble [breakIntoDigits](#) (std::vector< [DoubleCRT](#) > &dgts) const
 

*Break into  $n$  digits, according to the primeSets in context.digits. See Section 3.1.6 of the design document (re-linearization) Returns the sum of the canonical embedding of the digits.*
- void [addPrimes](#) (const [IndexSet](#) &s1, NTL::ZZX \*poly\_p=0)
 

*Expand the index set by  $s1$ . It is assumed that  $s1$  is disjoint from the current index set. If  $poly\_p \neq 0$ , then  $*poly\_p$  will first be set to the result of applying [toPoly](#).*
- double [addPrimesAndScale](#) (const [IndexSet](#) &s1)
 

*Expand index set by  $s1$ , and multiply by  $Prod_{\{q \text{ in } s1\}}$ .  $s1$  is disjoint from the current index set, returns  $\log(\text{product})$ .*
- void [removePrimes](#) (const [IndexSet](#) &s1)
 

*Remove  $s1$  from the index set.*
- void [setPrimes](#) (const [IndexSet](#) &s1)
 

*@ brief make prime set equal to  $s1$*
- const [Context](#) & [getContext](#) () const
- const [IndexMap](#)< NTL::vec\_long > & [getMap](#) () const
- const [IndexSet](#) & [getIndexSet](#) () const
- void [randomize](#) (const NTL::ZZ \*seed=nullptr)
 

*Fills each row  $i$  with random ints mod  $p_i$ , uses [NTL](#)'s PRG.*
- double [sampleSmall](#) ()
 

*Coefficients are  $-1/0/1$ , Prob[0]=1/2.*
- double [sampleSmallBounded](#) ()
- double [sampleHWt](#) (long Hwt)
 

*Coefficients are  $-1/0/1$  with pre-specified number of nonzeros.*
- double [sampleHWtBounded](#) (long Hwt)
- double [sampleGaussian](#) (double stdev=0.0)
 

*Coefficients are Gaussians Return a high probability bound on  $L$ -inf norm of canonical embedding.*
- double [sampleGaussianBounded](#) (double stdev=0.0)
- double [sampleUniform](#) (long B)
 

*Coefficients are uniform in  $[-B..B]$ .*
- NTL::xdouble [sampleUniform](#) (const NTL::ZZ &B)
- void [scaleDownToSet](#) (const [IndexSet](#) &s, long ptxtSpace, NTL::ZZX &delta)
- void [FFT](#) (const NTL::ZZX &poly, const [IndexSet](#) &s)
- void [FFT](#) (const [zzX](#) &poly, const [IndexSet](#) &s)
- void [reduce](#) () const
- void [read](#) (std::istream &str)
- void [write](#) (std::ostream &str) const

### Arithmetic operation

Only the "destructive" versions are used, i.e.,  $a += b$  is implemented but not  $a + b$ .

- [DoubleCRT](#) & [Negate](#) (const [DoubleCRT](#) &other)
- [DoubleCRT](#) & [Negate](#) ()
- [DoubleCRT](#) & [operator+=](#) (const [DoubleCRT](#) &other)



- [DoubleCRT](#) & [operator+=](#) (const NTL::ZZX &poly)
- [DoubleCRT](#) & [operator+=](#) (const NTL::ZZ &num)
- [DoubleCRT](#) & [operator+=](#) (long num)
- [DoubleCRT](#) & [operator-=](#) (const [DoubleCRT](#) &other)
- [DoubleCRT](#) & [operator-=](#) (const NTL::ZZX &poly)
- [DoubleCRT](#) & [operator-=](#) (const NTL::ZZ &num)
- [DoubleCRT](#) & [operator-=](#) (long num)
- [DoubleCRT](#) & [operator++](#) ()
- [DoubleCRT](#) & [operator--](#) ()
- void [operator++](#) (int)
- void [operator--](#) (int)
- [DoubleCRT](#) & [operator\\*=](#) (const [DoubleCRT](#) &other)
- [DoubleCRT](#) & [operator\\*=](#) (const NTL::ZZX &poly)
- [DoubleCRT](#) & [operator\\*=](#) (const NTL::ZZ &num)
- [DoubleCRT](#) & [operator\\*=](#) (long num)
- void [Add](#) (const [DoubleCRT](#) &other, bool matchIndexSets=true)
- void [Sub](#) (const [DoubleCRT](#) &other, bool matchIndexSets=true)
- void [Mul](#) (const [DoubleCRT](#) &other, bool matchIndexSets=true)
- [DoubleCRT](#) & [operator/=](#) (const NTL::ZZ &num)
- [DoubleCRT](#) & [operator/=](#) (long num)
- void [Exp](#) (long k)  
*Small-exponent polynomial exponentiation.*
- void [automorph](#) (long k)  
*Apply the automorphism  $F(X) \mapsto F(X^k)$  (with  $\gcd(k,m)=1$ )*
- [DoubleCRT](#) & [operator>>=](#) (long k)
- void [complexConj](#) ()  
*Compute the complex conjugate, the same as [automorph\(m-1\)](#)*

## Friends

- std::ostream & [operator<<](#) (std::ostream &s, const [DoubleCRT](#) &d)
- std::istream & [operator>>](#) (std::istream &s, [DoubleCRT](#) &d)

### 7.38.1 Detailed Description

Implementing polynomials (elements in the ring  $R_Q$ ) in double-CRT form.

Double-CRT form is a matrix of  $L$  rows and  $\phi(m)$  columns. The  $i$ 'th row contains the FFT of the element wrt the  $i$ th prime, i.e. the evaluations of the polynomial at the primitive  $m$ th roots of unity mod the  $i$ th prime. The polynomial thus represented is defined modulo the product of all the primes in use.

The list of primes is defined by the data member `indexMap`. `indexMap.getIndexSet()` defines the set of indices of primes associated with this [DoubleCRT](#) object: they index the primes stored in the associated [Context](#).

Arithmetic operations are computed modulo the product of the primes in use and also modulo  $\Phi_m(X)$ . Arithmetic operations can only be applied to [DoubleCRT](#) objects relative to the same context, trying to add/multiply objects that have different [Context](#) objects will raise an error.

### 7.38.2 Constructor & Destructor Documentation

**7.38.2.1 DoubleCRT()** [1/4]

```
helib::DoubleCRT::DoubleCRT (
    const DoubleCRT & other ) [default]
```

**7.38.2.2 DoubleCRT()** [2/4]

```
helib::DoubleCRT::DoubleCRT (
    const NTL::ZZX & poly,
    const Context & _context,
    const IndexSet & indexSet )
```

Initializing [DoubleCRT](#) from a ZZX polynomial.

**Parameters**

<i>poly</i>	The ring element itself, zero if not specified
<i>_context</i>	The context for this <a href="#">DoubleCRT</a> object, use "current active context" if not specified
<i>indexSet</i>	Which primes to use for this object, if not specified then use all of them

### 7.38.2.3 DoubleCRT() [3/4]

```
helib::DoubleCRT::DoubleCRT (
    const ZZ & poly,
    const Context & _context,
    const IndexSet & indexSet )
```

Same as above, but with ZZ's.

### 7.38.2.4 DoubleCRT() [4/4]

```
helib::DoubleCRT::DoubleCRT (
    const Context & _context,
    const IndexSet & indexSet )
```

Also specify the [IndexSet](#) explicitly.

## 7.38.3 Member Function Documentation

### 7.38.3.1 Add()

```
void helib::DoubleCRT::Add (
    const DoubleCRT & other,
    bool matchIndexSets = true ) [inline]
```

### 7.38.3.2 addPrimes()

```
void helib::DoubleCRT::addPrimes (
    const IndexSet & s1,
    NTL::ZZX * poly_p = 0 )
```

Expand the index set by s1. It is assumed that s1 is disjoint from the current index set. If poly\_p != 0, then \*poly\_p will first be set to the result of applying toPoly.

### 7.38.3.3 addPrimesAndScale()

```
double helib::DoubleCRT::addPrimesAndScale (
    const IndexSet & s1 )
```

Expand index set by s1, and multiply by Prod\_{q in s1}. s1 is disjoint from the current index set, returns log(product).

**7.38.3.4 automorph()**

```
void helib::DoubleCRT::automorph (
    long k )
```

Apply the automorphism  $F(X) \mapsto F(X^k)$  (with  $\gcd(k,m)=1$ )

**7.38.3.5 breakIntoDigits()**

```
NTL::xdouble helib::DoubleCRT::breakIntoDigits (
    std::vector< DoubleCRT > & dgt ) const
```

Break into  $n$  digits, according to the primeSets in context.digits. See Section 3.1.6 of the design document (re-linearization) Returns the sum of the canonical embedding of the digits.

**7.38.3.6 complexConj()**

```
void helib::DoubleCRT::complexConj ( )
```

Compute the complex conjugate, the same as automorph(m-1)

**7.38.3.7 Exp()**

```
void helib::DoubleCRT::Exp (
    long k )
```

Small-exponent polynomial exponentiation.

**7.38.3.8 FFT() [1/2]**

```
void helib::DoubleCRT::FFT (
    const NTL::ZZX & poly,
    const IndexSet & s )
```

**7.38.3.9 FFT() [2/2]**

```
void helib::DoubleCRT::FFT (
    const ZZ & poly,
    const IndexSet & s )
```

### 7.38.3.10 getContext()

```
const Context& helib::DoubleCRT::getContext ( ) const [inline]
```

### 7.38.3.11 getIndexSet()

```
const IndexSet& helib::DoubleCRT::getIndexSet ( ) const [inline]
```

### 7.38.3.12 getMap()

```
const IndexMap<NTL::vec_long>& helib::DoubleCRT::getMap ( ) const [inline]
```

### 7.38.3.13 getOneRow() [1/2]

```
long helib::DoubleCRT::getOneRow (
    NTL::Vec< long > & row,
    long idx,
    bool positive = false ) const
```

Get one row of a polynomial.

### 7.38.3.14 getOneRow() [2/2]

```
long helib::DoubleCRT::getOneRow (
    NTL::zz_pX & row,
    long idx ) const
```

### 7.38.3.15 Mul()

```
void helib::DoubleCRT::Mul (
    const DoubleCRT & other,
    bool matchIndexSets = true ) [inline]
```

#### 7.38.3.16 Negate() [1/2]

```
DoubleCRT& helib::DoubleCRT::Negate ( ) [inline]
```

#### 7.38.3.17 Negate() [2/2]

```
DoubleCRT & helib::DoubleCRT::Negate (
    const DoubleCRT & other )
```

#### 7.38.3.18 operator"!=() [1/2]

```
bool helib::DoubleCRT::operator!= (
    const DoubleCRT & other ) const [inline]
```

#### 7.38.3.19 operator\*=( ) [1/4]

```
DoubleCRT& helib::DoubleCRT::operator*= (
    const DoubleCRT & other ) [inline]
```

#### 7.38.3.20 operator\*=( ) [2/4]

```
DoubleCRT& helib::DoubleCRT::operator*= (
    const NTL::ZZ & num ) [inline]
```

#### 7.38.3.21 operator\*=( ) [3/4]

```
DoubleCRT& helib::DoubleCRT::operator*= (
    const NTL::ZZX & poly ) [inline]
```

#### 7.38.3.22 operator\*=( ) [4/4]

```
DoubleCRT& helib::DoubleCRT::operator*= (
    long num ) [inline]
```

**7.38.3.23 operator++()** [1/2]

```
DoubleCRT& helib::DoubleCRT::operator++ ( ) [inline]
```

**7.38.3.24 operator++()** [2/2]

```
void helib::DoubleCRT::operator++ (
    int ) [inline]
```

**7.38.3.25 operator+=()** [1/4]

```
DoubleCRT& helib::DoubleCRT::operator+= (
    const DoubleCRT & other ) [inline]
```

**7.38.3.26 operator+=()** [2/4]

```
DoubleCRT& helib::DoubleCRT::operator+= (
    const NTL::ZZ & num ) [inline]
```

**7.38.3.27 operator+=()** [3/4]

```
DoubleCRT& helib::DoubleCRT::operator+= (
    const NTL::ZZX & poly ) [inline]
```

**7.38.3.28 operator+=()** [4/4]

```
DoubleCRT& helib::DoubleCRT::operator+= (
    long num ) [inline]
```

**7.38.3.29 operator--()** [1/2]

```
DoubleCRT& helib::DoubleCRT::operator-- ( ) [inline]
```

### 7.38.3.30 operator--() [2/2]

```
void helib::DoubleCRT::operator-- (
    int ) [inline]
```

### 7.38.3.31 operator-=() [1/4]

```
DoubleCRT& helib::DoubleCRT::operator-= (
    const DoubleCRT & other ) [inline]
```

### 7.38.3.32 operator-=() [2/4]

```
DoubleCRT& helib::DoubleCRT::operator-= (
    const NTL::ZZ & num ) [inline]
```

### 7.38.3.33 operator-=() [3/4]

```
DoubleCRT& helib::DoubleCRT::operator-= (
    const NTL::ZZX & poly ) [inline]
```

### 7.38.3.34 operator-=() [4/4]

```
DoubleCRT& helib::DoubleCRT::operator-= (
    long num ) [inline]
```

### 7.38.3.35 operator/=() [1/2]

```
DoubleCRT & helib::DoubleCRT::operator/= (
    const NTL::ZZ & num )
```

### 7.38.3.36 operator/=() [2/2]

```
DoubleCRT& helib::DoubleCRT::operator/= (
    long num ) [inline]
```



**7.38.3.37 operator=()** [1/5]

```
DoubleCRT & helib::DoubleCRT::operator= (
    const DoubleCRT & other )
```

**7.38.3.38 operator=()** [2/5]

```
DoubleCRT& helib::DoubleCRT::operator= (
    const long num ) [inline]
```

**7.38.3.39 operator=()** [3/5]

```
DoubleCRT & helib::DoubleCRT::operator= (
    const NTL::ZZ & num )
```

**7.38.3.40 operator=()** [4/5]

```
DoubleCRT & helib::DoubleCRT::operator= (
    const NTL::ZZX & poly )
```

**7.38.3.41 operator=()** [5/5]

```
DoubleCRT & helib::DoubleCRT::operator= (
    const zzX & poly )
```

**7.38.3.42 operator==( )**

```
bool helib::DoubleCRT::operator== (
    const DoubleCRT & other ) const [inline]
```

**7.38.3.43 operator>>=()**

```
DoubleCRT& helib::DoubleCRT::operator>>= (
    long k ) [inline]
```

#### 7.38.3.44 `randomize()`

```
void helib::DoubleCRT::randomize (
    const NTL::ZZ * seed = nullptr )
```

Fills each row  $i$  with random ints mod  $p_i$ , uses NTL's PRG.

#### 7.38.3.45 `read()`

```
void helib::DoubleCRT::read (
    std::istream & str )
```

#### 7.38.3.46 `reduce()`

```
void helib::DoubleCRT::reduce ( ) const [inline]
```

#### 7.38.3.47 `removePrimes()`

```
void helib::DoubleCRT::removePrimes (
    const IndexSet & s1 ) [inline]
```

Remove  $s1$  from the index set.

#### 7.38.3.48 `sampleGaussian()`

```
double helib::DoubleCRT::sampleGaussian (
    double stdev = 0.0 )
```

Coefficients are Gaussians Return a high probability bound on L-infty norm of canonical embedding.

#### 7.38.3.49 `sampleGaussianBounded()`

```
double helib::DoubleCRT::sampleGaussianBounded (
    double stdev = 0.0 )
```

**7.38.3.50 sampleHWt()**

```
double helib::DoubleCRT::sampleHWt (
    long Hwt )
```

Coefficients are  $-1/0/1$  with pre-specified number of nonzeros.

**7.38.3.51 sampleHWtBounded()**

```
double helib::DoubleCRT::sampleHWtBounded (
    long Hwt )
```

**7.38.3.52 sampleSmall()**

```
double helib::DoubleCRT::sampleSmall ( )
```

Coefficients are  $-1/0/1$ , Prob[0]=1/2.

Sampling routines: Each of these return a high probability bound on L-infty norm of canonical embedding

**7.38.3.53 sampleSmallBounded()**

```
double helib::DoubleCRT::sampleSmallBounded ( )
```

**7.38.3.54 sampleUniform() [1/2]**

```
NTL::xdouble helib::DoubleCRT::sampleUniform (
    const NTL::ZZ & B )
```

**7.38.3.55 sampleUniform() [2/2]**

```
double helib::DoubleCRT::sampleUniform (
    long B )
```

Coefficients are uniform in  $[-B..B]$ .

### 7.38.3.56 scaleDownToSet()

```
void helib::DoubleCRT::scaleDownToSet (
    const IndexSet & s,
    long ptxtSpace,
    NTL::ZZX & delta )
```

### 7.38.3.57 SetOne()

```
DoubleCRT& helib::DoubleCRT::SetOne ( ) [inline]
```

### 7.38.3.58 setPrimes()

```
void helib::DoubleCRT::setPrimes (
    const IndexSet & s1 ) [inline]
```

@ brief make prime set equal to s1

### 7.38.3.59 SetZero()

```
DoubleCRT& helib::DoubleCRT::SetZero ( ) [inline]
```

### 7.38.3.60 Sub()

```
void helib::DoubleCRT::Sub (
    const DoubleCRT & other,
    bool matchIndexSets = true ) [inline]
```

### 7.38.3.61 toPoly() [1/2]

```
void helib::DoubleCRT::toPoly (
    NTL::ZZX & p,
    bool positive = false ) const
```

### 7.38.3.62 toPoly() [2/2]

```
void helib::DoubleCRT::toPoly (
    NTL::ZZX & p,
    const IndexSet & s,
    bool positive = false ) const
```

Recovering the polynomial in coefficient representation. This yields an integer polynomial with coefficients in  $[-P/2, P/2]$ , unless the positive flag is set to true, in which case we get coefficients in  $[0, P-1]$  ( $P$  is the product of all moduli used). Using the optional [IndexSet](#) param we compute the polynomial reduced modulo the product of only the primes in that set.

### 7.38.3.63 write()

```
void helib::DoubleCRT::write (
    std::ostream & str ) const
```

## 7.38.4 Friends And Related Function Documentation

### 7.38.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const DoubleCRT & d ) [friend]
```

### 7.38.4.2 operator>>

```
std::istream& operator>> (
    std::istream & s,
    DoubleCRT & d ) [friend]
```

The documentation for this class was generated from the following files:

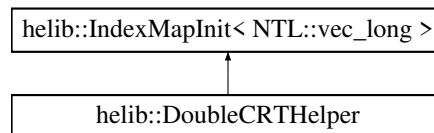
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/DoubleCRT.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/DoubleCRT.cpp](#)

## 7.39 helib::DoubleCRTHelper Class Reference

A helper class to enforce consistency within an [DoubleCRTHelper](#) object.

```
#include <DoubleCRT.h>
```

Inheritance diagram for helib::DoubleCRTHelper:



### Public Member Functions

- [DoubleCRTHelper](#) (const [Context](#) &context)
- virtual void [init](#) (NTL::vec\_long &v)  
*the init method ensures that all rows have the same size*
- virtual [IndexMapInit](#)< NTL::vec\_long > \* [clone](#) () const  
*clone allocates a new object and copies the content*

#### 7.39.1 Detailed Description

A helper class to enforce consistency within an [DoubleCRTHelper](#) object.

See Section 2.6.2 of the design document ([IndexMap](#))

#### 7.39.2 Constructor & Destructor Documentation

##### 7.39.2.1 DoubleCRTHelper()

```
helib::DoubleCRTHelper::DoubleCRTHelper (
    const Context & context )
```

#### 7.39.3 Member Function Documentation

##### 7.39.3.1 clone()

```
virtual IndexMapInit<NTL::vec_long>* helib::DoubleCRTHelper::clone ( ) const [inline], [virtual]
```

clone allocates a new object and copies the content

### 7.39.3.2 init()

```
virtual void helib::DoubleCRTHelper::init (
    NTL::vec_long & v ) [inline], [virtual]
```

the init method ensures that all rows have the same size

Implements [helib::IndexMapInit< NTL::vec\\_long >](#).

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[DoubleCRT.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[DoubleCRT.cpp](#)

## 7.40 helib::DynamicCtxtPowers Class Reference

Store powers of X, compute them dynamically as needed.

```
#include <polyEval.h>
```

### Public Member Functions

- [DynamicCtxtPowers](#) (const [Ctxt](#) &c, long nPowers)
- [Ctxt](#) & [getPower](#) (long e)  
*Returns the e'th power, computing it as needed.*
- [Ctxt](#) & [at](#) (long i)  
*dp.at(i) and dp[i] both return the i+1st power*
- [Ctxt](#) & [operator\[\]](#) (long i)
- const std::vector< [Ctxt](#) > & [getVector](#) () const
- long [size](#) () const
- bool [isPowerComputed](#) (long i)

### 7.40.1 Detailed Description

Store powers of X, compute them dynamically as needed.

### 7.40.2 Constructor & Destructor Documentation

#### 7.40.2.1 DynamicCtxtPowers()

```
helib::DynamicCtxtPowers::DynamicCtxtPowers (
    const Ctxt & c,
    long nPowers ) [inline]
```

### 7.40.3 Member Function Documentation

#### 7.40.3.1 at()

```
Ctxt& helib::DynamicCtxtPowers::at (
    long i ) [inline]
```

dp.at(i) and dp[i] both return the i+1st power

#### 7.40.3.2 getPower()

```
Ctxt & helib::DynamicCtxtPowers::getPower (
    long e )
```

Returns the e'th power, computing it as needed.

#### 7.40.3.3 getVector()

```
const std::vector<Ctxt>& helib::DynamicCtxtPowers::getVector ( ) const [inline]
```

#### 7.40.3.4 isPowerComputed()

```
bool helib::DynamicCtxtPowers::isPowerComputed (
    long i ) [inline]
```

#### 7.40.3.5 operator[]()

```
Ctxt& helib::DynamicCtxtPowers::operator[] (
    long i ) [inline]
```

#### 7.40.3.6 size()

```
long helib::DynamicCtxtPowers::size ( ) const [inline]
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[polyEval.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[polyEval.cpp](#)



## 7.41 helib::encode\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const std::vector< long > &array)
- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const std::vector< NTL::ZZX > &array)

### 7.41.1 Member Function Documentation

#### 7.41.1.1 [apply\(\)](#) [1/2]

```
template<typename type >
static void helib::encode_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const std::vector< long > & array ) [inline], [static]
```

#### 7.41.1.2 [apply\(\)](#) [2/2]

```
template<typename type >
static void helib::encode_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const std::vector< NTL::ZZX > & array ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EncryptedArray.cpp](#)

## 7.42 helib::EncryptedArray Class Reference

A simple wrapper for a smart pointer to an [EncryptedArrayBase](#). This is the interface that higher-level code should use.

```
#include <EncryptedArray.h>
```

## Public Member Functions

- [EncryptedArray](#) (const [Context](#) &context, const NTL::ZZX &G=NTL::ZZX(1, 1))  
*constructor: G defaults to the monomial X, [PAAlgebraMod](#) from context*
- [EncryptedArray](#) (const [Context](#) &context, const [PAAlgebraMod](#) &\_alMod)  
*constructor: G defaults to F0, [PAAlgebraMod](#) explicitly given*
- [EncryptedArray](#) & operator= (const [EncryptedArray](#) &other)
- template<typename type >  
const [EncryptedArrayDerived](#)< type > & [getDerived](#) (type) const  
*downcast operator example: const EncryptedArrayDerived<PA\_GF2> & rep = ea.getDerived(PA\_GF2());*
- const [EncryptedArrayCx](#) & [getCx](#) () const

## Direct access to EncryptedArrayBase methods

- [PA\\_tag](#) [getTag](#) () const
- template<template< typename > class T, typename... Args>  
void [dispatch](#) (Args &&... args) const
- const [Context](#) & [getContext](#) () const
- const [PAAlgebraMod](#) & [getAlMod](#) () const
- const [PAAlgebra](#) & [getPAAlgebra](#) () const
- long [getDegree](#) () const
- void [rotate](#) ([Ctxt](#) &ctxt, long k) const
- void [shift](#) ([Ctxt](#) &ctxt, long k) const
- void [rotate1D](#) ([Ctxt](#) &ctxt, long i, long k, bool dc=false) const
- void [shift1D](#) ([Ctxt](#) &ctxt, long i, long k) const
- void [badDimensionAutomorphCorrection](#) ([Ctxt](#) &ctxt, long i, long amt) const
- template<typename PTXT, typename ARRAY >  
void [encode](#) (PTXT &ptxt, const ARRAY &array) const
- void [encodeUnitSelector](#) ([zzX](#) &ptxt, long i) const
- template<typename PTXT, typename ARRAY >  
void [decode](#) (ARRAY &array, const PTXT &ptxt) const
- template<typename T >  
void [random](#) (std::vector< T > &array) const
- template<typename T >  
void [encrypt](#) ([Ctxt](#) &ctxt, const [PubKey](#) &pKey, const T &ptxt) const
- template<typename T >  
void [decrypt](#) (const [Ctxt](#) &ctxt, const [SecKey](#) &sKey, T &ptxt) const
- void [buildLinPolyCoeffs](#) (std::vector< NTL::ZZX > &C, const std::vector< NTL::ZZX > &L) const
- void [restoreContext](#) () const
- void [restoreContextForG](#) () const
- long [size](#) () const
- long [dimension](#) () const
- long [sizeOfDimension](#) (long i) const
- long [nativeDimension](#) (long i) const
- long [coordinate](#) (long i, long k) const
- long [addCoord](#) (long i, long k, long offset) const
- template<typename U >  
void [rotate1D](#) (std::vector< U > &out, const std::vector< U > &in, long i, long offset) const  
*rotate an array by offset in the i'th dimension (output should not alias input)*

### 7.42.1 Detailed Description

A simple wrapper for a smart pointer to an [EncryptedArrayBase](#). This is the interface that higher-level code should use.

## 7.42.2 Constructor & Destructor Documentation

### 7.42.2.1 EncryptedArray() [1/2]

```
helib::EncryptedArray::EncryptedArray (
    const Context & context,
    const NTL::ZZX & G = NTL::ZZX(1, 1) ) [inline]
```

constructor: G defaults to the monomial X, [PAlgebraMod](#) from context

### 7.42.2.2 EncryptedArray() [2/2]

```
helib::EncryptedArray::EncryptedArray (
    const Context & context,
    const PAlgebraMod & _alMod ) [inline]
```

constructor: G defaults to F0, [PAlgebraMod](#) explicitly given

## 7.42.3 Member Function Documentation

### 7.42.3.1 addCoord()

```
long helib::EncryptedArray::addCoord (
    long i,
    long k,
    long offset ) const [inline]
```

### 7.42.3.2 badDimensionAutomorphCorrection()

```
void helib::EncryptedArray::badDimensionAutomorphCorrection (
    Ctxt & ctxt,
    long i,
    long amt ) const [inline]
```

**7.42.3.3 buildLinPolyCoeffs()**

```
void helib::EncryptedArray::buildLinPolyCoeffs (
    std::vector< NTL::ZZX > & C,
    const std::vector< NTL::ZZX > & L ) const [inline]
```

**7.42.3.4 coordinate()**

```
long helib::EncryptedArray::coordinate (
    long i,
    long k ) const [inline]
```

**7.42.3.5 decode()**

```
template<typename PTXT , typename ARRAY >
void helib::EncryptedArray::decode (
    ARRAY & array,
    const PTXT & ptxt ) const [inline]
```

**7.42.3.6 decrypt()**

```
template<typename T >
void helib::EncryptedArray::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    T & ptxt ) const [inline]
```

**7.42.3.7 dimension()**

```
long helib::EncryptedArray::dimension ( ) const [inline]
```

**7.42.3.8 dispatch()**

```
template<template< typename > class T, typename... Args>
void helib::EncryptedArray::dispatch (
    Args &&... args ) const [inline]
```

### 7.42.3.9 encode()

```
template<typename PTXT , typename ARRAY >
void helib::EncryptedArray::encode (
    PTXT & ptxt,
    const ARRAY & array ) const [inline]
```

### 7.42.3.10 encodeUnitSelector()

```
void helib::EncryptedArray::encodeUnitSelector (
    zzX & ptxt,
    long i ) const [inline]
```

### 7.42.3.11 encrypt()

```
template<typename T >
void helib::EncryptedArray::encrypt (
    Ctxt & ctxt,
    const PubKey & pKey,
    const T & ptxt ) const [inline]
```

### 7.42.3.12 getAlMod()

```
const PAlgebraMod& helib::EncryptedArray::getAlMod ( ) const [inline]
```

### 7.42.3.13 getContext()

```
const Context& helib::EncryptedArray::getContext ( ) const [inline]
```

### 7.42.3.14 getCx()

```
const EncryptedArrayCx& helib::EncryptedArray::getCx ( ) const [inline]
```

#### 7.42.3.15 getDegree()

```
long helib::EncryptedArray::getDegree ( ) const [inline]
```

#### 7.42.3.16 getDerived()

```
template<typename type >
const EncryptedArrayDerived<type>& helib::EncryptedArray::getDerived (
    type ) const [inline]
```

downcast operator example: `const EncryptedArrayDerived<PA_GF2>& rep = ea.getDerived(PA_GF2());`

#### 7.42.3.17 getPAlgebra()

```
const PAlgebra& helib::EncryptedArray::getPAlgebra ( ) const [inline]
```

#### 7.42.3.18 getTag()

```
PA_tag helib::EncryptedArray::getTag ( ) const [inline]
```

#### 7.42.3.19 nativeDimension()

```
long helib::EncryptedArray::nativeDimension (
    long i ) const [inline]
```

#### 7.42.3.20 operator=()

```
EncryptedArray& helib::EncryptedArray::operator= (
    const EncryptedArray & other ) [inline]
```

#### 7.42.3.21 random()

```
template<typename T >
void helib::EncryptedArray::random (
    std::vector< T > & array ) const [inline]
```

### 7.42.3.22 restoreContext()

```
void helib::EncryptedArray::restoreContext ( ) const [inline]
```

### 7.42.3.23 restoreContextForG()

```
void helib::EncryptedArray::restoreContextForG ( ) const [inline]
```

### 7.42.3.24 rotate()

```
void helib::EncryptedArray::rotate (
    Ctxt & ctxt,
    long k ) const [inline]
```

### 7.42.3.25 rotate1D() [1/2]

```
void helib::EncryptedArray::rotate1D (
    Ctxt & ctxt,
    long i,
    long k,
    bool dc = false ) const [inline]
```

### 7.42.3.26 rotate1D() [2/2]

```
template<typename U >
void helib::EncryptedArray::rotate1D (
    std::vector< U > & out,
    const std::vector< U > & in,
    long i,
    long offset ) const [inline]
```

rotate an array by offset in the i'th dimension (output should not alias input)

### 7.42.3.27 shift()

```
void helib::EncryptedArray::shift (
    Ctxt & ctxt,
    long k ) const [inline]
```

**7.42.3.28 shift1D()**

```
void helib::EncryptedArray::shift1D (
    Ctxt & ctxt,
    long i,
    long k ) const [inline]
```

**7.42.3.29 size()**

```
long helib::EncryptedArray::size ( ) const [inline]
```

**7.42.3.30 sizeOfDimension()**

```
long helib::EncryptedArray::sizeOfDimension (
    long i ) const [inline]
```

The documentation for this class was generated from the following file:

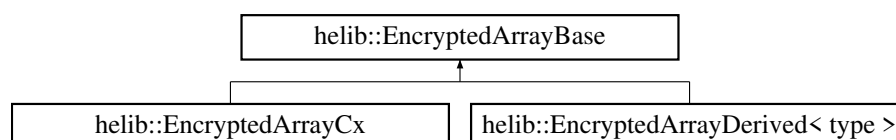
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EncryptedArray.h](#)

**7.43 helib::EncryptedArrayBase Class Reference**

virtual class for data-movement operations on arrays of slots

```
#include <EncryptedArray.h>
```

Inheritance diagram for helib::EncryptedArrayBase:





## Public Member Functions

- virtual [~EncryptedArrayBase](#) ()
- virtual [EncryptedArrayBase \\* clone](#) () const =0
- virtual [PA\\_tag](#) [getTag](#) () const =0
- virtual const [Context](#) & [getContext](#) () const =0
- virtual const [PAlgebra](#) & [getPAlgebra](#) () const =0
- virtual long [getDegree](#) () const =0
- virtual long [getP2R](#) () const =0
- virtual void [rotate](#) ([Ctxt](#) &ctxt, long k) const =0
 

*Right rotation as a linear array. E.g., rotating ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(n 1 2 ... n-1)*
- virtual void [shift](#) ([Ctxt](#) &ctxt, long k) const =0
 

*Non-cyclic right shift with zero fill E.g., shifting ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(0 1 2... n-1)*
- virtual void [rotate1D](#) ([Ctxt](#) &ctxt, long i, long k, bool dc=false) const =0
 

*right-rotate k positions along the i'th dimension*
- virtual void [shift1D](#) ([Ctxt](#) &ctxt, long i, long k) const =0
 

*Right shift k positions along the i'th dimension with zero fill.*
- virtual void [badDimensionAutomorphCorrection](#) ([Ctxt](#) &ctxt, long i, long amt) const =0
 

*Correct an automorphism in a bad dimension.*
- virtual void [buildLinPolyCoeffs](#) (std::vector< [NTL::ZZX](#) > &C, const std::vector< [NTL::ZZX](#) > &L) const =0
 

*Linearized polynomials. L describes a linear map M by describing its action on the standard power basis:  $M(x^j \bmod G) = (L[j] \bmod G)$ , for  $j = 0..d-1$ . The result is a coefficient std::vector C for the linearized polynomial representing M: a polynomial h in  $\mathbb{Z}/(p^r)[X]$  of degree < d is sent to.*
- virtual void [restoreContext](#) () const
- virtual void [restoreContextForG](#) () const
- long [size](#) () const
 

*Total size (# of slots) of hypercube.*
- long [dimension](#) () const
 

*Number of dimensions of hypercube.*
- long [sizeOfDimension](#) (long i) const
 

*Size of given dimension.*
- bool [nativeDimension](#) (long i) const
 

*Is rotations in given dimension a "native" operation?*
- long [coordinate](#) (long i, long k) const
 

*returns coordinate of index k along the i'th dimension*
- long [addCoord](#) (long i, long k, long offset) const
 

*adds offset to index k in the i'th dimension*
- template<typename U >
  - void [rotate1D](#) (std::vector< U > &out, const std::vector< U > &in, long i, long offset) const
 

*rotate an array by offset in the i'th dimension (output should not alias input)*

## Encoding/decoding methods

- virtual void [encode](#) ([zzX](#) &ptxt, const std::vector< long > &array) const =0
- virtual void [encode](#) ([NTL::ZZX](#) &ptxt, const std::vector< long > &array) const =0
- virtual void [encode](#) ([zzX](#) &ptxt, const std::vector< [zzX](#) > &array) const =0
- virtual void [encode](#) ([zzX](#) &ptxt, const [PlaintextArray](#) &array) const =0
- virtual void [encode](#) ([NTL::ZZX](#) &ptxt, const std::vector< [NTL::ZZX](#) > &array) const =0
- virtual void [encode](#) ([NTL::ZZX](#) &ptxt, const [PlaintextArray](#) &array) const =0
- void [encode](#) ([zzX](#) &ptxt, const std::vector< [NTL::ZZX](#) > &array) const
- virtual void [decode](#) (std::vector< long > &array, const [NTL::ZZX](#) &ptxt) const =0
- virtual void [decode](#) (std::vector< [NTL::ZZX](#) > &array, const [NTL::ZZX](#) &ptxt) const =0
- virtual void [decode](#) ([PlaintextArray](#) &array, const [NTL::ZZX](#) &ptxt) const =0
- virtual void [random](#) (std::vector< long > &array) const =0
- virtual void [random](#) (std::vector< [NTL::ZZX](#) > &array) const =0

- long [decode1Slot](#) (const NTL::ZZX &ptxt, long i) const
- void [decode1Slot](#) (NTL::ZZX &slot, const NTL::ZZX &ptxt, long i) const
- virtual void [encodeUnitSelector](#) (zzX &ptxt, long i) const =0

*Encodes a std::vector with 1 at position i and 0 everywhere else.*

### Encoding+encryption/decryption+decoding

- template<typename PTXT >  
void [encrypt](#) (Ctxt &ctxt, const [PubKey](#) &key, const PTXT &ptxt) const
- virtual void [decrypt](#) (const Ctxt &ctxt, const [SecKey](#) &sKey, std::vector< long > &ptxt) const =0
- virtual void [decrypt](#) (const Ctxt &ctxt, const [SecKey](#) &sKey, std::vector< NTL::ZZX > &ptxt) const =0
- virtual void [decrypt](#) (const Ctxt &ctxt, const [SecKey](#) &sKey, [PlaintextArray](#) &ptxt) const =0
- virtual void [decrypt](#) (const Ctxt &ctxt, const [SecKey](#) &sKey, std::vector< double > &ptxt) const =0
- virtual void [decrypt](#) (const Ctxt &ctxt, const [SecKey](#) &sKey, std::vector< [cx\\_double](#) > &ptxt) const =0
- long [decrypt1Slot](#) (const Ctxt &ctxt, const [SecKey](#) &sKey, long i) const
- void [decrypt1Slot](#) (NTL::ZZX &slot, const Ctxt &ctxt, const [SecKey](#) &sKey, long i) const

## 7.43.1 Detailed Description

virtual class for data-movement operations on arrays of slots

An object [ea](#) of type [EncryptedArray](#) stores information about an [Context](#) context, and a monic polynomial  $G$ . If context defines parameters  $m$ ,  $p$ , and  $r$ , then [ea](#) is a helper object that supports encoding/decoding and encryption/decryption of std::vectors of plaintext slots over the ring  $(\mathbb{Z}/(p^r)[X])/(G)$ .

The polynomial  $G$  should be irreducible over  $\mathbb{Z}/(p^r)$  (this is not checked). The degree of  $G$  should divide the multiplicative order of  $p$  modulo  $m$  (this is checked). Currently, the following restriction is imposed:

either  $r == 1$  or  $\deg(G) == 1$  or  $G == \text{factors}[0]$ .

[ea](#) stores objects in the polynomial ring  $\mathbb{Z}/(p^r)[X]$ .

Just as for the class [PAlgebraMod](#), if  $p == 2$  and  $r == 1$ , then these polynomials are represented as GF2X's, and otherwise as zz\_pX's. Thus, the types of these objects are not determined until run time. As such, we need to use a class hierarchy, which mirrors that of [PAlgebraMod](#), as follows.

[EncryptedArrayBase](#) is a virtual class

[EncryptedArrayDerived](#)<type> is a derived template class, where type is either PA\_GF2 or PA\_zz\_p.

The class [EncryptedArray](#) is a simple wrapper around a smart pointer to an [EncryptedArrayBase](#) object: copying an [EncryptedArray](#) object results in a "deep copy" of the underlying object of the derived class.

## 7.43.2 Constructor & Destructor Documentation

### 7.43.2.1 ~EncryptedArrayBase()

```
virtual helib::EncryptedArrayBase::~~EncryptedArrayBase ( ) [inline], [virtual]
```

### 7.43.3 Member Function Documentation

#### 7.43.3.1 addCoord()

```
long helib::EncryptedArrayBase::addCoord (
    long i,
    long k,
    long offset ) const [inline]
```

adds offset to index k in the i'th dimension

#### 7.43.3.2 badDimensionAutomorphCorrection()

```
virtual void helib::EncryptedArrayBase::badDimensionAutomorphCorrection (
    Ctxt & ctxt,
    long i,
    long amt ) const [pure virtual]
```

Correct an automorphism in a bad dimension.

##### Parameters

<i>ctxt</i>	Ctxt to perform the correction on.
<i>i</i>	Dimension of which to correct.
<i>amt</i>	Exponent of the auto- morphism.

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

#### 7.43.3.3 buildLinPolyCoeffs()

```
virtual void helib::EncryptedArrayBase::buildLinPolyCoeffs (
    std::vector< NTL::ZZX > & C,
    const std::vector< NTL::ZZX > & L ) const [pure virtual]
```

Linearized polynomials. L describes a linear map M by describing its action on the standard power basis:  $M(x^j \bmod G) = (L[j] \bmod G)$ , for  $j = 0..d-1$ . The result is a coefficient `std::vector C` for the linearized polynomial representing M: a polynomial  $h$  in  $\mathbb{Z}/(p^r)[X]$  of degree  $< d$  is sent to.

$$//!M(h(X) \bmod G) = \sum_{i=0}^{d-1} (C[i] \cdot h(X^{p^i})) \bmod G.//!$$

Implemented in [helib::EncryptedArrayDerived< type >](#).

#### 7.43.3.4 clone()

```
virtual EncryptedArrayBase* helib::EncryptedArrayBase::clone ( ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

#### 7.43.3.5 coordinate()

```
long helib::EncryptedArrayBase::coordinate (
    long i,
    long k ) const [inline]
```

returns coordinate of index k along the i'th dimension

#### 7.43.3.6 decode() [1/3]

```
virtual void helib::EncryptedArrayBase::decode (
    PlaintextArray & array,
    const NTL::ZZX & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

#### 7.43.3.7 decode() [2/3]

```
virtual void helib::EncryptedArrayBase::decode (
    std::vector< long > & array,
    const NTL::ZZX & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.8 decode()** [3/3]

```
virtual void helib::EncryptedArrayBase::decode (
    std::vector< NTL::ZZX > & array,
    const NTL::ZZX & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.9 decode1Slot()** [1/2]

```
long helib::EncryptedArrayBase::decode1Slot (
    const NTL::ZZX & ptxt,
    long i ) const [inline]
```

**7.43.3.10 decode1Slot()** [2/2]

```
void helib::EncryptedArrayBase::decode1Slot (
    NTL::ZZX & slot,
    const NTL::ZZX & ptxt,
    long i ) const [inline]
```

**7.43.3.11 decrypt()** [1/5]

```
virtual void helib::EncryptedArrayBase::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    PlaintextArray & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.12 decrypt()** [2/5]

```
virtual void helib::EncryptedArrayBase::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< cx_double > & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#).

**7.43.3.13 decrypt()** [3/5]

```
virtual void helib::EncryptedArrayBase::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< double > & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#).

**7.43.3.14 decrypt()** [4/5]

```
virtual void helib::EncryptedArrayBase::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< long > & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.15 decrypt()** [5/5]

```
virtual void helib::EncryptedArrayBase::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< NTL::ZZX > & ptxt ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.16 decrypt1Slot()** [1/2]

```
long helib::EncryptedArrayBase::decrypt1Slot (
    const Ctxt & ctxt,
    const SecKey & sKey,
    long i ) const [inline]
```

**7.43.3.17 decrypt1Slot()** [2/2]

```
void helib::EncryptedArrayBase::decrypt1Slot (
    NTL::ZZX & slot,
    const Ctxt & ctxt,
    const SecKey & sKey,
    long i ) const [inline]
```

### 7.43.3.18 dimension()

```
long helib::EncryptedArrayBase::dimension ( ) const [inline]
```

Number of dimensions of hypercube.

### 7.43.3.19 encode() [1/7]

```
virtual void helib::EncryptedArrayBase::encode (
    NTL::ZZX & ptxt,
    const PlaintextArray & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

### 7.43.3.20 encode() [2/7]

```
virtual void helib::EncryptedArrayBase::encode (
    NTL::ZZX & ptxt,
    const std::vector< long > & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

### 7.43.3.21 encode() [3/7]

```
virtual void helib::EncryptedArrayBase::encode (
    NTL::ZZX & ptxt,
    const std::vector< NTL::ZZX > & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

### 7.43.3.22 encode() [4/7]

```
virtual void helib::EncryptedArrayBase::encode (
    zzX & ptxt,
    const PlaintextArray & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.23 encode()** [5/7]

```
virtual void helib::EncryptedArrayBase::encode (
    zzX & ptxt,
    const std::vector< long > & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.24 encode()** [6/7]

```
void helib::EncryptedArrayBase::encode (
    zzX & ptxt,
    const std::vector< NTL::ZZX > & array ) const [inline]
```

**7.43.3.25 encode()** [7/7]

```
virtual void helib::EncryptedArrayBase::encode (
    zzX & ptxt,
    const std::vector< zzX > & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.26 encodeUnitSelector()**

```
virtual void helib::EncryptedArrayBase::encodeUnitSelector (
    zzX & ptxt,
    long i ) const [pure virtual]
```

Encodes a std::vector with 1 at position i and 0 everywhere else.

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.27 encrypt()**

```
template<typename PTXT >
void helib::EncryptedArrayBase::encrypt (
    Ctxt & ctxt,
    const PubKey & key,
    const PTXT & ptxt ) const [inline]
```



**7.43.3.28 getContext()**

```
virtual const Context& helib::EncryptedArrayBase::getContext ( ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.29 getDegree()**

```
virtual long helib::EncryptedArrayBase::getDegree ( ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.30 getP2R()**

```
virtual long helib::EncryptedArrayBase::getP2R ( ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.31 getPAlgebra()**

```
virtual const PAlgebra& helib::EncryptedArrayBase::getPAlgebra ( ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.32 getTag()**

```
virtual PA_tag helib::EncryptedArrayBase::getTag ( ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.33 nativeDimension()**

```
bool helib::EncryptedArrayBase::nativeDimension (
    long i ) const [inline]
```

Is rotations in given dimension a "native" operation?

**7.43.3.34 random()** [1/2]

```
virtual void helib::EncryptedArrayBase::random (
    std::vector< long > & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.35 random()** [2/2]

```
virtual void helib::EncryptedArrayBase::random (
    std::vector< NTL::ZZX > & array ) const [pure virtual]
```

Implemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.36 restoreContext()**

```
virtual void helib::EncryptedArrayBase::restoreContext ( ) const [inline], [virtual]
```

Reimplemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.37 restoreContextForG()**

```
virtual void helib::EncryptedArrayBase::restoreContextForG ( ) const [inline], [virtual]
```

Reimplemented in [helib::EncryptedArrayDerived< type >](#).

**7.43.3.38 rotate()**

```
virtual void helib::EncryptedArrayBase::rotate (
    Ctxt & ctxt,
    long k ) const [pure virtual]
```

Right rotation as a linear array. E.g., rotating ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(n 1 2 ... n-1)

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.39 rotate1D()** [1/2]

```
virtual void helib::EncryptedArrayBase::rotate1D (
    Ctxt & ctxt,
    long i,
    long k,
    bool dc = false ) const [pure virtual]
```

right-rotate k positions along the i'th dimension

## Parameters

<i>dc</i>	means "don't care", which means that the caller guarantees that only zero elements rotate off the end – this allows for some optimizations that would not otherwise be possible
-----------	---

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

### 7.43.3.40 rotate1D() [2/2]

```
template<typename U >
void helib::EncryptedArrayBase::rotate1D (
    std::vector< U > & out,
    const std::vector< U > & in,
    long i,
    long offset ) const [inline]
```

rotate an array by offset in the i'th dimension (output should not alias input)

**7.43.3.41 shift()**

```
virtual void helib::EncryptedArrayBase::shift (
    Ctxt & ctxt,
    long k ) const [pure virtual]
```

Non-cyclic right shift with zero fill E.g., shifting  $\text{ctxt} = \text{Enc}(1\ 2\ 3 \dots n)$  by  $k=1$  gives  $\text{Enc}(0\ 1\ 2 \dots n-1)$

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.42 shift1D()**

```
virtual void helib::EncryptedArrayBase::shift1D (
    Ctxt & ctxt,
    long i,
    long k ) const [pure virtual]
```

Right shift  $k$  positions along the  $i$ 'th dimension with zero fill.

Implemented in [helib::EncryptedArrayCx](#), and [helib::EncryptedArrayDerived< type >](#).

**7.43.3.43 size()**

```
long helib::EncryptedArrayBase::size ( ) const [inline]
```

Total size (# of slots) of hypercube.

**7.43.3.44 sizeOfDimension()**

```
long helib::EncryptedArrayBase::sizeOfDimension (
    long i ) const [inline]
```

Size of given dimension.

The documentation for this class was generated from the following file:

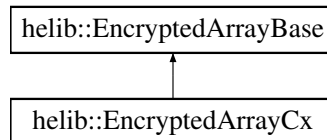
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/EncryptedArray.h](#)

## 7.44 helib::EncryptedArrayCx Class Reference

A different derived class to be used for the approximate-numbers scheme.

```
#include <EncryptedArray.h>
```

Inheritance diagram for helib::EncryptedArrayCx:



### Public Member Functions

- double [encodei](#) ([zzX](#) &ptxt, long precision=-1) const
- [EncryptedArrayCx](#) (const [Context](#) &\_context)
- [EncryptedArrayCx](#) (const [Context](#) &\_context, const [PAlgebraModCx](#) &\_alMod)
- [EncryptedArrayBase](#) \* [clone](#) () const override
- const [zzX](#) & [getiEncoded](#) () const
- [PA\\_tag](#) [getTag](#) () const override
- const [Context](#) & [getContext](#) () const override
- const [PAlgebra](#) & [getPAlgebra](#) () const override
- long [getDegree](#) () const override
- void [rotate](#) ([Ctxt](#) &ctxt, long k) const override
 

*Right rotation as a linear array. E.g., rotating ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(n 1 2 ... n-1)*
- void [shift](#) ([Ctxt](#) &ctxt, long k) const override
 

*Non-cyclic right shift with zero fill E.g., shifting ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(0 1 2... n-1)*
- void [rotate1D](#) ([Ctxt](#) &ctxt, long i, long k, bool dc=false) const override
 

*right-rotate k positions along the i'th dimension*
- void [shift1D](#) ([Ctxt](#) &ctxt, long i, long k) const override
 

*Right shift k positions along the i'th dimension with zero fill.*
- long [getP2R](#) () const override
- void [encode](#) ([UNUSED](#) [zzX](#) &ptxt, [UNUSED](#) const std::vector< long > &array) const override
 

*Unimplemented encode function for BGV. It will always throw helib::LogicError.*
- void [encode](#) ([UNUSED](#) [NTL::ZZX](#) &ptxt, [UNUSED](#) const std::vector< long > &array) const override
 

*Unimplemented encode function for BGV. It will always throw helib::LogicError.*
- void [encode](#) ([UNUSED](#) [zzX](#) &ptxt, [UNUSED](#) const std::vector< [zzX](#) > &array) const override
 

*Unimplemented encode function for BGV. It will always throw helib::LogicError.*
- void [encode](#) ([UNUSED](#) [zzX](#) &ptxt, [UNUSED](#) const [PlaintextArray](#) &array) const override
 

*Unimplemented encode function for BGV. It will always throw helib::LogicError.*
- void [encode](#) ([UNUSED](#) [NTL::ZZX](#) &ptxt, [UNUSED](#) const std::vector< [NTL::ZZX](#) > &array) const override
 

*Unimplemented encode function for BGV. It will always throw helib::LogicError.*
- void [encode](#) ([UNUSED](#) [NTL::ZZX](#) &ptxt, [UNUSED](#) const [PlaintextArray](#) &array) const override
 

*Unimplemented encode function for BGV. It will always throw helib::LogicError.*
- void [decode](#) ([UNUSED](#) std::vector< long > &array, [UNUSED](#) const [NTL::ZZX](#) &ptxt) const override
 

*Unimplemented decode function for BGV. It will always throw helib::LogicError.*
- void [decode](#) ([UNUSED](#) std::vector< [NTL::ZZX](#) > &array, [UNUSED](#) const [NTL::ZZX](#) &ptxt) const override
 

*Unimplemented decode function for BGV. It will always throw helib::LogicError.*
- void [decode](#) ([UNUSED](#) [PlaintextArray](#) &array, [UNUSED](#) const [NTL::ZZX](#) &ptxt) const override

- Unimplemented decode function for BGV. It will always throw [helib::LogicError](#).*

  - void [random](#) ([UNUSED](#) std::vector< NTL::ZZX > &array) const override
- Unimplemented random function for BGV. It will always throw [helib::LogicError](#).*

  - void [decrypt](#) ([UNUSED](#) const Ctxt &ctxt, [UNUSED](#) const SecKey &sKey, [UNUSED](#) std::vector< long > &ptxt) const override
- Unimplemented decrypt function for BGV. It will always throw [helib::LogicError](#).*

  - void [decrypt](#) ([UNUSED](#) const Ctxt &ctxt, [UNUSED](#) const SecKey &sKey, [UNUSED](#) std::vector< NTL::ZZX > &ptxt) const override
- Unimplemented decrypt function for BGV. It will always throw [helib::LogicError](#).*

  - void [decrypt](#) ([UNUSED](#) const Ctxt &ctxt, [UNUSED](#) const SecKey &sKey, [UNUSED](#) PlaintextArray &ptxt) const override
- Unimplemented decrypt function for BGV. It will always throw [helib::LogicError](#).*

  - void [buildLinPolyCoeffs](#) ([UNUSED](#) std::vector< NTL::ZZX > &C, [UNUSED](#) const std::vector< NTL::ZZX > &L) const override
- Unimplemented buildLinPolyCoeffs function for BGV. It will always throw [helib::LogicError](#).*

  - double [encode](#) (zzX &ptxt, const std::vector< [cx\\_double](#) > &array, double useThisSize, long precision=-1) const
  - double [encode](#) (zzX &ptxt, const std::vector< double > &array, double useThisSize, long precision=-1) const
  - double [encode](#) (zzX &ptxt, const std::vector< long > &array, double useThisSize, long precision=-1) const
  - template<typename Scheme >  
double [encode](#) (zzX &out, const Ptxt< Scheme > &ptxt, double useThisSize, long precision=-1) const

*Encode a Ptxt object into a zzX.*

  - double [encode](#) (zzX &ptxt, double aSingleNumber, double useThisSize=-1, long precision=-1) const
  - template<typename PTXT >  
double [encode](#) (NTL::ZZX &ptxt, const PTXT &pt, double useThisSize=-1, long precision=-1) const
  - void [encryptOneNum](#) (Ctxt &ctxt, const PubKey &key, double num, double useThisSize=-1, long precision=-1) const
  - template<typename PTXT >  
void [encrypt](#) (Ctxt &ctxt, const PubKey &key, const PTXT &ptxt, double useThisSize, long precision=-1) const
  - template<typename PTXT >  
void [encrypt](#) (Ctxt &ctxt, const PubKey &key, const PTXT &ptxt) const
  - void [encodeUnitSelector](#) (zzX &ptxt, long i) const override

*Encodes a std::vector with 1 at position i and 0 everywhere else.*

  - double [encodeRoundingError](#) () const
  - long [encodeScalingFactor](#) (long precision=-1, double roundErr=-1.0) const
  - void [decode](#) (std::vector< [cx\\_double](#) > &array, const zzX &ptxt, double scaling) const
  - void [decode](#) (std::vector< [cx\\_double](#) > &array, const NTL::ZZX &ptxt, double scaling) const
  - void [decode](#) (std::vector< double > &array, const zzX &ptxt, double scaling) const
  - void [decode](#) (std::vector< double > &array, const NTL::ZZX &ptxt, double scaling) const
  - void [random](#) (std::vector< [cx\\_double](#) > &array, double rad=1.0) const
  - void [random](#) (std::vector< double > &array, double rad=1.0) const
  - void [random](#) (std::vector< long > &array) const override
  - void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, std::vector< [cx\\_double](#) > &ptxt) const override
  - void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, std::vector< double > &ptxt) const override
  - template<typename Scheme >  
void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, Ptxt< Scheme > &ptxt) const

*Decrypt ciphertext to a plaintext relative to a specific scheme.*

  - void [extractRealPart](#) (Ctxt &c) const
  - template<typename Scheme >  
void [extractRealPart](#) (Ptxt< Scheme > &p) const

*Extract the real part of a CKKS plaintext.*

  - template<typename Scheme >  
void [extractImPart](#) (Ptxt< Scheme > &p) const

*Extract the imaginary part of a CKKS plaintext.*

- void `extractImPart` (Ctxt &c, DoubleCRT \*dcrt=nullptr) const
- void `badDimensionAutomorphCorrection` (Ctxt &ctxt, long i, long k) const override  
*Correct an automorphism in a bad dimension.*

### Linearized polynomials for EncryptedArrayCx

`buildLinPolyCoeffs` returns in  $C$  two encoded constants such that the linear transformation(s) defined as  $L(1) = \text{oneImage}$  and  $L(i)=\text{iImage}$  can be computed as:  $L(x) = C[0]*x + C[1]*\text{conjugate}(x)$ . Once  $C$  is computed, we can apply this  $L$  to a ciphertext by calling `applyLinPolyLL(ctxt, C, 2)`. Alternatively, we can convert  $C$  to a vector of two `DoubleCRT` objects, then call `applyLinPolyLL(ctxt, dcrtVec, 2)`. This lets us compute the `DoubleCRT` object just once, then use them many times.

- void `buildLinPolyCoeffs` (std::vector< zzX > &C, const cx\_double &oneImage, const cx\_double &iImage, long precision=0) const  
*First variant: same linear transformation in all the slots.*
- void `buildLinPolyCoeffs` (std::vector< zzX > &C, const std::vector< cx\_double > &oneImages, const std::vector< cx\_double > &iImages, long precision=0) const  
*Second variant: different linear transformation in each slots.*

## Static Public Member Functions

- static double `roundedSize` (double x)
- static void `convert` (std::vector< cx\_double > &out, const std::vector< double > &in)
- static void `convert` (std::vector< double > &out, const std::vector< cx\_double > &in)
- static void `convert` (std::vector< cx\_double > &out, const std::vector< long > &in)
- static void `convert` (std::vector< long > &out, const std::vector< cx\_double > &in)

### 7.44.1 Detailed Description

A different derived class to be used for the approximate-numbers scheme.

### 7.44.2 Constructor & Destructor Documentation

#### 7.44.2.1 EncryptedArrayCx() [1/2]

```
helib::EncryptedArrayCx::EncryptedArrayCx (
    const Context & _context ) [inline], [explicit]
```

#### 7.44.2.2 EncryptedArrayCx() [2/2]

```
helib::EncryptedArrayCx::EncryptedArrayCx (
    const Context & _context,
    const PAlgebraModCx & _alMod ) [inline]
```

### 7.44.3 Member Function Documentation

#### 7.44.3.1 badDimensionAutomorphCorrection()

```
void helib::EncryptedArrayCx::badDimensionAutomorphCorrection (
    Ctxt & ctxt,
    long i,
    long amt ) const [override], [virtual]
```

Correct an automorphism in a bad dimension.

##### Parameters

<i>ctxt</i>	Ctxt to perform the correction on.
<i>i</i>	Dimension of which to correct.
<i>amt</i>	Exponent of the automorphism.

Implements [helib::EncryptedArrayBase](#).

#### 7.44.3.2 buildLinPolyCoeffs() [1/3]

```
void helib::EncryptedArrayCx::buildLinPolyCoeffs (
    std::vector< zzX > & C,
    const cx_double & oneImage,
    const cx_double & iImage,
    long precision = 0 ) const
```

First variant: same linear transformation in all the slots.



**7.44.3.3 buildLinPolyCoeffs() [2/3]**

```
void helib::EncryptedArrayCx::buildLinPolyCoeffs (
    std::vector< zzX > & C,
    const std::vector< cx_double > & oneImages,
    const std::vector< cx_double > & iImages,
    long precision = 0 ) const
```

Second variant: different linear transformation in each slots.

**7.44.3.4 buildLinPolyCoeffs() [3/3]**

```
void helib::EncryptedArrayCx::buildLinPolyCoeffs (
    UNUSED std::vector< NTL::ZZX > & C,
    UNUSED const std::vector< NTL::ZZX > & L ) const [inline], [override]
```

Unimplemented buildLinPolyCoeffs function for BGV. It will always throw [helib::LogicError](#).

**Parameters**

<i>C</i>	Unused.
<i>L</i>	Unused.

**7.44.3.5 clone()**

```
EncryptedArrayBase* helib::EncryptedArrayCx::clone ( ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.44.3.6 convert() [1/4]**

```
static void helib::EncryptedArrayCx::convert (
    std::vector< cx_double > & out,
    const std::vector< double > & in ) [inline], [static]
```

**7.44.3.7 convert() [2/4]**

```
static void helib::EncryptedArrayCx::convert (
    std::vector< cx_double > & out,
    const std::vector< long > & in ) [inline], [static]
```

**7.44.3.8 convert() [3/4]**

```
static void helib::EncryptedArrayCx::convert (
    std::vector< double > & out,
    const std::vector< cx_double > & in ) [inline], [static]
```

**7.44.3.9 convert() [4/4]**

```
static void helib::EncryptedArrayCx::convert (
    std::vector< long > & out,
    const std::vector< cx_double > & in ) [inline], [static]
```

**7.44.3.10 decode() [1/7]**

```
void helib::EncryptedArrayCx::decode (
    std::vector< cx_double > & array,
    const NTL::ZZX & ptxt,
    double scaling ) const [inline]
```

**7.44.3.11 decode() [2/7]**

```
void helib::EncryptedArrayCx::decode (
    std::vector< cx_double > & array,
    const zzX & ptxt,
    double scaling ) const
```

**7.44.3.12 decode() [3/7]**

```
void helib::EncryptedArrayCx::decode (
    std::vector< double > & array,
    const NTL::ZZX & ptxt,
    double scaling ) const [inline]
```

**7.44.3.13 decode() [4/7]**

```
void helib::EncryptedArrayCx::decode (
    std::vector< double > & array,
    const zzX & ptxt,
    double scaling ) const [inline]
```

**7.44.3.14 decode() [5/7]**

```
void helib::EncryptedArrayCx::decode (
    UNUSED PlaintextArray & array,
    UNUSED const NTL::ZZX & ptxt ) const [inline], [override]
```

Unimplemented decode function for BGV. It will always throw [helib::LogicError](#).

## Parameters

<i>array</i>	Unused.
<i>ptxt</i>	Unused.

**7.44.3.15 decode()** [6/7]

```
void helib::EncryptedArrayCx::decode (
    UNUSED std::vector< long > & array,
    UNUSED const NTL::ZZX & ptxt ) const [inline], [override]
```

Unimplemented decode function for [BGV](#). It will always throw [helib::LogicError](#).

## Parameters

<i>array</i>	Unused.
<i>ptxt</i>	Unused.

**7.44.3.16 decode()** [7/7]

```
void helib::EncryptedArrayCx::decode (
    UNUSED std::vector< NTL::ZZX > & array,
    UNUSED const NTL::ZZX & ptxt ) const [inline], [override]
```

Unimplemented decode function for [BGV](#). It will always throw [helib::LogicError](#).

## Parameters

<i>array</i>	Unused.
<i>ptxt</i>	Unused.

**7.44.3.17 decrypt()** [1/6]

```
template<typename Scheme >
void helib::EncryptedArrayCx::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    Ptxt< Scheme > & ptxt ) const [inline]
```

Decrypt ciphertext to a plaintext relative to a specific scheme.

## Template Parameters

<i>Scheme</i>	Encryption scheme to be used (either <a href="#">BGV</a> or <a href="#">CKKS</a> ).
---------------	---

## Parameters

<i>ctxt</i>	Ciphertext to de-crypt.
<i>sKey</i>	Secret key to be used for de-cryption.
<i>ptxt</i>	Plaintext into which to de-crypt. Decrypt a <a href="#">Ctxt</a> cipher-text object to a <a href="#">Ptxt</a> plain-text one relative to a specific scheme.

**7.44.3.18 decrypt()** [2/6]

```
void helib::EncryptedArrayCx::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< cx\_double > & ptxt ) const [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.44.3.19 decrypt()** [3/6]

```
void helib::EncryptedArrayCx::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< double > & ptxt ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.44.3.20 decrypt()** [4/6]

```
void helib::EncryptedArrayCx::decrypt (
    UNUSED const Ctxt & ctxt,
    UNUSED const SecKey & sKey,
    UNUSED PlaintextArray & ptxt ) const [inline], [override]
```

Unimplemented decrypt function for [BGV](#). It will always throw [helib::LogicError](#).

**Parameters**

<i>ctxt</i>	Unused.
<i>sKey</i>	Unused.
<i>ptxt</i>	Unused.

**7.44.3.21 decrypt()** [5/6]

```
void helib::EncryptedArrayCx::decrypt (
    UNUSED const Ctxt & ctxt,
    UNUSED const SecKey & sKey,
    UNUSED std::vector< long > & ptxt ) const [inline], [override]
```

Unimplemented decrypt function for [BGV](#). It will always throw [helib::LogicError](#).

**Parameters**

<i>ctxt</i>	Unused.
<i>sKey</i>	Unused.
<i>ptxt</i>	Unused.

**7.44.3.22 decrypt()** [6/6]

```
void helib::EncryptedArrayCx::decrypt (
    UNUSED const Ctxt & ctxt,
    UNUSED const SecKey & sKey,
    UNUSED std::vector< NTL::ZZX > & ptxt ) const [inline], [override]
```

Unimplemented decrypt function for [BGV](#). It will always throw [helib::LogicError](#).

**Parameters**

<i>ctxt</i>	Unused.
<i>sKey</i>	Unused.
<i>ptxt</i>	Unused.

**7.44.3.23 encode()** [1/12]

```
template<typename PTXT >
double helib::EncryptedArrayCx::encode (
    NTL::ZZX & ptxt,
    const PTXT & pt,
    double useThisSize = -1,
    long precision = -1 ) const [inline]
```

**7.44.3.24 encode()** [2/12]

```
void helib::EncryptedArrayCx::encode (
    UNUSED NTL::ZZX & ptxt,
    UNUSED const PlaintextArray & array ) const [inline], [override]
```

Unimplemented encode function for [BGV](#). It will always throw [helib::LogicError](#).

**Parameters**

<i>ptxt</i>	Unused.
<i>array</i>	Unused.

**7.44.3.25 encode()** [3/12]

```
void helib::EncryptedArrayCx::encode (
    UNUSED NTL::ZZX & ptxt,
    UNUSED const std::vector< long > & array ) const [inline], [override]
```

Unimplemented encode function for [BGV](#). It will always throw [helib::LogicError](#).

**Parameters**

<i>ptxt</i>	Unused.
<i>array</i>	Unused.

**7.44.3.26 encode()** [4/12]

```
void helib::EncryptedArrayCx::encode (
    UNUSED NTL::ZZX & ptxt,
    UNUSED const std::vector< NTL::ZZX > & array ) const [inline], [override]
```

Unimplemented encode function for [BGV](#). It will always throw [helib::LogicError](#).

## Parameters

<i>ptxt</i>	Unused.
<i>array</i>	Unused.

**7.44.3.27 encode()** [5/12]

```
void helib::EncryptedArrayCx::encode (
    UNUSED zzX & ptxt,
    UNUSED const PlaintextArray & array ) const [inline], [override]
```

Unimplemented encode function for [BGV](#). It will always throw [helib::LogicError](#).

## Parameters

<i>ptxt</i>	Unused.
<i>array</i>	Unused.

**7.44.3.28 encode()** [6/12]

```
void helib::EncryptedArrayCx::encode (
    UNUSED zzX & ptxt,
    UNUSED const std::vector< long > & array ) const [inline], [override]
```

Unimplemented encode function for [BGV](#). It will always throw [helib::LogicError](#).

## Parameters

<i>ptxt</i>	Unused.
<i>array</i>	Unused.

**7.44.3.29 encode()** [7/12]

```
void helib::EncryptedArrayCx::encode (
    UNUSED zzX & ptxt,
    UNUSED const std::vector< zzX > & array ) const [inline], [override]
```

Unimplemented encode function for [BGV](#). It will always throw [helib::LogicError](#).

## Parameters

<i>ptxt</i>	Unused.
<i>array</i>	Unused.

### 7.44.3.30 encode() [8/12]

```
template<typename Scheme >
double helib::EncryptedArrayCx::encode (
    zzX & out,
    const Ptxt< Scheme > & ptxt,
    double useThisSize,
    long precision = -1 ) const [inline]
```

Encode a [Ptxt](#) object into a [zzX](#).

#### Template Parameters

<i>Scheme</i>	Encryption scheme to be used (either <a href="#">BGV</a> or <a href="#">CKKS</a> ).
---------------	---

#### Parameters

<i>out</i>	Polynomial to encode into.
<i>ptxt</i>	Plaintext <a href="#">Ptxt</a> object to encode.
<i>useThisSize</i>	Size to use.
<i>precision</i>	Precision to use.

#### Returns

The scaling factor used in the encoding routine.

### 7.44.3.31 encode() [9/12]

```
double helib::EncryptedArrayCx::encode (
    zzX & ptxt,
    const std::vector< cx_double > & array,
    double useThisSize,
    long precision = -1 ) const
```



**7.44.3.32 encode()** [10/12]

```
double helib::EncryptedArrayCx::encode (
    zzX & ptxt,
    const std::vector< double > & array,
    double useThisSize,
    long precision = -1 ) const [inline]
```

**7.44.3.33 encode()** [11/12]

```
double helib::EncryptedArrayCx::encode (
    zzX & ptxt,
    const std::vector< long > & array,
    double useThisSize,
    long precision = -1 ) const [inline]
```

**7.44.3.34 encode()** [12/12]

```
double helib::EncryptedArrayCx::encode (
    zzX & ptxt,
    double aSingleNumber,
    double useThisSize = -1,
    long precision = -1 ) const
```

**7.44.3.35 encodei()**

```
double helib::EncryptedArrayCx::encodei (
    zzX & ptxt,
    long precision = -1 ) const
```

**7.44.3.36 encodeRoundingError()**

```
double helib::EncryptedArrayCx::encodeRoundingError ( ) const [inline]
```

**7.44.3.37 encodeScalingFactor()**

```
long helib::EncryptedArrayCx::encodeScalingFactor (
    long precision = -1,
    double roundErr = -1.0 ) const [inline]
```

**7.44.3.38 encodeUnitSelector()**

```
void helib::EncryptedArrayCx::encodeUnitSelector (
    zzX & ptxt,
    long i ) const [inline], [override], [virtual]
```

Encodes a `std::vector` with 1 at position `i` and 0 everywhere else.

Implements [helib::EncryptedArrayBase](#).

**7.44.3.39 encrypt() [1/2]**

```
template<typename PTXT >
void helib::EncryptedArrayCx::encrypt (
    Ctxt & ctxt,
    const PubKey & key,
    const PTXT & ptxt ) const [inline]
```

**7.44.3.40 encrypt() [2/2]**

```
template<typename PTXT >
void helib::EncryptedArrayCx::encrypt (
    Ctxt & ctxt,
    const PubKey & key,
    const PTXT & ptxt,
    double useThisSize,
    long precision = -1 ) const [inline]
```

**7.44.3.41 encryptOneNum()**

```
void helib::EncryptedArrayCx::encryptOneNum (
    Ctxt & ctxt,
    const PubKey & key,
    double num,
    double useThisSize = -1,
    long precision = -1 ) const [inline]
```

**7.44.3.42 extractImPart() [1/2]**

```
void helib::EncryptedArrayCx::extractImPart (
    Ctxt & c,
    DoubleCRT * dcrt = nullptr ) const
```

Note: If called with `dcrt==nullptr`, `extractImPart` will perform FFT's when encoding `i` as a [DoubleCRT](#) object. If called with `dcrt!=nullptr`, it assumes that `dcrt` points to an object that encodes `i`.

#### 7.44.3.43 extractImPart() [2/2]

```
template<typename Scheme >  
void helib::EncryptedArrayCx::extractImPart (  
    Ptxt< Scheme > & p ) const [inline]
```

Extract the imaginary part of a [CKKS](#) plaintext.

## Template Parameters

<i>Scheme</i>	Encryption scheme to be used (must be <a href="#">CKKS</a> ).
---------------	---

## Parameters

<i>p</i>	Plaintext on which to operate.
----------	--------------------------------

**7.44.3.44 extractRealPart()** [1/2]

```
void helib::EncryptedArrayCx::extractRealPart (
    Ctxt & c ) const
```

**7.44.3.45 extractRealPart()** [2/2]

```
template<typename Scheme >
void helib::EncryptedArrayCx::extractRealPart (
    Ptxt< Scheme > & p ) const [inline]
```

Extract the real part of a [CKKS](#) plaintext.

## Template Parameters

<i>Scheme</i>	Encryption scheme to be used (must be <a href="#">CKKS</a> ).
---------------	---

## Parameters

<i>p</i>	Plaintext on which to operate.
----------	--------------------------------

**7.44.3.46 getContext()**

```
const Context& helib::EncryptedArrayCx::getContext ( ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

#### 7.44.3.47 getDegree()

```
long helib::EncryptedArrayCx::getDegree ( ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

#### 7.44.3.48 getiEncoded()

```
const zzX & helib::EncryptedArrayCx::getiEncoded ( ) const
```

#### 7.44.3.49 getP2R()

```
long helib::EncryptedArrayCx::getP2R ( ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

#### 7.44.3.50 getPAlgebra()

```
const PAlgebra& helib::EncryptedArrayCx::getPAlgebra ( ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

#### 7.44.3.51 getTag()

```
PA_tag helib::EncryptedArrayCx::getTag ( ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

#### 7.44.3.52 random() [1/4]

```
void helib::EncryptedArrayCx::random (
    std::vector< cx_double > & array,
    double rad = 1.0 ) const
```

**7.44.3.53 random()** [2/4]

```
void helib::EncryptedArrayCx::random (
    std::vector< double > & array,
    double rad = 1.0 ) const [inline]
```

**7.44.3.54 random()** [3/4]

```
void helib::EncryptedArrayCx::random (
    std::vector< long > & array ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.44.3.55 random()** [4/4]

```
void helib::EncryptedArrayCx::random (
    UNUSED std::vector< NTL::ZZX > & array ) const [inline], [override]
```

Unimplemented random function for [BGV](#). It will always throw [helib::LogicError](#).

**Parameters**

<i>array</i>	Unused.
--------------	---------

**7.44.3.56 rotate()**

```
void helib::EncryptedArrayCx::rotate (
    Ctxt & ctxt,
    long k ) const [override], [virtual]
```

Right rotation as a linear array. E.g., rotating ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(n 1 2 ... n-1)

Implements [helib::EncryptedArrayBase](#).

**7.44.3.57 rotate1D()**

```
void helib::EncryptedArrayCx::rotate1D (
    Ctxt & ctxt,
    long i,
    long k,
    bool dc = false ) const [override], [virtual]
```

right-rotate k positions along the i'th dimension

## Parameters

<i>dc</i>	means "don't care", which means that the caller guarantees that only zero elements rotate off the end – this allows for some optimizations that would not otherwise be possible
-----------	---

Implements [helib::EncryptedArrayBase](#).

**7.44.3.58 roundedSize()**

```
static double helib::EncryptedArrayCx::roundedSize (
    double x ) [inline], [static]
```

**7.44.3.59 shift()**

```
void helib::EncryptedArrayCx::shift (
    Ctxt & ctxt,
    long k ) const [override], [virtual]
```

Non-cyclic right shift with zero fill E.g., shifting ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(0 1 2... n-1)

Implements [helib::EncryptedArrayBase](#).

### 7.44.3.60 shift1D()

```
void helib::EncryptedArrayCx::shift1D (
    Ctxt & ctxt,
    long i,
    long k ) const [override], [virtual]
```

Right shift k positions along the i'th dimension with zero fill.

Implements [helib::EncryptedArrayBase](#).

The documentation for this class was generated from the following files:

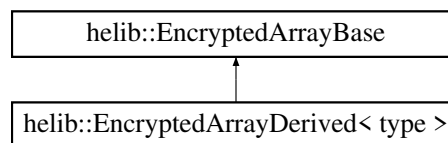
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EncryptedArray.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EaCx.cpp](#)

## 7.45 helib::EncryptedArrayDerived< type > Class Template Reference

Derived concrete implementation of [EncryptedArrayBase](#).

```
#include <EncryptedArray.h>
```

Inheritance diagram for helib::EncryptedArrayDerived< type >:



### Public Member Functions

- [EncryptedArrayDerived](#) (const [Context](#) &\_context, const RX &\_G, const [PAlgebraMod](#) &\_tab)
- [EncryptedArrayDerived](#) (const [EncryptedArrayDerived](#) &other)
- [EncryptedArrayDerived](#) & operator= (const [EncryptedArrayDerived](#) &other)
- virtual [EncryptedArrayBase](#) \* clone () const override
- virtual [PA\\_tag](#) getTag () const override
- template<typename > class T, typename... Args>  
void [dispatch](#) (Args &&... args) const
- const RX & [getG](#) () const
- const NTL::Mat< R > & [getNormalBasisMatrix](#) () const
- const NTL::Mat< R > & [getNormalBasisMatrixInverse](#) () const
- void [initNormalBasisMatrix](#) () const
- virtual void [restoreContext](#) () const override
- virtual void [restoreContextForG](#) () const override
- virtual const [Context](#) & [getContext](#) () const override
- virtual const [PAlgebra](#) & [getPAlgebra](#) () const override
- virtual long [getDegree](#) () const override
- const [PAlgebraModDerived](#)< type > & [getTab](#) () const
- virtual void [rotate](#) (Ctxt &ctxt, long k) const override

*Right rotation as a linear array. E.g., rotating ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(n 1 2 ... n-1)*



- virtual void [shift](#) (Ctxt &ctxt, long k) const override  
*Non-cyclic right shift with zero fill E.g., shifting ctxt=Enc(1 2 3 ... n) by k=1 gives Enc(0 1 2... n-1)*
- virtual void [rotate1D](#) (Ctxt &ctxt, long i, long k, bool dc=false) const override  
*right-rotate k positions along the i'th dimension*
- virtual void [badDimensionAutomorphCorrection](#) (Ctxt &ctxt, long i, long k) const override  
*Correct an automorphism in a bad dimension.*
- long [getP2R](#) () const override
- template<typename U >  
void [rotate1D](#) (std::vector< U > &out, const std::vector< U > &in, long i, long offset) const
- virtual void [shift1D](#) (Ctxt &ctxt, long i, long k) const override  
*Right shift k positions along the i'th dimension with zero fill.*
- void [decrypt](#) (UNUSED const Ctxt &ctxt, UNUSED const SecKey &sKey, UNUSED std::vector< double > &ptxt) const override  
*Unimplemented decrypt function for CKKS. It will always throw helib::LogicError.*
- void [decrypt](#) (UNUSED const Ctxt &ctxt, UNUSED const SecKey &sKey, UNUSED std::vector< cx\_double > &ptxt) const override  
*Unimplemented decrypt function for CKKS. It will always throw helib::LogicError.*
- virtual void [encode](#) (NTL::ZZX &ptxt, const std::vector< long > &array) const override
- virtual void [encode](#) (zzX &ptxt, const std::vector< long > &array) const override
- virtual void [encode](#) (NTL::ZZX &ptxt, const std::vector< NTL::ZZX > &array) const override
- virtual void [encode](#) (zzX &ptxt, const std::vector< zzX > &array) const override
- virtual void [encode](#) (NTL::ZZX &ptxt, const PlaintextArray &array) const override
- virtual void [encode](#) (zzX &ptxt, const PlaintextArray &array) const override
- virtual void [encodeUnitSelector](#) (zzX &ptxt, long i) const override  
*Encodes a std::vector with 1 at position i and 0 everywhere else.*
- virtual void [decode](#) (std::vector< long > &array, const NTL::ZZX &ptxt) const override
- virtual void [decode](#) (std::vector< NTL::ZZX > &array, const NTL::ZZX &ptxt) const override
- virtual void [decode](#) (PlaintextArray &array, const NTL::ZZX &ptxt) const override
- virtual void [decode](#) (PlaintextArray &array, const zzX &ptxt) const
- virtual void [random](#) (std::vector< long > &array) const override
- virtual void [random](#) (std::vector< NTL::ZZX > &array) const override
- virtual void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, std::vector< long > &ptxt) const override
- virtual void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, std::vector< NTL::ZZX > &ptxt) const override
- virtual void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, PlaintextArray &ptxt) const override
- virtual void [buildLinPolyCoeffs](#) (std::vector< NTL::ZZX > &C, const std::vector< NTL::ZZX > &L) const override  
*Linearized polynomials. L describes a linear map M by describing its action on the standard power basis:  $M(x^j \bmod G) = (L[j] \bmod G)$ , for  $j = 0..d-1$ . The result is a coefficient std::vector C for the linearized polynomial representing M: a polynomial h in  $\mathbb{Z}/(p^r)[X]$  of degree < d is sent to.*
- void [encode](#) (zzX &ptxt, const std::vector< RX > &array) const
- void [decode](#) (std::vector< RX > &array, const zzX &ptxt) const
- void [encode](#) (NTL::ZZX &ptxt, const std::vector< RX > &array) const
- void [decode](#) (std::vector< RX > &array, const NTL::ZZX &ptxt) const
- void [encode](#) (RX &ptxt, const std::vector< RX > &array) const
- void [decode](#) (std::vector< RX > &array, const RX &ptxt) const
- void [random](#) (std::vector< RX > &array) const
- void [decrypt](#) (const Ctxt &ctxt, const SecKey &sKey, std::vector< RX > &ptxt) const
- virtual void [buildLinPolyCoeffs](#) (std::vector< RX > &C, const std::vector< RX > &L) const

### 7.45.1 Detailed Description

template<typename type>

class helib::EncryptedArrayDerived< type >

Derived concrete implementation of [EncryptedArrayBase](#).

## 7.45.2 Constructor & Destructor Documentation

### 7.45.2.1 EncryptedArrayDerived() [1/2]

```
template<typename type >
helib::EncryptedArrayDerived< type >::EncryptedArrayDerived (
    const Context & _context,
    const RX & _G,
    const PAlgebraMod & _tab ) [explicit]
```

### 7.45.2.2 EncryptedArrayDerived() [2/2]

```
template<typename type >
helib::EncryptedArrayDerived< type >::EncryptedArrayDerived (
    const EncryptedArrayDerived< type > & other ) [inline]
```

## 7.45.3 Member Function Documentation

### 7.45.3.1 badDimensionAutomorphCorrection()

```
template<typename type >
void helib::EncryptedArrayDerived< type >::badDimensionAutomorphCorrection (
    Ctxt & ctxt,
    long i,
    long amt ) const [override], [virtual]
```

Correct an automorphism in a bad dimension.

#### Parameters

<i>ctxt</i>	Ctxt to perform the correction on.
<i>i</i>	Dimension of which to correct.
<i>amt</i>	Exponent of the auto- mor- phism.

Implements [helib::EncryptedArrayBase](#).

### 7.45.3.2 buildLinPolyCoeffs() [1/2]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::buildLinPolyCoeffs (
    std::vector< NTL::ZZX > & C,
    const std::vector< NTL::ZZX > & L ) const [override], [virtual]
```

Linearized polynomials. L describes a linear map M by describing its action on the standard power basis:  $M(x^j \bmod G) = (L[j] \bmod G)$ , for  $j = 0..d-1$ . The result is a coefficient `std::vector C` for the linearized polynomial representing M: a polynomial  $h$  in  $\mathbb{Z}/(p^r)[X]$  of degree  $< d$  is sent to.

$$//!M(h(X) \bmod G) = \sum_{i=0}^{d-1} (C[i] \cdot h(X^{p^i})) \bmod G) ./!$$

Implements [helib::EncryptedArrayBase](#).

### 7.45.3.3 buildLinPolyCoeffs() [2/2]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::buildLinPolyCoeffs (
    std::vector< RX > & C,
    const std::vector< RX > & L ) const [virtual]
```

### 7.45.3.4 clone()

```
template<typename type >
virtual EncryptedArrayBase* helib::EncryptedArrayDerived< type >::clone ( ) const [inline],
[override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

### 7.45.3.5 decode() [1/7]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decode (
    PlaintextArray & array,
    const NTL::ZZX & ptxt ) const [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.6 decode()** [2/7]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decode (
    PlaintextArray & array,
    const ZZ & ptxt ) const [virtual]
```

**7.45.3.7 decode()** [3/7]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::decode (
    std::vector< long > & array,
    const NTL::ZZ & ptxt ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.8 decode()** [4/7]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::decode (
    std::vector< NTL::ZZ > & array,
    const NTL::ZZ & ptxt ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.9 decode()** [5/7]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decode (
    std::vector< RX > & array,
    const NTL::ZZ & ptxt ) const
```

**7.45.3.10 decode()** [6/7]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decode (
    std::vector< RX > & array,
    const RX & ptxt ) const
```

**7.45.3.11 decode()** [7/7]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decode (
    std::vector< RX > & array,
    const ZZ & ptxt ) const
```

**7.45.3.12 decrypt()** [1/6]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    PlaintextArray & ptxt ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.13 decrypt()** [2/6]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< long > & ptxt ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.14 decrypt()** [3/6]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< NTL::ZZ > & ptxt ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.15 decrypt()** [4/6]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decrypt (
    const Ctxt & ctxt,
    const SecKey & sKey,
    std::vector< RX > & ptxt ) const [inline]
```

**7.45.3.16 decrypt()** [5/6]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decrypt (
    UNUSED const Ctxt & ctxt,
    UNUSED const SecKey & sKey,
    UNUSED std::vector< cx_double > & ptxt ) const [inline], [override]
```

Unimplemented decrypt function for CKKS. It will always throw [helib::LogicError](#).

**Parameters**

<i>ctxt</i>	Unused.
<i>sKey</i>	Unused.
<i>ptxt</i>	Unused.

**7.45.3.17 decrypt()** [6/6]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::decrypt (
    UNUSED const Ctxt & ctxt,
    UNUSED const SecKey & sKey,
    UNUSED std::vector< double > & ptxt ) const [inline], [override]
```

Unimplemented decrypt function for CKKS. It will always throw [helib::LogicError](#).

**Parameters**

<i>ctxt</i>	Unused.
<i>sKey</i>	Unused.
<i>ptxt</i>	Unused.

**7.45.3.18 dispatch()**

```
template<typename type >
template<template< typename > class T, typename... Args>
void helib::EncryptedArrayDerived< type >::dispatch (
    Args &&... args ) const [inline]
```

**7.45.3.19 encode()** [1/9]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::encode (
    NTL::ZZX & ptxt,
    const PlaintextArray & array ) const [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.20 `encode()`** [2/9]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::encode (
    NTL::ZZX & ptxt,
    const std::vector< long > & array ) const [inline], [override], [virtual]
```

Implements `helib::EncryptedArrayBase`.

**7.45.3.21 `encode()`** [3/9]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::encode (
    NTL::ZZX & ptxt,
    const std::vector< NTL::ZZX > & array ) const [inline], [override], [virtual]
```

Implements `helib::EncryptedArrayBase`.

**7.45.3.22 `encode()`** [4/9]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::encode (
    NTL::ZZX & ptxt,
    const std::vector< RX > & array ) const
```

**7.45.3.23 `encode()`** [5/9]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::encode (
    RX & ptxt,
    const std::vector< RX > & array ) const
```

**7.45.3.24 `encode()`** [6/9]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::encode (
    zzX & ptxt,
    const PlaintextArray & array ) const [override], [virtual]
```

Implements `helib::EncryptedArrayBase`.

**7.45.3.25 encode()** [7/9]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::encode (
    zzX & ptxt,
    const std::vector< long > & array ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.26 encode()** [8/9]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::encode (
    zzX & ptxt,
    const std::vector< RX > & array ) const
```

**7.45.3.27 encode()** [9/9]

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::encode (
    zzX & ptxt,
    const std::vector< zzX > & array ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.28 encodeUnitSelector()**

```
template<typename type >
void helib::EncryptedArrayDerived< type >::encodeUnitSelector (
    zzX & ptxt,
    long i ) const [override], [virtual]
```

Encodes a std::vector with 1 at position i and 0 everywhere else.

Implements [helib::EncryptedArrayBase](#).

**7.45.3.29 getContext()**

```
template<typename type >
virtual const Context& helib::EncryptedArrayDerived< type >::getContext ( ) const [inline],
[override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).



### 7.45.3.30 `getDegree()`

```
template<typename type >
virtual long helib::EncryptedArrayDerived< type >::getDegree \( \) const [inline], [override],
[virtual]
```

Implements [helib::EncryptedArrayBase](#).

### 7.45.3.31 `getG()`

```
template<typename type >
const RX& helib::EncryptedArrayDerived< type >::getG \( \) const [inline]
```

### 7.45.3.32 `getNormalBasisMatrix()`

```
template<typename type >
const NTL::Mat<R>& helib::EncryptedArrayDerived< type >::getNormalBasisMatrix \( \) const [inline]
```

### 7.45.3.33 `getNormalBasisMatrixInverse()`

```
template<typename type >
const NTL::Mat<R>& helib::EncryptedArrayDerived< type >::getNormalBasisMatrixInverse \( \)
const [inline]
```

### 7.45.3.34 `getP2R()`

```
template<typename type >
long helib::EncryptedArrayDerived< type >::getP2R \( \) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

### 7.45.3.35 `getPAlgebra()`

```
template<typename type >
virtual const PAlgebra& helib::EncryptedArrayDerived< type >::getPAlgebra \( \) const [inline],
[override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.36 getTab()**

```
template<typename type >
const PAlgebraModDerived<type>& helib::EncryptedArrayDerived< type >::getTab ( ) const [inline]
```

**7.45.3.37 getTag()**

```
template<typename type >
virtual PA\_tag helib::EncryptedArrayDerived< type >::getTag ( ) const [inline], [override],
[virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.38 initNormalBasisMatrix()**

```
template<typename type >
void helib::EncryptedArrayDerived< type >::initNormalBasisMatrix
```

**7.45.3.39 operator=()**

```
template<typename type >
EncryptedArrayDerived& helib::EncryptedArrayDerived< type >::operator= (
    const EncryptedArrayDerived< type > & other ) [inline]
```

**7.45.3.40 random() [1/3]**

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::random (
    std::vector< long > & array ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.41 random() [2/3]**

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::random (
    std::vector< NTL::ZZX > & array ) const [inline], [override], [virtual]
```

Implements [helib::EncryptedArrayBase](#).

**7.45.3.42** `random()` [3/3]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::random (
    std::vector< RX > & array ) const [inline]
```

**7.45.3.43** `restoreContext()`

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::restoreContext ( ) const [inline], [override],
[virtual]
```

Reimplemented from `helib::EncryptedArrayBase`.

**7.45.3.44** `restoreContextForG()`

```
template<typename type >
virtual void helib::EncryptedArrayDerived< type >::restoreContextForG ( ) const [inline],
[override], [virtual]
```

Reimplemented from `helib::EncryptedArrayBase`.

**7.45.3.45** `rotate()`

```
template<typename type >
void helib::EncryptedArrayDerived< type >::rotate (
    Ctxt & ctxt,
    long k ) const [override], [virtual]
```

Right rotation as a linear array. E.g., rotating `ctxt=Enc(1 2 3 ... n)` by `k=1` gives `Enc(n 1 2 ... n-1)`

Implements `helib::EncryptedArrayBase`.

**7.45.3.46** `rotate1D()` [1/2]

```
template<typename type >
void helib::EncryptedArrayDerived< type >::rotate1D (
    Ctxt & ctxt,
    long i,
    long k,
    bool dc = false ) const [override], [virtual]
```

right-rotate *k* positions along the *i*'th dimension

## Parameters

<i>dc</i>	means "don't care", which means that the caller guarantees that only zero elements rotate off the end – this allows for some optimizations that would not otherwise be possible
-----------	---

Implements [helib::EncryptedArrayBase](#).

#### 7.45.3.47 rotate1D() [2/2]

```
template<typename type >
template<typename U >
void helib::EncryptedArrayDerived< type >::rotate1D (
    std::vector< U > & out,
    const std::vector< U > & in,
    long i,
    long offset ) const [inline]
```

**7.45.3.48 `shift()`**

```
template<typename type >
void helib::EncryptedArrayDerived< type >::shift (
    Ctxt & ctxt,
    long k ) const [override], [virtual]
```

Non-cyclic right shift with zero fill E.g., shifting `ctxt=Enc(1 2 3 ... n)` by `k=1` gives `Enc(0 1 2... n-1)`

Implements [helib::EncryptedArrayBase](#).

**7.45.3.49 `shift1D()`**

```
template<typename type >
void helib::EncryptedArrayDerived< type >::shift1D (
    Ctxt & ctxt,
    long i,
    long k ) const [override], [virtual]
```

Right shift `k` positions along the `i`'th dimension with zero fill.

Implements [helib::EncryptedArrayBase](#).

The documentation for this class was generated from the following files:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/EncryptedArray.h`
- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/EncryptedArray.cpp`

**7.46 `helib::equals_pa_impl< type >` Class Template Reference****Static Public Member Functions**

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, bool &res, const [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)
- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, bool &res, const [PlaintextArray](#) &pa, const `std::vector< long >` &other)
- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, bool &res, const [PlaintextArray](#) &pa, const `std::vector< NTL::ZZX >` &other)

**7.46.1 Member Function Documentation**

**7.46.1.1 apply() [1/3]**

```
template<typename type >
static void helib::equals_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    bool & res,
    const PlaintextArray & pa,
    const PlaintextArray & other ) [inline], [static]
```

**7.46.1.2 apply() [2/3]**

```
template<typename type >
static void helib::equals_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    bool & res,
    const PlaintextArray & pa,
    const std::vector< long > & other ) [inline], [static]
```

**7.46.1.3 apply() [3/3]**

```
template<typename type >
static void helib::equals_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    bool & res,
    const PlaintextArray & pa,
    const std::vector< NTL::ZZX > & other ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EncryptedArray.cpp](#)

**7.47 helib::EvalMap Class Reference**

Class that provides the functionality for the linear transforms used in bootstrapping. The constructor is invoked with three arguments:

```
#include <EvalMap.h>
```

**Public Member Functions**

- [EvalMap](#) (const [EncryptedArray](#) &\_ea, bool minimal, const NTL::Vec< long > &mvec, bool \_invert, bool build\_cache, bool normal\_basis=true)
- void [upgrade](#) ()
- void [apply](#) (Ctxt &ctxt) const

### 7.47.1 Detailed Description

Class that provides the functionality for the linear transforms used in bootstrapping. The constructor is invoked with three arguments:

- an [EncryptedArray](#) object `ea`
- an integer vector `mvec`
- a boolean flag `invert` The `mvec` vector specifies the factorization of `m` to use in the "powerful basis" decomposition.

If the `invert` flag is false, the forward transformation is used. This transformation views the slots as being packed with powerful-basis coefficients and performs a multi-point polynomial evaluation. This is the second transformation used in bootstrapping.

If `invert` flag is true, the inverse transformation is used. In addition, the current implementation folds into the inverse transformation a transformation that moves the coefficients in each slot into a normal-basis representation, which helps with the unpacking procedure.

The constructor precomputes certain values, and the linear transformation itself is effected using the `apply` method.

Note that the factorization in `mvec` must correspond to the generators used in [PAlgebra](#). The best way to ensure this is to directly use the output of the program in `params.cpp`: that program computes values for `mvec` (to be used here), and `gens` and `ords` (to be used in initialization of the [Context](#)).

### 7.47.2 Constructor & Destructor Documentation

#### 7.47.2.1 EvalMap()

```
helib::EvalMap::EvalMap (
    const EncryptedArray & _ea,
    bool minimal,
    const NTL::Vec< long > & mvec,
    bool _invert,
    bool build_cache,
    bool normal_basis = true )
```

### 7.47.3 Member Function Documentation

#### 7.47.3.1 apply()

```
void helib::EvalMap::apply (
    Ctxt & ctxt ) const
```

### 7.47.3.2 upgrade()

```
void helib::EvalMap::upgrade ( )
```

The documentation for this class was generated from the following files:

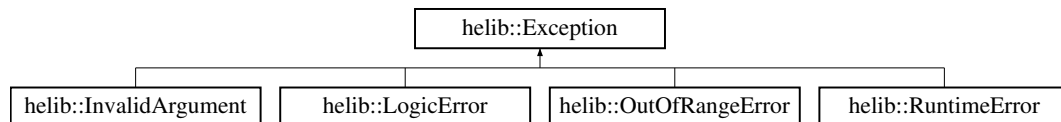
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EvalMap.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EvalMap.cpp](#)

## 7.48 helib::Exception Class Reference

Base class that other HElib exception classes inherit from.

```
#include <exceptions.h>
```

Inheritance diagram for helib::Exception:



### Public Member Functions

- virtual [~Exception](#) ()=default
- virtual const char \* [what](#) () const noexcept=0

### Protected Member Functions

- [Exception](#) ()=default

### 7.48.1 Detailed Description

Base class that other HElib exception classes inherit from.

### 7.48.2 Constructor & Destructor Documentation

#### 7.48.2.1 ~Exception()

```
virtual helib::Exception::~~Exception ( ) [virtual], [default]
```



### 7.48.2.2 Exception()

```
helib::Exception::Exception ( ) [protected], [default]
```

## 7.48.3 Member Function Documentation

### 7.48.3.1 what()

```
virtual const char* helib::Exception::what ( ) const [pure virtual], [noexcept]
```

Implemented in [helib::InvalidArgument](#), [helib::RuntimeError](#), [helib::OutOfRangeError](#), and [helib::LogicError](#).

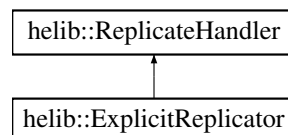
The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/exceptions.h](#)

## 7.49 helib::ExplicitReplicator Class Reference

An implementation of [ReplicateHandler](#) that explicitly returns all the replicated ciphertexts in one big vector.

Inheritance diagram for helib::ExplicitReplicator:



### Public Member Functions

- [ExplicitReplicator](#) (std::vector< [Ctxt](#) > &\_v)
- virtual void [handle](#) (const [Ctxt](#) &ctxt)

### 7.49.1 Detailed Description

An implementation of [ReplicateHandler](#) that explicitly returns all the replicated ciphertexts in one big vector.

This is useful mostly for debugging purposes, for real parameters it would take a lot of memory.

### 7.49.2 Constructor & Destructor Documentation

### 7.49.2.1 ExplicitReplicator()

```
helib::ExplicitReplicator::ExplicitReplicator (
    std::vector< Ctxt > & _v ) [inline]
```

## 7.49.3 Member Function Documentation

### 7.49.3.1 handle()

```
virtual void helib::ExplicitReplicator::handle (
    const Ctxt & ctxt ) [inline], [virtual]
```

Implements [helib::ReplicateHandler](#).

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[replicate.cpp](#)

## 7.50 helib::fhe\_stats\_record Struct Reference

```
#include <fhe_stats.h>
```

### Public Member Functions

- [fhe\\_stats\\_record](#) (const char \*\_name)
- void [update](#) (double val)
- void [save](#) (double val)

### Public Attributes

- const char \* [name](#)
- long [count](#)
- double [sum](#)
- double [max](#)
- std::vector< double > [saved\\_values](#)

### Static Public Attributes

- static std::vector< [fhe\\_stats\\_record](#) \* > [map](#)

### 7.50.1 Constructor & Destructor Documentation

### 7.50.1.1 fhe\_stats\_record()

```
helib::fhe_stats_record::fhe_stats_record (
    const char * _name )
```

## 7.50.2 Member Function Documentation

### 7.50.2.1 save()

```
void helib::fhe_stats_record::save (
    double val )
```

### 7.50.2.2 update()

```
void helib::fhe_stats_record::update (
    double val )
```

## 7.50.3 Member Data Documentation

### 7.50.3.1 count

```
long helib::fhe_stats_record::count
```

### 7.50.3.2 map

```
std::vector<fhe_stats_record*> helib::fhe_stats_record::map [static]
```

### 7.50.3.3 max

```
double helib::fhe_stats_record::max
```

#### 7.50.3.4 name

```
const char* helib::fhe_stats_record::name
```

#### 7.50.3.5 saved\_values

```
std::vector<double> helib::fhe_stats_record::saved_values
```

#### 7.50.3.6 sum

```
double helib::fhe_stats_record::sum
```

The documentation for this struct was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[fhe\\_stats.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[fhe\\_stats.cpp](#)

## 7.51 helib::FHEtimer Class Reference

A simple class to accumulate time.

```
#include <timing.h>
```

### Public Member Functions

- [FHEtimer](#) (const char \*\_name, const char \*\_loc)
- void [reset](#) ()
- double [getTime](#) () const
- long [getNumCalls](#) () const

### Public Attributes

- const char \* [name](#)
- const char \* [loc](#)
- [HELIB\\_atomic\\_ulong](#) counter
- [HELIB\\_atomic\\_long](#) numCalls

#### 7.51.1 Detailed Description

A simple class to accumulate time.

## 7.51.2 Constructor & Destructor Documentation

### 7.51.2.1 FHEtimer()

```
helib::FHEtimer::FHEtimer (
    const char * _name,
    const char * _loc ) [inline]
```

## 7.51.3 Member Function Documentation

### 7.51.3.1 getNumCalls()

```
long helib::FHEtimer::getNumCalls ( ) const
```

### 7.51.3.2 getTime()

```
double helib::FHEtimer::getTime ( ) const
```

### 7.51.3.3 reset()

```
void helib::FHEtimer::reset ( )
```

## 7.51.4 Member Data Documentation

### 7.51.4.1 counter

```
HELIB_atomic_ulong helib::FHEtimer::counter
```

### 7.51.4.2 loc

```
const char* helib::FHEtimer::loc
```

#### 7.51.4.3 name

```
const char* helib::FHEtimer::name
```

#### 7.51.4.4 numCalls

```
HELIB_atomic_long helib::FHEtimer::numCalls
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[timing.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[timing.cpp](#)

## 7.52 helib::FlowEdge Class Reference

An edge in a flow graph.

```
#include <matching.h>
```

### Public Member Functions

- [FlowEdge](#) (long c=0, long f=0)

### Public Attributes

- long [capacity](#)
- long [flow](#)

#### 7.52.1 Detailed Description

An edge in a flow graph.

#### 7.52.2 Constructor & Destructor Documentation

##### 7.52.2.1 FlowEdge()

```
helib::FlowEdge::FlowEdge (  
    long c = 0,  
    long f = 0 ) [inline], [explicit]
```

### 7.52.3 Member Data Documentation

#### 7.52.3.1 capacity

```
long helib::FlowEdge::capacity
```

#### 7.52.3.2 flow

```
long helib::FlowEdge::flow
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matching.h](#)

## 7.53 helib::frobeniusAutomorph\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, long j)
- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const NTL::Vec< long > &vec)

### 7.53.1 Member Function Documentation

#### 7.53.1.1 apply() [1/2]

```
template<typename type >
static void helib::frobeniusAutomorph_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const NTL::Vec< long > & vec ) [inline], [static]
```

### 7.53.1.2 apply() [2/2]

```
template<typename type >
static void helib::frobeniusAutomorph_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    long j ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EncryptedArray.cpp

## 7.54 helib::FullBinaryTree Class Reference

A simple implementation of full binary trees (each non-leaf has 2 children)

```
#include <permutations.h>
```

### Public Member Functions

- [FullBinaryTree](#) (long \_aux=0)
- [FullBinaryTree](#) (const T &d, long \_aux=0)
- void [putDataInRoot](#) (const T &d)
- long [size](#) ()
- [TreeNode](#)< T > & [operator\[\]](#) (long i)
- const [TreeNode](#)< T > & [operator\[\]](#) (long i) const
- [TreeNode](#)< T > & [at](#) (long i)
- const [TreeNode](#)< T > & [at](#) (long i) const
- T & [DataOfNode](#) (long i)
- const T & [DataOfNode](#) (long i) const
- long [getAuxKey](#) () const
- void [setAuxKey](#) (long \_aux)
- long [getNleaves](#) () const
- long [firstLeaf](#) () const
- long [nextLeaf](#) (long i) const
- long [prevLeaf](#) (long i) const
- long [lastLeaf](#) () const
- long [rootIdx](#) () const
- long [parentIdx](#) (long i) const
- long [leftChildIdx](#) (long i) const
- long [rightChildIdx](#) (long i) const
- void [printout](#) (std::ostream &s, long idx=0) const
- long [addChildren](#) (long prntIdx, const T &leftData, const T &rightData)
- void [collapseToRoot](#) ()

*Remove all nodes in the tree except for the root.*

### 7.54.1 Detailed Description

A simple implementation of full binary trees (each non-leaf has 2 children)



## 7.54.2 Constructor & Destructor Documentation

### 7.54.2.1 FullBinaryTree() [1/2]

```
helib::FullBinaryTree::FullBinaryTree (  
    long _aux = 0 )    [inline]
```

### 7.54.2.2 FullBinaryTree() [2/2]

```
helib::FullBinaryTree::FullBinaryTree (  
    const T & d,  
    long _aux = 0 )    [inline], [explicit]
```

## 7.54.3 Member Function Documentation

### 7.54.3.1 addChildren()

```
long helib::FullBinaryTree::addChildren (  
    long prntIdx,  
    const T & leftData,  
    const T & rightData )
```

If the parent is a leaf, add to it to children with the given data, else just update the data of the two children of this parent. Returns the index of the left child, the right-child index is one more than the left-child index.

### 7.54.3.2 at() [1/2]

```
TreeNode<T>& helib::FullBinaryTree::at (  
    long i )    [inline]
```

### 7.54.3.3 at() [2/2]

```
const TreeNode<T>& helib::FullBinaryTree::at (  
    long i ) const    [inline]
```

#### 7.54.3.4 collapseToRoot()

```
void helib::FullBinaryTree::collapseToRoot ( ) [inline]
```

Remove all nodes in the tree except for the root.

#### 7.54.3.5 DataOfNode() [1/2]

```
T& helib::FullBinaryTree::DataOfNode (
    long i ) [inline]
```

#### 7.54.3.6 DataOfNode() [2/2]

```
const T& helib::FullBinaryTree::DataOfNode (
    long i ) const [inline]
```

#### 7.54.3.7 firstLeaf()

```
long helib::FullBinaryTree::firstLeaf ( ) const [inline]
```

#### 7.54.3.8 getAuxKey()

```
long helib::FullBinaryTree::getAuxKey ( ) const [inline]
```

#### 7.54.3.9 getNleaves()

```
long helib::FullBinaryTree::getNleaves ( ) const [inline]
```

#### 7.54.3.10 lastLeaf()

```
long helib::FullBinaryTree::lastLeaf ( ) const [inline]
```

#### 7.54.3.11 leftChildIdx()

```
long helib::FullBinaryTree::leftChildIdx (
    long i ) const [inline]
```

#### 7.54.3.12 nextLeaf()

```
long helib::FullBinaryTree::nextLeaf (
    long i ) const [inline]
```

#### 7.54.3.13 operator[]() [1/2]

```
TreeNode<T>& helib::FullBinaryTree::operator[] (
    long i ) [inline]
```

#### 7.54.3.14 operator[]() [2/2]

```
const TreeNode<T>& helib::FullBinaryTree::operator[] (
    long i ) const [inline]
```

#### 7.54.3.15 parentIdx()

```
long helib::FullBinaryTree::parentIdx (
    long i ) const [inline]
```

#### 7.54.3.16 prevLeaf()

```
long helib::FullBinaryTree::prevLeaf (
    long i ) const [inline]
```

#### 7.54.3.17 printout()

```
void helib::FullBinaryTree::printout (
    std::ostream & s,
    long idx = 0 ) const [inline]
```

**7.54.3.18 putDataInRoot()**

```
void helib::FullBinaryTree::putDataInRoot (
    const T & d ) [inline]
```

**7.54.3.19 rightChildIdx()**

```
long helib::FullBinaryTree::rightChildIdx (
    long i ) const [inline]
```

**7.54.3.20 rootIdx()**

```
long helib::FullBinaryTree::rootIdx ( ) const [inline]
```

**7.54.3.21 setAuxKey()**

```
void helib::FullBinaryTree::setAuxKey (
    long _aux ) [inline]
```

**7.54.3.22 size()**

```
long helib::FullBinaryTree::size ( ) [inline]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[permutations.h](#)

**7.55 helib::GenDescriptor Class Reference**

A minimal description of a generator for the purpose of building tree.

```
#include <permutations.h>
```

**Public Member Functions**

- [GenDescriptor](#) (long \_order, bool \_good, long gen=0)
- [GenDescriptor](#) ()

## Public Attributes

- long [genIdx](#)
- long [order](#)
- bool [good](#)

### 7.55.1 Detailed Description

A minimal description of a generator for the purpose of building tree.

### 7.55.2 Constructor & Destructor Documentation

#### 7.55.2.1 GenDescriptor() [1/2]

```
helib::GenDescriptor::GenDescriptor (
    long _order,
    bool _good,
    long gen = 0 ) [inline]
```

#### 7.55.2.2 GenDescriptor() [2/2]

```
helib::GenDescriptor::GenDescriptor ( ) [inline]
```

### 7.55.3 Member Data Documentation

#### 7.55.3.1 genIdx

```
long helib::GenDescriptor::genIdx
```

#### 7.55.3.2 good

```
bool helib::GenDescriptor::good
```

### 7.55.3.3 order

```
long helib::GenDescriptor::order
```

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/permutations.h](#)

## 7.56 helib::general\_range< T > Class Template Reference

```
#include <range.h>
```

### Classes

- class [iterator](#)

### Public Member Functions

- [iterator begin](#) () const
- [iterator end](#) () const
- [general\\_range](#) (T *begin*, T *end*)

### 7.56.1 Constructor & Destructor Documentation

#### 7.56.1.1 general\_range()

```
template<typename T >  
helib::general_range< T >::general_range (   
    T begin,  
    T end ) [inline]
```

### 7.56.2 Member Function Documentation

#### 7.56.2.1 begin()

```
template<typename T >  
iterator helib::general_range< T >::begin ( ) const [inline]
```

### 7.56.2.2 end()

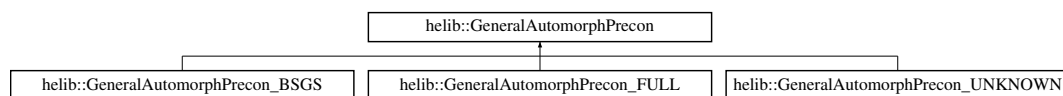
```
template<typename T >
iterator helib::general_range< T >::end ( ) const [inline]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[range.h](#)

## 7.57 helib::GeneralAutomorphPrecon Class Reference

Inheritance diagram for helib::GeneralAutomorphPrecon:



### Public Member Functions

- virtual [~GeneralAutomorphPrecon](#) ()
- virtual std::shared\_ptr< [Ctxt](#) > [automorph](#) (long i) const =0

### 7.57.1 Constructor & Destructor Documentation

#### 7.57.1.1 ~GeneralAutomorphPrecon()

```
virtual helib::GeneralAutomorphPrecon::~GeneralAutomorphPrecon ( ) [inline], [virtual]
```

### 7.57.2 Member Function Documentation

#### 7.57.2.1 automorph()

```
virtual std::shared_ptr<Ctxt> helib::GeneralAutomorphPrecon::automorph (
    long i ) const [pure virtual]
```

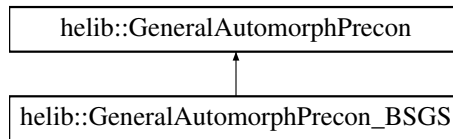
Implemented in [helib::GeneralAutomorphPrecon\\_BSGS](#), [helib::GeneralAutomorphPrecon\\_FULL](#), and [helib::GeneralAutomorphPrecon\\_UNKNOWN](#).

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.58 helib::GeneralAutomorphPrecon\_BSGS Class Reference

Inheritance diagram for helib::GeneralAutomorphPrecon\_BSGS:



### Public Member Functions

- [GeneralAutomorphPrecon\\_BSGS](#) (const [Ctxt](#) &\_ctxt, long \_dim, const [EncryptedArray](#) &ea)
- std::shared\_ptr< [Ctxt](#) > [automorph](#) (long i) const override

### 7.58.1 Constructor & Destructor Documentation

#### 7.58.1.1 GeneralAutomorphPrecon\_BSGS()

```

helib::GeneralAutomorphPrecon_BSGS::GeneralAutomorphPrecon_BSGS (
    const Ctxt & _ctxt,
    long _dim,
    const EncryptedArray & ea ) [inline]
  
```

### 7.58.2 Member Function Documentation

#### 7.58.2.1 automorph()

```

std::shared_ptr<Ctxt> helib::GeneralAutomorphPrecon_BSGS::automorph (
    long i ) const [inline], [override], [virtual]
  
```

Implements [helib::GeneralAutomorphPrecon](#).

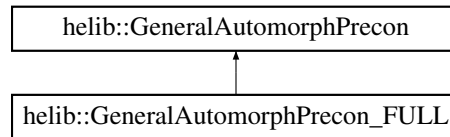
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)



## 7.59 `helib::GeneralAutomorphPrecon_FULL` Class Reference

Inheritance diagram for `helib::GeneralAutomorphPrecon_FULL`:



### Public Member Functions

- [`GeneralAutomorphPrecon\_FULL`](#) (const [`Ctxt`](#) &`_ctxt`, long `_dim`, const [`EncryptedArray`](#) &`ea`)
- `std::shared_ptr< Ctxt > automorph` (long `i`) const override

### 7.59.1 Constructor & Destructor Documentation

#### 7.59.1.1 `GeneralAutomorphPrecon_FULL()`

```

helib::GeneralAutomorphPrecon_FULL::GeneralAutomorphPrecon_FULL (
    const Ctxt & _ctxt,
    long _dim,
    const EncryptedArray & ea ) [inline]

```

### 7.59.2 Member Function Documentation

#### 7.59.2.1 `automorph()`

```

std::shared_ptr<Ctxt> helib::GeneralAutomorphPrecon_FULL::automorph (
    long i ) const [inline], [override], [virtual]

```

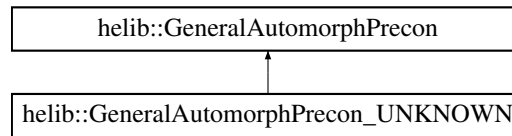
Implements [`helib::GeneralAutomorphPrecon`](#).

The documentation for this class was generated from the following file:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/matmul.cpp`

## 7.60 helib::GeneralAutomorphPrecon\_UNKNOWN Class Reference

Inheritance diagram for helib::GeneralAutomorphPrecon\_UNKNOWN:



### Public Member Functions

- [GeneralAutomorphPrecon\\_UNKNOWN](#) (const [Ctxt](#) &\_ctxt, long \_dim, const [EncryptedArray](#) &ea)
- std::shared\_ptr< [Ctxt](#) > [automorph](#) (long i) const override

### 7.60.1 Constructor & Destructor Documentation

#### 7.60.1.1 GeneralAutomorphPrecon\_UNKNOWN()

```

helib::GeneralAutomorphPrecon_UNKNOWN::GeneralAutomorphPrecon_UNKNOWN (
    const Ctxt & _ctxt,
    long _dim,
    const EncryptedArray & ea ) [inline]
  
```

### 7.60.2 Member Function Documentation

#### 7.60.2.1 automorph()

```

std::shared_ptr<Ctxt> helib::GeneralAutomorphPrecon_UNKNOWN::automorph (
    long i ) const [inline], [override], [virtual]
  
```

Implements [helib::GeneralAutomorphPrecon](#).

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.61 helib::GeneralBenesNetwork Class Reference

Implementation of generalized Benes Permutation Network.

```
#include <permutations.h>
```

## Public Member Functions

- long [getDepth](#) () const
- long [getSize](#) () const
- long [getNumLevels](#) () const
- const NTL::Vec< short > & [getLevel](#) (long i) const
- long [levelToDepthMap](#) (long i) const
- long [shamt](#) (long i) const
- [GeneralBenesNetwork](#) (const [Permut](#) &perm)
- bool [testNetwork](#) (const [Permut](#) &perm) const

## Static Public Member Functions

- static long [depth](#) (long n)
- static long [levelToDepthMap](#) ([UNUSED](#) long n, long k, long i)
- static long [shamt](#) (long n, long k, long i)

### 7.61.1 Detailed Description

Implementation of generalized Benes Permutation Network.

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 GeneralBenesNetwork()

```
helib::GeneralBenesNetwork::GeneralBenesNetwork (
    const Permut & perm )
```

### 7.61.3 Member Function Documentation

#### 7.61.3.1 depth()

```
static long helib::GeneralBenesNetwork::depth (
    long n ) [inline], [static]
```

computes recursion depth k for generalized Benes network of size n. the actual number of levels in the network is  $2*k-1$

#### 7.61.3.2 getDepth()

```
long helib::GeneralBenesNetwork::getDepth ( ) const [inline]
```

**7.61.3.3 getLevel()**

```
const NTL::Vec<short>& helib::GeneralBenesNetwork::getLevel (
    long i ) const [inline]
```

**7.61.3.4 getNumLevels()**

```
long helib::GeneralBenesNetwork::getNumLevels ( ) const [inline]
```

**7.61.3.5 getSize()**

```
long helib::GeneralBenesNetwork::getSize ( ) const [inline]
```

**7.61.3.6 levelToDepthMap() [1/2]**

```
long helib::GeneralBenesNetwork::levelToDepthMap (
    long i ) const [inline]
```

**7.61.3.7 levelToDepthMap() [2/2]**

```
static long helib::GeneralBenesNetwork::levelToDepthMap (
    UNUSED long n,
    long k,
    long i ) [inline], [static]
```

maps a level number  $i = 0..2*k-2$  to a recursion depth  $d = 0..k-1$  using the formula  $d = (k-1) - |(k-1)-i|$

**7.61.3.8 shamt() [1/2]**

```
long helib::GeneralBenesNetwork::shamt (
    long i ) const [inline]
```

**7.61.3.9 shamt() [2/2]**

```
static long helib::GeneralBenesNetwork::shamt (
    long n,
    long k,
    long i ) [inline], [static]
```

shift amount for level number  $i=0..2*k-2$  using the formula  $\text{ceil}(\text{floor}(n/2^d) / 2)$ , where  $d = \text{levelToDepthMap}(i)$

### 7.61.3.10 testNetwork()

```
bool helib::GeneralBenesNetwork::testNetwork (
    const Permut & perm ) const
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[permutations.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[BenesNetwork.cpp](#)

## 7.62 helib::GeneratorTrees Class Reference

A std::vector of generator trees, one per generator in  $Z_{m^*}/(p)$

```
#include <permutations.h>
```

### Public Member Functions

- [GeneratorTrees](#) ()
- long [numLayers](#) () const
- long [numTrees](#) () const
- long [getSize](#) () const
- [OneGeneratorTree](#) & [operator\[\]](#) (long i)
- const [OneGeneratorTree](#) & [operator\[\]](#) (long i) const
- [OneGeneratorTree](#) & [at](#) (long i)
- const [OneGeneratorTree](#) & [at](#) (long i) const
- [OneGeneratorTree](#) & [getGenTree](#) (long i)
- const [OneGeneratorTree](#) & [getGenTree](#) (long i) const
- const [Permut](#) & [mapToCube](#) () const
- const [Permut](#) & [mapToArray](#) () const
- [Permut](#) & [mapToCube](#) ()
- [Permut](#) & [mapToArray](#) ()
- long [mapToCube](#) (long i) const
- long [mapToArray](#) (long i) const
- void [getCubeDims](#) (NTL::Vec< long > &dims) const
- void [getCubeSubDims](#) (NTL::Vec< long > &dims) const
- long [buildOptimalTrees](#) (const NTL::Vec< [GenDescriptor](#) > &vec, long depthBound)
- void [ComputeCubeMapping](#) ()

*Computes permutations mapping between linear array and the cube.*

### Friends

- std::ostream & [operator<<](#) (std::ostream &s, const [GeneratorTrees](#) &t)

### 7.62.1 Detailed Description

A std::vector of generator trees, one per generator in  $Z_{m^*}/(p)$

## 7.62.2 Constructor & Destructor Documentation

### 7.62.2.1 GeneratorTrees()

```
helib::GeneratorTrees::GeneratorTrees ( ) [inline]
```

## 7.62.3 Member Function Documentation

### 7.62.3.1 at() [1/2]

```
OneGeneratorTree& helib::GeneratorTrees::at (
    long i ) [inline]
```

### 7.62.3.2 at() [2/2]

```
const OneGeneratorTree& helib::GeneratorTrees::at (
    long i ) const [inline]
```

### 7.62.3.3 buildOptimalTrees()

```
long helib::GeneratorTrees::buildOptimalTrees (
    const NTL::Vec< GenDescriptor > & vec,
    long depthBound )
```

Compute the trees corresponding to the "optimal" way of breaking a permutation into dimensions, subject to some constraints. Returns the cost (# of 1D shifts) of this solution. Returns NTL\_MAX\_LONG if no solution

### 7.62.3.4 ComputeCubeMapping()

```
void helib::GeneratorTrees::ComputeCubeMapping ( )
```

Computes permutations mapping between linear array and the cube.

If the cube dimensions (i.e., leaves of tree) are  $n_1, n_2, \dots, n_t$  and  $N = \prod_j n_j$  is the size of the cube, then an integer  $i$  can be represented in either the mixed base of the  $n_j$ 's or in "CRT basis" relative to the leaves: Namely either  $i = \sum_{j \leq t} i_j * \prod_{k > j} n_k$ , or  $i = \sum_{leaf} i_{leaf} * leaf.e \bmod N$ .

The breakPermByDim procedure expects its input in the mixed-base representation, and the maps are used to convert back and forth. Specifically, let  $(i'_1, \dots, i'_t)$  be the CRT representation of  $i$  in this cube, and  $j = \sum_{j=1}^t i'_j * \prod_{k > j} n_k$ , then we have  $map2cube[i] = j$  and  $map2array[j] = i$ .

### 7.62.3.5 getCubeDims()

```
void helib::GeneratorTrees::getCubeDims (
    NTL::Vec< long > & dims ) const
```

Get the "crude" cube dimensions corresponding to the vector of trees, the ordered vector with one dimension per tree

### 7.62.3.6 getCubeSubDims()

```
void helib::GeneratorTrees::getCubeSubDims (
    NTL::Vec< long > & dims ) const
```

Get the "fine" cube dimensions corresponding to the vector of trees, the ordered vector with one dimension per leaf in all the trees.

### 7.62.3.7 getGenTree() [1/2]

```
OneGeneratorTree& helib::GeneratorTrees::getGenTree (
    long i ) [inline]
```

### 7.62.3.8 getGenTree() [2/2]

```
const OneGeneratorTree& helib::GeneratorTrees::getGenTree (
    long i ) const [inline]
```

### 7.62.3.9 getSize()

```
long helib::GeneratorTrees::getSize ( ) const [inline]
```

### 7.62.3.10 mapToArray() [1/3]

```
Permut& helib::GeneratorTrees::mapToArray ( ) [inline]
```

### 7.62.3.11 mapToArray() [2/3]

```
const Permut& helib::GeneratorTrees::mapToArray ( ) const [inline]
```

**7.62.3.12 mapToArray()** [3/3]

```
long helib::GeneratorTrees::mapToArray (
    long i ) const [inline]
```

**7.62.3.13 mapToCube()** [1/3]

```
Permut& helib::GeneratorTrees::mapToCube ( ) [inline]
```

**7.62.3.14 mapToCube()** [2/3]

```
const Permut& helib::GeneratorTrees::mapToCube ( ) const [inline]
```

**7.62.3.15 mapToCube()** [3/3]

```
long helib::GeneratorTrees::mapToCube (
    long i ) const [inline]
```

**7.62.3.16 numLayers()**

```
long helib::GeneratorTrees::numLayers ( ) const [inline]
```

**7.62.3.17 numTrees()**

```
long helib::GeneratorTrees::numTrees ( ) const [inline]
```

**7.62.3.18 operator[]()** [1/2]

```
OneGeneratorTree& helib::GeneratorTrees::operator[] (
    long i ) [inline]
```



### 7.62.3.19 operator[]() [2/2]

```
const OneGeneratorTree& helib::GeneratorTrees::operator[] (
    long i ) const [inline]
```

## 7.62.4 Friends And Related Function Documentation

### 7.62.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const GeneratorTrees & t ) [friend]
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[permutations.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[OptimizePermutations.cpp](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[permutations.cpp](#)

## 7.63 helib::half\_FFT Struct Reference

```
#include <PAlgebra.h>
```

### Public Member Functions

- [half\\_FFT](#) (long m)

### Public Attributes

- [PGFFT](#) fft
- std::vector< std::complex< double > > [pow](#)

## 7.63.1 Constructor & Destructor Documentation

### 7.63.1.1 half\_FFT()

```
helib::half_FFT::half_FFT (
    long m )
```

## 7.63.2 Member Data Documentation

### 7.63.2.1 fft

`PGFFT helib::half_FFT::fft`

### 7.63.2.2 pow

`std::vector<std::complex<double> > helib::half_FFT::pow`

The documentation for this struct was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PAgebra.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/PAgebra.cpp](#)

## 7.64 HighLvlTimingData Class Reference

### Public Member Functions

- [HighLvlTimingData](#) (long \_lvl=-1)

### Public Attributes

- long [lvl](#)
- double [rotate](#)
- double [shift](#)
- double [permute](#)
- double [matmul](#)
- double [replicate](#)
- double [replAll](#)

## 7.64.1 Constructor & Destructor Documentation

### 7.64.1.1 HighLvlTimingData()

```
HighLvlTimingData::HighLvlTimingData (  
    long _lvl = -1 ) [inline], [explicit]
```

## 7.64.2 Member Data Documentation

### 7.64.2.1 lvl

```
long HighLvlTimingData::lvl
```

### 7.64.2.2 matmul

```
double HighLvlTimingData::matmul
```

### 7.64.2.3 permute

```
double HighLvlTimingData::permute
```

### 7.64.2.4 replAll

```
double HighLvlTimingData::replAll
```

### 7.64.2.5 replicate

```
double HighLvlTimingData::replicate
```

### 7.64.2.6 rotate

```
double HighLvlTimingData::rotate
```

### 7.64.2.7 shift

```
double HighLvlTimingData::shift
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[Test\\_Timing.cpp](#)

## 7.65 helib::HyperCube< T > Class Template Reference

A multi-dimensional cube.

```
#include <hypercube.h>
```

### Public Member Functions

- [HyperCube](#) (const [HyperCube](#) &other)=default
- [HyperCube](#) (const [CubeSignature](#) &\_sig)  
*initialize a [HyperCube](#) with a [CubeSignature](#)*
- [HyperCube](#) & [operator=](#) (const [HyperCube](#)< T > &other)  
*assignment: signatures must be the same*
- bool [operator==](#) (const [HyperCube](#)< T > &other) const  
*equality testing: signatures must be the same*
- bool [operator!=](#) (const [HyperCube](#)< T > &other) const
- const [CubeSignature](#) & [getSig](#) () const  
*const ref to signature*
- NTL::Vec< T > & [getData](#) ()
- const NTL::Vec< T > & [getData](#) () const  
*read-only ref to data vector*
- long [getSize](#) () const  
*total size of cube*
- long [getNumDims](#) () const  
*number of dimensions*
- long [getDim](#) (long d) const  
*size of dimension d*
- long [getProd](#) (long d) const  
*product of sizes of dimensions d, d+1, ...*
- long [getProd](#) (long from, long to) const  
*product of sizes of dimensions from, from+1, ..., to-1*
- long [getCoord](#) (long i, long d) const  
*get coordinate in dimension d of index i*
- long [addCoord](#) (long i, long d, long offset) const  
*add offset to coordinate in dimension d of index i*
- long [numSlices](#) (long d=1) const  
*number of slices*
- long [sliceSize](#) (long d=1) const  
*size of one slice*
- long [numCols](#) () const  
*number of columns*
- T & [at](#) (long i)  
*reference to element at position i, with bounds check*
- T & [operator\[\]](#) (long i)  
*reference to element at position i, without bounds check*
- const T & [at](#) (long i) const  
*read-only reference to element at position i, with bounds check*
- const T & [operator\[\]](#) (long i) const  
*read-only reference to element at position i, without bounds check*
- void [rotate1D](#) (long i, long k)  
*rotate k positions along the i'th dimension*
- void [shift1D](#) (long i, long k)  
*Shift k positions along the i'th dimension with zero fill.*

## 7.65.1 Detailed Description

```
template<typename T>
class helib::HyperCube< T >
```

A multi-dimensional cube.

Such an object is initialized with a [CubeSignature](#): a reference to the signature is stored with the cube, and so the signature must remain alive during the lifetime of the cube, to prevent dangling pointers.

## 7.65.2 Constructor & Destructor Documentation

### 7.65.2.1 HyperCube() [1/2]

```
template<typename T >
helib::HyperCube< T >::HyperCube (
    const HyperCube< T > & other ) [default]
```

### 7.65.2.2 HyperCube() [2/2]

```
template<typename T >
helib::HyperCube< T >::HyperCube (
    const CubeSignature & _sig ) [inline]
```

initialize a [HyperCube](#) with a [CubeSignature](#)

## 7.65.3 Member Function Documentation

### 7.65.3.1 addCoord()

```
template<typename T >
long helib::HyperCube< T >::addCoord (
    long i,
    long d,
    long offset ) const [inline]
```

add offset to coordinate in dimension d of index i

**7.65.3.2 at() [1/2]**

```
template<typename T >
T& helib::HyperCube< T >::at (
    long i ) [inline]
```

reference to element at position i, with bounds check

**7.65.3.3 at() [2/2]**

```
template<typename T >
const T& helib::HyperCube< T >::at (
    long i ) const [inline]
```

read-only reference to element at position i, with bounds check

**7.65.3.4 getCoord()**

```
template<typename T >
long helib::HyperCube< T >::getCoord (
    long i,
    long d ) const [inline]
```

get coordinate in dimension d of index i

**7.65.3.5 getData() [1/2]**

```
template<typename T >
NTL::Vec<T>& helib::HyperCube< T >::getData ( ) [inline]
```

read/write ref to the data vector. Note that the length of data is fixed upon construction, so it cannot be changed through this ref.

**7.65.3.6 getData() [2/2]**

```
template<typename T >
const NTL::Vec<T>& helib::HyperCube< T >::getData ( ) const [inline]
```

read-only ref to data vector

### 7.65.3.7 getDim()

```
template<typename T >
long helib::HyperCube< T >::getDim (
    long d ) const [inline]
```

size of dimension d

### 7.65.3.8 getNumDims()

```
template<typename T >
long helib::HyperCube< T >::getNumDims ( ) const [inline]
```

number of dimensions

### 7.65.3.9 getProd() [1/2]

```
template<typename T >
long helib::HyperCube< T >::getProd (
    long d ) const [inline]
```

product of sizes of dimensions d, d+1, ...

### 7.65.3.10 getProd() [2/2]

```
template<typename T >
long helib::HyperCube< T >::getProd (
    long from,
    long to ) const [inline]
```

product of sizes of dimensions from, from+1, ..., to-1

### 7.65.3.11 getSig()

```
template<typename T >
const CubeSignature& helib::HyperCube< T >::getSig ( ) const [inline]
```

const ref to signature

#### 7.65.3.12 getSize()

```
template<typename T >
long helib::HyperCube< T >::getSize ( ) const [inline]
```

total size of cube

#### 7.65.3.13 numCols()

```
template<typename T >
long helib::HyperCube< T >::numCols ( ) const [inline]
```

number of columns

#### 7.65.3.14 numSlices()

```
template<typename T >
long helib::HyperCube< T >::numSlices (
    long d = 1 ) const [inline]
```

number of slices

#### 7.65.3.15 operator"!="()

```
template<typename T >
bool helib::HyperCube< T >::operator!= (
    const HyperCube< T > & other ) const [inline]
```

#### 7.65.3.16 operator=()

```
template<typename T >
HyperCube& helib::HyperCube< T >::operator= (
    const HyperCube< T > & other ) [inline]
```

assignment: signatures must be the same



### 7.65.3.17 operator==( )

```
template<typename T >
bool helib::HyperCube< T >::operator==(
    const HyperCube< T > & other ) const [inline]
```

equality testing: signatures must be the same

### 7.65.3.18 operator[]( ) [1/2]

```
template<typename T >
T& helib::HyperCube< T >::operator[] (
    long i ) [inline]
```

reference to element at position i, without bounds check

### 7.65.3.19 operator[]( ) [2/2]

```
template<typename T >
const T& helib::HyperCube< T >::operator[] (
    long i ) const [inline]
```

read-only reference to element at position i, without bounds check

### 7.65.3.20 rotate1D( )

```
template<typename T >
void helib::HyperCube< T >::rotate1D (
    long i,
    long k )
```

rotate k positions along the i'th dimension

### 7.65.3.21 shift1D( )

```
template<typename T >
void helib::HyperCube< T >::shift1D (
    long i,
    long k )
```

Shift k positions along the i'th dimension with zero fill.

### 7.65.3.22 sliceSize()

```
template<typename T >
long helib::HyperCube< T >::sliceSize (
    long d = 1 ) const [inline]
```

size of one slice

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/hypercube.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/hypercube.cpp

## 7.66 helib::IndexMap< T > Class Template Reference

IndexMap<T> implements a generic map indexed by a dynamic index set.

```
#include <IndexMap.h>
```

### Public Member Functions

- [IndexMap](#) ()  
*The empty map.*
- [IndexMap](#) ([IndexMapInit](#)< T > \*\_init)  
*A map with an initialization object. This associates a method for initializing new elements in the map. When a new index j is added to the index set, an object t of type T is created using the default constructor for T, after which the function \_init->init(t) is called (t is passed by reference). To use this feature, you need to derive a subclass of [IndexMapInit](#)<T> that defines the init function. This "helper object" should be created using operator new, and the pointer is "exclusively owned" by the map object.*
- const [IndexSet](#) & [getIndexSet](#) () const  
*Get the underlying index set.*
- T & [operator\[\]](#) (long j)  
*Access functions: will raise an error if j does not belong to the current index set.*
- const T & [operator\[\]](#) (long j) const
- void [insert](#) (long j)  
*Insert indexes to the [IndexSet](#). Insertion will cause new T objects to be created, using the default constructor, and possibly initialized via the [IndexMapInit](#)<T> pointer.*
- void [insert](#) (const [IndexSet](#) &s)
- void [remove](#) (long j)  
*Delete indexes from [IndexSet](#), may cause objects to be destroyed.*
- void [remove](#) (const [IndexSet](#) &s)
- void [clear](#) ()

### 7.66.1 Detailed Description

```
template<typename T>
class helib::IndexMap< T >
```

IndexMap<T> implements a generic map indexed by a dynamic index set.

Additionally, it allows new elements of the map to be initialized in a flexible manner.

## 7.66.2 Constructor & Destructor Documentation

### 7.66.2.1 IndexMap() [1/2]

```
template<typename T >
helib::IndexMap< T >::IndexMap ( )
```

The empty map.

### 7.66.2.2 IndexMap() [2/2]

```
template<typename T >
helib::IndexMap< T >::IndexMap (
    IndexMapInit< T > * _init ) [inline], [explicit]
```

A map with an initialization object. This associates a method for initializing new elements in the map. When a new index  $j$  is added to the index set, an object  $t$  of type  $T$  is created using the default constructor for  $T$ , after which the function `_init->init(t)` is called ( $t$  is passed by reference). To use this feature, you need to derive a subclass of `IndexMapInit<T>` that defines the `init` function. This "helper object" should be created using operator `new`, and the pointer is "exclusively owned" by the map object.

## 7.66.3 Member Function Documentation

### 7.66.3.1 clear()

```
template<typename T >
void helib::IndexMap< T >::clear ( ) [inline]
```

### 7.66.3.2 getIndexSet()

```
template<typename T >
const IndexSet& helib::IndexMap< T >::getIndexSet ( ) const [inline]
```

Get the underlying index set.

### 7.66.3.3 insert() [1/2]

```
template<typename T >
void helib::IndexMap< T >::insert (
    const IndexSet & s ) [inline]
```

### 7.66.3.4 insert() [2/2]

```
template<typename T >
void helib::IndexMap< T >::insert (
    long j ) [inline]
```

Insert indexes to the [IndexSet](#). Insertion will cause new T objects to be created, using the default constructor, and possibly initialized via the `IndexMapInit<T>` pointer.

### 7.66.3.5 operator[]() [1/2]

```
template<typename T >
T& helib::IndexMap< T >::operator[] (
    long j ) [inline]
```

Access functions: will raise an error if j does not belong to the current index set.

### 7.66.3.6 operator[]() [2/2]

```
template<typename T >
const T& helib::IndexMap< T >::operator[] (
    long j ) const [inline]
```

### 7.66.3.7 remove() [1/2]

```
template<typename T >
void helib::IndexMap< T >::remove (
    const IndexSet & s ) [inline]
```

**7.66.3.8 remove()** [2/2]

```
template<typename T >
void helib::IndexMap< T >::remove (
    long j ) [inline]
```

Delete indexes from [IndexSet](#), may cause objects to be destroyed.

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[IndexMap.h](#)

**7.67 helib::IndexMapInit< T > Class Template Reference**

Initializing elements in an [IndexMap](#).

```
#include <IndexMap.h>
```

**Public Member Functions**

- virtual void [init](#) (T &)=0  
*Initialization function, override with initialization code.*
- virtual [IndexMapInit](#)< T > \* [clone](#) () const =0  
*Cloning a pointer, override with code to create a fresh copy.*
- virtual [~IndexMapInit](#) ()

**7.67.1 Detailed Description**

```
template<typename T>
class helib::IndexMapInit< T >
```

Initializing elements in an [IndexMap](#).

**7.67.2 Constructor & Destructor Documentation****7.67.2.1 ~IndexMapInit()**

```
template<typename T >
virtual helib::IndexMapInit< T >::~~IndexMapInit ( ) [inline], [virtual]
```

**7.67.3 Member Function Documentation**

### 7.67.3.1 clone()

```
template<typename T >
virtual IndexMapInit<T>* helib::IndexMapInit< T >::clone ( ) const [pure virtual]
```

Cloning a pointer, override with code to create a fresh copy.

### 7.67.3.2 init()

```
template<typename T >
virtual void helib::IndexMapInit< T >::init (
    T & ) [pure virtual]
```

Initialization function, override with initialization code.

Implemented in [helib::DoubleCRTHelper](#).

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[IndexMap.h](#)

## 7.68 helib::IndexSet Class Reference

A dynamic set of non-negative integers.

```
#include <IndexSet.h>
```

### Classes

- class [iterator](#)

### Public Member Functions

- [IndexSet](#) ()
- [IndexSet](#) (long low, long high)
- [IndexSet](#) (long j)
- long [first](#) () const  
*Returns the first element, 0 if the set is empty.*
- long [last](#) () const  
*Returns the last element, -1 if the set is empty.*
- long [next](#) (long j) const  
*Returns the next element after j, if any; otherwise j+1.*
- long [prev](#) (long j) const
- long [card](#) () const  
*The cardinality of the set.*
- bool [contains](#) (long j) const  
*Returns true iff the set contains j.*

- bool `contains` (const `IndexSet` &s) const  
*Returns true iff the set contains s.*
- bool `disjointFrom` (const `IndexSet` &s) const  
*Returns true iff the set is disjoint from s.*
- bool `operator==` (const `IndexSet` &s) const
- bool `operator!=` (const `IndexSet` &s) const
- void `clear` ()  
*Set to the empty set.*
- void `insert` (long j)  
*Add j to the set.*
- void `remove` (long j)  
*Remove j from the set.*
- void `insert` (const `IndexSet` &s)  
*Add s to the set (union)*
- void `remove` (const `IndexSet` &s)  
*Remove s from the set (set minus)*
- void `retain` (const `IndexSet` &s)  
*Retains only those elements that are also in s (intersection)*
- bool `isInterval` () const  
*Is this set a contiguous interval?*
- void `read` (std::istream &str)
- void `write` (std::ostream &str) const
- `iterator begin` () const
- `iterator end` () const

## Static Public Member Functions

- static const `IndexSet` & `emptySet` ()  
*Read-only access to an empty set.*

### 7.68.1 Detailed Description

A dynamic set of non-negative integers.

You can iterate through a set as follows:

```
for (long i = s.first(); i <= s.last(); i = s.next(i)) ...
for (long i = s.last(); i >= s.first(); i = s.prev(i)) ...
```

### 7.68.2 Constructor & Destructor Documentation

#### 7.68.2.1 IndexSet() [1/3]

```
helib::IndexSet::IndexSet ( ) [inline]
```

### 7.68.2.2 IndexSet() [2/3]

```
helib::IndexSet::IndexSet (
    long low,
    long high ) [inline]
```

### 7.68.2.3 IndexSet() [3/3]

```
helib::IndexSet::IndexSet (
    long j ) [inline], [explicit]
```

## 7.68.3 Member Function Documentation

### 7.68.3.1 begin()

```
iterator helib::IndexSet::begin ( ) const [inline]
```

### 7.68.3.2 card()

```
long helib::IndexSet::card ( ) const [inline]
```

The cardinality of the set.

### 7.68.3.3 clear()

```
void helib::IndexSet::clear ( )
```

Set to the empty set.

### 7.68.3.4 contains() [1/2]

```
bool helib::IndexSet::contains (
    const IndexSet & s ) const
```

Returns true iff the set contains s.



### 7.68.3.5 contains() [2/2]

```
bool helib::IndexSet::contains (
    long j ) const
```

Returns true iff the set contains j.

### 7.68.3.6 disjointFrom()

```
bool helib::IndexSet::disjointFrom (
    const IndexSet & s ) const
```

Returns true iff the set is disjoint from s.

### 7.68.3.7 emptySet()

```
const IndexSet & helib::IndexSet::emptySet ( ) [static]
```

Read-only access to an empty set.

### 7.68.3.8 end()

```
iterator helib::IndexSet::end ( ) const [inline]
```

### 7.68.3.9 first()

```
long helib::IndexSet::first ( ) const [inline]
```

Returns the first element, 0 if the set is empty.

### 7.68.3.10 insert() [1/2]

```
void helib::IndexSet::insert (
    const IndexSet & s )
```

Add s to the set (union)

#### 7.68.3.11 insert() [2/2]

```
void helib::IndexSet::insert (
    long j )
```

Add j to the set.

#### 7.68.3.12 isInterval()

```
bool helib::IndexSet::isInterval ( ) const [inline]
```

Is this set a contiguous interval?

#### 7.68.3.13 last()

```
long helib::IndexSet::last ( ) const [inline]
```

Returns the last element, -1 if the set is empty.

#### 7.68.3.14 next()

```
long helib::IndexSet::next (
    long j ) const
```

Returns the next element after j, if any; otherwise j+1.

#### 7.68.3.15 operator!=(())

```
bool helib::IndexSet::operator!= (
    const IndexSet & s ) const [inline]
```

#### 7.68.3.16 operator==(())

```
bool helib::IndexSet::operator== (
    const IndexSet & s ) const
```

### 7.68.3.17 prev()

```
long helib::IndexSet::prev (
    long j ) const
```

### 7.68.3.18 read()

```
void helib::IndexSet::read (
    std::istream & str )
```

### 7.68.3.19 remove() [1/2]

```
void helib::IndexSet::remove (
    const IndexSet & s )
```

Remove s from the set (set minus)

### 7.68.3.20 remove() [2/2]

```
void helib::IndexSet::remove (
    long j )
```

Remove j from the set.

### 7.68.3.21 retain()

```
void helib::IndexSet::retain (
    const IndexSet & s )
```

Retains only those elements that are also in s (intersection)

### 7.68.3.22 write()

```
void helib::IndexSet::write (
    std::ostream & str ) const
```

The documentation for this class was generated from the following files:

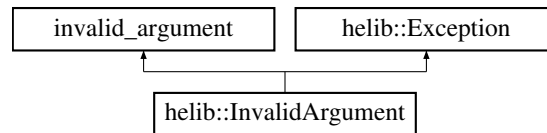
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[IndexSet.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[IndexSet.cpp](#)

## 7.69 helib::InvalidArgument Class Reference

Inherits from [Exception](#) and `std::invalid_argument`.

```
#include <exceptions.h>
```

Inheritance diagram for `helib::InvalidArgument`:



### Public Member Functions

- [InvalidArgument](#) (const std::string &what\_arg)
- [InvalidArgument](#) (const char \*what\_arg)
- virtual [~InvalidArgument](#) ()
- virtual const char \* [what](#) () const noexcept override

### Additional Inherited Members

#### 7.69.1 Detailed Description

Inherits from [Exception](#) and `std::invalid_argument`.

#### 7.69.2 Constructor & Destructor Documentation

##### 7.69.2.1 InvalidArgument() [1/2]

```
helib::InvalidArgument::InvalidArgument (  
    const std::string & what_arg ) [inline], [explicit]
```

##### 7.69.2.2 InvalidArgument() [2/2]

```
helib::InvalidArgument::InvalidArgument (  
    const char * what_arg ) [inline], [explicit]
```

### 7.69.2.3 ~InvalidArgument()

```
virtual helib::InvalidArgument::~InvalidArgument ( ) [inline], [virtual]
```

## 7.69.3 Member Function Documentation

### 7.69.3.1 what()

```
virtual const char* helib::InvalidArgument::what ( ) const [inline], [override], [virtual],
[noexcept]
```

Implements [helib::Exception](#).

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[exceptions.h](#)

## 7.70 helib::IndexSet::iterator Class Reference

```
#include <IndexSet.h>
```

### Public Member Functions

- long [operator\\*](#) () const
- [iterator](#) & [operator++](#) ()
- bool [operator==](#) (const [iterator](#) &other) const
- bool [operator!=](#) (const [iterator](#) &other) const

### Protected Member Functions

- [iterator](#) (const [IndexSet](#) &s, long i)

### Friends

- class [IndexSet](#)

### 7.70.1 Constructor & Destructor Documentation

### 7.70.1.1 iterator()

```
helib::IndexSet::iterator::iterator (
    const IndexSet & s,
    long i ) [inline], [protected]
```

## 7.70.2 Member Function Documentation

### 7.70.2.1 operator!=(())

```
bool helib::IndexSet::iterator::operator!= (
    const iterator & other ) const [inline]
```

### 7.70.2.2 operator\*()

```
long helib::IndexSet::iterator::operator* ( ) const [inline]
```

### 7.70.2.3 operator++()

```
iterator& helib::IndexSet::iterator::operator++ ( ) [inline]
```

### 7.70.2.4 operator==(())

```
bool helib::IndexSet::iterator::operator== (
    const iterator & other ) const [inline]
```

## 7.70.3 Friends And Related Function Documentation

### 7.70.3.1 IndexSet

```
friend class IndexSet [friend]
```

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/\[IndexSet.h\]\(#\)](#)

## 7.71 helib::general\_range< T >::iterator Class Reference

```
#include <range.h>
```

### Public Member Functions

- T [operator\\*](#) () const
- [iterator](#) & [operator++](#) ()
- bool [operator==](#) (const [iterator](#) &other) const
- bool [operator!=](#) (const [iterator](#) &other) const

### Protected Member Functions

- [iterator](#) (T start)

### Friends

- class [general\\_range](#)

## 7.71.1 Constructor & Destructor Documentation

### 7.71.1.1 iterator()

```
template<typename T >  
helib::general_range< T >::iterator::iterator (  
    T start ) [inline], [protected]
```

## 7.71.2 Member Function Documentation

### 7.71.2.1 operator"!=()"

```
template<typename T >  
bool helib::general_range< T >::iterator::operator!= (  
    const iterator & other ) const [inline]
```

### 7.71.2.2 operator\*()

```
template<typename T >
T helib::general_range< T >::iterator::operator* ( ) const [inline]
```

### 7.71.2.3 operator++()

```
template<typename T >
iterator& helib::general_range< T >::iterator::operator++ ( ) [inline]
```

### 7.71.2.4 operator==()

```
template<typename T >
bool helib::general_range< T >::iterator::operator== (
    const iterator & other ) const [inline]
```

## 7.71.3 Friends And Related Function Documentation

### 7.71.3.1 general\_range

```
template<typename T >
friend class general_range [friend]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[range.h](#)

## 7.72 helib::KeySwitch Class Reference

Key-switching matrices.

```
#include <keySwitching.h>
```



## Public Member Functions

- [KeySwitch](#) (long sPow=0, long xPow=0, long fromID=0, long toID=0, long p=0)
- [KeySwitch](#) (const [SKHandle](#) &\_fromKey, long fromID=0, long toID=0, long p=0)
- bool [operator==](#) (const [KeySwitch](#) &other) const
- bool [operator!=](#) (const [KeySwitch](#) &other) const
- unsigned long [NumCols](#) () const
- bool [isDummy](#) () const
- void [verify](#) ([SecKey](#) &sk)  
*A debugging method.*
- void [readMatrix](#) (std::istream &str, const [Context](#) &context)  
*Read a key-switching matrix from input.*
- void [read](#) (std::istream &str, const [Context](#) &context)  
*Raw IO.*
- void [write](#) (std::ostream &str) const

## Static Public Member Functions

- static const [KeySwitch](#) & [dummy](#) ()  
*returns a dummy static matrix with toKeyId == -1*

## Public Attributes

- [SKHandle](#) [fromKey](#)
- long [toKeyId](#)
- long [ptxtSpace](#)
- std::vector< [DoubleCRT](#) > [b](#)
- NTL::ZZ [prgSeed](#)
- NTL::xdouble [noiseBound](#)

### 7.72.1 Detailed Description

Key-switching matrices.

There are basically two approaches for how to do key-switching: either decompose the mod- $q$  ciphertext into bits (or digits) to make it low-norm, or perform the key-switching operation mod  $Q \gg q$ . The tradeoff is that when decomposing the (coefficients of the) ciphertext into  $t$  digits, we need to increase the size of the key-switching matrix by a factor of  $t$  (and the running time similarly grows). On the other hand if we do not decompose at all then we need to work modulo  $Q \gg q^2$ , which means that the bitsize of our largest modulus  $q_0$  more than doubles (and hence also the parameter  $m$  more than doubles). In general if we decompose into digits of size  $B$  then we need to work with  $Q \gg q \cdot B$ .)

The part of the spectrum where we expect to find the sweet spot is when we decompose the ciphertext into digits of size  $B = q_0^{1/t}$  for some small constant  $t$  (maybe  $t=2,3$  or so). This means that our largest modulus has to be  $Q \gg q_0^{1+1/t}$ , which increases also the parameter  $m$  by a factor  $(1+1/t)$ . It also means that for key-switching in the top levels we would break the ciphertext to  $t$  digits, hence the key-switching matrix will have  $t$  columns.

A key-switch matrix  $W[s' \rightarrow s]$  converts a ciphertext-part with respect to secret-key polynomial  $s'$  into a canonical ciphertext (i.e. a two-part ciphertext with respect to  $(1,s)$ ). The matrix  $W$  is a 2-by- $t$  matrix of [DoubleCRT](#) objects. The bottom row are just (pseudo)random elements. Then for column  $j$ , if the bottom element is  $a_j$  then the top element is set as  $b_j = P \cdot B_j \cdot s' + p \cdot e_j - s \cdot a_j \bmod P \cdot q_0$ , where  $p$  is the plaintext space (i.e. 2 or  $2^r$ , or 1 for [CKKS](#)) and  $B_j$  is the product of the digits-sizes corresponding to columns  $0 \dots i-1$ . (For example if we have digit sizes 3,5,7

then  $B_0=1$ ,  $B_1=3$ ,  $B_2=15$  and  $B_3=105$ .) Also,  $q_0$  is the product of all the "ciphertext primes" and  $P$  is roughly the product of all the special primes. (Actually, for [BGV](#), if  $Q$  is the product of all the special primes then  $P=Q*(Q^{-1} \bmod p)$ .)

In this implementation we save some space, by keeping only a PRG seed for generating the pseudo-random elements, rather than the elements themselves.

To convert a ciphertext part  $R$ , we break  $R$  into digits  $R = \sum_j B_j R_j$ , then set  $(q_0, q_1)^T = \sum_j R_j * \text{column-}j$ . Note that we have  $\langle (1, s), (q_0, q_1) \rangle = \sum_j R_j * (s * a_j - s * a_j + p * e_j + P * B_j * s) = P * \sum_j B_j * R_j * s' + p \sum_j R_j * e_j = P * R * s' + p * \text{a-small-element} \pmod{P * q_0}$  where the last element is small since the  $e_j$ 's are small and  $|R_j| < B$ . Note that if the ciphertext is encrypted relative to plaintext space  $p'$  and then key-switched with matrices  $W$  relative to plaintext space  $p$ , then we get a new ciphertext with noise  $p' * \text{small} + p * \text{small}$ , so it is valid relative to plaintext space  $\text{GCD}(p', p)$ .

The matrix  $W$  is defined modulo  $Q > t * B * \sigma * q_0$  (with  $\sigma$  a bound on the size of the  $e_j$ 's), and  $Q$  is the product of all the small primes in our moduli chain. However, if  $p$  is much smaller than  $B$  then it is enough to use  $W \bmod Q_i$  with  $Q_i$  a smaller modulus,  $Q > p * \sigma * q_0$ . Also note that if  $p < B_r$  then we will be using only first  $r$  columns of the matrix  $W$ .

## 7.72.2 Constructor & Destructor Documentation

### 7.72.2.1 KeySwitch() [1/2]

```
helib::KeySwitch::KeySwitch (
    long sPow = 0,
    long xPow = 0,
    long fromID = 0,
    long toID = 0,
    long p = 0 ) [explicit]
```

### 7.72.2.2 KeySwitch() [2/2]

```
helib::KeySwitch::KeySwitch (
    const SKHandle & _fromKey,
    long fromID = 0,
    long toID = 0,
    long p = 0 ) [explicit]
```

## 7.72.3 Member Function Documentation

### 7.72.3.1 dummy()

```
const KeySwitch & helib::KeySwitch::dummy ( ) [static]
```

returns a dummy static matrix with toKeyId == -1

### 7.72.3.2 isDummy()

```
bool helib::KeySwitch::isDummy ( ) const
```

### 7.72.3.3 NumCols()

```
unsigned long helib::KeySwitch::NumCols ( ) const
```

### 7.72.3.4 operator"!="()

```
bool helib::KeySwitch::operator!= (
    const KeySwitch & other ) const
```

### 7.72.3.5 operator==()

```
bool helib::KeySwitch::operator== (
    const KeySwitch & other ) const
```

### 7.72.3.6 read()

```
void helib::KeySwitch::read (
    std::istream & str,
    const Context & context )
```

Raw IO.

### 7.72.3.7 readMatrix()

```
void helib::KeySwitch::readMatrix (
    std::istream & str,
    const Context & context )
```

Read a key-switching matrix from input.

#### 7.72.3.8 verify()

```
void helib::KeySwitch::verify (
    SecKey & sk )
```

A debugging method.

#### 7.72.3.9 write()

```
void helib::KeySwitch::write (
    std::ostream & str ) const
```

### 7.72.4 Member Data Documentation

#### 7.72.4.1 b

```
std::vector<DoubleCRT> helib::KeySwitch::b
```

#### 7.72.4.2 fromKey

```
SKHandle helib::KeySwitch::fromKey
```

#### 7.72.4.3 noiseBound

```
NTL::xdouble helib::KeySwitch::noiseBound
```

#### 7.72.4.4 prgSeed

```
NTL::ZZ helib::KeySwitch::prgSeed
```

#### 7.72.4.5 ptxtSpace

```
long helib::KeySwitch::ptxtSpace
```

#### 7.72.4.6 toKeyID

```
long helib::KeySwitch::toKeyID
```

The documentation for this class was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/keySwitching.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/keySwitching.cpp](#)

## 7.73 helib::LabeledEdge Class Reference

A generic directed edge in a graph with some labels.

```
#include <matching.h>
```

### Public Member Functions

- [LabeledEdge](#) (long f, long t, long l=0, long c=0)

### Public Attributes

- long [from](#)
- long [to](#)
- long [label](#)
- long [color](#)

### 7.73.1 Detailed Description

A generic directed edge in a graph with some labels.

### 7.73.2 Constructor & Destructor Documentation

#### 7.73.2.1 LabeledEdge()

```
helib::LabeledEdge::LabeledEdge (  
    long f,  
    long t,  
    long l = 0,  
    long c = 0 ) [inline]
```

### 7.73.3 Member Data Documentation

### 7.73.3.1 color

```
long helib::LabeledEdge::color
```

### 7.73.3.2 from

```
long helib::LabeledEdge::from
```

### 7.73.3.3 label

```
long helib::LabeledEdge::label
```

### 7.73.3.4 to

```
long helib::LabeledEdge::to
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matching.h](#)

## 7.74 helib::LabeledVertex Class Reference

A generic node in a graph with some labels.

```
#include <matching.h>
```

### Public Member Functions

- [LabeledVertex](#) (long n, long l=0)
- void [addEdge](#) (long nn, long l=0, long c=0)
- void [addNeighbor](#) (long nn, long l=0, long c=0)

### Public Attributes

- long [name](#)
- long [label](#)
- [LNeighborList](#) [neighbors](#)

### 7.74.1 Detailed Description

A generic node in a graph with some labels.

### 7.74.2 Constructor & Destructor Documentation

#### 7.74.2.1 LabeledVertex()

```
helib::LabeledVertex::LabeledVertex (
    long n,
    long l = 0 ) [inline], [explicit]
```

### 7.74.3 Member Function Documentation

#### 7.74.3.1 addEdge()

```
void helib::LabeledVertex::addEdge (
    long nn,
    long l = 0,
    long c = 0 ) [inline]
```

#### 7.74.3.2 addNeighbor()

```
void helib::LabeledVertex::addNeighbor (
    long nn,
    long l = 0,
    long c = 0 ) [inline]
```

### 7.74.4 Member Data Documentation

#### 7.74.4.1 label

```
long helib::LabeledVertex::label
```

#### 7.74.4.2 name

```
long helib::LabeledVertex::name
```

#### 7.74.4.3 neighbors

```
LNeighborList helib::LabeledVertex::neighbors
```

The documentation for this class was generated from the following file:

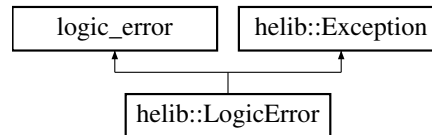
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matching.h](#)

## 7.75 helib::LogicError Class Reference

Inherits from [Exception](#) and `std::logic_error`.

```
#include <exceptions.h>
```

Inheritance diagram for `helib::LogicError`:



### Public Member Functions

- [LogicError](#) (const `std::string` &what\_arg)
- [LogicError](#) (const `char` \*what\_arg)
- virtual [~LogicError](#) ()
- virtual const `char` \* [what](#) () const noexcept override

### Additional Inherited Members

#### 7.75.1 Detailed Description

Inherits from [Exception](#) and `std::logic_error`.

#### 7.75.2 Constructor & Destructor Documentation



**7.75.2.1 LogicError() [1/2]**

```
helib::LogicError::LogicError (
    const std::string & what_arg ) [inline], [explicit]
```

**7.75.2.2 LogicError() [2/2]**

```
helib::LogicError::LogicError (
    const char * what_arg ) [inline], [explicit]
```

**7.75.2.3 ~LogicError()**

```
virtual helib::LogicError::~~LogicError ( ) [inline], [virtual]
```

**7.75.3 Member Function Documentation****7.75.3.1 what()**

```
virtual const char* helib::LogicError::what ( ) const [inline], [override], [virtual], [noexcept]
```

Implements [helib::Exception](#).

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/exceptions.h](#)

**7.76 LowLvlTimingData Class Reference****Public Member Functions**

- [LowLvlTimingData](#) (long *\_lvl*=-1)

**Public Attributes**

- long *lvl*
- double [addConst](#)
- double [add](#)
- double [multConst](#)
- double [mult](#)
- double [multBy2](#)
- double [autoNative](#)
- double [autoTypical](#)
- double [innerProd](#)

## 7.76.1 Constructor & Destructor Documentation

### 7.76.1.1 LowLvlTimingData()

```
LowLvlTimingData::LowLvlTimingData (
    long _lvl = -1 )    [inline], [explicit]
```

## 7.76.2 Member Data Documentation

### 7.76.2.1 add

```
double LowLvlTimingData::add
```

### 7.76.2.2 addConst

```
double LowLvlTimingData::addConst
```

### 7.76.2.3 autoNative

```
double LowLvlTimingData::autoNative
```

### 7.76.2.4 autoTypical

```
double LowLvlTimingData::autoTypical
```

### 7.76.2.5 innerProd

```
double LowLvlTimingData::innerProd
```

### 7.76.2.6 lvl

```
long LowLvlTimingData::lvl
```

### 7.76.2.7 mult

```
double LowLvlTimingData::mult
```

### 7.76.2.8 multBy2

```
double LowLvlTimingData::multBy2
```

### 7.76.2.9 multConst

```
double LowLvlTimingData::multConst
```

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/Test\\_Timing.cpp](#)

## 7.77 helib::MappingData< type > Class Template Reference

Auxiliary structure to support encoding/decoding slots.

```
#include <PAlgebra.h>
```

### Public Member Functions

- const RX & [getG](#) () const
- long [getDegG](#) () const
- void [restoreContextForG](#) () const

### Friends

- class [PAlgebraModDerived< type >](#)

### 7.77.1 Detailed Description

```
template<typename type>  
class helib::MappingData< type >
```

Auxiliary structure to support encoding/decoding slots.

### 7.77.2 Member Function Documentation

#### 7.77.2.1 getDegG()

```
template<typename type >  
long helib::MappingData< type >::getDegG ( ) const [inline]
```

#### 7.77.2.2 getG()

```
template<typename type >  
const RX& helib::MappingData< type >::getG ( ) const [inline]
```

#### 7.77.2.3 restoreContextForG()

```
template<typename type >  
void helib::MappingData< type >::restoreContextForG ( ) const [inline]
```

### 7.77.3 Friends And Related Function Documentation

#### 7.77.3.1 PAlgebraModDerived< type >

```
template<typename type >  
friend class PAlgebraModDerived< type > [friend]
```

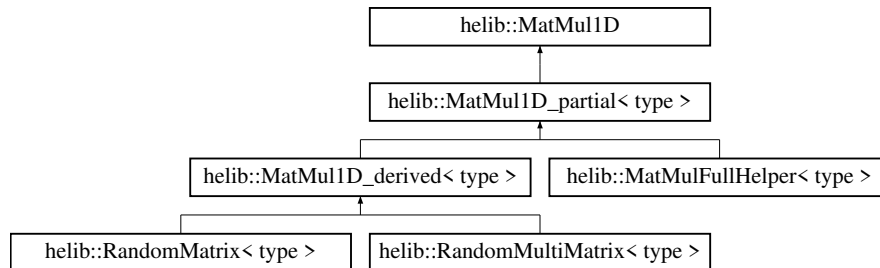
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PAlgebra.h](#)

## 7.78 helib::MatMul1D Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::MatMul1D:



### Public Types

- typedef [MatMul1DExec](#) ExecType

### Public Member Functions

- virtual [~MatMul1D](#) ()
- virtual const [EncryptedArray](#) & [getEA](#) () const =0
- virtual long [getDim](#) () const =0

### 7.78.1 Member Typedef Documentation

#### 7.78.1.1 ExecType

```
typedef MatMul1DExec helib::MatMul1D::ExecType
```

### 7.78.2 Constructor & Destructor Documentation

#### 7.78.2.1 ~MatMul1D()

```
virtual helib::MatMul1D::~~MatMul1D ( ) [inline], [virtual]
```

### 7.78.3 Member Function Documentation

### 7.78.3.1 getDim()

```
virtual long helib::MatMul1D::getDim ( ) const [pure virtual]
```

Implemented in [helib::MatMulFullHelper< type >](#), [helib::RandomMultiMatrix< type >](#), and [helib::RandomMatrix< type >](#).

### 7.78.3.2 getEA()

```
virtual const EncryptedArray& helib::MatMul1D::getEA ( ) const [pure virtual]
```

Implemented in [helib::MatMulFullHelper< type >](#), [helib::RandomMultiMatrix< type >](#), and [helib::RandomMatrix< type >](#).

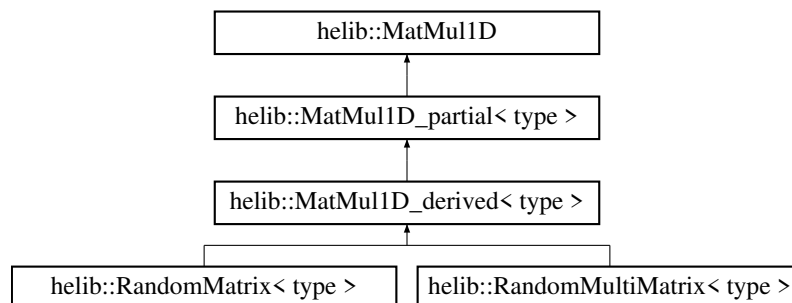
The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/matmul.h](#)

## 7.79 helib::MatMul1D\_derived< type > Class Template Reference

```
#include <matmul.h>
```

Inheritance diagram for `helib::MatMul1D_derived< type >`:



### Public Member Functions

- virtual bool [multipleTransforms](#) ( ) const =0
- virtual bool [get](#) (RX &out, long i, long j, long k) const =0
- void [processDiagonal](#) (RX &poly, long i, const [EncryptedArrayDerived< type >](#) &ea) const override

### Additional Inherited Members

#### 7.79.1 Member Function Documentation

### 7.79.1.1 `get()`

```
template<typename type >
virtual bool helib::MatMul1D\_derived< type >::get (
    RX & out,
    long i,
    long j,
    long k ) const [pure virtual]
```

Implemented in [helib::RandomMultiMatrix< type >](#).

### 7.79.1.2 `multipleTransforms()`

```
template<typename type >
virtual bool helib::MatMul1D\_derived< type >::multipleTransforms ( ) const [pure virtual]
```

Implemented in [helib::RandomMultiMatrix< type >](#), and [helib::RandomMatrix< type >](#).

### 7.79.1.3 `processDiagonal()`

```
template<typename type >
template void helib::MatMul1D\_derived< type >::processDiagonal (
    RX & poly,
    long i,
    const EncryptedArrayDerived< type > & ea ) const [override], [virtual]
```

Implements [helib::MatMul1D\\_partial< type >](#).

The documentation for this class was generated from the following files:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/matmul.h`
- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/src/matmul.cpp`

## 7.80 `helib::MatMul1D_derived_impl< type >` Struct Template Reference

### Static Public Member Functions

- static void [processDiagonal1](#) (RX &poly, long i, const [EncryptedArrayDerived< type > &ea](#), const [MatMul1D\\_derived< type > &mat](#))
- static void [processDiagonal2](#) (RX &poly, long idx, const [EncryptedArrayDerived< type > &ea](#), const [MatMul1D\\_derived< type > &mat](#))
- static void [processDiagonal](#) (RX &poly, long i, const [EncryptedArrayDerived< type > &ea](#), const [MatMul1D\\_derived< type > &mat](#))

## 7.80.1 Member Function Documentation

### 7.80.1.1 processDiagonal()

```
template<typename type >
static void helib::MatMul1D_derived_impl< type >::processDiagonal (
    RX & poly,
    long i,
    const EncryptedArrayDerived< type > & ea,
    const MatMul1D_derived< type > & mat ) [inline], [static]
```

### 7.80.1.2 processDiagonal1()

```
template<typename type >
static void helib::MatMul1D_derived_impl< type >::processDiagonal1 (
    RX & poly,
    long i,
    const EncryptedArrayDerived< type > & ea,
    const MatMul1D_derived< type > & mat ) [inline], [static]
```

### 7.80.1.3 processDiagonal2()

```
template<typename type >
static void helib::MatMul1D_derived_impl< type >::processDiagonal2 (
    RX & poly,
    long idx,
    const EncryptedArrayDerived< type > & ea,
    const MatMul1D_derived< type > & mat ) [inline], [static]
```

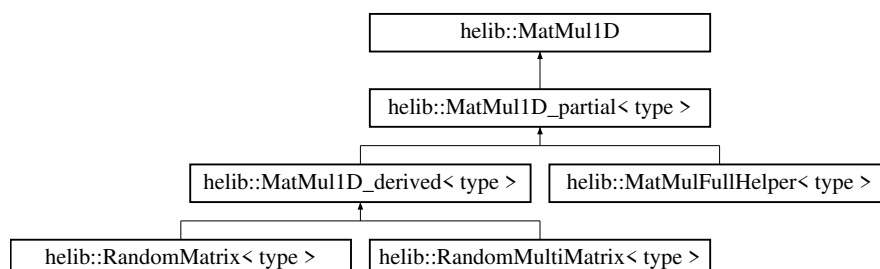
The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.81 helib::MatMul1D\_partial< type > Class Template Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::MatMul1D\_partial< type >:





## Public Member Functions

- virtual void [processDiagonal](#) (RX &poly, long i, const [EncryptedArrayDerived](#)< type > &ea) const =0

## Additional Inherited Members

### 7.81.1 Member Function Documentation

#### 7.81.1.1 processDiagonal()

```
template<typename type >
virtual void helib::MatMul1D\_partial< type >::processDiagonal (
    RX & poly,
    long i,
    const EncryptedArrayDerived< type > & ea ) const [pure virtual]
```

Implemented in [helib::MatMul1D\\_derived](#)< type >, and [helib::MatMulFullHelper](#)< type >.

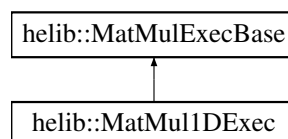
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)

## 7.82 helib::MatMul1DExec Class Reference

```
#include <matmul.h>
```

Inheritance diagram for [helib::MatMul1DExec](#):



## Public Member Functions

- [MatMul1DExec](#) (const [MatMul1D](#) &mat, bool [minimal](#)=false)
- void [mul](#) (Ctxt &ctxt) const override
- void [upgrade](#) () override
- const [EncryptedArray](#) & [getEA](#) () const override

## Public Attributes

- const [EncryptedArray](#) & [ea](#)
- long [dim](#)
- long [D](#)
- bool [native](#)
- bool [minimal](#)
- long [g](#)
- [ConstMultiplierCache](#) [cache](#)
- [ConstMultiplierCache](#) [cache1](#)

## 7.82.1 Constructor & Destructor Documentation

### 7.82.1.1 MatMul1DExec()

```
helib::MatMul1DExec::MatMul1DExec (  
    const MatMul1D & mat,  
    bool minimal = false ) [explicit]
```

## 7.82.2 Member Function Documentation

### 7.82.2.1 getEA()

```
const EncryptedArray& helib::MatMul1DExec::getEA ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.82.2.2 mul()

```
void helib::MatMul1DExec::mul (  
    Ctxt & ctxt ) const [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.82.2.3 upgrade()

```
void helib::MatMul1DExec::upgrade ( ) [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

## 7.82.3 Member Data Documentation

### 7.82.3.1 cache

`ConstMultiplierCache` helib::MatMul1DExec::cache

### 7.82.3.2 cache1

`ConstMultiplierCache` helib::MatMul1DExec::cache1

### 7.82.3.3 D

`long` helib::MatMul1DExec::D

### 7.82.3.4 dim

`long` helib::MatMul1DExec::dim

### 7.82.3.5 ea

`const EncryptedArray&` helib::MatMul1DExec::ea

### 7.82.3.6 g

`long` helib::MatMul1DExec::g

### 7.82.3.7 minimal

`bool` helib::MatMul1DExec::minimal

### 7.82.3.8 native

```
bool helib::MatMul1DExec::native
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.83 helib::MatMul1DExec\_construct< type > Struct Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, const [MatMul1D](#) &mat\_basetype, std::vector< std::shared\_ptr< [ConstMultiplier](#) >> &vec, std::vector< std::shared\_ptr< [ConstMultiplier](#) >> &vec1, long g)

### 7.83.1 Member Function Documentation

#### 7.83.1.1 apply()

```
template<typename type >
static void helib::MatMul1DExec_construct< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    const MatMul1D & mat_basetype,
    std::vector< std::shared_ptr< ConstMultiplier >> & vec,
    std::vector< std::shared_ptr< ConstMultiplier >> & vec1,
    long g ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.84 helib::MatMulFullExec\_construct< type >::MatMulDimComp Struct Reference

### Public Member Functions

- [MatMulDimComp](#) (const [EncryptedArrayDerived](#)< type > \*\_ea)
- bool [operator\(\)](#) (long i, long j)

## Public Attributes

- const [EncryptedArrayDerived](#)< type > \* [ea](#)

## 7.84.1 Constructor & Destructor Documentation

### 7.84.1.1 MatMulDimComp()

```
template<typename type >
helib::MatMulFullExec\_construct< type >::MatMulDimComp (
    const EncryptedArrayDerived< type > * _ea ) [inline]
```

## 7.84.2 Member Function Documentation

### 7.84.2.1 operator()()

```
template<typename type >
bool helib::MatMulFullExec\_construct< type >::MatMulDimComp::operator() (
    long i,
    long j ) [inline]
```

## 7.84.3 Member Data Documentation

### 7.84.3.1 ea

```
template<typename type >
const EncryptedArrayDerived<type>* helib::MatMulFullExec\_construct< type >::MatMulDimComp::ea
```

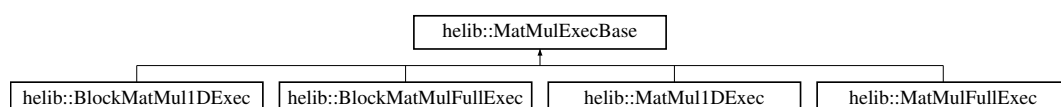
The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[matmul.cpp](#)

## 7.85 helib::MatMulExecBase Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::MatMulExecBase:



## Public Member Functions

- virtual [~MatMulExecBase](#) ()
- virtual const [EncryptedArray](#) & [getEA](#) () const =0
- virtual void [upgrade](#) ()=0
- virtual void [mul](#) ([Ctxt](#) &ctxt) const =0

## 7.85.1 Constructor & Destructor Documentation

### 7.85.1.1 ~MatMulExecBase()

```
virtual helib::MatMulExecBase::~~MatMulExecBase ( ) [inline], [virtual]
```

## 7.85.2 Member Function Documentation

### 7.85.2.1 getEA()

```
virtual const EncryptedArray& helib::MatMulExecBase::getEA ( ) const [pure virtual]
```

Implemented in [helib::BlockMatMulFullExec](#), [helib::MatMulFullExec](#), [helib::BlockMatMul1DExec](#), and [helib::MatMul1DExec](#).

### 7.85.2.2 mul()

```
virtual void helib::MatMulExecBase::mul (
    Ctxt & ctxt ) const [pure virtual]
```

Implemented in [helib::BlockMatMulFullExec](#), [helib::MatMulFullExec](#), [helib::BlockMatMul1DExec](#), and [helib::MatMul1DExec](#).

### 7.85.2.3 upgrade()

```
virtual void helib::MatMulExecBase::upgrade ( ) [pure virtual]
```

Implemented in [helib::BlockMatMulFullExec](#), [helib::MatMulFullExec](#), [helib::BlockMatMul1DExec](#), and [helib::MatMul1DExec](#).

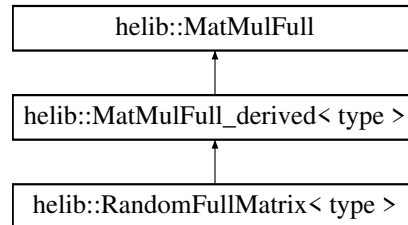
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)

## 7.86 helib::MatMulFull Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::MatMulFull:



### Public Types

- typedef [MatMulFullExec](#) [ExecType](#)

### Public Member Functions

- virtual [~MatMulFull](#) ()
- virtual const [EncryptedArray](#) & [getEA](#) () const =0

### 7.86.1 Member Typedef Documentation

#### 7.86.1.1 ExecType

```
typedef MatMulFullExec helib::MatMulFull::ExecType
```

### 7.86.2 Constructor & Destructor Documentation

#### 7.86.2.1 ~MatMulFull()

```
virtual helib::MatMulFull::~MatMulFull ( ) \[inline\], \[virtual\]
```

### 7.86.3 Member Function Documentation

### 7.86.3.1 getEA()

```
virtual const EncryptedArray& helib::MatMulFull::getEA ( ) const [pure virtual]
```

Implemented in [helib::RandomFullMatrix< type >](#).

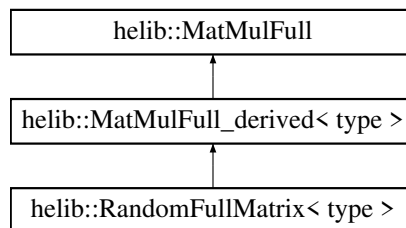
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)

## 7.87 helib::MatMulFull\_derived< type > Class Template Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::MatMulFull\_derived< type >:



### Public Member Functions

- virtual bool [get](#) (RX &out, long i, long j) const =0

### Additional Inherited Members

### 7.87.1 Member Function Documentation

#### 7.87.1.1 get()

```
template<typename type >
virtual bool helib::MatMulFull_derived< type >::get (
    RX & out,
    long i,
    long j ) const [pure virtual]
```

Implemented in [helib::RandomFullMatrix< type >](#).

The documentation for this class was generated from the following file:

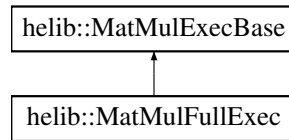
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[matmul.h](#)



## 7.88 helib::MatMulFullExec Class Reference

```
#include <matmul.h>
```

Inheritance diagram for helib::MatMulFullExec:



### Public Member Functions

- [MatMulFullExec](#) (const [MatMulFull](#) &mat, bool [minimal](#)=false)
- void [mul](#) (Ctxt &ctxt) const override
- void [upgrade](#) () override
- const [EncryptedArray](#) & [getEA](#) () const override
- long [rec\\_mul](#) (Ctxt &acc, const Ctxt &ctxt, long dim, long idx) const

### Public Attributes

- const [EncryptedArray](#) & [ea](#)
- bool [minimal](#)
- std::vector< long > [dims](#)
- std::vector< [MatMul1DExec](#) > [transforms](#)

## 7.88.1 Constructor & Destructor Documentation

### 7.88.1.1 MatMulFullExec()

```

helib::MatMulFullExec::MatMulFullExec (
    const MatMulFull & mat,
    bool minimal = false ) [explicit]
  
```

## 7.88.2 Member Function Documentation

### 7.88.2.1 getEA()

```
const EncryptedArray& helib::MatMulFullExec::getEA ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.88.2.2 mul()

```
void helib::MatMulFullExec::mul (
    Ctxt & ctxt ) const [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

### 7.88.2.3 rec\_mul()

```
long helib::MatMulFullExec::rec_mul (
    Ctxt & acc,
    const Ctxt & ctxt,
    long dim,
    long idx ) const
```

### 7.88.2.4 upgrade()

```
void helib::MatMulFullExec::upgrade ( ) [inline], [override], [virtual]
```

Implements [helib::MatMulExecBase](#).

## 7.88.3 Member Data Documentation

### 7.88.3.1 dims

```
std::vector<long> helib::MatMulFullExec::dims
```

### 7.88.3.2 ea

```
const EncryptedArray& helib::MatMulFullExec::ea
```

### 7.88.3.3 minimal

```
bool helib::MatMulFullExec::minimal
```

### 7.88.3.4 transforms

```
std::vector<MatMul1DExec> helib::MatMulFullExec::transforms
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/matmul.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.89 helib::MatMulFullExec\_construct< type > Struct Template Reference

### Classes

- struct [MatMulDimComp](#)

### Static Public Member Functions

- static long [rec\\_mul](#) (long dim, long idx, const std::vector< long > &idxes, std::vector< [MatMul1DExec](#) > &transforms, bool minimal, const std::vector< long > &dims, const [EncryptedArray](#) &ea\_basetype, const [EncryptedArrayDerived](#)< type > &ea, const [MatMulFull\\_derived](#)< type > &mat)
- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, const [EncryptedArray](#) &ea\_basetype, const [MatMulFull](#) &mat\_basetype, std::vector< [MatMul1DExec](#) > &transforms, bool minimal, std::vector< long > &dims)

### 7.89.1 Member Function Documentation

#### 7.89.1.1 apply()

```
template<typename type >
static void helib::MatMulFullExec_construct< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    const EncryptedArray & ea_basetype,
    const MatMulFull & mat_basetype,
    std::vector< MatMul1DExec > & transforms,
    bool minimal,
    std::vector< long > & dims ) [inline], [static]
```

### 7.89.1.2 rec\_mul()

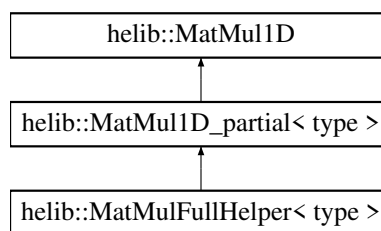
```
template<typename type >
static long helib::MatMulFullExec_construct< type >::rec_mul (
    long dim,
    long idx,
    const std::vector< long > & idxes,
    std::vector< MatMul1DExec > & transforms,
    bool minimal,
    const std::vector< long > & dims,
    const EncryptedArray & ea_basetype,
    const EncryptedArrayDerived< type > & ea,
    const MatMulFull_derived< type > & mat ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.90 helib::MatMulFullHelper< type > Class Template Reference

Inheritance diagram for helib::MatMulFullHelper< type >:



### Public Member Functions

- [MatMulFullHelper](#) (const [EncryptedArray](#) & ea\_basetype, const [MatMulFull\\_derived](#)< type > & \_mat, const std::vector< long > & \_init\_idxes, long \_dim)
- void [processDiagonal](#) (RX &epmat, long offset, const [EncryptedArrayDerived](#)< type > &ea) const override
- const [EncryptedArray](#) & [getEA](#) () const override
- long [getDim](#) () const override

### Public Attributes

- const [EncryptedArray](#) & ea\_basetype
- const [MatMulFull\\_derived](#)< type > & mat
- std::vector< long > init\_idxes
- long dim

### Additional Inherited Members

#### 7.90.1 Constructor & Destructor Documentation

### 7.90.1.1 MatMulFullHelper()

```
template<typename type >
helib::MatMulFullHelper< type >::MatMulFullHelper (
    const EncryptedArray & _ea_basetype,
    const MatMulFull_derived< type > & _mat,
    const std::vector< long > & _init_idxes,
    long _dim ) [inline]
```

## 7.90.2 Member Function Documentation

### 7.90.2.1 getDim()

```
template<typename type >
long helib::MatMulFullHelper< type >::getDim ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D](#).

### 7.90.2.2 getEA()

```
template<typename type >
const EncryptedArray& helib::MatMulFullHelper< type >::getEA ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D](#).

### 7.90.2.3 processDiagonal()

```
template<typename type >
void helib::MatMulFullHelper< type >::processDiagonal (
    RX & epmat,
    long offset,
    const EncryptedArrayDerived< type > & ea ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D\\_partial< type >](#).

## 7.90.3 Member Data Documentation

### 7.90.3.1 dim

```
template<typename type >
long helib::MatMulFullHelper< type >::dim
```

### 7.90.3.2 ea\_basetype

```
template<typename type >
const EncryptedArray& helib::MatMulFullHelper< type >::ea_basetype
```

### 7.90.3.3 init\_idxes

```
template<typename type >
std::vector<long> helib::MatMulFullHelper< type >::init_idxes
```

### 7.90.3.4 mat

```
template<typename type >
const MatMulFull_derived<type>& helib::MatMulFullHelper< type >::mat
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp

## 7.91 helib::ModuliSizes Class Reference

A helper class to map required modulo-sizes to primeSets.

```
#include <primeChain.h>
```

### Public Types

- typedef std::pair< double, [IndexSet](#) > [Entry](#)

### Public Member Functions

- void [init](#) (const std::vector< [Cmodulus](#) > &chain, const [IndexSet](#) &ctxtPrimes, const [IndexSet](#) &smallPrimes)  
*initialize helper table for a given chain*
- [IndexSet](#) [getSet4Size](#) (double low, double high, const [IndexSet](#) &fromSet, bool [reverse](#)) const
- [IndexSet](#) [getSet4Size](#) (double low, double high, const [IndexSet](#) &from1, const [IndexSet](#) &from2, bool [reverse](#)) const
- void [read](#) (std::istream &str)
- void [write](#) (std::ostream &str) const

## Friends

- `std::istream & operator>> (std::istream &s, ModuliSizes &szs)`
- `std::ostream & operator<< (std::ostream &s, const ModuliSizes &szs)`

### 7.91.1 Detailed Description

A helper class to map required modulo-sizes to primeSets.

### 7.91.2 Member Typedef Documentation

#### 7.91.2.1 Entry

```
typedef std::pair<double, IndexSet> helib::ModuliSizes::Entry
```

### 7.91.3 Member Function Documentation

#### 7.91.3.1 `getSet4Size()` [1/2]

```
IndexSet helib::ModuliSizes::getSet4Size (
    double low,
    double high,
    const IndexSet & from1,
    const IndexSet & from2,
    bool reverse ) const
```

Find a suitable [IndexSet](#) of primes whose total size is in the target interval [low,high], trying to minimize the total number of primes dropped from both from1, from2. If no [IndexSet](#) exists that fits in the target interval, returns the [IndexSet](#) that gives the largest value smaller than low. (or the smallest value greater than low if reverse flag is set).

Find a suitable [IndexSet](#) of primes whose total size is in the target interval [low,high], trying to minimize the total number of primes dropped from both from1, from2. If no [IndexSet](#) exists that fits in the target interval, returns the [IndexSet](#) that gives the largest value smaller than low.

#### 7.91.3.2 `getSet4Size()` [2/2]

```
IndexSet helib::ModuliSizes::getSet4Size (
    double low,
    double high,
    const IndexSet & fromSet,
    bool reverse ) const
```

Find a suitable [IndexSet](#) of primes whose total size is in the target interval [low,high], trying to minimize the number of primes dropped from fromSet. If no [IndexSet](#) exists that fits in the target interval, returns the [IndexSet](#) that gives the largest value smaller than low (or the smallest value greater than low if reverse flag is set).

### 7.91.3.3 init()

```
void helib::ModuliSizes::init (
    const std::vector< Cmodulus > & chain,
    const IndexSet & ctxtPrimes,
    const IndexSet & smallPrimes )
```

initialize helper table for a given chain

### 7.91.3.4 read()

```
void helib::ModuliSizes::read (
    std::istream & str )
```

### 7.91.3.5 write()

```
void helib::ModuliSizes::write (
    std::ostream & str ) const
```

## 7.91.4 Friends And Related Function Documentation

### 7.91.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const ModuliSizes & szs ) [friend]
```

### 7.91.4.2 operator>>

```
std::istream& operator>> (
    std::istream & s,
    ModuliSizes & szs ) [friend]
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[primeChain.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[primeChain.cpp](#)



## 7.92 helib::mul\_BlockMatMul1D\_impl< type > Struct Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const [BlockMatMul1D](#) &mat\_basetype)

### 7.92.1 Member Function Documentation

#### 7.92.1.1 apply()

```
template<typename type >
static void helib::mul_BlockMatMul1D_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const BlockMatMul1D & mat_basetype ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.93 helib::mul\_BlockMatMulFull\_impl< type > Struct Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const [BlockMatMulFull](#) &mat\_basetype)

### 7.93.1 Member Function Documentation

#### 7.93.1.1 apply()

```
template<typename type >
static void helib::mul_BlockMatMulFull_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const BlockMatMulFull & mat_basetype ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.94 helib::mul\_MatMul1D\_impl< type > Struct Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const [MatMul1D](#) &mat\_↵  
basetype)

### 7.94.1 Member Function Documentation

#### 7.94.1.1 apply()

```
template<typename type >
static void helib::mul_MatMul1D_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const MatMul1D & mat_basetype ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.95 helib::mul\_MatMulFull\_impl< type > Struct Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const [MatMulFull](#) &mat\_↵  
basetype)

### 7.95.1 Member Function Documentation

#### 7.95.1.1 apply()

```
template<typename type >
static void helib::mul_MatMulFull_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const MatMulFull & mat_basetype ) [inline], [static]
```

The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/matmul.cpp](#)

## 7.96 helib::mul\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)

### 7.96.1 Member Function Documentation

#### 7.96.1.1 apply()

```
template<typename type >
static void helib::mul_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const PlaintextArray & other ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EncryptedArray.cpp](#)

## 7.97 MyClass Class Reference

### Public Member Functions

- [MyClass](#) (int i)
- int [get](#) () const
- void [set](#) (int i)

### 7.97.1 Constructor & Destructor Documentation

#### 7.97.1.1 MyClass()

```
MyClass::MyClass (
    int i ) [inline]
```

### 7.97.2 Member Function Documentation

### 7.97.2.1 get()

```
int MyClass::get ( ) const [inline]
```

### 7.97.2.2 set()

```
void MyClass::set (
    int i ) [inline]
```

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/Test\\_PtrVector.cpp](#)

## 7.98 helib::negate\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa)

### 7.98.1 Member Function Documentation

#### 7.98.1.1 apply()

```
template<typename type >
static void helib::negate_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa ) [inline], [static]
```

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/EncryptedArray.cpp](#)

## 7.99 OtherTimingData Class Reference

### Public Attributes

- double [init2](#)
- double [init4](#)
- double [keyGen](#)
- double [encode2](#)
- double [encode2d](#)
- double [encode4](#)
- double [encode4d](#)
- double [encrypt](#)
- double [decode2](#)
- double [decode4](#)
- double [decrypt](#)

## 7.99.1 Member Data Documentation

### 7.99.1.1 decode2

`double OtherTimingData::decode2`

### 7.99.1.2 decode4

`double OtherTimingData::decode4`

### 7.99.1.3 decrypt

`double OtherTimingData::decrypt`

### 7.99.1.4 encode2

`double OtherTimingData::encode2`

### 7.99.1.5 encode2d

`double OtherTimingData::encode2d`

### 7.99.1.6 encode4

`double OtherTimingData::encode4`

### 7.99.1.7 encode4d

`double OtherTimingData::encode4d`

### 7.99.1.8 encrypt

```
double OtherTimingData::encrypt
```

### 7.99.1.9 init2

```
double OtherTimingData::init2
```

### 7.99.1.10 init4

```
double OtherTimingData::init4
```

### 7.99.1.11 keyGen

```
double OtherTimingData::keyGen
```

The documentation for this class was generated from the following file:

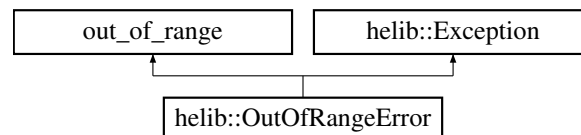
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[Test\\_Timing.cpp](#)

## 7.100 helib::OutOfRangeException Class Reference

Inherits from [Exception](#) and `std::out_of_range`.

```
#include <exceptions.h>
```

Inheritance diagram for `helib::OutOfRangeException`:



### Public Member Functions

- [OutOfRangeException](#) (const std::string &what\_arg)
- [OutOfRangeException](#) (const char \*what\_arg)
- virtual [~OutOfRangeException](#) ()
- virtual const char \* [what](#) () const noexcept override

## Additional Inherited Members

### 7.100.1 Detailed Description

Inherits from [Exception](#) and `std::out_of_range`.

### 7.100.2 Constructor & Destructor Documentation

#### 7.100.2.1 OutOfRangeError() [1/2]

```
helib::OutOfRangeError::OutOfRangeError (
    const std::string & what_arg ) [inline], [explicit]
```

#### 7.100.2.2 OutOfRangeError() [2/2]

```
helib::OutOfRangeError::OutOfRangeError (
    const char * what_arg ) [inline], [explicit]
```

#### 7.100.2.3 ~OutOfRangeError()

```
virtual helib::OutOfRangeError::~~OutOfRangeError ( ) [inline], [virtual]
```

### 7.100.3 Member Function Documentation

#### 7.100.3.1 what()

```
virtual const char* helib::OutOfRangeError::what ( ) const [inline], [override], [virtual],
[noexcept]
```

Implements [helib::Exception](#).

The documentation for this class was generated from the following file:

- `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_v1.0.2/HElib/include/helib/exceptions.h`

## 7.101 helib::PAlgebra Class Reference

The structure of  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$

```
#include <PAlgebra.h>
```

### Public Member Functions

- [PAlgebra](#) (long mm, long pp=2, const std::vector< long > &\_gens=std::vector< long >(), const std::vector< long > &\_ords=std::vector< long >())
- bool [operator==](#) (const [PAlgebra](#) &other) const
- bool [operator!=](#) (const [PAlgebra](#) &other) const
- void [printout](#) () const  
*Prints the structure in a readable form.*
- void [printAll](#) () const
- long [getM](#) () const  
*Returns m.*
- long [getP](#) () const  
*Returns p.*
- long [getPhiM](#) () const  
*Returns  $\phi(m)$*
- long [getOrdP](#) () const  
*The order of p in  $(\mathbb{Z}/m\mathbb{Z})^*$ .*
- long [getNFactors](#) () const  
*The number of distinct prime factors of m.*
- long [getRadM](#) () const  
 *$\text{getRadM}()$  = prod of distinct prime factors of m*
- double [getNormBnd](#) () const  
 *$\text{max-norm-on-pwfl-basis} \leq \text{normBnd} * \text{max-norm-canon-embed}$*
- double [getPolyNormBnd](#) () const  
 *$\text{max-norm-on-pwfl-basis} \leq \text{polyNormBnd} * \text{max-norm-canon-embed}$*
- long [getNSlots](#) () const  
*The number of plaintext slots =  $\phi(m)/\text{ord}(p)$*
- long [getPow2](#) () const  
*if  $m = 2^k$ , then  $\text{pow2} == k$ ; otherwise,  $\text{pow2} == 0$*
- const NTL::ZZX & [getPhimX](#) () const  
*The cyclotomic polynomial  $\Phi_m(X)$*
- void [set\\_cM](#) (double c)  
*The "ring constant" cM.*
- double [get\\_cM](#) () const
- long [numOfGens](#) () const  
*The prime-power factorization of m.*
- long [ZmStarGen](#) (long i) const  
*the i'th generator in  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$  (if any)*
- long [genToPow](#) (long i, long j) const  
*the i'th generator to the power j mod m*
- long [frobeniusPow](#) (long j) const
- long [OrderOf](#) (long i) const  
*The order of i'th generator (if any)*
- long [ProdOrdsFrom](#) (long i) const



- *The product  $\prod_{j=i}^{n-1} \text{OrderOf}(i)$*
- bool [SameOrd](#) (long i) const  
*Is ord(*i*'th generator) the same as its order in  $(\mathbb{Z}/m\mathbb{Z})^*$ ?*
- long [FrobPerturb](#) (long i) const

### Translation between index, representatives, and exponents

- long [ith\\_rep](#) (long i) const  
*Returns the *i*'th element in *T*.*
- long [indexOfRep](#) (long t) const  
*Returns the index of *t* in *T*.*
- bool [isRep](#) (long t) const  
*Is *t* in *T*?*
- long [indexInZmstar](#) (long t) const  
*Returns the index of *t* in  $(\mathbb{Z}/m\mathbb{Z})^*$ .*
- long [indexInZmstar\\_unchecked](#) (long t) const  
*Returns the index of *t* in  $(\mathbb{Z}/m\mathbb{Z})^*$  – no range checking.*
- long [replInZmstar\\_unchecked](#) (long idx) const  
*Returns rep whose index is *i*.*
- bool [inZmStar](#) (long t) const
- long [exponentiate](#) (const std::vector< long > &exps, bool onlySameOrd=false) const  
*Returns  $\prod_i g_i^{\text{exps}[i]} \bmod m$ . If onlySameOrd=true, use only generators that have the same order as in  $(\mathbb{Z}/m\mathbb{Z})^*$ .*
- long [coordinate](#) (long i, long k) const  
*Returns coordinate of index *k* along the *i*'th dimension.*
- std::pair< long, long > [breakIndexByDim](#) (long idx, long dim) const
- long [assembleIndexByDim](#) (std::pair< long, long > idx, long dim) const  
*The inverse of breakIndexByDim.*
- long [addCoord](#) (long i, long k, long offset) const  
*adds offset to index *k* in the *i*'th dimension*
- bool [nextExpVector](#) (std::vector< long > &exps) const
- long [fftSizeNeeded](#) () const  
*The largest FFT we need to handle degree-*m* polynomials.*
- const [PGFFT](#) & [getFFTInfo](#) () const
- const [half\\_FFT](#) & [getHalfFFTInfo](#) () const
- const [quarter\\_FFT](#) & [getQuarterFFTInfo](#) () const

## 7.101.1 Detailed Description

The structure of  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$

A [PAlgebra](#) object is determined by an integer *m* and a prime *p*, where *p* does not divide *m*. It holds information describing the structure of  $(\mathbb{Z}/m\mathbb{Z})^*$ , which is isomorphic to the Galois group over  $A = \mathbb{Z}[X]/\text{Phi}_m(X)$ .

We represent  $(\mathbb{Z}/m\mathbb{Z})^*$  as  $(\mathbb{Z}/m\mathbb{Z})^* = (p) \times (g_1, g_2, \dots) \times (h_1, h_2, \dots)$  where the group generated by *g*<sub>1</sub>, *g*<sub>2</sub>, ... consists of the elements that have the same order in  $(\mathbb{Z}/m\mathbb{Z})^*$  as in  $(\mathbb{Z}/m\mathbb{Z})^* / (p, g_1, \dots, g_{i-1})$ , and *h*<sub>1</sub>, *h*<sub>2</sub>, ... generate the remaining quotient group  $(\mathbb{Z}/m\mathbb{Z})^* / (p, g_1, g_2, \dots)$ .

We let *T* subset  $(\mathbb{Z}/m\mathbb{Z})^*$  be a set of representatives for the quotient group  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$ , defined as  $T = \{ \prod_i g_i^{e_i} * \prod_j h_j^{f_j} \}$  where the *e*<sub>*i*</sub>'s range over 0, 1, ..., ord(*g*<sub>*i*</sub>)-1 and the *f*<sub>*j*</sub>'s range over 0, 1, ..., ord(*h*<sub>*j*</sub>)-1 (these last orders are in  $(\mathbb{Z}/m\mathbb{Z})^* / (p, g_1, g_2, \dots)$ ).

$\text{Phi}_m(X)$  is factored as  $\text{Phi}_m(X) = \prod_{t \in T} F_t(X) \bmod p$ , where the *F*<sub>*t*</sub>'s are irreducible modulo *p*. An arbitrary factor is chosen as *F*<sub>1</sub>, then for each *t* in *T* we associate with the index *t* the factor  $F_t(X) = \text{GCD}(F_1(X^t), \text{Phi}_m(X))$ .

Note that fixing a representation of the field  $R = (\mathbb{Z}/p\mathbb{Z})[X]/F_1(X)$  and letting *z* be a root of *F*<sub>1</sub> in *R* (which is a primitive *m*-th root of unity in *R*), we get that *F*<sub>*t*</sub> is the minimal polynomial of  $z^{1/t}$ .

## 7.101.2 Constructor & Destructor Documentation

### 7.101.2.1 PAlgebra()

```
helib::PAlgebra::PAlgebra (
    long mm,
    long pp = 2,
    const std::vector< long > & _gens = std::vector<long>(),
    const std::vector< long > & _ords = std::vector<long>() )
```

## 7.101.3 Member Function Documentation

### 7.101.3.1 addCoord()

```
long helib::PAlgebra::addCoord (
    long i,
    long k,
    long offset ) const [inline]
```

adds offset to index k in the i'th dimension

### 7.101.3.2 assembleIndexByDim()

```
long helib::PAlgebra::assembleIndexByDim (
    std::pair< long, long > idx,
    long dim ) const [inline]
```

The inverse of breakIndexByDim.

### 7.101.3.3 breakIndexByDim()

```
std::pair<long, long> helib::PAlgebra::breakIndexByDim (
    long idx,
    long dim ) const [inline]
```

Break an index into the hypercube to index of the dimension-dim subcube and index inside that subcube.

#### 7.101.3.4 coordinate()

```
long helib::PAlgebra::coordinate (
    long i,
    long k ) const [inline]
```

Returns coordinate of index k along the i'th dimension.

#### 7.101.3.5 exponentiate()

```
long helib::PAlgebra::exponentiate (
    const std::vector< long > & exps,
    bool onlySameOrd = false ) const
```

Returns  $\prod_i g_i^{\text{exps}[i]} \bmod m$ . If onlySameOrd=true, use only generators that have the same order as in  $(\mathbb{Z}/m\mathbb{Z})^*$ .

#### 7.101.3.6 fftSizeNeeded()

```
long helib::PAlgebra::fftSizeNeeded ( ) const [inline]
```

The largest FFT we need to handle degree-m polynomials.

#### 7.101.3.7 frobeniusPow()

```
long helib::PAlgebra::frobeniusPow (
    long j ) const
```

#### 7.101.3.8 FrobPerturb()

```
long helib::PAlgebra::FrobPerturb (
    long i ) const [inline]
```

#### 7.101.3.9 genToPow()

```
long helib::PAlgebra::genToPow (
    long i,
    long j ) const
```

the i'th generator to the power j mod m

#### 7.101.3.10 `get_cM()`

```
double helib::PAlgebra::get_cM ( ) const [inline]
```

#### 7.101.3.11 `getFFTInfo()`

```
const PGFFT& helib::PAlgebra::getFFTInfo ( ) const [inline]
```

#### 7.101.3.12 `getHalfFFTInfo()`

```
const half_FFT& helib::PAlgebra::getHalfFFTInfo ( ) const [inline]
```

#### 7.101.3.13 `getM()`

```
long helib::PAlgebra::getM ( ) const [inline]
```

Returns m.

#### 7.101.3.14 `getNFactors()`

```
long helib::PAlgebra::getNFactors ( ) const [inline]
```

The number of distinct prime factors of m.

#### 7.101.3.15 `getNormBnd()`

```
double helib::PAlgebra::getNormBnd ( ) const [inline]
```

max-norm-on-pwfl-basis  $\leq$  normBnd \* max-norm-canon-embed

#### 7.101.3.16 `getNSlots()`

```
long helib::PAlgebra::getNSlots ( ) const [inline]
```

The number of plaintext slots =  $\phi(m)/\text{ord}(p)$

**7.101.3.17 getOrdP()**

```
long helib::PAlgebra::getOrdP ( ) const [inline]
```

The order of  $p$  in  $(\mathbb{Z}/m\mathbb{Z})^*$ .

**7.101.3.18 getP()**

```
long helib::PAlgebra::getP ( ) const [inline]
```

Returns  $p$ .

**7.101.3.19 getPhiM()**

```
long helib::PAlgebra::getPhiM ( ) const [inline]
```

Returns  $\phi(m)$

**7.101.3.20 getPhimX()**

```
const NTL::ZZX& helib::PAlgebra::getPhimX ( ) const [inline]
```

The cyclotomix polynomial  $\Phi_m(X)$

**7.101.3.21 getPolyNormBnd()**

```
double helib::PAlgebra::getPolyNormBnd ( ) const [inline]
```

max-norm-on-pwfl-basis  $\leq$  polyNormBnd \* max-norm-canon-embed

**7.101.3.22 getPow2()**

```
long helib::PAlgebra::getPow2 ( ) const [inline]
```

if  $m = 2^k$ , then  $\text{pow2} == k$ ; otherwise,  $\text{pow2} == 0$

#### 7.101.3.23 getQuarterFFTInfo()

```
const quarter_FFT& helib::PAlgebra::getQuarterFFTInfo ( ) const [inline]
```

#### 7.101.3.24 getRadM()

```
long helib::PAlgebra::getRadM ( ) const [inline]
```

`getRadM()` = prod of distinct prime factors of m

#### 7.101.3.25 indexInZmstar()

```
long helib::PAlgebra::indexInZmstar (
    long t ) const [inline]
```

Returns the index of t in  $(\mathbb{Z}/m\mathbb{Z})^*$ .

#### 7.101.3.26 indexInZmstar\_unchecked()

```
long helib::PAlgebra::indexInZmstar_unchecked (
    long t ) const [inline]
```

Returns the index of t in  $(\mathbb{Z}/m\mathbb{Z})^*$  – no range checking.

#### 7.101.3.27 indexOfRep()

```
long helib::PAlgebra::indexOfRep (
    long t ) const [inline]
```

Returns the index of t in T.

#### 7.101.3.28 inZmStar()

```
bool helib::PAlgebra::inZmStar (
    long t ) const [inline]
```

**7.101.3.29 isRep()**

```
bool helib::PAlgebra::isRep (
    long t ) const [inline]
```

Is t in T?

**7.101.3.30 ith\_rep()**

```
long helib::PAlgebra::ith_rep (
    long i ) const [inline]
```

Returns the i'th element in T.

**7.101.3.31 nextExpVector()**

```
bool helib::PAlgebra::nextExpVector (
    std::vector< long > & exps ) const [inline]
```

exps is an array of exponents (the dLog of some t in T), this function increment exps lexicographic order, return false if it cannot be incremented (because it is at its maximum value)

**7.101.3.32 numOfGens()**

```
long helib::PAlgebra::numOfGens ( ) const [inline]
```

The prime-power factorization of m.

The number of generators in  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$

**7.101.3.33 operator"!="()**

```
bool helib::PAlgebra::operator!= (
    const PAlgebra & other ) const [inline]
```

**7.101.3.34 operator==( )**

```
bool helib::PAlgebra::operator== (
    const PAlgebra & other ) const
```

#### 7.101.3.35 OrderOf()

```
long helib::PAlgebra::OrderOf (
    long i ) const [inline]
```

The order of i'th generator (if any)

#### 7.101.3.36 printAll()

```
void helib::PAlgebra::printAll ( ) const
```

#### 7.101.3.37 printout()

```
void helib::PAlgebra::printout ( ) const
```

Prints the structure in a readable form.

#### 7.101.3.38 ProdOrdsFrom()

```
long helib::PAlgebra::ProdOrdsFrom (
    long i ) const [inline]
```

The product  $\prod_{j=i}^{n-1} \text{OrderOf}(i)$

#### 7.101.3.39 repInZmstar\_unchecked()

```
long helib::PAlgebra::repInZmstar_unchecked (
    long idx ) const [inline]
```

Returns rep whose index is i.

#### 7.101.3.40 SameOrd()

```
bool helib::PAlgebra::SameOrd (
    long i ) const [inline]
```

Is ord(i'th generator) the same as its order in  $(\mathbb{Z}/m\mathbb{Z})^{*?}$



**7.101.3.41 set\_cM()**

```
void helib::PAlgebra::set_cM (
    double c ) [inline]
```

The "ring constant" cM.

**7.101.3.42 ZmStarGen()**

```
long helib::PAlgebra::ZmStarGen (
    long i ) const [inline]
```

the i'th generator in  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$  (if any)

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/PAlgebra.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/PAlgebra.cpp

**7.102 helib::PAlgebraMod Class Reference**

The structure of  $\mathbb{Z}[X]/(\Phi_m(X), p)$

```
#include <PAlgebra.h>
```

**Public Member Functions**

- [PAlgebraMod](#) (const [PAlgebra](#) &zMStar, long r)
- template<typename type >  
const [PAlgebraModDerived](#)< type > & [getDerived](#) (type) const
- const [PAlgebraModCx](#) & [getCx](#) () const
- bool [operator==](#) (const [PAlgebraMod](#) &other) const
- bool [operator!=](#) (const [PAlgebraMod](#) &other) const
- [PA\\_tag](#) [getTag](#) () const  
*Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.*
- const [PAlgebra](#) & [getZMStar](#) () const  
*Returns reference to underlying PAlgebra object.*
- const std::vector< NTL::ZZX > & [getFactorsOverZZ](#) () const  
*Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZ's.*
- long [getR](#) () const  
*The value r.*
- long [getPPowR](#) () const  
*The value  $p^r$ .*
- void [restoreContext](#) () const  
*Restores the NTL context for  $p^r$ .*
- [zzX](#) [getMask\\_zzX](#) (long i, long j) const

### 7.102.1 Detailed Description

The structure of  $\mathbb{Z}[X]/(\Phi_m(X), p)$

An object of type [PAlgebraMod](#) stores information about a [PAlgebra](#) object `zMStar`, and an integer `r`. It also provides support for encoding and decoding plaintext slots.

the [PAlgebra](#) object `zMStar` defines  $(\mathbb{Z}/m\mathbb{Z})^{\wedge *}/(0)$ , and the [PAlgebraMod](#) object stores various tables related to the polynomial ring  $\mathbb{Z}/(p^{\wedge r})[X]$ . To do this most efficiently, if `p == 2` and `r == 1`, then these polynomials are represented as GF2X's, and otherwise as `zz_pX`'s. Thus, the types of these objects are not determined until run time. As such, we need to use a class hierarchy, as follows.

- [PAlgebraModBase](#) is a virtual class
- [PAlgebraModDerived<type>](#) is a derived template class, where `type` is either `PA_GF2` or `PA_zz_p`.
- The class [PAlgebraMod](#) is a simple wrapper around a smart pointer to a [PAlgebraModBase](#) object: copying a [PAlgebra](#) object results in a "deep copy" of the underlying object of the derived class. It provides `dDirect` access to the virtual methods of [PAlgebraModBase](#), along with a "downcast" operator to get a reference to the object as a derived type, and also `==` and `!=` operators.

### 7.102.2 Constructor & Destructor Documentation

#### 7.102.2.1 PAlgebraMod()

```
helib::PAlgebraMod::PAlgebraMod (
    const PAlgebra & zMStar,
    long r ) [inline], [explicit]
```

### 7.102.3 Member Function Documentation

#### 7.102.3.1 getCx()

```
const PAlgebraModCx& helib::PAlgebraMod::getCx ( ) const [inline]
```

#### 7.102.3.2 getDerived()

```
template<typename type >
const PAlgebraModDerived<type>& helib::PAlgebraMod::getDerived (
    type ) const [inline]
```

Downcast operator example: `const PAlgebraModDerived<PA_GF2>& rep = alMod.getDerived(PA_GF2());`

### 7.102.3.3 getFactorsOverZZ()

```
const std::vector<NTL::ZZX>& helib::PAlgebraMod::getFactorsOverZZ ( ) const [inline]
```

Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZX's.

### 7.102.3.4 getMask\_zzX()

```
zzX helib::PAlgebraMod::getMask_zzX (
    long i,
    long j ) const [inline]
```

### 7.102.3.5 getPPowR()

```
long helib::PAlgebraMod::getPPowR ( ) const [inline]
```

The value  $p^r$ .

### 7.102.3.6 getR()

```
long helib::PAlgebraMod::getR ( ) const [inline]
```

The value  $r$ .

### 7.102.3.7 getTag()

```
PA_tag helib::PAlgebraMod::getTag ( ) const [inline]
```

Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.

### 7.102.3.8 getZMStar()

```
const PAlgebra& helib::PAlgebraMod::getZMStar ( ) const [inline]
```

Returns reference to underlying PAlgebra object.

### 7.102.3.9 operator"!=()

```
bool helib::PAlgebraMod::operator!= (
    const PAlgebraMod & other ) const [inline]
```

### 7.102.3.10 operator==(

```
bool helib::PAlgebraMod::operator== (
    const PAlgebraMod & other ) const [inline]
```

### 7.102.3.11 restoreContext()

```
void helib::PAlgebraMod::restoreContext ( ) const [inline]
```

Restores the [NTL](#) context for  $p^r$ .

The documentation for this class was generated from the following file:

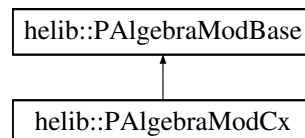
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PAlgebra.h](#)

## 7.103 helib::PAlgebraModBase Class Reference

Virtual base class for [PAlgebraMod](#).

```
#include <PAlgebra.h>
```

Inheritance diagram for helib::PAlgebraModBase:



### Public Member Functions

- virtual [~PAlgebraModBase](#) ()
- virtual [PAlgebraModBase](#) \* [clone](#) () const =0
- virtual [PA\\_tag](#) [getTag](#) () const =0  
Returns the type tag: *PA\_GF2\_tag* or *PA\_zz\_p\_tag*.
- virtual const [PAlgebra](#) & [getZMStar](#) () const =0  
Returns reference to underlying [PAlgebra](#) object.
- virtual const std::vector< [NTL::ZZX](#) > & [getFactorsOverZZ](#) () const =0  
Returns reference to the factorization of  $\text{Phi}_m(X) \bmod p^r$ , but as *ZZX*'s.
- virtual long [getR](#) () const =0  
The value *r*.
- virtual long [getPPowR](#) () const =0  
The value  $p^r$ .
- virtual void [restoreContext](#) () const =0  
Restores the [NTL](#) context for  $p^r$ .
- virtual [zzX](#) [getMask\\_zzX](#) (long i, long j) const =0

### 7.103.1 Detailed Description

Virtual base class for [PAlgebraMod](#).

### 7.103.2 Constructor & Destructor Documentation

#### 7.103.2.1 ~PAlgebraModBase()

```
virtual helib::PAlgebraModBase::~PAlgebraModBase ( ) [inline], [virtual]
```

### 7.103.3 Member Function Documentation

#### 7.103.3.1 clone()

```
virtual PAlgebraModBase* helib::PAlgebraModBase::clone ( ) const [pure virtual]
```

Implemented in [helib::PAlgebraModCx](#).

#### 7.103.3.2 getFactorsOverZZ()

```
virtual const std::vector<NTL::ZZX>& helib::PAlgebraModBase::getFactorsOverZZ ( ) const [pure virtual]
```

Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZ's.

Implemented in [helib::PAlgebraModCx](#).

#### 7.103.3.3 getMask\_zzX()

```
virtual ZZX helib::PAlgebraModBase::getMask_zzX (
    long i,
    long j ) const [pure virtual]
```

#### 7.103.3.4 getPPowR()

```
virtual long helib::PAlgebraModBase::getPPowR ( ) const [pure virtual]
```

The value  $p^r$ .

Implemented in [helib::PAlgebraModCx](#).

#### 7.103.3.5 getR()

```
virtual long helib::PAlgebraModBase::getR ( ) const [pure virtual]
```

The value  $r$ .

Implemented in [helib::PAlgebraModCx](#).

#### 7.103.3.6 getTag()

```
virtual PA_tag helib::PAlgebraModBase::getTag ( ) const [pure virtual]
```

Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.

Implemented in [helib::PAlgebraModCx](#).

#### 7.103.3.7 getZMStar()

```
virtual const PAlgebra& helib::PAlgebraModBase::getZMStar ( ) const [pure virtual]
```

Returns reference to underlying [PAlgebra](#) object.

Implemented in [helib::PAlgebraModCx](#).

#### 7.103.3.8 restoreContext()

```
virtual void helib::PAlgebraModBase::restoreContext ( ) const [pure virtual]
```

Restores the [NTL](#) context for  $p^r$ .

Implemented in [helib::PAlgebraModCx](#).

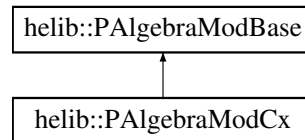
The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PAlgebra.h](#)

## 7.104 helib::PAlgebraModCx Class Reference

```
#include <PAlgebra.h>
```

Inheritance diagram for helib::PAlgebraModCx:



### Public Member Functions

- [PAlgebraModCx](#) (const [PAlgebra](#) &palg, long \_r)
- [PAlgebraModBase](#) \* [clone](#) () const override
- [PA\\_tag](#) [getTag](#) () const override  
*Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.*
- const [PAlgebra](#) & [getZMStar](#) () const override  
*Returns reference to underlying [PAlgebra](#) object.*
- long [getR](#) () const override  
*The value  $r$ .*
- long [getPPowR](#) () const override  
*The value  $p^r$ .*
- void [restoreContext](#) () const override  
*Restores the [NTL](#) context for  $p^r$ .*
- const std::vector< NTL::ZZX > & [getFactorsOverZZ](#) () const override  
*Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZX's.*
- [zzX](#) [getMask\\_zzX](#) ([UNUSED](#) long i, [UNUSED](#) long j) const override

### 7.104.1 Detailed Description

A different derived class to be used for the approximate-numbers scheme This is mostly a dummy class, but needed since the context always has a [PAlgebraMod](#) data member.

### 7.104.2 Constructor & Destructor Documentation

#### 7.104.2.1 PAlgebraModCx()

```

helib::PAlgebraModCx::PAlgebraModCx (
    const PAlgebra & palg,
    long _r ) [inline]
  
```

### 7.104.3 Member Function Documentation

#### 7.104.3.1 clone()

```
PAlgebraModBase* helib::PAlgebraModCx::clone ( ) const [inline], [override], [virtual]
```

Implements [helib::PAlgebraModBase](#).

#### 7.104.3.2 getFactorsOverZZ()

```
const std::vector<NTL::ZZX>& helib::PAlgebraModCx::getFactorsOverZZ ( ) const [inline], [override], [virtual]
```

Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZ's.

Implements [helib::PAlgebraModBase](#).

#### 7.104.3.3 getMask\_zzX()

```
zzX helib::PAlgebraModCx::getMask_zzX (
    UNUSED long i,
    UNUSED long j ) const [inline], [override]
```

#### 7.104.3.4 getPPowR()

```
long helib::PAlgebraModCx::getPPowR ( ) const [inline], [override], [virtual]
```

The value  $p^r$ .

Implements [helib::PAlgebraModBase](#).

#### 7.104.3.5 getR()

```
long helib::PAlgebraModCx::getR ( ) const [inline], [override], [virtual]
```

The value  $r$ .

Implements [helib::PAlgebraModBase](#).



### 7.104.3.6 getTag()

```
PA_tag helib::PAlgebraModCx::getTag ( ) const [inline], [override], [virtual]
```

Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.

Implements [helib::PAlgebraModBase](#).

### 7.104.3.7 getZMStar()

```
const PAlgebra& helib::PAlgebraModCx::getZMStar ( ) const [inline], [override], [virtual]
```

Returns reference to underlying [PAlgebra](#) object.

Implements [helib::PAlgebraModBase](#).

### 7.104.3.8 restoreContext()

```
void helib::PAlgebraModCx::restoreContext ( ) const [inline], [override], [virtual]
```

Restores the [NTL](#) context for  $p^r$ .

Implements [helib::PAlgebraModBase](#).

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PAlgebra.h](#)

## 7.105 helib::PAlgebraModDerived< type > Class Template Reference

A concrete instantiation of the virtual class.

```
#include <PAlgebra.h>
```

## Public Member Functions

- [PAlgebraModDerived](#) (const [PAlgebra](#) &zMStar, long r)
- [PAlgebraModDerived](#) (const [PAlgebraModDerived](#) &other)
- [PAlgebraModDerived](#) & operator= (const [PAlgebraModDerived](#) &other)
- virtual [PAlgebraModBase](#) \* clone () const override  
*Returns a pointer to a "clone".*
- virtual [PA\\_tag](#) getTag () const override  
*Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.*
- virtual const [PAlgebra](#) & getZMStar () const override  
*Returns reference to underlying [PAlgebra](#) object.*
- virtual const std::vector< NTL::ZZX > & getFactorsOverZZ () const override  
*Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZX's.*
- virtual long getR () const override  
*The value  $r$ .*
- virtual long getPPowR () const override  
*The value  $p^r$ .*
- virtual void restoreContext () const override  
*Restores the [NTL](#) context for  $p^r$ .*
- const RXModulus & getPhimXMod () const  
*Returns reference to an RXModulus representing  $\Phi_m(X) \bmod p^r$*
- const vec\_RX & getFactors () const  
*Returns reference to the factors of  $\Phi_m(X)$  modulo  $p^r$ .*
- const vec\_RX & getCrtCoeffs () const  
*Returns the CRT coefficients: element  $i$  contains  $(\prod_{j \neq i} F_j)^{-1} \bmod F_i$ , where  $F_0 F_1 \dots$  is the factorization of  $\Phi_m(X) \bmod p^r$ .*
- const std::vector< std::vector< RX > > & getMaskTable () const  
*Returns ref to maskTable, which is used to implement rotations (in the [EncryptedArray](#) module).*
- [zzX](#) getMask\_zzX (long i, long j) const override

## Embedding in the plaintext slots and decoding back

In all the functions below,  $G$  must be irreducible mod  $p$ , and the order of  $G$  must divide the order of  $p$  modulo  $m$  (as returned by `zMStar.getOrdP()`). In addition, when  $r > 1$ ,  $G$  must be the monomial  $X$  ( $RX(1, 1)$ )

- void [CRT\\_decompose](#) (std::vector< RX > &crt, const RX &H) const  
*Returns a std::vector crt[] such that  $crt[i] = H \bmod Ft$  (with  $t = T[i]$ )*
- void [CRT\\_reconstruct](#) (RX &H, std::vector< RX > &crt) const  
*Returns  $H$  in  $R[X]/\Phi_m(X)$  s.t. for every  $i < nSlots$  and  $t = T[i]$ , we have  $H == crt[i] \bmod Ft$*
- void [mapToSlots](#) ([MappingData](#)< type > &mappingData, const RX &G) const  
*Compute the maps for all the slots. In the current implementation, we if  $r > 1$ , then we must have either  $\deg(G) == 1$  or  $G == factors[0]$ .*
- void [embedInAllSlots](#) (RX &H, const RX &alpha, const [MappingData](#)< type > &mappingData) const  
*Returns  $H$  in  $R[X]/\Phi_m(X)$  s.t. for every  $t$  in  $T$ , the element  $Ht = (H \bmod Ft)$  in  $R[X]/Ft(X)$  represents the same element as  $\alpha$  in  $R[X]/G(X)$ .*
- void [embedInSlots](#) (RX &H, const std::vector< RX > &alphas, const [MappingData](#)< type > &mappingData) const  
*Returns  $H$  in  $R[X]/\Phi_m(X)$  s.t. for every  $t$  in  $T$ , the element  $Ht = (H \bmod Ft)$  in  $R[X]/Ft(X)$  represents the same element as  $alphas[i]$  in  $R[X]/G(X)$ .*
- void [decodePlaintext](#) (std::vector< RX > &alphas, const RX &ptxt, const [MappingData](#)< type > &mappingData) const  
*Return an array such that  $alphas[i]$  in  $R[X]/G(X)$  represent the same element as  $rt = (H \bmod Ft)$  in  $R[X]/Ft(X)$  where  $t = T[i]$ .*
- void [buildLinPolyCoeffs](#) (std::vector< RX > &C, const std::vector< RX > &L, const [MappingData](#)< type > &mappingData) const  
*Returns a coefficient std::vector  $C$  for the linearized polynomial representing  $M$ .*

### 7.105.1 Detailed Description

```
template<typename type>
class helib::PAlgebraModDerived< type >
```

A concrete instantiation of the virtual class.

### 7.105.2 Constructor & Destructor Documentation

#### 7.105.2.1 PAlgebraModDerived() [1/2]

```
template<typename type >
helib::PAlgebraModDerived< type >::PAlgebraModDerived (
    const PAlgebra & zMStar,
    long r )
```

#### 7.105.2.2 PAlgebraModDerived() [2/2]

```
template<typename type >
helib::PAlgebraModDerived< type >::PAlgebraModDerived (
    const PAlgebraModDerived< type > & other ) [inline]
```

### 7.105.3 Member Function Documentation

#### 7.105.3.1 buildLinPolyCoeffs()

```
template<typename type >
void helib::PAlgebraModDerived< type >::buildLinPolyCoeffs (
    std::vector< RX > & C,
    const std::vector< RX > & L,
    const MappingData< type > & mappingData ) const
```

Returns a coefficient std::vector C for the linearized polynomial representing M.

For h in  $\mathbb{Z}/(p^r)[X]$  of degree  $< d$ ,

$$M(h(X) \bmod G) = \sum_{i=0}^{d-1} (C[i] \bmod G) * (h(X^{p^i}) \bmod G).$$

G is assumed to be defined in mappingData, with  $d = \deg(G)$ . L describes a linear map M by describing its action on the standard power basis:  $M(x^j \bmod G) = (L[j] \bmod G)$ , for  $j = 0..d-1$ .

### 7.105.3.2 clone()

```
template<typename type >
virtual PAlgebraModBase* helib::PAlgebraModDerived< type >::clone ( ) const [inline], [override],
[virtual]
```

Returns a pointer to a "clone".

### 7.105.3.3 CRT\_decompose()

```
template<typename type >
void helib::PAlgebraModDerived< type >::CRT_decompose (
    std::vector< RX > & crt,
    const RX & H ) const
```

Returns a `std::vector crt[]` such that  $crt[i] = H \bmod Ft$  (with  $t = T[i]$ )

### 7.105.3.4 CRT\_reconstruct()

```
template<typename type >
void helib::PAlgebraModDerived< type >::CRT_reconstruct (
    RX & H,
    std::vector< RX > & crt ) const
```

Returns  $H$  in  $R[X]/\Phi_m(X)$  s.t. for every  $i < nSlots$  and  $t=T[i]$ , we have  $H == crt[i] \pmod{Ft}$

### 7.105.3.5 decodePlaintext()

```
template<typename type >
void helib::PAlgebraModDerived< type >::decodePlaintext (
    std::vector< RX > & alphas,
    const RX & ptxt,
    const MappingData< type > & mappingData ) const
```

Return an array such that  $alphas[i]$  in  $R[X]/G(X)$  represent the same element as  $rt = (H \bmod Ft)$  in  $R[X]/Ft(X)$  where  $t=T[i]$ .

The `mappingData` argument should contain the output of `mapToSlots(G)`.

### 7.105.3.6 embedInAllSlots()

```
template<typename type >
void helib::PAlgebraModDerived< type >::embedInAllSlots (
    RX & H,
    const RX & alpha,
    const MappingData< type > & mappingData ) const
```

Returns  $H$  in  $R[X]/\Phi_m(X)$  s.t. for every  $t$  in  $T$ , the element  $Ht = (H \bmod Ft)$  in  $R[X]/Ft(X)$  represents the same element as  $alpha$  in  $R[X]/G(X)$ .

Must have  $\deg(alpha) < \deg(G)$ . The `mappingData` argument should contain the output of `mapToSlots(G)`.

**7.105.3.7 embedInSlots()**

```
template<typename type >
void helib::PAlgebraModDerived< type >::embedInSlots (
    RX & H,
    const std::vector< RX > & alphas,
    const MappingData< type > & mappingData ) const
```

Returns H in  $R[X]/\Phi_m(X)$  s.t. for every  $t$  in  $T$ , the element  $H_t = (H \bmod F_t)$  in  $R[X]/F_t(X)$  represents the same element as  $\text{alphas}[i]$  in  $R[X]/G(X)$ .

Must have  $\deg(\text{alpha}[i]) < \deg(G)$ . The `mappingData` argument should contain the output of `mapToSlots(G)`.

**7.105.3.8 getCrtCoeffs()**

```
template<typename type >
const vec_RX& helib::PAlgebraModDerived< type >::getCrtCoeffs ( ) const [inline]
```

Returns the CRT coefficients: element  $i$  contains  $(\prod_{j \neq i} F_j)^{-1} \bmod F_i$ , where  $F_0 F_1 \dots$  is the factorization of  $\Phi_m(X) \bmod p^r$ .

**7.105.3.9 getFactors()**

```
template<typename type >
const vec_RX& helib::PAlgebraModDerived< type >::getFactors ( ) const [inline]
```

Returns reference to the factors of  $\Phi_m(X)$  modulo  $p^r$ .

**7.105.3.10 getFactorsOverZZ()**

```
template<typename type >
virtual const std::vector<NTL::ZZX>& helib::PAlgebraModDerived< type >::getFactorsOverZZ ( )
const [inline], [override], [virtual]
```

Returns reference to the factorization of  $\Phi_m(X) \bmod p^r$ , but as ZZX's.

**7.105.3.11 getMask\_zzX()**

```
template<typename type >
zzX helib::PAlgebraModDerived< type >::getMask_zzX (
    long i,
    long j ) const [inline], [override]
```

**7.105.3.12 getMaskTable()**

```
template<typename type >
const std::vector<std::vector<RX> >& helib::PAlgebraModDerived< type >::getMaskTable ( )
const [inline]
```

Returns ref to maskTable, which is used to implement rotations (in the [EncryptedArray](#) module).

maskTable[i][j] is a polynomial representation of a mask that is 1 in all slots whose i'th coordinate is at least j, and 0 elsewhere. We have:

```
maskTable.size() == zMStar.numOfGens()      // # of generators
for i = 0..maskTable.size()-1:
    maskTable[i].size() == zMStar.OrderOf(i)+1 // order of generator i
```

**7.105.3.13 getPhimXMod()**

```
template<typename type >
const RXModulus& helib::PAlgebraModDerived< type >::getPhimXMod ( ) const [inline]
```

Returns reference to an RXModulus representing  $\Phi_m(X)$  (mod  $p^r$ )

**7.105.3.14 getPPowR()**

```
template<typename type >
virtual long helib::PAlgebraModDerived< type >::getPPowR ( ) const [inline], [override],
[virtual]
```

The value  $p^r$ .

**7.105.3.15 getR()**

```
template<typename type >
virtual long helib::PAlgebraModDerived< type >::getR ( ) const [inline], [override], [virtual]
```

The value  $r$ .

**7.105.3.16 getTag()**

```
template<typename type >
virtual PA_tag helib::PAlgebraModDerived< type >::getTag ( ) const [inline], [override],
[virtual]
```

Returns the type tag: PA\_GF2\_tag or PA\_zz\_p\_tag.

**7.105.3.17 getZMStar()**

```
template<typename type >
virtual const PAlgebra& helib::PAlgebraModDerived< type >::getZMStar ( ) const [inline],
[override], [virtual]
```

Returns reference to underlying [PAlgebra](#) object.

**7.105.3.18 mapToSlots()**

```
template<typename type >
void helib::PAlgebraModDerived< type >::mapToSlots (
    MappingData< type > & mappingData,
    const RX & G ) const
```

Compute the maps for all the slots. In the current implementation, we if  $r > 1$ , then we must have either  $\deg(G) == 1$  or  $G == \text{factors}[0]$ .

**7.105.3.19 operator=()**

```
template<typename type >
PAlgebraModDerived& helib::PAlgebraModDerived< type >::operator= (
    const PAlgebraModDerived< type > & other ) [inline]
```

**7.105.3.20 restoreContext()**

```
template<typename type >
virtual void helib::PAlgebraModDerived< type >::restoreContext ( ) const [inline], [override],
[virtual]
```

Restores the [NTL](#) context for  $p^r$ .

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PAlgebra.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[PAlgebra.cpp](#)

**7.106 helib::PermNetLayer Class Reference**

The information needed to apply one layer of a permutation network.

```
#include <permutations.h>
```

## Public Member Functions

- long [getGenIdx](#) () const
- long [getE](#) () const
- const NTL::Vec< long > & [getShifts](#) () const
- bool [isIdentity](#) () const

## Friends

- class [PermNetwork](#)
- std::ostream & [operator<<](#) (std::ostream &s, const [PermNetwork](#) &net)

## 7.106.1 Detailed Description

The information needed to apply one layer of a permutation network.

## 7.106.2 Member Function Documentation

### 7.106.2.1 [getE\(\)](#)

```
long helib::PermNetLayer::getE ( ) const [inline]
```

### 7.106.2.2 [getGenIdx\(\)](#)

```
long helib::PermNetLayer::getGenIdx ( ) const [inline]
```

### 7.106.2.3 [getShifts\(\)](#)

```
const NTL::Vec<long>& helib::PermNetLayer::getShifts ( ) const [inline]
```

### 7.106.2.4 [isIdentity\(\)](#)

```
bool helib::PermNetLayer::isIdentity ( ) const [inline]
```

## 7.106.3 Friends And Related Function Documentation



### 7.106.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const PermNetwork & net ) [friend]
```

### 7.106.3.2 PermNetwork

```
friend class PermNetwork [friend]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[permutations.h](#)

## 7.107 helib::PermNetwork Class Reference

A full permutation network.

```
#include <permutations.h>
```

### Public Member Functions

- [PermNetwork](#) ()
- [PermNetwork](#) (const [Permut](#) &pi, const [GeneratorTrees](#) &trees)
- long [depth](#) () const
- void [buildNetwork](#) (const [Permut](#) &pi, const [GeneratorTrees](#) &trees)
- void [applyToCtxt](#) ([Ctxt](#) &c, const [EncryptedArray](#) &ea) const  
*Apply network to permute a ciphertext.*
- void [applyToCube](#) ([HyperCube](#)< long > &v) const  
*Apply network to array, used mostly for debugging.*
- void [applyToPtxt](#) (NTL::ZZX &p, const [EncryptedArray](#) &ea) const  
*Apply network to plaintext polynomial, used mostly for debugging.*
- const [PermNetLayer](#) & [getLayer](#) (long i) const

### Friends

- std::ostream & [operator<<](#) (std::ostream &s, const [PermNetwork](#) &net)

### 7.107.1 Detailed Description

A full permutation network.

## 7.107.2 Constructor & Destructor Documentation

### 7.107.2.1 PermNetwork() [1/2]

```
helib::PermNetwork::PermNetwork ( ) [inline]
```

### 7.107.2.2 PermNetwork() [2/2]

```
helib::PermNetwork::PermNetwork (
    const Permut & pi,
    const GeneratorTrees & trees ) [inline]
```

## 7.107.3 Member Function Documentation

### 7.107.3.1 applyToCtxt()

```
void helib::PermNetwork::applyToCtxt (
    Ctxt & c,
    const EncryptedArray & ea ) const
```

Apply network to permute a ciphertext.

### 7.107.3.2 applyToCube()

```
void helib::PermNetwork::applyToCube (
    HyperCube< long > & v ) const
```

Apply network to array, used mostly for debugging.

### 7.107.3.3 applyToPtxt()

```
void helib::PermNetwork::applyToPtxt (
    NTL::ZZX & p,
    const EncryptedArray & ea ) const
```

Apply network to plaintext polynomial, used mostly for debugging.

### 7.107.3.4 buildNetwork()

```
void helib::PermNetwork::buildNetwork (
    const Permut & pi,
    const GeneratorTrees & trees )
```

Take as input a permutation pi and the trees of all the generators, and prepares the permutation network for this pi

### 7.107.3.5 depth()

```
long helib::PermNetwork::depth ( ) const [inline]
```

### 7.107.3.6 getLayer()

```
const PermNetLayer& helib::PermNetwork::getLayer (
    long i ) const [inline]
```

## 7.107.4 Friends And Related Function Documentation

### 7.107.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const PermNetwork & net ) [friend]
```

The documentation for this class was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/permutations.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/PermNetwork.cpp](#)

## 7.108 helib::PGFFT Class Reference

```
#include <PGFFT.h>
```

### Classes

- class [aligned\\_allocator](#)

### Public Types

- template<class T >  
using [aligned\\_vector](#) = std::vector< T, [aligned\\_allocator](#)< T > >

## Public Member Functions

- [PGFFT](#) (long n)
- void [apply](#) (const std::complex< double > \*src, std::complex< double > \*dst) const
- void [apply](#) (std::complex< double > \*v) const
- [PGFFT](#) (const [PGFFT](#) &)=delete
- [PGFFT](#) ([PGFFT](#) &&)=delete
- [PGFFT](#) & [operator=](#) (const [PGFFT](#) &)=delete
- [PGFFT](#) & [operator=](#) ([PGFFT](#) &&)=delete

## Static Public Member Functions

- static bool [simd\\_enabled](#) ()
- static void \* [aligned\\_allocate](#) (std::size\_t n, std::size\_t nelts)
- static void [aligned\\_deallocate](#) (void \*p)

## 7.108.1 Member Typedef Documentation

### 7.108.1.1 aligned\_vector

```
template<class T >
using helib::PGFFT::aligned\_vector = std::vector<T,aligned\_allocator<T> >
```

## 7.108.2 Constructor & Destructor Documentation

### 7.108.2.1 PGFFT() [1/3]

```
helib::PGFFT::PGFFT (
    long n ) [explicit]
```

### 7.108.2.2 PGFFT() [2/3]

```
helib::PGFFT::PGFFT (
    const PGFFT & ) [delete]
```

### 7.108.2.3 PGFFT() [3/3]

```
helib::PGFFT::PGFFT (
    PGFFT && ) [delete]
```

## 7.108.3 Member Function Documentation

### 7.108.3.1 aligned\_allocate()

```
void * helib::PGFFT::aligned_allocate (
    std::size_t n,
    std::size_t nelts ) [static]
```

### 7.108.3.2 aligned\_deallocate()

```
void helib::PGFFT::aligned_deallocate (
    void * p ) [static]
```

### 7.108.3.3 apply() [1/2]

```
void helib::PGFFT::apply (
    const std::complex< double > * src,
    std::complex< double > * dst ) const
```

### 7.108.3.4 apply() [2/2]

```
void helib::PGFFT::apply (
    std::complex< double > * v ) const [inline]
```

### 7.108.3.5 operator=() [1/2]

```
PGFFT& helib::PGFFT::operator= (
    const PGFFT & ) [delete]
```

### 7.108.3.6 operator=() [2/2]

```
PGFFT& helib::PGFFT::operator= (
    PGFFT && ) [delete]
```

### 7.108.3.7 simd\_enabled()

```
bool helib::PGFFT::simd_enabled ( ) [static]
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PGFFT.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[PGFFT.cpp](#)

## 7.109 helib::PlaintextArray Class Reference

```
#include <EncryptedArray.h>
```

### Public Member Functions

- [PlaintextArray](#) (const [EncryptedArray](#) &ea)
- [PlaintextArray](#) (const [PlaintextArray](#) &other)
- [PlaintextArray](#) & [operator=](#) (const [PlaintextArray](#) &other)
- template<typename type >  
std::vector< typename type::RX > & [getData](#) ()
- template<typename type >  
const std::vector< typename type::RX > & [getData](#) () const
- void [print](#) (std::ostream &s) const

### 7.109.1 Constructor & Destructor Documentation

#### 7.109.1.1 PlaintextArray() [1/2]

```
helib::PlaintextArray::PlaintextArray (
    const EncryptedArray & ea ) [inline]
```

#### 7.109.1.2 PlaintextArray() [2/2]

```
helib::PlaintextArray::PlaintextArray (
    const PlaintextArray & other ) [inline]
```

### 7.109.2 Member Function Documentation

**7.109.2.1** `getData()` [1/2]

```
template<typename type >
std::vector<typename type::RX>& helib::PlaintextArray::getData ( ) [inline]
```

**7.109.2.2** `getData()` [2/2]

```
template<typename type >
const std::vector<typename type::RX>& helib::PlaintextArray::getData ( ) const [inline]
```

**7.109.2.3** `operator=()`

```
PlaintextArray& helib::PlaintextArray::operator= (
    const PlaintextArray & other ) [inline]
```

**7.109.2.4** `print()`

```
void helib::PlaintextArray::print (
    std::ostream & s ) const [inline]
```

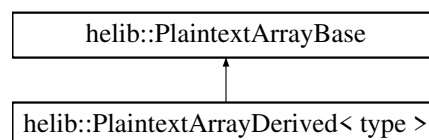
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EncryptedArray.h](#)

**7.110 helib::PlaintextArrayBase Class Reference**

```
#include <EncryptedArray.h>
```

Inheritance diagram for helib::PlaintextArrayBase:

**Public Member Functions**

- virtual [~PlaintextArrayBase](#) ()
- virtual void [print](#) (std::ostream &s) const =0

### 7.110.1 Constructor & Destructor Documentation

#### 7.110.1.1 ~PlaintextArrayBase()

```
virtual helib::PlaintextArrayBase::~~PlaintextArrayBase ( ) [inline], [virtual]
```

### 7.110.2 Member Function Documentation

#### 7.110.2.1 print()

```
virtual void helib::PlaintextArrayBase::print (
    std::ostream & s ) const [pure virtual]
```

Implemented in [helib::PlaintextArrayDerived< type >](#).

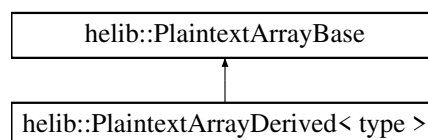
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EncryptedArray.h](#)

## 7.111 helib::PlaintextArrayDerived< type > Class Template Reference

```
#include <EncryptedArray.h>
```

Inheritance diagram for helib::PlaintextArrayDerived< type >:



### Public Member Functions

- virtual void [print](#) (std::ostream &s) const

### Public Attributes

- std::vector< RX > [data](#)



## 7.111.1 Member Function Documentation

### 7.111.1.1 print()

```
template<typename type >
virtual void helib::PlaintextArrayDerived< type >::print (
    std::ostream & s ) const [inline], [virtual]
```

Implements [helib::PlaintextArrayBase](#).

## 7.111.2 Member Data Documentation

### 7.111.2.1 data

```
template<typename type >
std::vector<RX> helib::PlaintextArrayDerived< type >::data
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EncryptedArray.h](#)

## 7.112 helib::PolyMod Class Reference

An object that contains an  $\text{NTL} : \mathbb{Z} \mathbb{Z} X$  polynomial along with a coefficient modulus  $p_{2r}$  and a polynomial modulus  $G$ .

```
#include <PolyMod.h>
```

### Public Member Functions

- [PolyMod](#) ()  
*Default constructor.*
- [PolyMod](#) (const std::shared\_ptr< [PolyModRing](#) > &ringDescriptor)  
*No-data constructor.*
- [PolyMod](#) (long input, const std::shared\_ptr< [PolyModRing](#) > &ringDescriptor)  
*Constant constructor.*
- [PolyMod](#) (const std::vector< long > &input, const std::shared\_ptr< [PolyModRing](#) > &ringDescriptor)  
*Coefficient vector constructor.*
- [PolyMod](#) (const  $\text{NTL} : \mathbb{Z} \mathbb{Z} X$  &input, const std::shared\_ptr< [PolyModRing](#) > &ringDescriptor)  
*Polynomial constructor.*
- [PolyMod](#) (const [PolyMod](#) &input)=default  
*Default copy constructor.*

- [PolyMod](#) ([PolyMod](#) &&input) noexcept=default  
*Default move constructor.*
- [~PolyMod](#) ()=default  
*Default destructor.*
- [PolyMod](#) & [operator=](#) (const [PolyMod](#) &input)=default  
*Assignment operator.*
- [PolyMod](#) & [operator=](#) ([PolyMod](#) &&input)=default  
*default move assignment operator*
- [PolyMod](#) & [operator=](#) (long input)  
*long assignment operator, creates a constant polynomial mod G and p2r.*
- [PolyMod](#) & [operator=](#) (const std::vector< long > &input)  
*std::vector<long> assignment operator, creates a polynomial mod G and p2r.*
- [PolyMod](#) & [operator=](#) (const std::initializer\_list< long > &input)  
*std::initializer\_list<long> assignment operator, creates a polynomial mod G and p2r.*
- [PolyMod](#) & [operator=](#) (const NTL::ZZX &input)  
*NTL::ZZX assignment operator, creates a polynomial mod G and p2r.*
- [operator long](#) () const  
*Explicit conversion to a long.*
- [operator std::vector< long > \(\)](#) const  
*Explicit conversion to std::vector<long> (coefficient vector).*
- [operator NTL::ZZX](#) () const  
*Explicit conversion to an NTL::ZZX.*
- bool [isValid](#) () const  
*Gets the validity of this. This will be false iff this was default constructed.*
- long [getp2r](#) () const  
*Get current  $p^r$  value.*
- NTL::ZZX [getG](#) () const  
*Get current G value.*
- NTL::ZZX [getData](#) () const  
*Getter function that returns the data of [PolyMod](#) as an NTL::ZZX.*
- bool [operator==](#) (const [PolyMod](#) &rhs) const  
*Equals operator between two [PolyMod](#) objects.*
- bool [operator==](#) (long rhs) const  
*Equals operator between a [PolyMod](#) and a long.*
- bool [operator==](#) (const std::vector< long > &rhs) const  
*Equals operator between two [PolyMod](#) objects.*
- bool [operator==](#) (const NTL::ZZX &rhs) const  
*Equals operator between two [PolyMod](#) objects.*
- template<typename T >  
bool [operator!=](#) (T &&rhs) const  
*Not equals operator.*
- [PolyMod](#) & [negate](#) ()  
*Negate function.*
- [PolyMod](#) [operator-](#) () const  
*Unary minus operator.*
- [PolyMod](#) [operator\\*](#) (const [PolyMod](#) &rhs) const  
*Infix multiplication operator.*
- [PolyMod](#) [operator\\*](#) (long rhs) const  
*Infix multiplication operator with long.*
- [PolyMod](#) [operator\\*](#) (const NTL::ZZX &rhs) const  
*Infix multiplication operator with NTL::ZZX.*

- **PolyMod operator+** (const **PolyMod** &rhs) const  
*Infix plus operator.*
- **PolyMod operator+** (long rhs) const  
*Infix plus operator with `long`.*
- **PolyMod operator+** (const NTL::ZZX &rhs) const  
*Infix plus operator with `NTL : :ZZX`.*
- **PolyMod operator-** (const **PolyMod** &rhs) const  
*Infix minus operator.*
- **PolyMod operator-** (long rhs) const  
*Infix minus operator with `long`.*
- **PolyMod operator-** (const NTL::ZZX &rhs) const  
*Infix minus operator with `NTL : :ZZX`.*
- **PolyMod & operator\*=** (const **PolyMod** &otherPoly)  
*Times equals operator with `PolyMod` rhs.*
- **PolyMod & operator\*=** (long scalar)  
*Times equals operator with `long` rhs.*
- **PolyMod & operator\*=** (const NTL::ZZX &otherPoly)  
*Times equals operator with `NTL : :ZZX` rhs.*
- **PolyMod & operator+=** (const **PolyMod** &otherPoly)  
*Plus equals operator with `PolyMod` rhs.*
- **PolyMod & operator+=** (long scalar)  
*Plus equals operator with `long` rhs.*
- **PolyMod & operator+=** (const NTL::ZZX &otherPoly)  
*Plus equals operator with `NTL : :ZZX` rhs.*
- **PolyMod & operator-=** (const **PolyMod** &otherPoly)  
*Minus equals operator with `PolyMod` rhs.*
- **PolyMod & operator-=** (long scalar)  
*Minus equals operator with `long` rhs.*
- **PolyMod & operator-=** (const NTL::ZZX &otherPoly)  
*Minus equals operator with `NTL : :ZZX` rhs.*

## Friends

- std::istream & **operator>>** (std::istream &is, **PolyMod** &poly)  
*Input shift operator.*
- std::ostream & **operator<<** (std::ostream &os, const **PolyMod** &poly)  
*Output shift operator.*

### 7.112.1 Detailed Description

An object that contains an `NTL : :ZZX` polynomial along with a coefficient modulus `p2r` and a polynomial modulus `G`.

A **PolyMod** object can be considered to be an element of  $\mathbb{Z}_{p^r}[x]/G$  where  $p^r$  and  $G$  are the passed-in parameters `p2r` and `G`.

This allows for inter-**PolyMod** operations with the modulo operations performed automatically.

General usage:

```
helib::PolyMod poly(input_data, p2r, G);
```

Calling an operation on a default constructed **PolyMod** will throw an `helib::LogicError`.

## 7.112.2 Constructor & Destructor Documentation

### 7.112.2.1 PolyMod() [1/7]

```
helib::PolyMod::PolyMod ( )
```

Default constructor.

#### Note

[PolyMod](#) objects constructed using this are marked as invalid. If used for any operation whether directly on a [PolyMod](#) or via another object such as [Ptxt](#) this will produce an error.

### 7.112.2.2 PolyMod() [2/7]

```
helib::PolyMod::PolyMod (
    const std::shared_ptr< PolyModRing > & ringDescriptor ) [explicit]
```

No-data constructor.

#### Parameters

<i>ringDescriptor</i>	Descriptor object for the plain-text ring. Contains $p^r$ and $G$ which are to be used for modular reduction.
-----------------------	---

#### Note

This constructor does not take in input data but can be assigned data later via `operator=`.

## 7.112.2.3 PolyMod() [3/7]

```
helib::PolyMod::PolyMod (
    long input,
    const std::shared_ptr< PolyModRing > & ringDescriptor )
```

Constant constructor.

## Parameters

<i>input</i>	Input data as a long.
<i>ringDescriptor</i>	Descriptor object for the plain-text ring. Contains $p^r$ and $G$ which are to be used for modular reduction.

## Note

This constructor accepts input data as a `long` and converts it into an NTL : : ZZ<sub>X</sub> polynomial.

## 7.112.2.4 PolyMod() [4/7]

```
helib::PolyMod::PolyMod (
    const std::vector< long > & input,
    const std::shared_ptr< PolyModRing > & ringDescriptor )
```

Coefficient vector constructor.

## Parameters

<i>input</i>	Input data as a vector of long (the i'th element of the vector corresponds to the coefficient of $x^i$ ).
<i>ringDescriptor</i>	Descriptor object for the plain-text ring. Contains $p^r$ and $G$ which are to be used for modular reduction.

## Note

This constructor accepts input data as a `std::vector<long>` and converts it into an `NTL::ZZX` polynomial.

## 7.112.2.5 PolyMod() [5/7]

```
helib::PolyMod::PolyMod (
    const NTL::ZZX & input,
    const std::shared_ptr< PolyModRing > & ringDescriptor )
```

Polynomial constructor.

## Parameters

<i>input</i>	Input data as an $\text{NTL} \leftarrow \mathbb{Z} \leftarrow \mathbb{Z}X$ .
<i>ringDescriptor</i>	Descriptor object for the plain-text ring. Contains $p^r$ and $G$ which are to be used for modular reduction.

## Note

This constructor accepts input data as an  $\text{NTL} : \mathbb{Z} \mathbb{Z} X$ .

## 7.112.2.6 PolyMod() [6/7]

```
helib::PolyMod::PolyMod (
    const PolyMod & input ) [default]
```

Default copy constructor.

## Parameters

<i>input</i>	<code>PolyMod</code> object that is copied.
--------------	---

## 7.112.2.7 PolyMod() [7/7]

```
helib::PolyMod::PolyMod (
    PolyMod && input ) [default], [noexcept]
```

Default move constructor.

#### 7.112.2.8 `~PolyMod()`

```
helib::PolyMod::~~PolyMod ( ) [default]
```

Default destructor.

### 7.112.3 Member Function Documentation

#### 7.112.3.1 `getData()`

```
NTL::ZZX helib::PolyMod::getData ( ) const
```

Getter function that returns the data of `PolyMod` as an `NTL::ZZX`.

#### 7.112.3.2 `getG()`

```
NTL::ZZX helib::PolyMod::getG ( ) const
```

Get current G value.

##### Returns

The current G value in use.

#### 7.112.3.3 `getp2r()`

```
long helib::PolyMod::getp2r ( ) const
```

Get current  $p^r$  value.

##### Returns

The current  $p^r$  value in use.



**7.112.3.4 isValid()**

```
bool helib::PolyMod::isValid ( ) const
```

Gets the validity of `this`. This will be `false` iff `this` was default constructed.

**Returns**

`true` if `this` is valid, `false` otherwise.

**7.112.3.5 negate()**

```
PolyMod & helib::PolyMod::negate ( )
```

Negate function.

**Returns**

Reference to `*this` post negation.

**7.112.3.6 operator long()**

```
helib::PolyMod::operator long ( ) const [explicit]
```

Explicit conversion to a `long`.

**Note**

This function returns only the constant term even if the polynomial contains higher order terms.

**7.112.3.7 operator NTL::ZZX()**

```
helib::PolyMod::operator NTL::ZZX ( ) const [explicit]
```

Explicit conversion to an `NTL::ZZX`.

**7.112.3.8 operator std::vector< long >()**

```
helib::PolyMod::operator std::vector< long > ( ) const [explicit]
```

Explicit conversion to `std::vector<long>` (coefficient vector).

**7.112.3.9 operator"!="()**

```
template<typename T >
bool helib::PolyMod::operator!= (
    T && rhs ) const [inline]
```

Not equals operator.

**Parameters**

<i>rhs</i>	Right-hand side of comparison.
------------	--------------------------------

**Returns**

`true` if not equal, `false` otherwise

**Note**

Simply forwards to the correct `operator==` method.

**7.112.3.10 `operator*()` [1/3]**

```
PolyMod helib::PolyMod::operator* (
    const NTL::ZZX & rhs ) const
```

Infix multiplication operator with `NTL : : ZZX`.

**Parameters**

<i>rhs</i>	Right hand side of multiplication.
------------	------------------------------------

**Returns**

Product of the two objects.

**7.112.3.11 `operator*()` [2/3]**

```
PolyMod helib::PolyMod::operator* (
    const PolyMod & rhs ) const
```

Infix multiplication operator.

## Parameters

<i>rhs</i>	Right hand side of multiplication.
------------	------------------------------------

## Returns

Product of the two [PolyMod](#) objects.

7.112.3.12 `operator*()` [3/3]

```
PolyMod helib::PolyMod::operator* (
    long rhs ) const
```

Infix multiplication operator with `long`.

## Parameters

<i>rhs</i>	Right hand side of multiplication.
------------	------------------------------------

## Returns

Product of the two values.

7.112.3.13 `operator*=( )` [1/3]

```
PolyMod & helib::PolyMod::operator*= (
    const NTL::ZZX & otherPoly )
```

Times equals operator with `NTL::ZZX` rhs.

## Parameters

<i>otherPoly</i>	Right hand side of multiplication.
------------------	------------------------------------

**Returns**

Reference to `*this` post multiplication.

**7.112.3.14 operator\*=( ) [2/3]**

```
PolyMod & helib::PolyMod::operator*= (
    const PolyMod & otherPoly )
```

Times equals operator with `PolyMod` rhs.

**Parameters**

<i>otherPoly</i>	Right hand side of multiplication.
------------------	------------------------------------

**Returns**

Reference to `*this` post multiplication.

**7.112.3.15 operator\*=( ) [3/3]**

```
PolyMod & helib::PolyMod::operator*= (
    long scalar )
```

Times equals operator with `long` rhs.

**Parameters**

<i>scalar</i>	Right hand side of multiplication.
---------------	------------------------------------

**Returns**

Reference to `*this` post multiplication.

**7.112.3.16 operator+()** [1/3]

```
PolyMod helib::PolyMod::operator+ (  
    const NTL::ZZX & rhs ) const
```

Infix plus operator with NTL : ZZX.

**Parameters**

<i>rhs</i>	Right hand side of addition.
------------	------------------------------

**Returns**

Sum of the two objects.

**7.112.3.17 operator+()** [2/3]

```
PolyMod helib::PolyMod::operator+ (  
    const PolyMod & rhs ) const
```

Infix plus operator.

**Parameters**

<i>rhs</i>	Right hand side of addition.
------------	------------------------------

**Returns**

Sum of the two PolyMod objects.

**7.112.3.18 operator+()** [3/3]

```
PolyMod helib::PolyMod::operator+ (  
    long rhs ) const
```

Infix plus operator with long.

## Parameters

<i>rhs</i>	Right hand side of addition.
------------	------------------------------

## Returns

Sum of the two values.

7.112.3.19 **operator+=()** [1/3]

```
PolyMod & helib::PolyMod::operator+= (
    const NTL::ZZX & otherPoly )
```

Plus equals operator with NTL : : ZZX rhs.

## Parameters

<i>otherPoly</i>	Right hand side of addition.
------------------	------------------------------

## Returns

Reference to `*this` post addition.

7.112.3.20 **operator+=()** [2/3]

```
PolyMod & helib::PolyMod::operator+= (
    const PolyMod & otherPoly )
```

Plus equals operator with PolyMod rhs.

## Parameters

<i>otherPoly</i>	Right hand side of addition.
------------------	------------------------------

**Returns**

Reference to `*this` post addition.

**7.112.3.21 operator+=() [3/3]**

```
PolyMod & helib::PolyMod::operator+= (
    long scalar )
```

Plus equals operator with `long` rhs.

**Parameters**

<i>scalar</i>	Right hand side of addition.
---------------	------------------------------

**Returns**

Reference to `*this` post addition.

**7.112.3.22 operator-() [1/4]**

```
PolyMod helib::PolyMod::operator- ( ) const
```

Unary minus operator.

**Returns**

Negation of the `PolyMod`.

**7.112.3.23 operator-() [2/4]**

```
PolyMod helib::PolyMod::operator- (
    const NTL::ZZX & rhs ) const
```

Infix minus operator with `NTL::ZZX`.

## Parameters

<i>rhs</i>	Right hand side of subtraction.
------------	---------------------------------

## Returns

Difference of the two objects.

**7.112.3.24 operator-() [3/4]**

```
PolyMod helib::PolyMod::operator- (
    const PolyMod & rhs ) const
```

Infix minus operator.

## Parameters

<i>rhs</i>	Right hand side of subtraction.
------------	---------------------------------

## Returns

Difference of the two [PolyMod](#) objects.

**7.112.3.25 operator-() [4/4]**

```
PolyMod helib::PolyMod::operator- (
    long rhs ) const
```

Infix minus operator with `long`.

## Parameters

<i>rhs</i>	Right hand side of subtraction.
------------	---------------------------------



**Returns**

Difference of the two values.

**7.112.3.26 operator-=( ) [1/3]**

```
PolyMod & helib::PolyMod::operator-= (
    const NTL::ZZX & otherPoly )
```

Minus equals operator with NTL : : ZZX rhs.

**Parameters**

<i>otherPoly</i>	Right hand side of subtraction.
------------------	---------------------------------

**Returns**

Reference to \*this post subtraction.

**7.112.3.27 operator-=( ) [2/3]**

```
PolyMod & helib::PolyMod::operator-= (
    const PolyMod & otherPoly )
```

Minus equals operator with PolyMod rhs.

**Parameters**

<i>otherPoly</i>	Right hand side of subtraction.
------------------	---------------------------------

**Returns**

Reference to \*this post subtraction.

**7.112.3.28 operator-=()** [3/3]

```
PolyMod & helib::PolyMod::operator-= (
    long scalar )
```

Minus equals operator with `long` rhs.

**Parameters**

<i>scalar</i>	Right hand side of subtraction.
---------------	---------------------------------

**Returns**

Reference to `*this` post subtraction.

**7.112.3.29 operator=()** [1/6]

```
PolyMod & helib::PolyMod::operator= (
    const NTL::ZZX & input )
```

NTL::ZZX assignment operator, creates a polynomial mod G and p2r.

**Parameters**

<i>input</i>	Polynomial.
--------------	-------------

**7.112.3.30 operator=()** [2/6]

```
PolyMod& helib::PolyMod::operator= (
    const PolyMod & input ) [default]
```

Assignment operator.

**Parameters**

<i>input</i>	Another PolyMod to copy.
--------------	--------------------------

**7.112.3.31 operator=()** [3/6]

```
PolyMod & helib::PolyMod::operator= (
    const std::initializer_list< long > & input )
```

`std::initializer_list<long>` assignment operator, creates a polynomial mod G and p2r.

**Parameters**

<i>input</i>	coefficient vector as an initial- izer list of long (the i'th el- ement of the vector corre- sponds to the coef- ficient of $x^i$ ).
--------------	---

**7.112.3.32 operator=()** [4/6]

```
PolyMod & helib::PolyMod::operator= (
    const std::vector< long > & input )
```

`std::vector<long>` assignment operator, creates a polynomial mod G and p2r.

**Parameters**

<i>input</i>	Coefficient vec- tor of long (the i'th el- ement of the vector corre- sponds to the coef- ficient of $x^i$ ).
--------------	---

**7.112.3.33 operator=()** [5/6]

```
PolyMod & helib::PolyMod::operator= (
    long input )
```

long assignment operator, creates a constant polynomial mod G and p2r.

**Parameters**

<i>input</i>	long datum.
--------------	----------------

**7.112.3.34 operator=()** [6/6]

```
PolyMod& helib::PolyMod::operator= (
    PolyMod && input ) [default]
```

default move assignment operator

**7.112.3.35 operator==( )** [1/4]

```
bool helib::PolyMod::operator== (
    const NTL::ZZX & rhs ) const
```

Equals operator between two [PolyMod](#) objects.

**Parameters**

<i>rhs</i>	Other <a href="#">PolyMod</a> to com- pare to.
------------	---

**Returns**

true if identical, false otherwise.

**Note**

Always returns false when called on invalid [PolyMod](#) objects.

**7.112.3.36 operator==( ) [2/4]**

```
bool helib::PolyMod::operator== (
    const PolyMod & rhs ) const
```

Equals operator between two [PolyMod](#) objects.

**Parameters**

<i>rhs</i>	Other <a href="#">PolyMod</a> to compare to.
------------	--

**Returns**

true if [PolyMod](#) objects are identical, false otherwise.

**7.112.3.37 operator==( ) [3/4]**

```
bool helib::PolyMod::operator== (
    const std::vector< long > & rhs ) const
```

Equals operator between two [PolyMod](#) objects.

**Parameters**

<i>rhs</i>	Other <a href="#">PolyMod</a> to compare to.
------------	--

**Returns**

true if identical, false otherwise.

**Note**

Always returns false when called on invalid [PolyMod](#) objects.

**7.112.3.38 operator==( ) [4/4]**

```
bool helib::PolyMod::operator== (
    long rhs ) const
```

Equals operator between a [PolyMod](#) and a long.

**Parameters**

<i>rhs</i>	long to com- pare the data against.
------------	---

**Returns**

`true` if identical, `false` otherwise.

**Note**

Always returns false when called on invalid [PolyMod](#) objects.

## 7.112.4 Friends And Related Function Documentation

### 7.112.4.1 `operator<<`

```
std::ostream& operator<< (
    std::ostream & os,
    const PolyMod & poly ) [friend]
```

Output shift operator.

**Parameters**

<i>os</i>	Output std↔ ::ostream.
<i>poly</i>	<a href="#">PolyMod</a> object to be writ- ten.

**Returns**

Input `std::ostream` post writing.

**Note**

`p2r` and `G` are not serialised, see note of `operator>>`.

## 7.112.4.2 operator&gt;&gt;

```
std::istream& operator>> (
    std::istream & is,
    PolyMod & poly ) [friend]
```

Input shift operator.

## Parameters

<i>is</i>	Input std← ::istream.
<i>poly</i>	Destination PolyMod object.

## Returns

Input std::istream post reading.

## Note

`poly` must be constructed with an appropriate `p2r` and `G` **BEFORE** calling this function. For example,

```
PolyMod my_poly(p2r, G);
std::cin >> my_poly;
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PolyMod.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[PolyMod.cpp](#)

## 7.113 helib::PolyModRing Struct Reference

Lightweight type for describing the structure of a single slot of the plaintext space.

```
#include <PolyModRing.h>
```

## Public Member Functions

- [PolyModRing](#) ()=delete
- [PolyModRing](#) & [operator=](#) (const [PolyModRing](#) &)=delete
- [PolyModRing](#) & [operator=](#) ([PolyModRing](#) &&)=delete
- [PolyModRing](#) (const [PolyModRing](#) &other)=default  
*Copy constructor.*
- [PolyModRing](#) ([PolyModRing](#) &&other)=default  
*Move constructor.*
- [~PolyModRing](#) ()=default  
*Destructor.*
- [PolyModRing](#) (long [p](#), long [r](#), const NTL::ZZX &G)  
*Constructor.*
- bool [operator==](#) (const [PolyModRing](#) &rhs) const noexcept  
*Equality operator.*
- bool [operator!=](#) (const [PolyModRing](#) &rhs) const noexcept  
*Not-equals operator.*

## Public Attributes

- const long `p`  
*The characteristic of the plaintext space. This should be prime.*
- const long `r`  
*The power of  $p$  used in the plaintext space coefficient modulus.*
- const NTL::ZZX `G`  
*The irreducible factor of  $\Phi_m(X)$  used for the algebra of the individual slots.*
- const long `p2r`  
*The plaintext space coefficient modulus, equal to  $p^r$ .*

## Friends

- std::ostream & `operator<<` (std::ostream &os, const PolyModRing &ring)  
*Output shift operator.*

### 7.113.1 Detailed Description

Lightweight type for describing the structure of a single slot of the plaintext space.

A single slot of the plaintext space is isomorphic to  $\mathbb{Z}[X]/(G(x), p^r)$  for some irreducible factor  $G$  of  $\Phi_m(X)$ , so the main useful members of this `struct` are `p`, `r`, `G`, and `p2r`.

The fields of this `struct` are all `const`, so they should be determined at the time of construction.

#### Note

This `struct` aggregates this often-useful information into a single placeholder for convenience.

### 7.113.2 Constructor & Destructor Documentation

#### 7.113.2.1 PolyModRing() [1/4]

```
helib::PolyModRing::PolyModRing ( ) [delete]
```

#### 7.113.2.2 PolyModRing() [2/4]

```
helib::PolyModRing::PolyModRing (
    const PolyModRing & other ) [default]
```

Copy constructor.



### 7.113.2.3 PolyModRing() [3/4]

```
helib::PolyModRing::PolyModRing (
    PolyModRing && other ) [default]
```

Move constructor.

### 7.113.2.4 ~PolyModRing()

```
helib::PolyModRing::~~PolyModRing ( ) [default]
```

Destructor.

### 7.113.2.5 PolyModRing() [4/4]

```
helib::PolyModRing::PolyModRing (
    long p,
    long r,
    const NTL::ZZX & G )
```

Constructor.

#### Parameters

$p$	The characteristic of the plain-text space.
$r$	The power of $p$ used in the plain-text space coefficient modulus.

## Parameters

<i>G</i>	The irreducible factor of $\text{Phi}_{\leftarrow\_m}(X)$ used for the algebra of the individual slots.
----------	---

$p^r$  will be calculated automatically.

## Note

$p$  should be a prime number.

### 7.113.3 Member Function Documentation

#### 7.113.3.1 `operator!=(())`

```
bool helib::PolyModRing::operator!= (
    const PolyModRing & rhs ) const [noexcept]
```

Not-equals operator.

#### 7.113.3.2 `operator=()` [1/2]

```
PolyModRing& helib::PolyModRing::operator= (
    const PolyModRing & ) [delete]
```

#### 7.113.3.3 `operator=()` [2/2]

```
PolyModRing& helib::PolyModRing::operator= (
    PolyModRing && ) [delete]
```

### 7.113.3.4 operator==( )

```
bool helib::PolyModRing::operator== (
    const PolyModRing & rhs ) const [noexcept]
```

Equality operator.

## 7.113.4 Friends And Related Function Documentation

### 7.113.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const PolyModRing & ring ) [friend]
```

Output shift operator.

#### Parameters

<i>os</i>	Output std:: ::ostream.
<i>ring</i>	PolyModRing object to be writ- ten.

#### Returns

Input std::ostream post writing.

## 7.113.5 Member Data Documentation

### 7.113.5.1 G

```
const NTL::ZZX helib::PolyModRing::G
```

The irreducible factor of  $\Phi_m(X)$  used for the algebra of the individual slots.

### 7.113.5.2 p

```
const long helib::PolyModRing::p
```

The characteristic of the plaintext space. This should be prime.

### 7.113.5.3 p2r

```
const long helib::PolyModRing::p2r
```

The plaintext space coefficient modulus, equal to  $p^r$ .

### 7.113.5.4 r

```
const long helib::PolyModRing::r
```

The power of p used in the plaintext space coefficient modulus.

The documentation for this struct was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PolyModRing.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[PolyModRing.cpp](#)

## 7.114 helib::PowerfulConversion Class Reference

Conversion between powerful representation in  $R_m/(q)$  and  $zz\_pX$ .

```
#include <powerful.h>
```

### Public Member Functions

- [PowerfulConversion](#) ()
- [PowerfulConversion](#) (const [PowerfulTranslationIndexes](#) &ind)
- void [initPConv](#) (const [PowerfulTranslationIndexes](#) &ind)
- void [restoreModulus](#) () const
- const [CubeSignature](#) & [getLongSig](#) () const
- const [CubeSignature](#) & [getShortSig](#) () const
- long [powerfulToPoly](#) (NTL::zz\_pX &poly, const [HyperCube](#)< NTL::zz\_p > &powerful) const
- long [polyToPowerful](#) ([HyperCube](#)< NTL::zz\_p > &powerful, const NTL::zz\_pX &poly) const

### 7.114.1 Detailed Description

Conversion between powerful representation in  $R_m/(q)$  and  $zz\_pX$ .

Usage pattern is as follows:

```
// compute tables for index translation PowerfulTranslationIndexes ind(mvec); // mvec is some factorization of m

// ... set the current zz_p::modulus to something before initializing PowerfulConversion pConv(ind);

// Alternatively use // PowerfulConversion pConv(); pConv.initPConv(ind); // Only the latter call needs zz_p::modulus
// to be defined

// A powerful basis is defined wrt same modulus and cube signature HyperCube<zz_p> powerful(pConv.get↵
ShortSig());

// ... some code here to initialize powerful // code can also do other stuff, perhaps changing zz_p::modulus

pConv.restoreModulus(); // restore zz_p::modulus zz_pX poly; // defined relative to the same modulus pConv.↵
powerfulToPoly(poly, powerful);

// ... some more code here, perhaps changing zz_p::modulus again

pConv.restoreModulus(); // restore zz_p::modulus pConv.polyToPowerful(powerful, poly);
```

### 7.114.2 Constructor & Destructor Documentation

#### 7.114.2.1 PowerfulConversion() [1/2]

```
helib::PowerfulConversion::PowerfulConversion ( ) [inline]
```

#### 7.114.2.2 PowerfulConversion() [2/2]

```
helib::PowerfulConversion::PowerfulConversion (
    const PowerfulTranslationIndexes & ind ) [inline], [explicit]
```

### 7.114.3 Member Function Documentation

#### 7.114.3.1 getLongSig()

```
const CubeSignature& helib::PowerfulConversion::getLongSig ( ) const [inline]
```

### 7.114.3.2 getShortSig()

```
const CubeSignature& helib::PowerfulConversion::getShortSig ( ) const [inline]
```

### 7.114.3.3 initPConv()

```
void helib::PowerfulConversion::initPConv (
    const PowerfulTranslationIndexes & ind ) [inline]
```

### 7.114.3.4 polyToPowerful()

```
long helib::PowerfulConversion::polyToPowerful (
    HyperCube< NTL::zz_p > & powerful,
    const NTL::zz_pX & poly ) const
```

### 7.114.3.5 powerfulToPoly()

```
long helib::PowerfulConversion::powerfulToPoly (
    NTL::zz_pX & poly,
    const HyperCube< NTL::zz_p > & powerful ) const
```

The conversion routines return the value of the modulus  $q$ . It is assumed that the modulus is already set before calling them

### 7.114.3.6 restoreModulus()

```
void helib::PowerfulConversion::restoreModulus ( ) const [inline]
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[powerful.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[powerful.cpp](#)

## 7.115 helib::PowerfulDCRT Class Reference

Conversion between powerful representation, [DoubleCRT](#), and ZZ $X$ .

```
#include <powerful.h>
```

## Public Member Functions

- [PowerfulDCRT](#) (const [Context](#) &\_context, const NTL::Vec< long > &mvec)
- const [PowerfulTranslationIndexes](#) & [getIndexTranslation](#) () const
- const [PowerfulConversion](#) & [getPConv](#) (long i) const
- void [dcrtToPowerful](#) (NTL::Vec< NTL::ZZ > &powerful, const [DoubleCRT](#) &dcrt) const
- void [ZZXtoPowerful](#) (NTL::Vec< NTL::ZZ > &powerful, const NTL::ZZX &poly) const
- void [powerfulToZZX](#) (NTL::ZZX &poly, const NTL::Vec< NTL::ZZ > &powerful) const

### 7.115.1 Detailed Description

Conversion between powerful representation, [DoubleCRT](#), and ZZ<sub>X</sub>.

### 7.115.2 Constructor & Destructor Documentation

#### 7.115.2.1 PowerfulDCRT()

```
helib::PowerfulDCRT::PowerfulDCRT (
    const Context & _context,
    const NTL::Vec< long > & mvec )
```

### 7.115.3 Member Function Documentation

#### 7.115.3.1 dcrtToPowerful()

```
void helib::PowerfulDCRT::dcrtToPowerful (
    NTL::Vec< NTL::ZZ > & powerful,
    const DoubleCRT & dcrt ) const
```

#### 7.115.3.2 getIndexTranslation()

```
const PowerfulTranslationIndexes& helib::PowerfulDCRT::getIndexTranslation ( ) const [inline]
```

#### 7.115.3.3 getPConv()

```
const PowerfulConversion& helib::PowerfulDCRT::getPConv (
    long i ) const [inline]
```

### 7.115.3.4 powerfulToZZX()

```
void helib::PowerfulDCRT::powerfulToZZX (
    NTL::ZZX & poly,
    const NTL::Vec< NTL::ZZ > & powerful ) const
```

### 7.115.3.5 ZZXtoPowerful()

```
void helib::PowerfulDCRT::ZZXtoPowerful (
    NTL::Vec< NTL::ZZ > & powerful,
    const NTL::ZZX & poly ) const
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[powerful.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[powerful.cpp](#)

## 7.116 helib::PowerfulTranslationIndexes Class Reference

Holds index tables for translation between powerful and zz\_pX.

```
#include <powerful.h>
```

### Public Member Functions

- [PowerfulTranslationIndexes](#) (const NTL::Vec< long > &mv)

### Public Attributes

- long [m](#)
- long [phim](#)
- NTL::Vec< long > [mvec](#)
- NTL::Vec< long > [phivec](#)
- NTL::Vec< long > [divvec](#)
- NTL::Vec< long > [invvec](#)
- [CubeSignature](#) [longSig](#)
- [CubeSignature](#) [shortSig](#)
- NTL::Vec< long > [polyToCubeMap](#)
- NTL::Vec< long > [cubeToPolyMap](#)
- NTL::Vec< long > [shortToLongMap](#)
- NTL::Vec< NTL::ZZX > [cycVec](#)
- NTL::ZZX [phimX](#)

### 7.116.1 Detailed Description

Holds index tables for translation between powerful and zz\_pX.



## 7.116.2 Constructor & Destructor Documentation

### 7.116.2.1 PowerfulTranslationIndexes()

```
helib::PowerfulTranslationIndexes::PowerfulTranslationIndexes (  
    const NTL::Vec< long > & mv )
```

## 7.116.3 Member Data Documentation

### 7.116.3.1 cubeToPolyMap

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::cubeToPolyMap
```

### 7.116.3.2 cycVec

```
NTL::Vec<NTL::ZZX> helib::PowerfulTranslationIndexes::cycVec
```

### 7.116.3.3 divvec

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::divvec
```

### 7.116.3.4 invvec

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::invvec
```

### 7.116.3.5 longSig

```
CubeSignature helib::PowerfulTranslationIndexes::longSig
```

**7.116.3.6 m**

```
long helib::PowerfulTranslationIndexes::m
```

**7.116.3.7 mvec**

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::mvec
```

**7.116.3.8 phim**

```
long helib::PowerfulTranslationIndexes::phim
```

**7.116.3.9 phimX**

```
NTL::ZZX helib::PowerfulTranslationIndexes::phimX
```

**7.116.3.10 phivec**

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::phivec
```

**7.116.3.11 polyToCubeMap**

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::polyToCubeMap
```

**7.116.3.12 shortSig**

```
CubeSignature helib::PowerfulTranslationIndexes::shortSig
```

### 7.116.3.13 shortToLongMap

```
NTL::Vec<long> helib::PowerfulTranslationIndexes::shortToLongMap
```

The documentation for this class was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/powerful.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/powerful.cpp](#)

## 7.117 helib::PrimeGenerator Struct Reference

### Public Member Functions

- [PrimeGenerator](#) (long \_len, long \_m)
- long [next](#) ()

### Public Attributes

- long [len](#)
- long [m](#)
- long [k](#)
- long [t](#)

## 7.117.1 Constructor & Destructor Documentation

### 7.117.1.1 PrimeGenerator()

```
helib::PrimeGenerator::PrimeGenerator (
    long _len,
    long _m ) [inline]
```

## 7.117.2 Member Function Documentation

### 7.117.2.1 next()

```
long helib::PrimeGenerator::next ( ) [inline]
```

## 7.117.3 Member Data Documentation

**7.117.3.1 k**

```
long helib::PrimeGenerator::k
```

**7.117.3.2 len**

```
long helib::PrimeGenerator::len
```

**7.117.3.3 m**

```
long helib::PrimeGenerator::m
```

**7.117.3.4 t**

```
long helib::PrimeGenerator::t
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[primeChain.cpp](#)

**7.118 helib::print\_pa\_impl< type > Class Template Reference****Static Public Member Functions**

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, std::ostream &s, const [PlaintextArray](#) &pa)

**7.118.1 Member Function Documentation****7.118.1.1 apply()**

```
template<typename type >
static void helib::print_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    std::ostream & s,
    const PlaintextArray & pa ) [inline], [static]
```

The documentation for this class was generated from the following file:

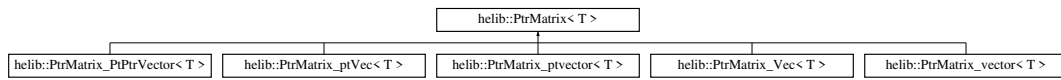
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EncryptedArray.cpp](#)

## 7.119 helib::PtrMatrix< T > Struct Template Reference

An abstract class for an array of PtrVectors.

```
#include <PtrMatrix.h>
```

Inheritance diagram for helib::PtrMatrix< T >:



### Public Member Functions

- virtual [PtrVector](#)< T > & [operator\[\]](#) (long)=0
- virtual const [PtrVector](#)< T > & [operator\[\]](#) (long) const =0
- virtual long [size](#) () const =0
- virtual void [resize](#) (long newSize)
- virtual [~PtrMatrix](#) ()
- virtual const T \* [ptr2nonNull](#) () const

### 7.119.1 Detailed Description

```
template<typename T>
struct helib::PtrMatrix< T >
```

An abstract class for an array of PtrVectors.

### 7.119.2 Constructor & Destructor Documentation

#### 7.119.2.1 ~PtrMatrix()

```
template<typename T >
virtual helib::PtrMatrix< T >::~~PtrMatrix ( ) [inline], [virtual]
```

### 7.119.3 Member Function Documentation

**7.119.3.1 operator[]()** [1/2]

```
template<typename T >
virtual const PtrVector<T>& helib::PtrMatrix< T >::operator[] (
    long ) const [pure virtual]
```

Implemented in [helib::PtrMatrix\\_PtPtrVector< T >](#), [helib::PtrMatrix\\_ptvector< T >](#), [helib::PtrMatrix\\_vector< T >](#), [helib::PtrMatrix\\_ptVec< T >](#), and [helib::PtrMatrix\\_Vec< T >](#).

**7.119.3.2 operator[]()** [2/2]

```
template<typename T >
virtual PtrVector<T>& helib::PtrMatrix< T >::operator[] (
    long ) [pure virtual]
```

Implemented in [helib::PtrMatrix\\_PtPtrVector< T >](#), [helib::PtrMatrix\\_ptvector< T >](#), [helib::PtrMatrix\\_vector< T >](#), [helib::PtrMatrix\\_ptVec< T >](#), and [helib::PtrMatrix\\_Vec< T >](#).

**7.119.3.3 ptr2nonNull()**

```
template<typename T >
virtual const T* helib::PtrMatrix< T >::ptr2nonNull ( ) const [inline], [virtual]
```

**7.119.3.4 resize()**

```
template<typename T >
virtual void helib::PtrMatrix< T >::resize (
    long newSize ) [inline], [virtual]
```

Reimplemented in [helib::PtrMatrix\\_vector< T >](#), and [helib::PtrMatrix\\_Vec< T >](#).

**7.119.3.5 size()**

```
template<typename T >
virtual long helib::PtrMatrix< T >::size ( ) const [pure virtual]
```

Implemented in [helib::PtrMatrix\\_PtPtrVector< T >](#), [helib::PtrMatrix\\_ptvector< T >](#), [helib::PtrMatrix\\_vector< T >](#), [helib::PtrMatrix\\_ptVec< T >](#), and [helib::PtrMatrix\\_Vec< T >](#).

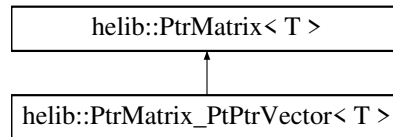
The documentation for this struct was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PtrMatrix.h](#)

## 7.120 helib::PtrMatrix\_PtPtrVector< T > Struct Template Reference

An implementation of [PtrMatrix](#) using `vector< PtrVector<T>*>`

Inheritance diagram for `helib::PtrMatrix_PtPtrVector< T >`:



### Public Member Functions

- [PtrMatrix\\_PtPtrVector](#) (`std::vector< PtrVector< T > * > &mat`)
- [PtrVector](#)< T > & [operator\[\]](#) (long i) override
- const [PtrVector](#)< T > & [operator\[\]](#) (long i) const override
- long [size](#) () const override

### Public Attributes

- `std::vector< PtrVector< T > * > & rows`

### 7.120.1 Detailed Description

```
template<typename T>
struct helib::PtrMatrix_PtPtrVector< T >
```

An implementation of [PtrMatrix](#) using `vector< PtrVector<T>*>`

### 7.120.2 Constructor & Destructor Documentation

#### 7.120.2.1 PtrMatrix\_PtPtrVector()

```
template<typename T >
helib::PtrMatrix_PtPtrVector< T >::PtrMatrix_PtPtrVector (
    std::vector< PtrVector< T > * > & mat ) [inline]
```

### 7.120.3 Member Function Documentation

**7.120.3.1 operator[]()** [1/2]

```
template<typename T >
const PtrVector<T>& helib::PtrMatrix_PtPtrVector< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.120.3.2 operator[]()** [2/2]

```
template<typename T >
PtrVector<T>& helib::PtrMatrix_PtPtrVector< T >::operator[] (
    long i ) [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.120.3.3 size()**

```
template<typename T >
long helib::PtrMatrix_PtPtrVector< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.120.4 Member Data Documentation****7.120.4.1 rows**

```
template<typename T >
std::vector<PtrVector<T>*>& helib::PtrMatrix_PtPtrVector< T >::rows
```

The documentation for this struct was generated from the following file:

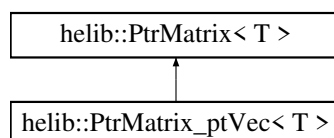
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[binaryArith.cpp](#)

**7.121 helib::PtrMatrix\_ptVec< T > Struct Template Reference**

An implementation of [PtrMatrix](#) using `Vec< Vec<T>*>`

```
#include <PtrMatrix.h>
```

Inheritance diagram for `helib::PtrMatrix_ptVec< T >`:





## Public Member Functions

- [PtrMatrix\\_ptVec](#) (NTL::Vec< NTL::Vec< T > \* > &mat)
- [PtrVector](#)< T > & [operator\[\]](#) (long i) override
- const [PtrVector](#)< T > & [operator\[\]](#) (long i) const override
- long [size](#) () const override

## Public Attributes

- NTL::Vec< NTL::Vec< T > \* > & [buffer](#)
- std::vector< [PtrVector\\_VecT](#)< T > > [rows](#)

### 7.121.1 Detailed Description

```
template<typename T>
struct helib::PtrMatrix_ptVec< T >
```

An implementation of [PtrMatrix](#) using `Vec< Vec<T>* >`

### 7.121.2 Constructor & Destructor Documentation

#### 7.121.2.1 PtrMatrix\_ptVec()

```
template<typename T >
helib::PtrMatrix_ptVec< T >::PtrMatrix_ptVec (
    NTL::Vec< NTL::Vec< T > * > & mat ) [inline]
```

### 7.121.3 Member Function Documentation

#### 7.121.3.1 operator[]() [1/2]

```
template<typename T >
const PtrVector<T>& helib::PtrMatrix\_ptVec< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix](#)< T >.

### 7.121.3.2 operator[]() [2/2]

```
template<typename T >
PtrVector<T>& helib::PtrMatrix_ptVec< T >::operator[] (
    long i ) [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

### 7.121.3.3 size()

```
template<typename T >
long helib::PtrMatrix_ptVec< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

## 7.121.4 Member Data Documentation

### 7.121.4.1 buffer

```
template<typename T >
NTL::Vec<NTL::Vec<T>*>& helib::PtrMatrix_ptVec< T >::buffer
```

### 7.121.4.2 rows

```
template<typename T >
std::vector<PtrVector_VecT<T> > helib::PtrMatrix_ptVec< T >::rows
```

The documentation for this struct was generated from the following file:

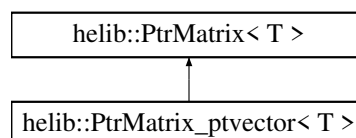
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PtrMatrix.h](#)

## 7.122 helib::PtrMatrix\_ptvector< T > Struct Template Reference

An implementation of [PtrMatrix](#) using `vector< vector<T>*>`

```
#include <PtrMatrix.h>
```

Inheritance diagram for `helib::PtrMatrix_ptvector< T >`:



## Public Member Functions

- [PtrMatrix\\_ptvector](#) (std::vector< std::vector< T > \* > &mat)
- [PtrVector](#)< T > & [operator\[\]](#) (long i) override
- const [PtrVector](#)< T > & [operator\[\]](#) (long i) const override
- long [size](#) () const override

## Public Attributes

- std::vector< std::vector< T > \* > & [buffer](#)
- std::vector< [PtrVector\\_vectorT](#)< T > > [rows](#)

### 7.122.1 Detailed Description

```
template<typename T>
struct helib::PtrMatrix_ptvector< T >
```

An implementation of [PtrMatrix](#) using vector< vector<T>\* >

### 7.122.2 Constructor & Destructor Documentation

#### 7.122.2.1 PtrMatrix\_ptvector()

```
template<typename T >
helib::PtrMatrix_ptvector< T >::PtrMatrix_ptvector (
    std::vector< std::vector< T > * > & mat ) [inline]
```

### 7.122.3 Member Function Documentation

#### 7.122.3.1 operator[]() [1/2]

```
template<typename T >
const PtrVector<T>& helib::PtrMatrix\_ptvector< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix](#)< T >.

**7.122.3.2 operator[]()** [2/2]

```
template<typename T >
PtrVector<T>& helib::PtrMatrix_ptvector< T >::operator[] (
    long i ) [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.122.3.3 size()**

```
template<typename T >
long helib::PtrMatrix_ptvector< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.122.4 Member Data Documentation****7.122.4.1 buffer**

```
template<typename T >
std::vector<std::vector<T>*>& helib::PtrMatrix_ptvector< T >::buffer
```

**7.122.4.2 rows**

```
template<typename T >
std::vector<PtrVector_vectorT<T> > helib::PtrMatrix_ptvector< T >::rows
```

The documentation for this struct was generated from the following file:

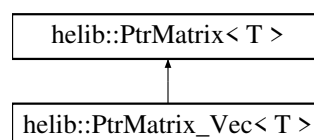
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PtrMatrix.h](#)

**7.123 helib::PtrMatrix\_Vec< T > Struct Template Reference**

An implementation of [PtrMatrix](#) using `Vec< Vec<T> >`

```
#include <PtrMatrix.h>
```

Inheritance diagram for `helib::PtrMatrix_Vec< T >`:



## Public Member Functions

- [PtrMatrix\\_Vec](#) (NTL::Vec< NTL::Vec< T >> &mat)
- [PtrVector](#)< T > & [operator\[\]](#) (long i) override
- const [PtrVector](#)< T > & [operator\[\]](#) (long i) const override
- long [size](#) () const override
- void [resize](#) (long newSize) override

## Public Attributes

- NTL::Vec< NTL::Vec< T >> & [buffer](#)
- std::vector< [PtrVector\\_VecT](#)< T >> [rows](#)

### 7.123.1 Detailed Description

```
template<typename T>
struct helib::PtrMatrix_Vec< T >
```

An implementation of [PtrMatrix](#) using `Vec< Vec<T>>`

### 7.123.2 Constructor & Destructor Documentation

#### 7.123.2.1 PtrMatrix\_Vec()

```
template<typename T >
helib::PtrMatrix_Vec< T >::PtrMatrix_Vec (
    NTL::Vec< NTL::Vec< T >> & mat ) [inline]
```

### 7.123.3 Member Function Documentation

#### 7.123.3.1 operator[]() [1/2]

```
template<typename T >
const PtrVector<T>& helib::PtrMatrix\_Vec< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix](#)< T >.

**7.123.3.2 operator[]()** [2/2]

```
template<typename T >
PtrVector<T>& helib::PtrMatrix_Vec< T >::operator[] (
    long i ) [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.123.3.3 resize()**

```
template<typename T >
void helib::PtrMatrix_Vec< T >::resize (
    long newSize ) [inline], [override], [virtual]
```

Reimplemented from [helib::PtrMatrix< T >](#).

**7.123.3.4 size()**

```
template<typename T >
long helib::PtrMatrix_Vec< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

**7.123.4 Member Data Documentation****7.123.4.1 buffer**

```
template<typename T >
Ntl::Vec<Ntl::Vec<T> >& helib::PtrMatrix_Vec< T >::buffer
```

**7.123.4.2 rows**

```
template<typename T >
std::vector<PtrVector_VecT<T> > helib::PtrMatrix_Vec< T >::rows
```

The documentation for this struct was generated from the following file:

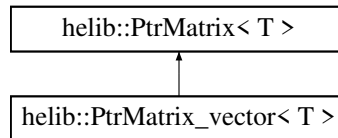
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PtrMatrix.h](#)

## 7.124 helib::PtrMatrix\_vector< T > Struct Template Reference

An implementation of [PtrMatrix](#) using `vector< vector<T> >`

```
#include <PtrMatrix.h>
```

Inheritance diagram for `helib::PtrMatrix_vector< T >`:



### Public Member Functions

- [PtrMatrix\\_vector](#) (`std::vector< std::vector< T >> &mat`)
- [PtrVector< T > & operator\[\]](#) (`long i`) override
- `const` [PtrVector< T > & operator\[\]](#) (`long i`) `const` override
- `long` [size](#) () `const` override
- `void` [resize](#) (`long newSize`) override

### Public Attributes

- `std::vector< std::vector< T > > &` [buffer](#)
- `std::vector< PtrVector\_vectorT< T > >` [rows](#)

### 7.124.1 Detailed Description

```
template<typename T>
struct helib::PtrMatrix_vector< T >
```

An implementation of [PtrMatrix](#) using `vector< vector<T> >`

### 7.124.2 Constructor & Destructor Documentation

#### 7.124.2.1 PtrMatrix\_vector()

```
template<typename T >
helib::PtrMatrix_vector< T >::PtrMatrix_vector (
    std::vector< std::vector< T >> & mat ) [inline]
```

### 7.124.3 Member Function Documentation

#### 7.124.3.1 operator[]() [1/2]

```
template<typename T >
const PtrVector<T>& helib::PtrMatrix_vector< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

#### 7.124.3.2 operator[]() [2/2]

```
template<typename T >
PtrVector<T>& helib::PtrMatrix_vector< T >::operator[] (
    long i ) [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

#### 7.124.3.3 resize()

```
template<typename T >
void helib::PtrMatrix_vector< T >::resize (
    long newSize ) [inline], [override], [virtual]
```

Reimplemented from [helib::PtrMatrix< T >](#).

#### 7.124.3.4 size()

```
template<typename T >
long helib::PtrMatrix_vector< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrMatrix< T >](#).

### 7.124.4 Member Data Documentation

#### 7.124.4.1 buffer

```
template<typename T >
std::vector<std::vector<T> >& helib::PtrMatrix_vector< T >::buffer
```



#### 7.124.4.2 rows

```
template<typename T >
std::vector<PtrVector_vectorT<T> > helib::PtrMatrix_vector< T >::rows
```

The documentation for this struct was generated from the following file:

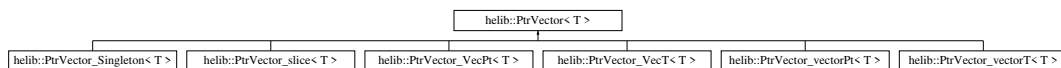
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PtrMatrix.h](#)

## 7.125 helib::PtrVector< T > Struct Template Reference

Abstract class for an array of objects.

```
#include <PtrVector.h>
```

Inheritance diagram for helib::PtrVector< T >:



### Public Member Functions

- virtual T \* [operator\[\]](#) (long) const =0
- virtual long [size](#) () const =0
- virtual void [resize](#) (long newSize, [UNUSED](#) const [PtrVector](#) \*another=nullptr)
- virtual [~PtrVector](#) ()
- bool [isSet](#) (long i) const
- virtual long [numNonNull](#) (long first=0, long last=LONG\_MAX) const
- virtual const T \* [ptr2nonNull](#) () const

### 7.125.1 Detailed Description

```
template<typename T>
struct helib::PtrVector< T >
```

Abstract class for an array of objects.

### 7.125.2 Constructor & Destructor Documentation

#### 7.125.2.1 ~PtrVector()

```
template<typename T >
virtual helib::PtrVector< T >::~~PtrVector ( ) [inline], [virtual]
```

## 7.125.3 Member Function Documentation

### 7.125.3.1 isSet()

```
template<typename T >
bool helib::PtrVector< T >::isSet (
    long i ) const [inline]
```

### 7.125.3.2 numNonNull()

```
template<typename T >
virtual long helib::PtrVector< T >::numNonNull (
    long first = 0,
    long last = LONG_MAX ) const [inline], [virtual]
```

Reimplemented in [helib::PtrVector\\_slice< T >](#), [helib::PtrVector\\_vectorT< T >](#), and [helib::PtrVector\\_VecT< T >](#).

### 7.125.3.3 operator[]()

```
template<typename T >
virtual T* helib::PtrVector< T >::operator[] (
    long ) const [pure virtual]
```

Implemented in [helib::PtrVector\\_Singleton< T >](#), [helib::PtrVector\\_slice< T >](#), [helib::PtrVector\\_vectorT< T >](#), [helib::PtrVector\\_VecT< T >](#), [helib::PtrVector\\_vectorPt< T >](#), and [helib::PtrVector\\_VecPt< T >](#).

### 7.125.3.4 ptr2nonNull()

```
template<typename T >
virtual const T* helib::PtrVector< T >::ptr2nonNull ( ) const [inline], [virtual]
```

Reimplemented in [helib::PtrVector\\_slice< T >](#), and [helib::PtrVector\\_VecT< T >](#).

### 7.125.3.5 resize()

```
template<typename T >
virtual void helib::PtrVector< T >::resize (
    long newSize,
    UNUSED const PtrVector< T > * another = nullptr ) [inline], [virtual]
```

Reimplemented in [helib::PtrVector\\_vectorPt< T >](#), and [helib::PtrVector\\_VecPt< T >](#).

### 7.125.3.6 size()

```
template<typename T >
virtual long helib::PtrVector< T >::size ( ) const [pure virtual]
```

Implemented in [helib::PtrVector\\_Singleton< T >](#), [helib::PtrVector\\_slice< T >](#), [helib::PtrVector\\_vectorT< T >](#), [helib::PtrVector\\_VecT< T >](#), [helib::PtrVector\\_vectorPt< T >](#), and [helib::PtrVector\\_VecPt< T >](#).

The documentation for this struct was generated from the following file:

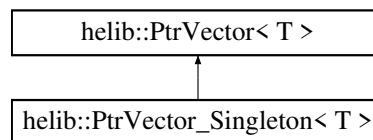
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PtrVector.h](#)

## 7.126 helib::PtrVector\_Singleton< T > Struct Template Reference

An implementation of [PtrVector](#) from a single T object.

```
#include <PtrVector.h>
```

Inheritance diagram for helib::PtrVector\_Singleton< T >:



### Public Member Functions

- [PtrVector\\_Singleton](#) (const T \* \_v)
- T \* [operator\[\]](#) (long i) const override
- long [size](#) () const override

### Public Attributes

- const T \* [v](#)

### 7.126.1 Detailed Description

```
template<typename T>
struct helib::PtrVector_Singleton< T >
```

An implementation of [PtrVector](#) from a single T object.

### 7.126.2 Constructor & Destructor Documentation

### 7.126.2.1 PtrVector\_Singleton()

```
template<typename T >
helib::PtrVector_Singleton< T >::PtrVector_Singleton (
    const T * _v ) [inline]
```

## 7.126.3 Member Function Documentation

### 7.126.3.1 operator[]()

```
template<typename T >
T* helib::PtrVector_Singleton< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

### 7.126.3.2 size()

```
template<typename T >
long helib::PtrVector_Singleton< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

## 7.126.4 Member Data Documentation

### 7.126.4.1 v

```
template<typename T >
const T* helib::PtrVector_Singleton< T >::v
```

The documentation for this struct was generated from the following file:

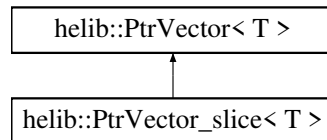
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PtrVector.h](#)

## 7.127 helib::PtrVector\_slice< T > Struct Template Reference

An implementation of [PtrVector](#) as a slice of another [PtrVector](#).

```
#include <PtrVector.h>
```

Inheritance diagram for helib::PtrVector\_slice< T >:



### Public Member Functions

- [PtrVector\\_slice](#) (const [PtrVector\\_slice](#)< T > &slice, long from, long \_sz=-1)
- [PtrVector\\_slice](#) (const [PtrVector](#)< T > &\_orig, long from, long \_sz=-1)
- T \* [operator\[\]](#) (long i) const override
- long [size](#) () const override
- long [numNonNull](#) (long first=0, long last=LONG\_MAX) const override
- const T \* [ptr2nonNull](#) () const override

### Public Attributes

- const [PtrVector](#)< T > & [orig](#)
- long [start](#)
- long [sz](#)

### 7.127.1 Detailed Description

```
template<typename T>
struct helib::PtrVector_slice< T >
```

An implementation of [PtrVector](#) as a slice of another [PtrVector](#).

### 7.127.2 Constructor & Destructor Documentation

#### 7.127.2.1 PtrVector\_slice() [1/2]

```
template<typename T>
helib::PtrVector_slice< T >::PtrVector_slice (
    const PtrVector\_slice< T > & slice,
    long from,
    long _sz = -1 ) [inline]
```

### 7.127.2.2 PtrVector\_slice() [2/2]

```
template<typename T >
helib::PtrVector_slice< T >::PtrVector_slice (
    const PtrVector< T > & _orig,
    long from,
    long _sz = -1 ) [inline]
```

## 7.127.3 Member Function Documentation

### 7.127.3.1 numNonNull()

```
template<typename T >
long helib::PtrVector_slice< T >::numNonNull (
    long first = 0,
    long last = LONG_MAX ) const [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).

### 7.127.3.2 operator[]()

```
template<typename T >
T* helib::PtrVector_slice< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

### 7.127.3.3 ptr2nonNull()

```
template<typename T >
const T* helib::PtrVector_slice< T >::ptr2nonNull ( ) const [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).

### 7.127.3.4 size()

```
template<typename T >
long helib::PtrVector_slice< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

## 7.127.4 Member Data Documentation

### 7.127.4.1 orig

```
template<typename T >
const PtrVector<T>& helib::PtrVector_slice< T >::orig
```

### 7.127.4.2 start

```
template<typename T >
long helib::PtrVector_slice< T >::start
```

### 7.127.4.3 sz

```
template<typename T >
long helib::PtrVector_slice< T >::sz
```

The documentation for this struct was generated from the following file:

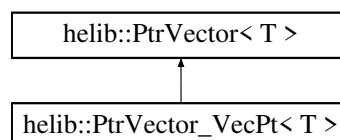
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/PtrVector.h

## 7.128 helib::PtrVector\_VecPt< T > Struct Template Reference

An implementation of [PtrVector](#) using `Vec<T*>`

```
#include <PtrVector.h>
```

Inheritance diagram for `helib::PtrVector_VecPt< T >`:



## Public Member Functions

- [PtrVector\\_VecPt](#) (Ntl::Vec< T \* > &\_v)
- T \* [operator\[\]](#) (long i) const override
- long [size](#) () const override
- void [resize](#) (long newSize, [UNUSED](#) const [PtrVector](#)< T > \*another=nullptr) override

## Public Attributes

- `Ntl::Vec< T * > & v`

### 7.128.1 Detailed Description

```
template<typename T>
struct helib::PtrVector_VecPt< T >
```

An implementation of [PtrVector](#) using `Vec<T*>`

### 7.128.2 Constructor & Destructor Documentation

#### 7.128.2.1 PtrVector\_VecPt()

```
template<typename T >
helib::PtrVector_VecPt< T >::PtrVector_VecPt (
    Ntl::Vec< T * > & _v ) [inline]
```

### 7.128.3 Member Function Documentation

#### 7.128.3.1 operator[]()

```
template<typename T >
T* helib::PtrVector_VecPt< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

#### 7.128.3.2 resize()

```
template<typename T >
void helib::PtrVector_VecPt< T >::resize (
    long newSize,
    UNUSED const PtrVector< T > * another = nullptr ) [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).



### 7.128.3.3 size()

```
template<typename T >
long helib::PtrVector_VecPt< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

## 7.128.4 Member Data Documentation

### 7.128.4.1 v

```
template<typename T >
NTL::Vec<T*>& helib::PtrVector_VecPt< T >::v
```

The documentation for this struct was generated from the following file:

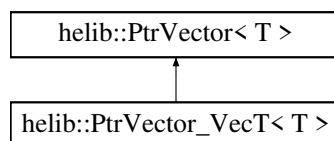
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PtrVector.h](#)

## 7.129 helib::PtrVector\_VecT< T > Struct Template Reference

An implementation of [PtrVector](#) using [Vec<T>](#)

```
#include <PtrVector.h>
```

Inheritance diagram for [helib::PtrVector\\_VecT< T >](#):



## Public Member Functions

- [PtrVector\\_VecT](#) (NTL::Vec< T > &\_v)
- T \* [operator\[\]](#) (long i) const override
- long [size](#) () const override
- void [resize](#) (long newSize, const [PtrVector](#)< T > \*another) override
- long [numNonNull](#) (long first=0, long last=LONG\_MAX) const override
- const T \* [ptr2nonNull](#) () const override

## Public Attributes

- NTL::Vec< T > & [v](#)

### 7.129.1 Detailed Description

```
template<typename T>
struct helib::PtrVector_VecT< T >
```

An implementation of [PtrVector](#) using [Vec<T>](#)

### 7.129.2 Constructor & Destructor Documentation

#### 7.129.2.1 PtrVector\_VecT()

```
template<typename T >
helib::PtrVector_VecT< T >::PtrVector_VecT (
    NTL::Vec< T > & _v ) [inline]
```

### 7.129.3 Member Function Documentation

#### 7.129.3.1 numNonNull()

```
template<typename T >
long helib::PtrVector_VecT< T >::numNonNull (
    long first = 0,
    long last = LONG_MAX ) const [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).

#### 7.129.3.2 operator[]()

```
template<typename T >
T* helib::PtrVector_VecT< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

#### 7.129.3.3 ptr2nonNull()

```
template<typename T >
const T* helib::PtrVector_VecT< T >::ptr2nonNull ( ) const [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).

### 7.129.3.4 resize()

```
template<typename T >
void helib::PtrVector_VecT< T >::resize (
    long newSize,
    const PtrVector< T > * another ) [inline], [override]
```

### 7.129.3.5 size()

```
template<typename T >
long helib::PtrVector_VecT< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

## 7.129.4 Member Data Documentation

### 7.129.4.1 v

```
template<typename T >
NTL::Vec<T>& helib::PtrVector_VecT< T >::v
```

The documentation for this struct was generated from the following file:

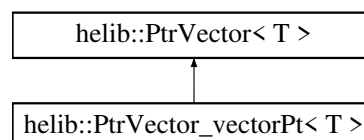
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PtrVector.h](#)

## 7.130 helib::PtrVector\_vectorPt< T > Struct Template Reference

An implementation of [PtrVector](#) using `vector<T*>`

```
#include <PtrVector.h>
```

Inheritance diagram for `helib::PtrVector_vectorPt< T >`:



## Public Member Functions

- [PtrVector\\_vectorPt](#) (`std::vector< T * > &_v`)
- `T * operator\[\]` (`long i`) `const` override
- `long size ()` `const` override
- `void resize (long newSize, UNUSED const PtrVector< T > *another=nullptr)` `override`

## Public Attributes

- `std::vector< T * > & v`

### 7.130.1 Detailed Description

```
template<typename T>
struct helib::PtrVector_vectorPt< T >
```

An implementation of [PtrVector](#) using `vector<T*>`

### 7.130.2 Constructor & Destructor Documentation

#### 7.130.2.1 PtrVector\_vectorPt()

```
template<typename T >
helib::PtrVector_vectorPt< T >::PtrVector_vectorPt (
    std::vector< T * > & _v ) [inline]
```

### 7.130.3 Member Function Documentation

#### 7.130.3.1 operator[]()

```
template<typename T >
T* helib::PtrVector_vectorPt< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

#### 7.130.3.2 resize()

```
template<typename T >
void helib::PtrVector_vectorPt< T >::resize (
    long newSize,
    UNUSED const PtrVector< T > * another = nullptr ) [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).

### 7.130.3.3 size()

```
template<typename T >
long helib::PtrVector_vectorPt< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

## 7.130.4 Member Data Documentation

### 7.130.4.1 v

```
template<typename T >
std::vector<T*>& helib::PtrVector_vectorPt< T >::v
```

The documentation for this struct was generated from the following file:

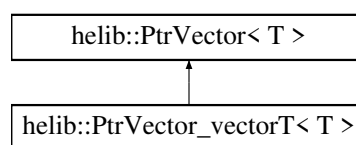
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/PtrVector.h](#)

## 7.131 helib::PtrVector\_vectorT< T > Struct Template Reference

An implementation of [PtrVector](#) using `vector<T>`

```
#include <PtrVector.h>
```

Inheritance diagram for `helib::PtrVector_vectorT< T >`:



## Public Member Functions

- [PtrVector\\_vectorT](#) (std::vector< T > &\_v)
- T \* [operator\[\]](#) (long i) const override
- long [size](#) () const override
- void [resize](#) (long newSize, const [PtrVector](#)< T > \*another) override
- long [numNonNull](#) (long first=0, long last=LONG\_MAX) const override

## Public Attributes

- std::vector< T > & [v](#)

### 7.131.1 Detailed Description

```
template<typename T>
struct helib::PtrVector_vectorT< T >
```

An implementation of [PtrVector](#) using `vector<T>`

### 7.131.2 Constructor & Destructor Documentation

#### 7.131.2.1 PtrVector\_vectorT()

```
template<typename T >
helib::PtrVector_vectorT< T >::PtrVector_vectorT (
    std::vector< T > & _v ) [inline]
```

### 7.131.3 Member Function Documentation

#### 7.131.3.1 numNonNull()

```
template<typename T >
long helib::PtrVector_vectorT< T >::numNonNull (
    long first = 0,
    long last = LONG_MAX ) const [inline], [override], [virtual]
```

Reimplemented from [helib::PtrVector< T >](#).

#### 7.131.3.2 operator[]()

```
template<typename T >
T* helib::PtrVector_vectorT< T >::operator[] (
    long i ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

#### 7.131.3.3 resize()

```
template<typename T >
void helib::PtrVector_vectorT< T >::resize (
    long newSize,
    const PtrVector< T > * another ) [inline], [override]
```

#### 7.131.3.4 size()

```
template<typename T >
long helib::PtrVector_vectorT< T >::size ( ) const [inline], [override], [virtual]
```

Implements [helib::PtrVector< T >](#).

### 7.131.4 Member Data Documentation

#### 7.131.4.1 v

```
template<typename T >
std::vector<T>& helib::PtrVector_vectorT< T >::v
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PtrVector.h](#)

## 7.132 helib::Ptxt Class Reference

An object that mimics the functionality of the [Ctxt](#) object, and acts as a convenient entry point for inputting/encoding data which is to be encrypted.

```
#include <Ctxt.h>
```

### Public Types

- using [SlotType](#) = typename Scheme::SlotType  
*Alias for type to be stored in the slots.*

### Public Member Functions

- [Ptxt](#) ()  
*Default constructor results in invalid [Ptxt](#) object which throws if used.*
- [Ptxt](#) (const [Context](#) &context)  
*[Context](#) only constructor, defaults all slots to 0.*
- [Ptxt](#) (const [Context](#) &context, const [SlotType](#) &value)  
*Single slot constructor, set all slots to *value*.*
- template<typename U = Scheme, std::enable\_if\_t< std::is\_same< U, BGV >::value > \* = nullptr> [Ptxt](#) (const [Context](#) &context, const NTL::ZZX &value)  
*[BGV](#) plaintext polynomial constructor, set all slots to the *value* polynomial.*
- [Ptxt](#) (const [Context](#) &context, const std::vector< [SlotType](#) > &data)  
*Slot vector constructor.*

- `template<typename T >`  
`Ptxt (const Context &context, const std::vector< T > &data)`  
*Generic slot vector constructor.*
- `Ptxt (const Ptxt< Scheme > &other)=default`  
*Default copy constructor.*
- `Ptxt (Ptxt< Scheme > &&other) noexcept=default`  
*Default move constructor.*
- `Ptxt< Scheme > & operator= (const Ptxt< Scheme > &v)=default`  
*Copy assignment operator with other Ptxt.*
- `Ptxt< Scheme > & operator= (Ptxt< Scheme > &&v) noexcept=default`  
*Move assignment operator with other Ptxt.*
- `~Ptxt ()=default`  
*Default destructor.*
- `bool isValid () const`  
*Check if a Ptxt is valid.*
- `size_t size () const`  
*Returns the size (number of slots) of a Ptxt.*
- `long lsize () const`  
*Returns the size (number of slots) of a Ptxt as long.*
- `void setData (const std::vector< SlotType > &data)`  
*Set the data.*
- `void setData (const SlotType &value)`  
*Set the data replicating the input on all slots.*
- `template<typename T = Scheme, typename std::enable_if_t< std::is_same< T, BGV >::value > * = nullptr>`  
`void setData (const NTL::ZZX &value)`  
*Set the Ptxt data replicating the input polynomial on all slots.*
- `template<typename T = Scheme, typename std::enable_if_t< std::is_same< T, BGV >::value > * = nullptr>`  
`void decodeSetData (const NTL::ZZX &data)`  
*Set the Ptxt slots using values from decoding data to slot representation.*
- `void clear ()`  
*Sets all slots to 0.*
- `Ptxt< Scheme > & random ()`  
*Populate slots with random data.*
- `const std::vector< SlotType > & getSlotRepr () const`  
*Get the data held in the slots as a std::vector<SlotType>.*
- `NTL::ZZX getPolyRepr () const`  
*Converts the slot data in this to its single polynomial representation.*
- `SlotType & operator[] (long i)`  
*Square bracket accessor operator.*
- `SlotType operator[] (long i) const`  
*const square bracket accessor operator.*
- `SlotType & at (long i)`  
*at accessor operator.*
- `SlotType at (long i) const`  
*const at accessor operator.*
- `bool operator== (const Ptxt< Scheme > &other) const`  
*Equals operator between two Ptxt objects.*
- `bool operator!= (const Ptxt< Scheme > &other) const`  
*Not equals operator between two Ptxt objects.*
- `Ptxt< Scheme > operator* (const Ptxt< Scheme > &rhs) const`  
*Infix multiplication operator.*



- `Ptxt< Scheme > operator+ (const Ptxt< Scheme > &rhs) const`  
*Infix addition operator.*
- `Ptxt< Scheme > operator- (const Ptxt< Scheme > &rhs) const`  
*Infix subtraction operator.*
- `Ptxt< Scheme > & operator*= (const Ptxt< Scheme > &otherPtxt)`  
*Times equals operator with another Ptxt.*
- `Ptxt< Scheme > & operator*= (const SlotType &scalar)`  
*Times equals operator with a single SlotType.*
- `template<typename Scalar >`  
`Ptxt< Scheme > & operator*= (const Scalar &scalar)`  
*Times equals operator with a scalar.*
- `Ptxt< Scheme > & operator+= (const Ptxt< Scheme > &otherPtxt)`  
*Plus equals operator with another Ptxt.*
- `Ptxt< Scheme > & operator+= (const SlotType &scalar)`  
*Plus equals operator with a single SlotType.*
- `template<typename Scalar >`  
`Ptxt< Scheme > & operator+= (const Scalar &scalar)`  
*Plus equals operator with a scalar.*
- `Ptxt< Scheme > & operator-= (const Ptxt< Scheme > &otherPtxt)`  
*Minus equals operator with another Ptxt.*
- `Ptxt< Scheme > & operator-= (const SlotType &scalar)`  
*Minus equals operator with a single SlotType.*
- `template<typename Scalar >`  
`Ptxt< Scheme > & operator-= (const Scalar &scalar)`  
*Minus equals operator with a scalar.*
- `Ptxt< Scheme > & negate ()`  
*Negate a Ptxt.*
- `template<typename T = Scheme, typename Scalar, typename std::enable_if_t< std::is_same< T, BGV >::value > * = nullptr>`  
`Ptxt< Scheme > & addConstant (const Scalar &scalar)`  
*Add a constant to a BGV Ptxt.*
- `template<typename T = Scheme, typename Scalar, typename std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr>`  
`Ptxt< Scheme > & addConstantCKKS (const Scalar &scalar)`  
*Add a constant to a CKKS Ptxt.*
- `Ptxt< Scheme > & multiplyBy (const Ptxt< Scheme > &otherPtxt)`  
*Multiplication function between two Ptxt objects.*
- `Ptxt< Scheme > & multiplyBy2 (const Ptxt &otherPtxt1, const Ptxt &otherPtxt2)`  
*Multiplication function between three Ptxt objects.*
- `Ptxt< Scheme > & square ()`  
*Square operation on a Ptxt.*
- `Ptxt< Scheme > & cube ()`  
*Cube operation on a Ptxt.*
- `Ptxt< Scheme > & power (long e)`  
*Power operation to raise a Ptxt to an arbitrary non-negative power.*
- `Ptxt< Scheme > & rotate (long amount)`  
*Rotate slots right by specified amount (slot i goes to slot i+1 mod size).*
- `Ptxt< Scheme > & rotate1D (long dim, long amount)`  
*Rotate slots right by specified amount along a specific dimension.*
- `Ptxt< Scheme > & shift (long amount)`  
*Shifts slots right by specified amount with 0 fill (slot i goes to slot i+1 mod size).*
- `Ptxt< Scheme > & shift1D (long dim, long amount)`  
*Shift slots right in one dimension of the hypercube structure with 0 fill.*

- `Ptxt< Scheme > & automorph (long k)`  
*Apply the automorphism  $a(X) \rightarrow a(X^k) \bmod \Phi_m(X)$ .*
- `template<typename T = Scheme, std::enable_if_t< std::is_same< T, BGV >::value > * = nullptr> Ptxt< Scheme > & frobeniusAutomorph (long j)`  
*Apply the frobenius automorphism  $a(X) \rightarrow a(X^{(p^j)}) \bmod \Phi_m(X)$ .*
- `Ptxt< Scheme > & replicate (long pos)`  
*Replicate single slot across all slots.*
- `std::vector< Ptxt< Scheme > > replicateAll () const`  
*Generate a vector of plaintexts with each slot replicated in each plaintext.*
- `template<typename T = Scheme, std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr> Ptxt< Scheme > & complexConj ()`  
*Apply complex conjugate of complex numbers in slots of a [CKKS Ptxt](#) object.*
- `template<typename T = Scheme, std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr> Ptxt< Scheme > real () const`  
*Extract the real part of a [CKKS](#) plaintext.*
- `template<typename T = Scheme, std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr> Ptxt< Scheme > imag () const`  
*Extract the imaginary part of a [CKKS](#) plaintext.*
- `Ptxt< Scheme > & runningSums ()`  
*Compute the running sum (each slot is the sum of the previous slots).*
- `Ptxt< Scheme > & totalSums ()`  
*Compute the total sum (each slot contains the total sum of every slot).*
- `Ptxt< Scheme > & incrementalProduct ()`  
*Compute the incremental product (each slot is the product of the previous slots).*
- `Ptxt< Scheme > & totalProduct ()`  
*Compute the total product (each slot contains the total product of every slot).*
- `Ptxt< Scheme > & mapTo01 ()`  
*Map all non-zero slots to 1, keeping zero slots as zero.*
- `PolyMod convertToSlot (const Context &context, long slot)`
- `std::complex< double > convertToSlot (const Context &, long slot)`
- `template<typename Scheme > Ptxt ()`
- `template<typename Scheme > Ptxt (const Context &context)`
- `template<typename Scheme > Ptxt (const Context &context, const SlotType &value)`
- `void setData (const NTL::ZZX &value)`
- `Ptxt (const Context &context, const NTL::ZZX &value)`
- `template<typename Scheme > Ptxt (const Context &context, const std::vector< SlotType > &data)`
- `template<typename Scheme > void setData (const std::vector< SlotType > &data)`
- `template<typename Scheme > void setData (const SlotType &value)`
- `void decodeSetData (const NTL::ZZX &data)`
- `NTL::ZZX getPolyRepr () const`  
*BGV specialisation of the `getPolyRepr` function.*
- `NTL::ZZX getPolyRepr () const`  
*CKKS specialisation of the `getPolyRepr` function.*
- `template<typename Scheme > Ptxt< Scheme > & operator*= (const Ptxt< Scheme > &otherPtxt)`
- `template<typename Scheme > Ptxt< Scheme > & operator*= (const SlotType &scalar)`

- `template<typename Scheme >`  
`Ptxt< Scheme > & operator+= (const Ptxt< Scheme > &otherPtxt)`
- `template<typename Scheme >`  
`Ptxt< Scheme > & operator+= (const SlotType &scalar)`
- `template<typename Scheme >`  
`Ptxt< Scheme > & operator-= (const Ptxt< Scheme > &otherPtxt)`
- `template<typename Scheme >`  
`Ptxt< Scheme > & operator-= (const SlotType &scalar)`
- `Ptxt< BGV > & automorph (long k)`
- `Ptxt< CKKS > & automorph (long k)`
- `Ptxt< BGV > & frobeniusAutomorph (long j)`
- `Ptxt< CKKS > & complexConj ()`
- `Ptxt< CKKS > real () const`
- `Ptxt< CKKS > imag () const`

## Static Public Member Functions

- static `SlotType convertToSlot (const Context &context, long slot)`  
*Conversion function from long to SlotType.*

## Friends

- `std::istream & operator>> (std::istream &is, Ptxt< Scheme > &ptxt)`  
*Input shift operator.*
- `std::ostream & operator<< (std::ostream &os, const Ptxt< Scheme > &ptxt)`  
*Output shift operator.*

### 7.132.1 Detailed Description

An object that mimics the functionality of the `Ctxt` object, and acts as a convenient entry point for inputting/encoding data which is to be encrypted.

`Ptxt` is templated on `Scheme`, which may be `CKKS` or `BGV`.

In the `BGV` case, `Ptxt` can be considered to be an element of  $\mathbb{Z}_p[x]/\Phi(m)$  viewed as a vector of slots with values each in  $\mathbb{Z}_p[x]/G$  where  $G$  is one of the irreducible factors of  $\Phi(m)$ , and all operations are performed entry-wise.

General usage:

```
helib::Ptxt<BGV> p1(bgv_context, data);
helib::Ptxt<BGV> p2(bgv_context, data);
p1 += p2;
std::cout << p1 << std::endl;
```

Internally, `Ptxt` objects store their data as a `std::vector<helib::PolyMod>`, where `PolyMod` is a convenience type representing an element of the above ring,  $\mathbb{Z}_p[x]/G$ . The `PolyMod` type can be easily converted via `static_cast` to more convenient types such as `long` and `NTL::ZZX`.

In the `CKKS` case, the slot type is `std::complex<double>`, and has sensible operator overloads supporting operations with other `Ptxt<CKKS>`, `Ctxt`, and `std::complex<double>` objects, as well as performing all operations slot-wise.

A large number of operator overloads are defined so that `Ptxt` objects should easily inter-operate, as well as providing interoperability with other logically compatible types e.g. `long` and `NTL::ZZX` in the `BGV` case, `std::complex<double>` in the `CKKS` case, and `helib::Ctxt` in both cases.

## 7.132.2 Member Typedef Documentation

### 7.132.2.1 SlotType

```
using helib::Ptxt::SlotType = typename Scheme::SlotType
```

Alias for type to be stored in the slots.

std::complex<double> for CKKS, helib::PolyMod for BGV.

## 7.132.3 Constructor & Destructor Documentation

### 7.132.3.1 Ptxt() [1/13]

```
helib::Ptxt::Ptxt ( )
```

Default constructor results in invalid Ptxt object which throws if used.

### 7.132.3.2 Ptxt() [2/13]

```
helib::Ptxt::Ptxt (
    const Context & context ) [explicit]
```

Context only constructor, defaults all slots to 0.

#### Parameters

<i>context</i>	Context to use.
----------------	-----------------

### 7.132.3.3 Ptxt() [3/13]

```
helib::Ptxt::Ptxt (
    const Context & context,
    const SlotType & value )
```

Single slot constructor, set all slots to value.

## Parameters

<i>context</i>	<a href="#">Context</a> to use.
<i>value</i>	Value to set all slots to.

**7.132.3.4 Ptxt()** [4/13]

```
template<typename U = Scheme, std::enable_if_t< std::is_same< U, BGV >::value > * = nullptr>
helib::Ptxt::Ptxt (
    const Context & context,
    const NTL::ZZX & value )
```

[BGV](#) plaintext polynomial constructor, set all slots to the `value` polynomial.

## Parameters

<i>context</i>	<a href="#">Context</a> to use.
<i>data</i>	Polynomial to be con- verted into slot repre- senta- tion.

## Note

Only exists for [BGV](#).

**7.132.3.5 Ptxt()** [5/13]

```
helib::Ptxt::Ptxt (
    const Context & context,
    const std::vector< SlotType > & data )
```

Slot vector constructor.

## Parameters

<i>context</i>	<a href="#">Context</a> to use.
----------------	------------------------------------

## Parameters

<i>data</i>	Data to populate the slots.
-------------	-----------------------------

**7.132.3.6 Ptxt()** [6/13]

```
template<typename T >
helib::Ptxt::Ptxt (
    const Context & context,
    const std::vector< T > & data ) [inline]
```

Generic slot vector constructor.

## Parameters

<i>context</i>	Context to use.
<i>data</i>	Data to populate the slots, must be convertible to Slot↵ Type.

**7.132.3.7 Ptxt()** [7/13]

```
helib::Ptxt::Ptxt (
    const Ptxt< Scheme > & other ) [default]
```

Default copy constructor.

## Parameters

<i>other</i>	Ptxt object to copy.
--------------	----------------------

**7.132.3.8 Ptxt()** [8/13]

```
helib::Ptxt::Ptxt (
    Ptxt< Scheme > && other ) [default], [noexcept]
```

Default move constructor.

**Parameters**

<i>other</i>	Ptxt to copy.
--------------	---------------------

**7.132.3.9 ~Ptxt()**

```
helib::Ptxt::~~Ptxt ( ) [default]
```

Default destructor.

**7.132.3.10 Ptxt()** [9/13]

```
template<typename Scheme >
helib::Ptxt::Ptxt ( )
```

**7.132.3.11 Ptxt()** [10/13]

```
template<typename Scheme >
helib::Ptxt::Ptxt (
    const Context & context )
```

**7.132.3.12 Ptxt()** [11/13]

```
template<typename Scheme >
helib::Ptxt::Ptxt (
    const Context & context,
    const SlotType & value )
```

**7.132.3.13 Ptxt()** [12/13]

```

helib::Ptxt< BGV >::Ptxt (
    const Context & context,
    const NTL::ZZX & value )

```

**7.132.3.14 Ptxt()** [13/13]

```

template<typename Scheme >
helib::Ptxt::Ptxt (
    const Context & context,
    const std::vector< SlotType > & data )

```

**7.132.4 Member Function Documentation****7.132.4.1 addConstant()**

```

template<typename T = Scheme, typename Scalar , typename std::enable_if_t< std::is_same< T,
BGV >::value > * = nullptr>
Ptxt<Scheme>& helib::Ptxt::addConstant (
    const Scalar & scalar ) [inline]

```

Add a constant to a [BGV Ptxt](#).

**Parameters**

<i>scalar</i>	Element to be added across all slots.
---------------	---------------------------------------

**Returns**

Reference to *\*this* post scalar addition.

**7.132.4.2 addConstantCKKS()**

```

template<typename T = Scheme, typename Scalar , typename std::enable_if_t< std::is_same< T,
CKKS >::value > * = nullptr>
Ptxt<Scheme>& helib::Ptxt::addConstantCKKS (
    const Scalar & scalar ) [inline]

```

Add a constant to a [CKKS Ptxt](#).



## Parameters

<i>scalar</i>	Element to be added across all slots.
---------------	---------------------------------------

## Returns

Reference to *\*this* post scalar addition.

**7.132.4.3 at()** [1/2]

```
Ptxt< Scheme >::SlotType & helib::Ptxt::at (  
    long i )
```

at accessor operator.

## Parameters

<i>i</i>	Index of the desired Ptxt slot.
----------	---------------------------------

## Returns

Reference to the data held at slot *i*.

## Note

throws if *i* is out of range.

**7.132.4.4 at()** [2/2]

```
Ptxt< Scheme >::SlotType helib::Ptxt::at (  
    long i ) const
```

const at accessor operator.

**Parameters**

<i>i</i>	Index of the de- sired <code>Ptxt</code> slot.
----------	---

**Returns**

Copy of the data held at slot *i*.

**Note**

throws if *i* is out of range.

**7.132.4.5 automorph() [1/3]**

```
Ptxt<Scheme>& helib::Ptxt::automorph (
    long k )
```

Apply the automorphism  $a(X) \rightarrow a(X^k) \bmod \Phi_m(X)$ .

**Parameters**

<i>k</i>	Exponent of the auto- mor- phism to apply.
----------	--

**Returns**

Reference to `*this` post automorphism application.

**Note**

*k* must be an element of  $\mathbb{Z}_m^*$

**7.132.4.6 automorph() [2/3]**

```
Ptxt< BGV > & helib::Ptxt< BGV >::automorph (
    long k )
```

**7.132.4.7 automorph()** [3/3]

```
Ptxt< CKKS > & helib::Ptxt< CKKS >::automorph (
    long k )
```

**7.132.4.8 clear()**

```
void helib::Ptxt::clear ( )
```

Sets all slots to 0.

**7.132.4.9 complexConj()** [1/2]

```
template<typename T = Scheme, std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr>
Ptxt<Scheme>& helib::Ptxt::complexConj ( )
```

Apply complex conjugate of complex numbers in slots of a [CKKS Ptxt](#) object.

**Returns**

Reference to `*this` post complex conjugation.

**Note**

Only valid for the [CKKS](#) scheme.

**7.132.4.10 complexConj()** [2/2]

```
Ptxt< CKKS > & helib::Ptxt< CKKS >::complexConj ( )
```

**7.132.4.11 convertToSlot()** [1/3]

```
std::complex< double > helib::Ptxt< CKKS >::convertToSlot (
    const Context & ,
    long slot )
```

**7.132.4.12 convertToSlot()** [2/3]

```
PolyMod helib::Ptxt< BGV >::convertToSlot (
    const Context & context,
    long slot )
```

**7.132.4.13 convertToSlot()** [3/3]

```
static SlotType helib::Ptxt::convertToSlot (
    const Context & context,
    long slot ) [static]
```

Conversion function from long to SlotType.

## Parameters

<i>context</i>	<a href="#">Context</a> which may be needed to extract algebraic info.
<i>slot</i>	Datum to be converted to a <code>Slot</code> ↔ <code>Type</code> .

## Returns

Converted slot.

**7.132.4.14 cube()**

```
Ptxt< Scheme > & helib::Ptxt::cube ( )
```

Cube operation on a [Ptxt](#).

## Returns

Reference to `*this` post cube operation.

**7.132.4.15 decodeSetData() [1/2]**

```
void helib::Ptxt< BGV >::decodeSetData (
    const NTL::ZZX & data )
```

**7.132.4.16 decodeSetData() [2/2]**

```
template<typename T = Scheme, typename std::enable_if_t< std::is_same< T, BGV >::value > * =
nullptr>
void helib::Ptxt::decodeSetData (
    const NTL::ZZX & data )
```

Set the [Ptxt](#) slots using values from decoding data to slot representation.

## Parameters

<i>data</i>	Polynomial to be de-coded and converted into slot data.
-------------	---

## Note

Only works in the [BGV](#) case.

**7.132.4.17 frobeniusAutomorph()** [1/2]

```
template<typename T = Scheme, std::enable_if_t< std::is_same< T, BGV >::value > * = nullptr>
Ptxt<Scheme>& helib::Ptxt::frobeniusAutomorph (
    long j )
```

Apply the frobenius automorphism  $a(X) \rightarrow a(X^{(p^j)}) \bmod \Phi_m(X)$ .

## Parameters

<i>j</i>	Exponent of the automorphism to apply.
----------	--

## Returns

Reference to `*this` post frobenius automorphism application.

## Note

Only valid for the [BGV](#) scheme.

**7.132.4.18 frobeniusAutomorph()** [2/2]

```
Ptxt< BGV > & helib::Ptxt< BGV >::frobeniusAutomorph (
    long j )
```

#### 7.132.4.19 `getPolyRepr()` [1/3]

```
NTL::ZZX helib::Ptxt< BGV >::getPolyRepr ( ) const
```

[BGV](#) specialisation of the `getPolyRepr` function.

##### Returns

Single encoded polynomial.

##### Note

Only enabled for the [BGV](#) scheme.

#### 7.132.4.20 `getPolyRepr()` [2/3]

```
NTL::ZZX helib::Ptxt< CKKS >::getPolyRepr ( ) const
```

[CKKS](#) specialisation of the `getPolyRepr` function.

##### Returns

Single encoded polynomial.

##### Note

Only enabled for the [CKKS](#) scheme.

#### 7.132.4.21 `getPolyRepr()` [3/3]

```
NTL::ZZX helib::Ptxt::getPolyRepr ( ) const
```

Converts the slot data in `this` to its single polynomial representation.

##### Returns

Single encoded polynomial.

##### Note

`NTL::ZZX` representation loses some precision in the [CKKS](#) case.

#### 7.132.4.22 getSlotRepr()

```
const std::vector< typename Ptxt< Scheme >::SlotType > & helib::Ptxt::getSlotRepr ( ) const
```

Get the data held in the slots as a `std::vector<SlotType>`.

##### Returns

Constant reference to the slot vector.

#### 7.132.4.23 imag() [1/2]

```
template<typename T = Scheme, std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr>  
Ptxt<Scheme> helib::Ptxt::imag ( ) const
```

Extract the imaginary part of a [CKKS](#) plaintext.

##### Returns

New plaintext containing the imaginary part of each slot.

##### Note

Only valid for the [CKKS](#) scheme.

#### 7.132.4.24 imag() [2/2]

```
Ptxt< CKKS > helib::Ptxt< CKKS >::imag ( ) const
```

#### 7.132.4.25 incrementalProduct()

```
Ptxt< Scheme > & helib::Ptxt::incrementalProduct ( )
```

Compute the incremental product (each slot is the product of the previous slots).

##### Returns

Reference to `*this` post multiplication.

#### 7.132.4.26 isValid()

```
bool helib::Ptxt::isValid ( ) const
```

Check if a `Ptxt` is valid.

##### Returns

`true` if valid, `false` otherwise.

#### 7.132.4.27 lsize()

```
long helib::Ptxt::lsize ( ) const
```

Returns the size (number of slots) of a `Ptxt` as long.

##### Returns

Number of slots of the `Ptxt`.

#### 7.132.4.28 mapTo01()

```
Ptxt< Scheme > & helib::Ptxt::mapTo01 ( )
```

Map all non-zero slots to 1, keeping zero slots as zero.

##### Returns

Reference to `*this` post mapping.

#### 7.132.4.29 multiplyBy()

```
Ptxt< Scheme > & helib::Ptxt::multiplyBy (
    const Ptxt< Scheme > & otherPtxt )
```

Multiplication function between two `Ptxt` objects.



## Parameters

<i>otherPtxt</i>	Right hand side of multiplication.
------------------	------------------------------------

## Returns

Reference to `*this` post multiplication.

## Note

This function is equivalent to operator `*=`.

**7.132.4.30 multiplyBy2()**

```
Ptxt< Scheme > & helib::Ptxt::multiplyBy2 (
    const Ptxt & otherPtxt1,
    const Ptxt & otherPtxt2 )
```

Multiplication function between three `Ptxt` objects.

## Parameters

<i>otherPtxt1</i>	First <code>Ptxt</code> to multiply with.
<i>otherPtxt2</i>	Second <code>Ptxt</code> to multiply with.

## Returns

Reference to `*this` post multiplication.

**7.132.4.31 negate()**

```
Ptxt< Scheme > & helib::Ptxt::negate ( )
```

Negate a `Ptxt`.

**Returns**

Reference to `*this` post negation.

**7.132.4.32 operator"!=()"**

```
bool helib::Ptxt::operator!= (
    const Ptxt< Scheme > & other ) const
```

Not equals operator between two `Ptxt` objects.

**Parameters**

<i>other</i>	<code>Ptxt</code> to com- pare to.
--------------	--

**Returns**

`true` if differ, `false` otherwise.

**7.132.4.33 operator\*()"**

```
Ptxt< Scheme > helib::Ptxt::operator* (
    const Ptxt< Scheme > & rhs ) const
```

Infix multiplication operator.

**Parameters**

<i>rhs</i>	Right hand side of multi- plica- tion.
------------	---

**Returns**

Product of the two `Ptxt` objects.

**7.132.4.34 operator\*=( ) [1/5]**

```
template<typename Scheme >
Ptxt<Scheme>& helib::Ptxt::operator*= (
    const Ptxt< Scheme > & otherPtxt )
```

**7.132.4.35 operator\*=( ) [2/5]**

```
Ptxt<Scheme>& helib::Ptxt::operator*= (
    const Ptxt< Scheme > & otherPtxt )
```

Times equals operator with another `Ptxt`.

**Parameters**

<i>otherPtxt</i>	Right hand side of multiplication.
------------------	------------------------------------

**Returns**

Reference to `*this` post multiplication.

**7.132.4.36 operator\*=( ) [3/5]**

```
template<typename Scalar >
Ptxt<Scheme>& helib::Ptxt::operator*= (
    const Scalar & scalar ) [inline]
```

Times equals operator with a scalar.

**Parameters**

<i>scalar</i>	Element to be added across all slots.
---------------	---------------------------------------

**Returns**

Reference to `*this` post scalar multiplication.

**7.132.4.37 operator\*=( ) [4/5]**

```
template<typename Scheme >
Ptxt<Scheme>& helib::Ptxt::operator*= (
    const SlotType & scalar )
```

**7.132.4.38 operator\*=( ) [5/5]**

```
Ptxt<Scheme>& helib::Ptxt::operator*= (
    const SlotType & scalar )
```

Times equals operator with a single SlotType.

**Parameters**

<i>scalar</i>	Element to be multiplied across all slots.
---------------	--

**Returns**

Reference to *\*this* post multiplication.

**7.132.4.39 operator+( )**

```
Ptxt< Scheme > helib::Ptxt::operator+ (
    const Ptxt< Scheme > & rhs ) const
```

Infix addition operator.

**Parameters**

<i>rhs</i>	Right hand side of addition.
------------	------------------------------

**Returns**

Sum of the two Ptxt objects.

**7.132.4.40 operator+=( ) [1/5]**

```
template<typename Scheme >
Ptxt<Scheme>& helib::Ptxt::operator+= (
    const Ptxt< Scheme > & otherPtxt )
```

**7.132.4.41 operator+=( ) [2/5]**

```
Ptxt<Scheme>& helib::Ptxt::operator+= (
    const Ptxt< Scheme > & otherPtxt )
```

Plus equals operator with another [Ptxt](#).

**Parameters**

<i>otherPtxt</i>	Right hand side of addition.
------------------	------------------------------

**Returns**

Reference to *\*this* post addition.

**7.132.4.42 operator+=( ) [3/5]**

```
template<typename Scalar >
Ptxt<Scheme>& helib::Ptxt::operator+= (
    const Scalar & scalar ) [inline]
```

Plus equals operator with a scalar.

**Parameters**

<i>scalar</i>	Element to be added across all slots.
---------------	---------------------------------------

**Returns**

Reference to *\*this* post scalar addition.

**7.132.4.43 operator+=()** [4/5]

```
template<typename Scheme >
Ptxt<Scheme>& helib::Ptxt::operator+= (
    const SlotType & scalar )
```

**7.132.4.44 operator+=()** [5/5]

```
Ptxt<Scheme>& helib::Ptxt::operator+= (
    const SlotType & scalar )
```

Plus equals operator with a single SlotType.

**Parameters**

<i>scalar</i>	Element to be added across all slots.
---------------	---------------------------------------

**Returns**

Reference to *\*this* post addition.

**7.132.4.45 operator-()**

```
Ptxt< Scheme > helib::Ptxt::operator- (
    const Ptxt< Scheme > & rhs ) const
```

Infix subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side of subtraction.
------------	---------------------------------

**Returns**

Difference of the two Ptxt objects.

**7.132.4.46 operator-=()** [1/5]

```
template<typename Scheme >
Ptxt<Scheme>& helib::Ptxt::operator-= (
    const Ptxt< Scheme > & otherPtxt )
```

**7.132.4.47 operator-=()** [2/5]

```
Ptxt<Scheme>& helib::Ptxt::operator-= (
    const Ptxt< Scheme > & otherPtxt )
```

Minus equals operator with another [Ptxt](#).

**Parameters**

<i>otherPtxt</i>	Right hand side of subtraction.
------------------	---------------------------------

**Returns**

Reference to *\*this* post subtraction.

**7.132.4.48 operator-=()** [3/5]

```
template<typename Scalar >
Ptxt<Scheme>& helib::Ptxt::operator-= (
    const Scalar & scalar ) [inline]
```

Minus equals operator with a scalar.

**Parameters**

<i>scalar</i>	Element to be subtracted across all slots.
---------------	--

**Returns**

Reference to *\*this* post scalar subtraction.

**7.132.4.49 operator-=()** [4/5]

```
template<typename Scheme >
Ptxt<Scheme>& helib::Ptxt::operator-= (
    const SlotType & scalar )
```

**7.132.4.50 operator-=()** [5/5]

```
Ptxt<Scheme>& helib::Ptxt::operator-= (
    const SlotType & scalar )
```

Minus equals operator with a single SlotType.

**Parameters**

<i>scalar</i>	Element to be subtracted across all slots.
---------------	--

**Returns**

Reference to *\*this* post subtraction.

**7.132.4.51 operator=()** [1/2]

```
Ptxt<Scheme>& helib::Ptxt::operator= (
    const Ptxt< Scheme > & v ) [default]
```

Copy assignment operator with other Ptxt.

**Parameters**

<i>other</i>	Ptxt to copy.
--------------	---------------



**7.132.4.52 operator=()** [2/2]

```
Ptxt<Scheme>& helib::Ptxt::operator= (
    Ptxt< Scheme > && v ) [default], [noexcept]
```

Move assignment operator with other `Ptxt`.

**Parameters**

<i>other</i>	<code>Ptxt</code> to copy.
--------------	----------------------------------

**7.132.4.53 operator==()**

```
bool helib::Ptxt::operator== (
    const Ptxt< Scheme > & other ) const
```

Equals operator between two `Ptxt` objects.

**Parameters**

<i>other</i>	<code>Ptxt</code> to com- pare to.
--------------	--

**Returns**

`true` if identical, `false` otherwise.

**7.132.4.54 operator[]()** [1/2]

```
Ptxt< Scheme >::SlotType & helib::Ptxt::operator[] (
    long i )
```

Square bracket accessor operator.

**Parameters**

<i>i</i>	Index of the de- sired <code>Ptxt</code> slot.
----------	---

**Returns**

Reference to the data held at slot *i*.

**7.132.4.55 operator[]() [2/2]**

```
Ptxt< Scheme >::SlotType helib::Ptxt::operator[] (
    long i ) const
```

const square bracket accessor operator.

**Parameters**

<i>i</i>	Index of the de- sired <i>Ptxt</i> slot.
----------	---

**Returns**

Copy of the data held at slot *i*.

**7.132.4.56 power()**

```
Ptxt< Scheme > & helib::Ptxt::power (
    long e )
```

Power operation to raise a *Ptxt* to an arbitrary non-negative power.

**Parameters**

<i>e</i>	Exponent to raise the <i>Ptxt</i> by.
----------	--

**Returns**

Reference to *\*this* post raising to the power *e*.

**7.132.4.57 random()**

```
Ptxt< Scheme > & helib::Ptxt::random ( )
```

Populate slots with random data.

**Returns**

Reference to *\*this* post population.

**7.132.4.58 real() [1/2]**

```
template<typename T = Scheme, std::enable_if_t< std::is_same< T, CKKS >::value > * = nullptr>
Ptxt<Scheme> helib::Ptxt::real ( ) const
```

Extract the real part of a [CKKS](#) plaintext.

**Returns**

New plaintext containing the real part of each slot.

**Note**

Only valid for the [CKKS](#) scheme.

**7.132.4.59 real() [2/2]**

```
Ptxt< CKKS > helib::Ptxt< CKKS >::real ( ) const
```

**7.132.4.60 replicate()**

```
Ptxt< Scheme > & helib::Ptxt::replicate (
    long pos )
```

Replicate single slot across all slots.

**Parameters**

<i>pos</i>	Position of the slot to replicate.
------------	------------------------------------

**Returns**

Reference to `*this` post replication.

**7.132.4.61 replicateAll()**

```
std::vector< Ptxt< Scheme > > helib::Ptxt::replicateAll ( ) const
```

Generate a vector of plaintexts with each slot replicated in each plaintext.

**Returns**

Vector of replicated plaintext slots. The order of the return vector agrees with the order of the slots. i.e. the  $i$ -th plaintext in the return value is a replication of `*this[i]`.

**7.132.4.62 rotate()**

```
Ptxt< Scheme > & helib::Ptxt::rotate (
    long amount )
```

Rotate slots right by specified amount (slot  $i$  goes to slot  $i+1 \bmod \text{size}$ ).

**Parameters**

<i>amount</i>	Number of slots to rotate by.
---------------	-------------------------------

**Returns**

Reference to `*this` post rotation.

**7.132.4.63 rotate1D()**

```
Ptxt< Scheme > & helib::Ptxt::rotate1D (
    long dim,
    long amount )
```

Rotate slots right by specified amount along a specific dimension.

## Parameters

<i>dim</i>	Dimension in which to rotate.
<i>amount</i>	Number of slots to rotate by.

## Returns

Reference to *\*this* post rotation.

**7.132.4.64 runningSums()**

```
Ptxt< Scheme > & helib::Ptxt::runningSums ( )
```

Compute the running sum (each slot is the sum of the previous slots).

## Returns

Reference to *\*this* post summation.

**7.132.4.65 setData() [1/6]**

```
void helib::Ptxt< BGV >::setData (
    const NTL::ZZX & value )
```

**7.132.4.66 setData() [2/6]**

```
template<typename T = Scheme, typename std::enable_if_t< std::is_same< T, BGV >::value > * =
nullptr>
void helib::Ptxt::setData (
    const NTL::ZZX & value )
```

Set the *Ptxt* data replicating the input polynomial on all slots.

**Parameters**

<i>data</i>	Polynomial to be repli- cate into slots.
-------------	---

**Note**

Only works in the [BGV](#) case.

**7.132.4.67 setData() [3/6]**

```
template<typename Scheme >
void helib::Ptxt::setData (
    const SlotType & value )
```

**7.132.4.68 setData() [4/6]**

```
void helib::Ptxt::setData (
    const SlotType & value )
```

Set the data replicating the input on all slots.

**Parameters**

<i>value</i>	value to set all slots to.
--------------	--

**7.132.4.69 setData() [5/6]**

```
template<typename Scheme >
void helib::Ptxt::setData (
    const std::vector< SlotType > & data )
```

**7.132.4.70 setData()** [6/6]

```
void helib::Ptxt::setData (
    const std::vector< SlotType > & data )
```

Set the data.

**Parameters**

<i>data</i>	Vector of SlotType to populate the slots.
-------------	---

**7.132.4.71 shift()**

```
Ptxt< Scheme > & helib::Ptxt::shift (
    long amount )
```

Shifts slots right by specified amount with 0 fill (slot *i* goes to slot *i*+1 mod size).

**Parameters**

<i>amount</i>	Number of slots to shift by.
---------------	------------------------------

**Returns**

Reference to *\*this* post shift.

**7.132.4.72 shift1D()**

```
Ptxt< Scheme > & helib::Ptxt::shift1D (
    long dim,
    long amount )
```

Shift slots right in one dimension of the hypercube structure with 0 fill.

**Parameters**

<i>dim</i>	Dimension in which to shift.
<i>amount</i>	Amount by which to shift.

**Returns**

Reference to `*this` post shift.

**7.132.4.73 size()**

```
size_t helib::Ptxt::size ( ) const
```

Returns the size (number of slots) of a [Ptxt](#).

**Returns**

Number of slots of the [Ptxt](#).

**7.132.4.74 square()**

```
Ptxt< Scheme > & helib::Ptxt::square ( )
```

Square operation on a [Ptxt](#).

**Returns**

Reference to `*this` post squaring.

**7.132.4.75 totalProduct()**

```
Ptxt< Scheme > & helib::Ptxt::totalProduct ( )
```

Compute the total product (each slot contains the total product of every slot).

**Returns**

Reference to `*this` post multiplication.



**7.132.4.76 totalSums()**

```
Ptxt< Scheme > & helib::Ptxt::totalSums ( )
```

Compute the total sum (each slot contains the total sum of every slot).

**Returns**

Reference to *\*this* post summation.

**7.132.5 Friends And Related Function Documentation****7.132.5.1 operator<<**

```
std::ostream& operator<< (
    std::ostream & os,
    const Ptxt< Scheme > & ptxt ) [friend]
```

Output shift operator.

**Parameters**

<i>os</i>	Output std:: ostream.
<i>ptxt</i>	Ptxt object to be writ- ten.

**Returns**

Input std::ostream post writing.

**Note**

Ptxt context is not serialised, see note of operator>>.

**7.132.5.2 operator>>**

```
std::istream& operator>> (
    std::istream & is,
    Ptxt< Scheme > & ptxt ) [friend]
```

Input shift operator.

## Parameters

<i>is</i>	Input std::istream.
<i>ptxt</i>	Destination Ptxt object.

## Returns

Input std::istream post reading.

## Note

ptxt must be constructed with an appropriate context **BEFORE** calling this function. For example,

```
Ptxt my_ptxt(context);
std::cin >> my_ptxt;
```

The documentation for this class was generated from the following files:

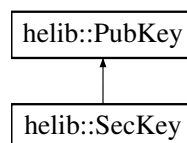
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/Ctxt.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/Ptxt.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Ptxt.cpp

## 7.133 helib::PubKey Class Reference

The public key.

```
#include <keys.h>
```

Inheritance diagram for helib::PubKey:



### Public Member Functions

- [PubKey](#) ()  
*This constructor thorws run-time error if activeContext=nullptr.*
- [PubKey](#) (const [Context](#) &\_context)
- [PubKey](#) (const [PubKey](#) &other)  
*Copy constructor.*
- virtual [~PubKey](#) ()=default  
*Default destructor.*
- virtual void [clear](#) ()  
*Clear all public-key data.*

- bool `operator==` (const `PubKey` &other) const
- bool `operator!=` (const `PubKey` &other) const
- const `Context` & `getContext` () const
- long `getPtxtSpace` () const
- bool `keyExists` (long keyID) const
- double `getSKeyBound` (long keyID=0) const  
*The size of the secret key.*
- bool `isReachable` (long k, long keyID=0) const  
*Is it possible to re-linearize the automorphism  $X \rightarrow X^k$  See Section 3.2.2 in the design document (KeySwitchMap)*
- void `setKeySwitchMap` (long keyID=0)  
*Compute the reachability graph of key-switching matrices See Section 3.2.2 in the design document (KeySwitchMap)*
- long `getKSStrategy` (long dim) const  
*get KS strategy for dimension dim dim == -1 is Frobenius*
- void `setKSStrategy` (long dim, int val)  
*set KS strategy for dimension dim dim == -1 is Frobenius*
- long `Encrypt` (Ctxt &ciphertxt, const NTL::ZZX &plaintext, long ptxtSpace, bool highNoise) const
- long `Encrypt` (Ctxt &ciphertxt, const `zzX` &plaintext, long ptxtSpace, bool highNoise) const
- void `CKKSencrypt` (Ctxt &ciphertxt, const NTL::ZZX &plaintext, double ptxtSize=1.0, double scaling=0.0) const
- void `CKKSencrypt` (Ctxt &ciphertxt, const `zzX` &plaintext, double ptxtSize=1.0, double scaling=0.0) const
- virtual long `Encrypt` (Ctxt &ciphertxt, const NTL::ZZX &plaintext, long ptxtSpace=0) const
- virtual long `Encrypt` (Ctxt &ciphertxt, const `zzX` &plaintext, long ptxtSpace=0) const
- template<typename Scheme >  
long `Encrypt` (Ctxt &ciphertxt, const Ptxt< Scheme > &plaintext, long ptxtSpace=0) const  
*Encrypts a plaintext into a ciphertext.*
- bool `isCKKS` () const
- bool `isBootstrappable` () const
- void `reCrypt` (Ctxt &ctxt) const
- void `thinReCrypt` (Ctxt &ctxt) const
- friend void `::helib::writePubKeyBinary` (std::ostream &str, const `PubKey` &pk)
- friend void `::helib::readPubKeyBinary` (std::istream &str, `PubKey` &pk)
- void `hackPtxtSpace` (long p2r)
- template<> long `Encrypt` (Ctxt &ciphertxt, const Ptxt< `BGV` > &plaintext, long ptxtSpace) const
- template<> long `Encrypt` (Ctxt &ciphertxt, const Ptxt< `CKKS` > &plaintext, `UNUSED` long ptxtSpace) const

### Find key-switching matrices

- const std::vector< `KeySwitch` > & `keySWlist` () const
- const `KeySwitch` & `getKeySWmatrix` (const `SKHandle` &from, long toID=0) const  
*Find a key-switching matrix by its indexes. If no such matrix exists it returns a dummy matrix with toKeyID== -1.*
- const `KeySwitch` & `getKeySWmatrix` (long fromSPower, long fromXPower, long fromID=0, long toID=0) const
- bool `haveKeySWmatrix` (const `SKHandle` &from, long toID=0) const
- bool `haveKeySWmatrix` (long fromSPower, long fromXPower, long fromID=0, long toID=0) const
- const `KeySwitch` & `getAnyKeySWmatrix` (const `SKHandle` &from) const  
*Is there a matrix from this key to any base key?*
- bool `haveAnyKeySWmatrix` (const `SKHandle` &from) const
- const `KeySwitch` & `getNextKSWmatrix` (long fromXPower, long fromID=0) const  
*Get the next matrix to use for multi-hop automorphism See Section 3.2.2 in the design document.*

### Static Public Member Functions

- static long `ePlusR` (long p)

## Friends

- class [SecKey](#)
- `std::ostream & operator<<` (`std::ostream &str`, `const PubKey &pk`)
- `std::istream & operator>>` (`std::istream &str`, `PubKey &pk`)

### 7.133.1 Detailed Description

The public key.

### 7.133.2 Constructor & Destructor Documentation

#### 7.133.2.1 `PubKey()` [1/3]

```
helib::PubKey::PubKey ( )
```

This constructor thorws run-time error if `activeContext=nullptr`.

#### 7.133.2.2 `PubKey()` [2/3]

```
helib::PubKey::PubKey (
    const Context & _context ) [explicit]
```

#### 7.133.2.3 `PubKey()` [3/3]

```
helib::PubKey::PubKey (
    const PubKey & other )
```

Copy constructor.

#### 7.133.2.4 `~PubKey()`

```
virtual helib::PubKey::~~PubKey ( ) [virtual], [default]
```

Default destructor.

### 7.133.3 Member Function Documentation

#### 7.133.3.1 CKKSencrypt() [1/2]

```
void helib::PubKey::CKKSencrypt (
    Ctxt & ciphertext,
    const NTL::ZZX & plaintext,
    double ptxtSize = 1.0,
    double scaling = 0.0 ) const
```

#### 7.133.3.2 CKKSencrypt() [2/2]

```
void helib::PubKey::CKKSencrypt (
    Ctxt & ciphertext,
    const zzX & plaintext,
    double ptxtSize = 1.0,
    double scaling = 0.0 ) const
```

#### 7.133.3.3 clear()

```
void helib::PubKey::clear ( ) [virtual]
```

Clear all public-key data.

Reimplemented in [helib::SecKey](#).

#### 7.133.3.4 Encrypt() [1/7]

```
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const NTL::ZZX & plaintext,
    long ptxtSpace,
    bool highNoise ) const
```

Encrypts plaintext, result returned in the ciphertext argument. When called with highNoise=true, returns a ciphertext with noise level approximately  $q/8$ . For [BGV](#), ptxtSpace is the intended plaintext space, which cannot be co-prime with pubEncrKey.ptxtSpace. The returned value is the plaintext-space for the resulting ciphertext, which is  $G \leftarrow CD(ptxtSpace, pubEncrKey.ptxtSpace)$ . For [CKKS](#), ptxtSpace is a bound on the size of the complex plaintext elements that are encoded in ptxt (before scaling), it is assumed that they are scaled by  $context.alMod.encode \leftarrow ScalingFactor()$ . The returned value is the same as the argument ptxtSpace.

**7.133.3.5 Encrypt()** [2/7]

```
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const NTL::ZZX & plaintext,
    long ptxtSpace = 0 ) const [virtual]
```

Reimplemented in [helib::SecKey](#).

**7.133.3.6 Encrypt()** [3/7]

```
template<>
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const Ptxt< BGV > & plaintext,
    long ptxtSpace ) const
```

**7.133.3.7 Encrypt()** [4/7]

```
template<>
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const Ptxt< CKKS > & plaintext,
    UNUSED long ptxtSpace ) const
```

**7.133.3.8 Encrypt()** [5/7]

```
template<typename Scheme >
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const Ptxt< Scheme > & plaintext,
    long ptxtSpace = 0 ) const
```

Encrypts a plaintext into a ciphertext.

**Template Parameters**

<i>Scheme</i>	Encryption scheme used (must be <a href="#">BGV</a> or <a href="#">CKKS</a> ).
---------------	--

## Parameters

<i>ciphertext</i>	Ciphertext into which to encrypt.
<i>plaintext</i>	Plaintext to encrypt.

## Returns

Plaintext space.

**7.133.3.9 Encrypt()** [6/7]

```
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const zzX & plaintext,
    long ptxtSpace,
    bool highNoise ) const
```

**7.133.3.10 Encrypt()** [7/7]

```
long helib::PubKey::Encrypt (
    Ctxt & ciphertext,
    const zzX & plaintext,
    long ptxtSpace = 0 ) const [virtual]
```

Reimplemented in [helib::SecKey](#).

**7.133.3.11 ePlusR()**

```
static long helib::PubKey::ePlusR (
    long p ) [static]
```

**7.133.3.12 getAnyKeySWmatrix()**

```
const KeySwitch & helib::PubKey::getAnyKeySWmatrix (
    const SKHandle & from ) const
```

Is there a matrix from this key to *any* base key?

**7.133.3.13 getContext()**

```
const Context & helib::PubKey::getContext ( ) const
```

**7.133.3.14 getKeySWmatrix() [1/2]**

```
const KeySwitch & helib::PubKey::getKeySWmatrix (
    const SKHandle & from,
    long toID = 0 ) const
```

Find a key-switching matrix by its indexes. If no such matrix exists it returns a dummy matrix with toKeyID== -1.

**7.133.3.15 getKeySWmatrix() [2/2]**

```
const KeySwitch & helib::PubKey::getKeySWmatrix (
    long fromSPower,
    long fromXPower,
    long fromID = 0,
    long toID = 0 ) const
```

**7.133.3.16 getKSStrategy()**

```
long helib::PubKey::getKSStrategy (
    long dim ) const
```

get KS strategy for dimension dim dim == -1 is Frobenius

**7.133.3.17 getNextKSWmatrix()**

```
const KeySwitch & helib::PubKey::getNextKSWmatrix (
    long fromXPower,
    long fromID = 0 ) const
```

Get the next matrix to use for multi-hop automorphism See Section 3.2.2 in the design document.

**7.133.3.18 getPtxtSpace()**

```
long helib::PubKey::getPtxtSpace ( ) const
```



### 7.133.3.19 getKeyBound()

```
double helib::PubKey::getKeyBound (
    long keyID = 0 ) const
```

The size of the secret key.

### 7.133.3.20 hackPtxtSpace()

```
void helib::PubKey::hackPtxtSpace (
    long p2r ) [inline]
```

### 7.133.3.21 haveAnyKeySWmatrix()

```
bool helib::PubKey::haveAnyKeySWmatrix (
    const SKHandle & from ) const
```

### 7.133.3.22 haveKeySWmatrix() [1/2]

```
bool helib::PubKey::haveKeySWmatrix (
    const SKHandle & from,
    long toID = 0 ) const
```

### 7.133.3.23 haveKeySWmatrix() [2/2]

```
bool helib::PubKey::haveKeySWmatrix (
    long fromSPower,
    long fromXPower,
    long fromID = 0,
    long toID = 0 ) const
```

### 7.133.3.24 isBootstrappable()

```
bool helib::PubKey::isBootstrappable ( ) const
```

#### 7.133.3.25 isCKKS()

```
bool helib::PubKey::isCKKS ( ) const
```

#### 7.133.3.26 isReachable()

```
bool helib::PubKey::isReachable (
    long k,
    long keyID = 0 ) const
```

Is it possible to re-linearize the automorphism  $X \rightarrow X^k$  See Section 3.2.2 in the design document (KeySwitchMap)

#### 7.133.3.27 keyExists()

```
bool helib::PubKey::keyExists (
    long keyID ) const
```

#### 7.133.3.28 keySWlist()

```
const std::vector< KeySwitch > & helib::PubKey::keySWlist ( ) const
```

#### 7.133.3.29 operator"!=()"

```
bool helib::PubKey::operator!= (
    const PubKey & other ) const
```

#### 7.133.3.30 operator=="()

```
bool helib::PubKey::operator== (
    const PubKey & other ) const
```

#### 7.133.3.31 reCrypt()

```
void helib::PubKey::reCrypt (
    Ctxt & ctxt ) const
```

### 7.133.3.32 setKeySwitchMap()

```
void helib::PubKey::setKeySwitchMap (
    long keyId = 0 )
```

Compute the reachability graph of key-switching matrices See Section 3.2.2 in the design document (KeySwitch↔Map)

### 7.133.3.33 setKSStrategy()

```
void helib::PubKey::setKSStrategy (
    long dim,
    int val )
```

set KS strategy for dimension dim dim == -1 is Frobenius

### 7.133.3.34 thinReCrypt()

```
void helib::PubKey::thinReCrypt (
    Ctxt & ctxt ) const
```

### 7.133.3.35 void ::helib::readPubKeyBinary()

```
helib::PubKey::void ::helib::readPubKeyBinary (
    std::istream & str,
    PubKey & pk )
```

### 7.133.3.36 void ::helib::writePubKeyBinary()

```
helib::PubKey::void ::helib::writePubKeyBinary (
    std::ostream & str,
    const PubKey & pk )
```

## 7.133.4 Friends And Related Function Documentation

#### 7.133.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & str,
    const PubKey & pk ) [friend]
```

#### 7.133.4.2 operator>>

```
std::istream& operator>> (
    std::istream & str,
    PubKey & pk ) [friend]
```

#### 7.133.4.3 SecKey

```
friend class SecKey [friend]
```

The documentation for this class was generated from the following files:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/keys.h](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/keys.cpp](#)
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/recryption.cpp](#)

## 7.134 helib::PubKeyHack Struct Reference

### Public Attributes

- const [Context](#) & context
- [Ctxt pubEncrKey](#)
- std::vector< long > [skHwts](#)
- std::vector< [KeySwitch](#) > [keySwitching](#)
- std::vector< std::vector< long > > [keySwitchMap](#)
- NTL::Vec< int > [KS\\_strategy](#)
- long [recryptKeyID](#)
- [Ctxt recryptEkey](#)

### 7.134.1 Member Data Documentation

#### 7.134.1.1 context

```
const Context& helib::PubKeyHack::context
```

### 7.134.1.2 keySwitching

```
std::vector<KeySwitch> helib::PubKeyHack::keySwitching
```

### 7.134.1.3 keySwitchMap

```
std::vector<std::vector<long> > helib::PubKeyHack::keySwitchMap
```

### 7.134.1.4 KS\_strategy

```
NTL::Vec<int> helib::PubKeyHack::KS_strategy
```

### 7.134.1.5 pubEncrKey

```
Ctxt helib::PubKeyHack::pubEncrKey
```

The public encryption key is an encryption of 0, relative to the first secret key

### 7.134.1.6 reencryptEkey

```
Ctxt helib::PubKeyHack::reencryptEkey
```

### 7.134.1.7 reencryptKeyID

```
long helib::PubKeyHack::reencryptKeyID
```

### 7.134.1.8 skHwts

```
std::vector<long> helib::PubKeyHack::skHwts
```

The documentation for this struct was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[recryption.cpp](#)

## 7.135 helib::quarter\_FFT Struct Reference

```
#include <PAlgebra.h>
```

### Public Member Functions

- [quarter\\_FFT](#) (long m)

### Public Attributes

- [PGFFT](#) [fft](#)
- `std::vector< std::complex< double > >` [pow1](#)
- `std::vector< std::complex< double > >` [pow2](#)

### 7.135.1 Constructor & Destructor Documentation

#### 7.135.1.1 quarter\_FFT()

```
helib::quarter_FFT::quarter_FFT (  
    long m )
```

### 7.135.2 Member Data Documentation

#### 7.135.2.1 fft

```
PGFFT helib::quarter_FFT::fft
```

#### 7.135.2.2 pow1

```
std::vector<std::complex<double> > helib::quarter_FFT::pow1
```

#### 7.135.2.3 pow2

```
std::vector<std::complex<double> > helib::quarter_FFT::pow2
```

The documentation for this struct was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[PAlgebra.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[PAlgebra.cpp](#)

## 7.136 helib::random\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa)

### 7.136.1 Member Function Documentation

#### 7.136.1.1 apply()

```
template<typename type >
static void helib::random_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa ) [inline], [static]
```

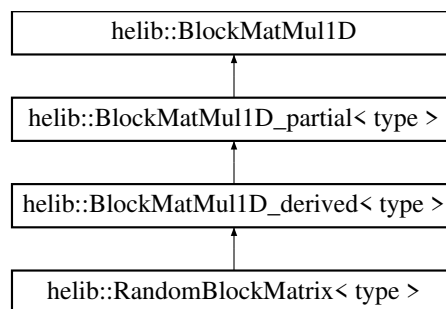
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EncryptedArray.cpp](#)

## 7.137 helib::RandomBlockMatrix< type > Class Template Reference

```
#include <randomMatrices.h>
```

Inheritance diagram for helib::RandomBlockMatrix< type >:



### Public Member Functions

- [RandomBlockMatrix](#) (const [EncryptedArray](#) &\_ea, long \_dim)
- bool [get](#) (mat\_R &out, long i, long j, [UNUSED](#) long k) const override
- const [EncryptedArray](#) & [getEA](#) () const override
- long [getDim](#) () const override
- bool [multipleTransforms](#) () const override

## Additional Inherited Members

### 7.137.1 Constructor & Destructor Documentation

#### 7.137.1.1 RandomBlockMatrix()

```
template<typename type >
helib::RandomBlockMatrix< type >::RandomBlockMatrix (
    const EncryptedArray & _ea,
    long _dim ) [inline]
```

### 7.137.2 Member Function Documentation

#### 7.137.2.1 get()

```
template<typename type >
bool helib::RandomBlockMatrix< type >::get (
    mat_R & out,
    long i,
    long j,
    UNUSED long k ) const [inline], [override]
```

#### 7.137.2.2 getDim()

```
template<typename type >
long helib::RandomBlockMatrix< type >::getDim ( ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMul1D](#).

#### 7.137.2.3 getEA()

```
template<typename type >
const EncryptedArray& helib::RandomBlockMatrix< type >::getEA ( ) const [inline], [override],
[virtual]
```

Implements [helib::BlockMatMul1D](#).



### 7.137.2.4 multipleTransforms()

```
template<typename type >
bool helib::RandomBlockMatrix< type >::multipleTransforms ( ) const [inline], [override],
[virtual]
```

Implements [helib::BlockMatMul1D\\_derived< type >](#).

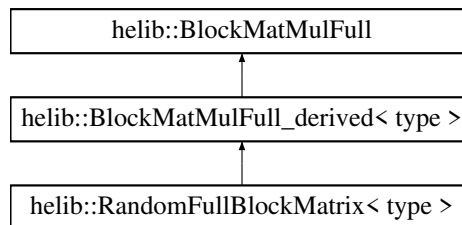
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[randomMatrices.h](#)

## 7.138 helib::RandomFullBlockMatrix< type > Class Template Reference

```
#include <randomMatrices.h>
```

Inheritance diagram for helib::RandomFullBlockMatrix< type >:



### Public Member Functions

- [RandomFullBlockMatrix](#) (const [EncryptedArray](#) &\_ea)
- bool [get](#) (mat\_R &out, long i, long j) const override
- const [EncryptedArray](#) & [getEA](#) () const override

### Additional Inherited Members

#### 7.138.1 Constructor & Destructor Documentation

##### 7.138.1.1 RandomFullBlockMatrix()

```
template<typename type >
helib::RandomFullBlockMatrix< type >::RandomFullBlockMatrix (
    const EncryptedArray & _ea ) [inline]
```

#### 7.138.2 Member Function Documentation

### 7.138.2.1 get()

```
template<typename type >
bool helib::RandomFullBlockMatrix< type >::get (
    mat_R & out,
    long i,
    long j ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMulFull\\_derived< type >](#).

### 7.138.2.2 getEA()

```
template<typename type >
const EncryptedArray& helib::RandomFullBlockMatrix< type >::getEA ( ) const [inline], [override],
[virtual]
```

Implements [helib::BlockMatMulFull](#).

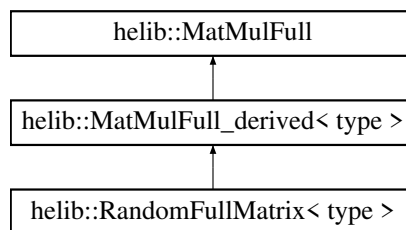
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[randomMatrices.h](#)

## 7.139 helib::RandomFullMatrix< type > Class Template Reference

```
#include <randomMatrices.h>
```

Inheritance diagram for `helib::RandomFullMatrix< type >`:



### Public Member Functions

- [RandomFullMatrix](#) (const [EncryptedArray](#) &\_ea)
- bool [get](#) (RX &out, long i, long j) const override
- const [EncryptedArray](#) & [getEA](#) () const override

### Additional Inherited Members

### 7.139.1 Constructor & Destructor Documentation

### 7.139.1.1 RandomFullMatrix()

```
template<typename type >
helib::RandomFullMatrix< type >::RandomFullMatrix (
    const EncryptedArray & _ea ) [inline]
```

## 7.139.2 Member Function Documentation

### 7.139.2.1 get()

```
template<typename type >
bool helib::RandomFullMatrix< type >::get (
    RX & out,
    long i,
    long j ) const [inline], [override], [virtual]
```

Implements [helib::MatMulFull\\_derived< type >](#).

### 7.139.2.2 getEA()

```
template<typename type >
const EncryptedArray& helib::RandomFullMatrix< type >::getEA ( ) const [inline], [override],
[virtual]
```

Implements [helib::MatMulFull](#).

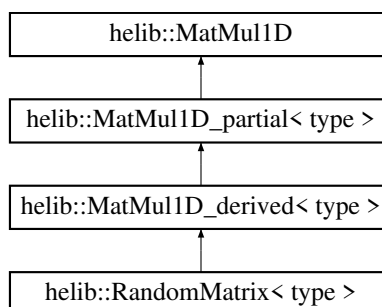
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[randomMatrices.h](#)

## 7.140 helib::RandomMatrix< type > Class Template Reference

```
#include <randomMatrices.h>
```

Inheritance diagram for [helib::RandomMatrix< type >](#):



## Public Member Functions

- virtual `~RandomMatrix ()`
- `RandomMatrix (const EncryptedArray &_ea, long _dim)`
- const `EncryptedArray & getEA ()` const override
- bool `multipleTransforms ()` const override
- long `getDim ()` const override
- bool `get (RX &out, long i, long j, UNUSED long k)` const override

## Additional Inherited Members

### 7.140.1 Constructor & Destructor Documentation

#### 7.140.1.1 `~RandomMatrix()`

```
template<typename type >
virtual helib::RandomMatrix< type >::~~RandomMatrix ( ) [inline], [virtual]
```

#### 7.140.1.2 `RandomMatrix()`

```
template<typename type >
helib::RandomMatrix< type >::RandomMatrix (
    const EncryptedArray & _ea,
    long _dim ) [inline]
```

### 7.140.2 Member Function Documentation

#### 7.140.2.1 `get()`

```
template<typename type >
bool helib::RandomMatrix< type >::get (
    RX & out,
    long i,
    long j,
    UNUSED long k ) const [inline], [override]
```

**7.140.2.2 getDim()**

```
template<typename type >
long helib::RandomMatrix< type >::getDim ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D](#).

**7.140.2.3 getEA()**

```
template<typename type >
const EncryptedArray& helib::RandomMatrix< type >::getEA ( ) const [inline], [override],
[virtual]
```

Implements [helib::MatMul1D](#).

**7.140.2.4 multipleTransforms()**

```
template<typename type >
bool helib::RandomMatrix< type >::multipleTransforms ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D\\_derived< type >](#).

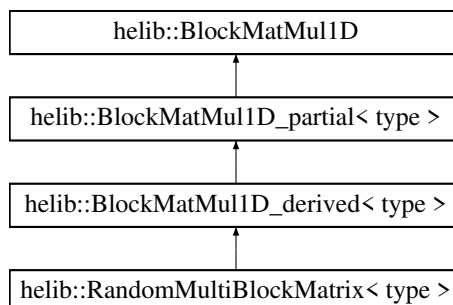
The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[randomMatrices.h](#)

**7.141 helib::RandomMultiBlockMatrix< type > Class Template Reference**

```
#include <randomMatrices.h>
```

Inheritance diagram for [helib::RandomMultiBlockMatrix< type >](#):



## Public Member Functions

- [RandomMultiBlockMatrix](#) (const [EncryptedArray](#) &\_ea, long \_dim)
- bool [get](#) (mat\_R &out, long i, long j, long k) const override
- const [EncryptedArray](#) & [getEA](#) () const override
- long [getDim](#) () const override
- bool [multipleTransforms](#) () const override

## Additional Inherited Members

### 7.141.1 Constructor & Destructor Documentation

#### 7.141.1.1 RandomMultiBlockMatrix()

```
template<typename type >
helib::RandomMultiBlockMatrix< type >::RandomMultiBlockMatrix (
    const EncryptedArray & _ea,
    long _dim ) [inline]
```

### 7.141.2 Member Function Documentation

#### 7.141.2.1 get()

```
template<typename type >
bool helib::RandomMultiBlockMatrix< type >::get (
    mat_R & out,
    long i,
    long j,
    long k ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMul1D\\_derived< type >](#).

#### 7.141.2.2 getDim()

```
template<typename type >
long helib::RandomMultiBlockMatrix< type >::getDim ( ) const [inline], [override], [virtual]
```

Implements [helib::BlockMatMul1D](#).

## 7.141.2.3 getEA()

```
template<typename type >
const EncryptedArray& helib::RandomMultiBlockMatrix< type >::getEA ( ) const [inline], [override],
[virtual]
```

Implements [helib::BlockMatMul1D](#).

## 7.141.2.4 multipleTransforms()

```
template<typename type >
bool helib::RandomMultiBlockMatrix< type >::multipleTransforms ( ) const [inline], [override],
[virtual]
```

Implements [helib::BlockMatMul1D\\_derived< type >](#).

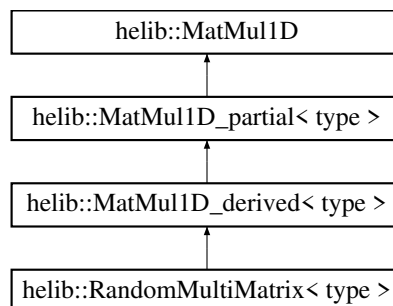
The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/randomMatrices.h](#)

## 7.142 helib::RandomMultiMatrix&lt; type &gt; Class Template Reference

```
#include <randomMatrices.h>
```

Inheritance diagram for helib::RandomMultiMatrix< type >:



## Public Member Functions

- virtual [~RandomMultiMatrix](#) ( )
- [RandomMultiMatrix](#) (const [EncryptedArray](#) &\_ea, long \_dim)
- const [EncryptedArray](#) & [getEA](#) ( ) const override
- bool [multipleTransforms](#) ( ) const override
- long [getDim](#) ( ) const override
- bool [get](#) (RX &out, long i, long j, long k) const override

## Additional Inherited Members

### 7.142.1 Constructor & Destructor Documentation

#### 7.142.1.1 `~RandomMultiMatrix()`

```
template<typename type >
virtual helib::RandomMultiMatrix< type >::~~RandomMultiMatrix ( ) [inline], [virtual]
```

#### 7.142.1.2 `RandomMultiMatrix()`

```
template<typename type >
helib::RandomMultiMatrix< type >::~RandomMultiMatrix (
    const EncryptedArray & _ea,
    long _dim ) [inline]
```

### 7.142.2 Member Function Documentation

#### 7.142.2.1 `get()`

```
template<typename type >
bool helib::RandomMultiMatrix< type >::get (
    RX & out,
    long i,
    long j,
    long k ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D\\_derived< type >](#).

#### 7.142.2.2 `getDim()`

```
template<typename type >
long helib::RandomMultiMatrix< type >::getDim ( ) const [inline], [override], [virtual]
```

Implements [helib::MatMul1D](#).



### 7.142.2.3 getEA()

```
template<typename type >
const EncryptedArray& helib::RandomMultiMatrix< type >::getEA ( ) const [inline], [override],
[virtual]
```

Implements [helib::MatMul1D](#).

### 7.142.2.4 multipleTransforms()

```
template<typename type >
bool helib::RandomMultiMatrix< type >::multipleTransforms ( ) const [inline], [override],
[virtual]
```

Implements [helib::MatMul1D\\_derived< type >](#).

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[randomMatrices.h](#)

## 7.143 helib::RandomState Class Reference

Facility for "restoring" the [NTL](#) PRG state.

```
#include <NumbTh.h>
```

### Public Member Functions

- [RandomState](#) ()
- void [restore](#) ()  
*Restore the PRG state of [NTL](#).*
- [~RandomState](#) ()

### 7.143.1 Detailed Description

Facility for "restoring" the [NTL](#) PRG state.

[NTL](#)'s random number generation facility is pretty limited, and does not provide a way to save/restore the state of a pseudo-random stream. This class gives us that ability: Constructing a [RandomState](#) object uses the PRG to generate 512 bits and stores them. Upon destruction (or an explicit call to [restore\(\)](#)), these bits are used to re-set the seed of the PRG. A typical usage of the class is as follows:

```
{
    RandomState r;           // save the random state
    SetSeed(something);      // set the PRG seed to something
    ...                      // more code that uses the new PRG seed
} // The destructor is called implicitly, PRG state is restored
```

## 7.143.2 Constructor & Destructor Documentation

### 7.143.2.1 RandomState()

```
helib::RandomState::RandomState ( ) [inline]
```

### 7.143.2.2 ~RandomState()

```
helib::RandomState::~~RandomState ( ) [inline]
```

## 7.143.3 Member Function Documentation

### 7.143.3.1 restore()

```
void helib::RandomState::restore ( ) [inline]
```

Restore the PRG state of [NTL](#).

The documentation for this class was generated from the following file:

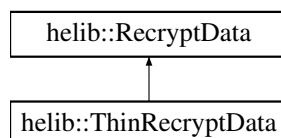
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[NumbTh.h](#)

## 7.144 helib::RecryptData Class Reference

A structure to hold reryption-related data inside the [Context](#).

```
#include <recryption.h>
```

Inheritance diagram for helib::RecryptData:



## Public Member Functions

- [RecryptData](#) ()
- void [init](#) (const [Context](#) &context, const NTL::Vec< long > &mvec\_, bool enableThick, long t=0, bool [build\\_cache](#)=false, bool minimal=false)  
*Initialize the recryption data in the context.*
- bool [operator==](#) (const [RecryptData](#) &other) const
- bool [operator!=](#) (const [RecryptData](#) &other) const

## Static Public Member Functions

- static long [setAE](#) (long &[e](#), long &[ePrime](#), const [Context](#) &context, long t=0)  
*Helper function for computing the recryption parameters.*

## Public Attributes

- NTL::Vec< long > [mvec](#)  
*Some data members that are only used for I/O.*
- long [e](#)  
*partition of m into co-prime factors*
- long [ePrime](#)
- long [skHwt](#)  
*Hamming weight of recryption secret key.*
- std::shared\_ptr< const [PAlgebraMod](#) > [alMod](#)  
*for plaintext space  $p^{\{e-e'+r\}}$*
- std::shared\_ptr< const [EncryptedArray](#) > [ea](#)  
*for plaintext space  $p^{\{e-e'+r\}}$*
- bool [build\\_cache](#)
- std::shared\_ptr< const [EvalMap](#) > [firstMap](#)  
*linear maps*
- std::shared\_ptr< const [EvalMap](#) > [secondMap](#)
- std::shared\_ptr< const [PowerfulDCRT](#) > [p2dConv](#)  
*conversion between ZZx and Powerful*
- std::vector< NTL::ZZx > [unpackSlotEncoding](#)  
*linPolys for unpacking the slots*

## Static Public Attributes

- static constexpr long [defSkHwt](#) = 100  
*default Hamming weight of recryption key*

### 7.144.1 Detailed Description

A structure to hold recryption-related data inside the [Context](#).

### 7.144.2 Constructor & Destructor Documentation

### 7.144.2.1 RecryptData()

```
helib::RecryptData::RecryptData ( ) [inline]
```

## 7.144.3 Member Function Documentation

### 7.144.3.1 init()

```
void helib::RecryptData::init (
    const Context & context,
    const NTL::Vec< long > & mvec_,
    bool enableThick,
    long t = 0,
    bool build_cache = false,
    bool minimal = false )
```

Initialize the recryption data in the context.

### 7.144.3.2 operator"!="()

```
bool helib::RecryptData::operator!= (
    const RecryptData & other ) const [inline]
```

### 7.144.3.3 operator==( )

```
bool helib::RecryptData::operator== (
    const RecryptData & other ) const
```

### 7.144.3.4 setAE()

```
long helib::RecryptData::setAE (
    long & e,
    long & ePrime,
    const Context & context,
    long t = 0 ) [static]
```

Helper function for computing the recryption parameters.

## 7.144.4 Member Data Documentation

### 7.144.4.1 alMod

`std::shared_ptr<const PAlgebraMod> helib::RecryptData::alMod`

for plaintext space  $p^{\{e-e'+r\}}$

### 7.144.4.2 build\_cache

`bool helib::RecryptData::build_cache`

### 7.144.4.3 defSkHwt

`constexpr long helib::RecryptData::defSkHwt = 100 [static], [constexpr]`

default Hamming weight of recryption key

### 7.144.4.4 e

`long helib::RecryptData::e`

partition of m into co-prime factors

skkey encrypted wrt space  $p^{\{e-e'+r\}}$

### 7.144.4.5 ea

`std::shared_ptr<const EncryptedArray> helib::RecryptData::ea`

for plaintext space  $p^{\{e-e'+r\}}$

### 7.144.4.6 ePrime

`long helib::RecryptData::ePrime`

#### 7.144.4.7 firstMap

```
std::shared_ptr<const EvalMap> helib::RecryptData::firstMap
```

linear maps

#### 7.144.4.8 mvec

```
NTL::Vec<long> helib::RecryptData::mvec
```

Some data members that are only used for I/O.

#### 7.144.4.9 p2dConv

```
std::shared_ptr<const PowerfulDCRT> helib::RecryptData::p2dConv
```

conversion between ZZ<sub>X</sub> and Powerful

#### 7.144.4.10 secondMap

```
std::shared_ptr<const EvalMap> helib::RecryptData::secondMap
```

#### 7.144.4.11 skHwt

```
long helib::RecryptData::skHwt
```

Hamming weight of recryption secret key.

#### 7.144.4.12 unpackSlotEncoding

```
std::vector<NTL::ZZX> helib::RecryptData::unpackSlotEncoding
```

linPolys for unpacking the slots

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[recryption.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[recryption.cpp](#)

## 7.145 helib::replicate\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, long i)

### 7.145.1 Member Function Documentation

#### 7.145.1.1 apply()

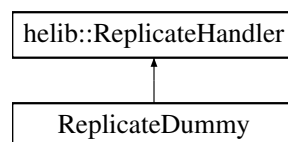
```
template<typename type >
static void helib::replicate_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    long i ) [inline], [static]
```

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/replicate.cpp](#)

## 7.146 ReplicateDummy Class Reference

Inheritance diagram for ReplicateDummy:



### Public Member Functions

- [ReplicateDummy](#) ()
- virtual void [handle](#) (const [Ctxt](#) &ctxt)

### 7.146.1 Constructor & Destructor Documentation

#### 7.146.1.1 ReplicateDummy()

```
ReplicateDummy::ReplicateDummy ( ) [inline]
```

## 7.146.2 Member Function Documentation

### 7.146.2.1 handle()

```
virtual void ReplicateDummy::handle (
    const Ctxt & ctxt ) [inline], [virtual]
```

Implements [helib::ReplicateHandler](#).

The documentation for this class was generated from the following file:

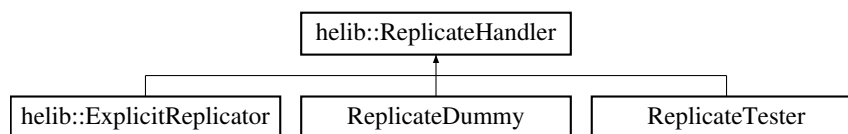
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[Test\\_Timing.cpp](#)

## 7.147 helib::ReplicateHandler Class Reference

An abstract class to handle call-backs to get the output of replicate.

```
#include <replicate.h>
```

Inheritance diagram for helib::ReplicateHandler:



### Public Member Functions

- virtual void [handle](#) (const [Ctxt](#) &ctxt)=0
- virtual [~ReplicateHandler](#) ()
- virtual bool [earlyStop](#) (long d, long k, long prodDim)

### 7.147.1 Detailed Description

An abstract class to handle call-backs to get the output of replicate.

### 7.147.2 Constructor & Destructor Documentation

#### 7.147.2.1 ~ReplicateHandler()

```
virtual helib::ReplicateHandler::~~ReplicateHandler ( ) [inline], [virtual]
```



### 7.147.3 Member Function Documentation

#### 7.147.3.1 earlyStop()

```
virtual bool helib::ReplicateHandler::earlyStop (
    long d,
    long k,
    long prodDim ) [inline], [virtual]
```

#### 7.147.3.2 handle()

```
virtual void helib::ReplicateHandler::handle (
    const Ctxt & ctxt ) [pure virtual]
```

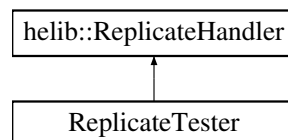
Implemented in [helib::ExplicitReplicator](#), [ReplicateDummy](#), and [ReplicateTester](#).

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/replicate.h](#)

## 7.148 ReplicateTester Class Reference

Inheritance diagram for ReplicateTester:



### Public Member Functions

- [ReplicateTester](#) (const [SecKey](#) & sKey, const [EncryptedArray](#) & \_ea, const [PlaintextArray](#) & \_pa, long \_B)
- virtual void [handle](#) (const [Ctxt](#) & ctxt)

### Public Attributes

- const [SecKey](#) & sKey
- const [EncryptedArray](#) & ea
- const [PlaintextArray](#) & pa
- long B
- double t\_last
- double t\_total
- long pos
- bool error

## 7.148.1 Constructor & Destructor Documentation

### 7.148.1.1 ReplicateTester()

```
ReplicateTester::ReplicateTester (
    const SecKey & _sKey,
    const EncryptedArray & _ea,
    const PlaintextArray & _pa,
    long _B ) [inline]
```

## 7.148.2 Member Function Documentation

### 7.148.2.1 handle()

```
virtual void ReplicateTester::handle (
    const Ctxt & ctxt ) [inline], [virtual]
```

Implements [helib::ReplicateHandler](#).

## 7.148.3 Member Data Documentation

### 7.148.3.1 B

```
long ReplicateTester::B
```

### 7.148.3.2 ea

```
const EncryptedArray& ReplicateTester::ea
```

### 7.148.3.3 error

```
bool ReplicateTester::error
```

#### 7.148.3.4 pa

```
const PlaintextArray& ReplicateTester::pa
```

#### 7.148.3.5 pos

```
long ReplicateTester::pos
```

#### 7.148.3.6 sKey

```
const SecKey& ReplicateTester::sKey
```

#### 7.148.3.7 t\_last

```
double ReplicateTester::t_last
```

#### 7.148.3.8 t\_total

```
double ReplicateTester::t_total
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[Test\\_Replicate.cpp](#)

## 7.149 helib::rotate\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, long k)

### 7.149.1 Member Function Documentation

### 7.149.1.1 apply()

```
template<typename type >
static void helib::rotate_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    long k ) [inline], [static]
```

The documentation for this class was generated from the following file:

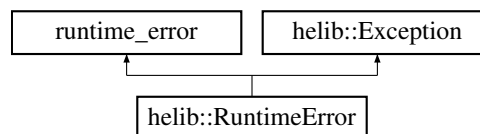
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EncryptedArray.cpp

## 7.150 helib::RuntimeError Class Reference

Inherits from [Exception](#) and `std::runtime_error`.

```
#include <exceptions.h>
```

Inheritance diagram for `helib::RuntimeError`:



### Public Member Functions

- [RuntimeError](#) (const std::string &what\_arg)
- [RuntimeError](#) (const char \*what\_arg)
- virtual [~RuntimeError](#) ()
- virtual const char \* [what](#) () const noexcept override

### Additional Inherited Members

#### 7.150.1 Detailed Description

Inherits from [Exception](#) and `std::runtime_error`.

#### 7.150.2 Constructor & Destructor Documentation

**7.150.2.1 RuntimeError() [1/2]**

```
helib::RuntimeError::RuntimeError (
    const std::string & what_arg ) [inline], [explicit]
```

**7.150.2.2 RuntimeError() [2/2]**

```
helib::RuntimeError::RuntimeError (
    const char * what_arg ) [inline], [explicit]
```

**7.150.2.3 ~RuntimeError()**

```
virtual helib::RuntimeError::~~RuntimeError ( ) [inline], [virtual]
```

**7.150.3 Member Function Documentation****7.150.3.1 what()**

```
virtual const char* helib::RuntimeError::what ( ) const [inline], [override], [virtual], [noexcept]
```

Implements [helib::Exception](#).

The documentation for this class was generated from the following file:

- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/include/helib/exceptions.h](#)

**7.151 helib::ScratchCell Class Reference**

A class to help manage the allocation of temporary [Ctxt](#) objects.

**Public Member Functions**

- [ScratchCell](#) (const [Ctxt](#) &c)
- [ScratchCell](#) ([ScratchCell](#) &&other)

**Public Attributes**

- `std::atomic_bool` [used](#)
- `std::unique_ptr< Ctxt >` [ct](#)

### 7.151.1 Detailed Description

A class to help manage the allocation of temporary [Ctxt](#) objects.

### 7.151.2 Constructor & Destructor Documentation

#### 7.151.2.1 ScratchCell() [1/2]

```
helib::ScratchCell::ScratchCell (  
    const Ctxt & c ) [inline]
```

#### 7.151.2.2 ScratchCell() [2/2]

```
helib::ScratchCell::ScratchCell (  
    ScratchCell && other ) [inline]
```

### 7.151.3 Member Data Documentation

#### 7.151.3.1 ct

```
std::unique_ptr<Ctxt> helib::ScratchCell::ct
```

#### 7.151.3.2 used

```
std::atomic_bool helib::ScratchCell::used
```

The documentation for this class was generated from the following file:

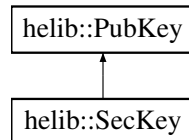
- [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/HElib/src/binaryArith.cpp](#)

## 7.152 helib::SecKey Class Reference

The secret key.

```
#include <keys.h>
```

Inheritance diagram for helib::SecKey:



### Public Member Functions

- [SecKey](#) ()=delete
- [~SecKey](#) () override=default
- [SecKey](#) (const [Context](#) &\_context)
- bool [operator==](#) (const [SecKey](#) &other) const
- bool [operator!=](#) (const [SecKey](#) &other) const
- void [clear](#) () override
- *Clear all secret-key data.*
- long [ImportSecKey](#) (const [DoubleCRT](#) &sKey, double bound, long ptxtSpace=0, long maxDegKswitch=3)
- long [GenSecKey](#) (long hwt=0, long ptxtSpace=0, long maxDegKswitch=3)
- void [GenKeySWmatrix](#) (long fromSPower, long fromXPower, long fromKeyIdx=0, long toKeyIdx=0, long ptxtSpace=0)
- void [Decrypt](#) (NTL::ZZX &plaintext, const [Ctxt](#) &ciphertext) const
- template<typename Scheme >  
void [Decrypt](#) (Ptxt< Scheme > &plaintext, const [Ctxt](#) &ciphertext) const
- *Decrypt a ciphertext into a plaintext.*
- void [Decrypt](#) (NTL::ZZX &plaintext, const [Ctxt](#) &ciphertext, NTL::ZZX &f) const
- *Debugging version, returns in f the polynomial before reduction modulo the ptxtSpace.*
- long [skEncrypt](#) ([Ctxt](#) &ctxt, const NTL::ZZX &ptxt, long ptxtSpace, long skldx) const
- *Symmetric encryption using the secret key.*
- long [skEncrypt](#) ([Ctxt](#) &ctxt, const [zzX](#) &ptxt, long ptxtSpace, long skldx) const
- long [Encrypt](#) ([Ctxt](#) &ciphertext, const NTL::ZZX &plaintext, long ptxtSpace=0) const override
- long [Encrypt](#) ([Ctxt](#) &ciphertext, const [zzX](#) &plaintext, long ptxtSpace=0) const override
- long [genRecryptData](#) ()
- *Generate bootstrapping data if needed, returns index of key.*
- friend void [::helib::writeSecKeyBinary](#) (std::ostream &str, const [SecKey](#) &sk)
- friend void [::helib::readSecKeyBinary](#) (std::istream &str, [SecKey](#) &sk)
- template<> void [Decrypt](#) (Ptxt< [BGV](#) > &plaintext, const [Ctxt](#) &ciphertext) const
- template<> void [Decrypt](#) (Ptxt< [CKKS](#) > &plaintext, const [Ctxt](#) &ciphertext) const

### Public Attributes

- std::vector< [DoubleCRT](#) > [sKeys](#)

## Friends

- `std::ostream & operator<<` (`std::ostream &str`, `const SecKey &sk`)
- `std::istream & operator>>` (`std::istream &str`, `SecKey &sk`)

## Additional Inherited Members

### 7.152.1 Detailed Description

The secret key.

### 7.152.2 Constructor & Destructor Documentation

#### 7.152.2.1 SecKey() [1/2]

```
helib::SecKey::SecKey ( ) [delete]
```

#### 7.152.2.2 ~SecKey()

```
helib::SecKey::~~SecKey ( ) [override], [default]
```

#### 7.152.2.3 SecKey() [2/2]

```
helib::SecKey::SecKey (
    const Context & _context ) [explicit]
```

### 7.152.3 Member Function Documentation

#### 7.152.3.1 clear()

```
void helib::SecKey::clear ( ) [override], [virtual]
```

Clear all secret-key data.

Reimplemented from [helib::PubKey](#).



**7.152.3.2 Decrypt()** [1/5]

```
void helib::SecKey::Decrypt (
    NTL::ZZX & plaintext,
    const Ctxt & ciphertext ) const
```

**7.152.3.3 Decrypt()** [2/5]

```
void helib::SecKey::Decrypt (
    NTL::ZZX & plaintext,
    const Ctxt & ciphertext,
    NTL::ZZX & f ) const
```

Debugging version, returns in *f* the polynomial before reduction modulo the *ptxtSpace*.

**7.152.3.4 Decrypt()** [3/5]

```
template<>
void helib::SecKey::Decrypt (
    Ptxt< BGV > & plaintext,
    const Ctxt & ciphertext ) const
```

**7.152.3.5 Decrypt()** [4/5]

```
template<>
void helib::SecKey::Decrypt (
    Ptxt< CKKS > & plaintext,
    const Ctxt & ciphertext ) const
```

**7.152.3.6 Decrypt()** [5/5]

```
template<typename Scheme >
void helib::SecKey::Decrypt (
    Ptxt< Scheme > & plaintext,
    const Ctxt & ciphertext ) const
```

Decrypt a ciphertext into a plaintext.

**Template Parameters**

<i>Scheme</i>	Encryption scheme used (must be <a href="#">BGV</a> or <a href="#">CKKS</a> ).
---------------	--

## Parameters

<i>plaintext</i>	Plaintext into which to de-crypt.
<i>ciphertext</i>	Ciphertext to de-crypt.

**7.152.3.7 Encrypt()** [1/2]

```
long helib::SecKey::Encrypt (
    Ctxt & ciphertext,
    const NTL::ZZX & plaintext,
    long ptxtSpace = 0 ) const [override], [virtual]
```

Reimplemented from [helib::PubKey](#).

**7.152.3.8 Encrypt()** [2/2]

```
long helib::SecKey::Encrypt (
    Ctxt & ciphertext,
    const zzX & plaintext,
    long ptxtSpace = 0 ) const [override], [virtual]
```

Reimplemented from [helib::PubKey](#).

**7.152.3.9 GenKeySWmatrix()**

```
void helib::SecKey::GenKeySWmatrix (
    long fromSPower,
    long fromXPower,
    long fromKeyIdx = 0,
    long toKeyIdx = 0,
    long ptxtSpace = 0 )
```

Generate a key-switching matrix and store it in the public key. The  $i$ 'th column of the matrix encrypts  $\text{fromKey} \leftarrow B_1 * B_2 * \dots * B_{i-1} * Q$  under  $\text{toKey}$ , relative to the largest modulus (i.e., all primes) and plaintext space  $p$ .  $Q$  is the product of special primes, and the  $B_i$ 's are the products of primes in the  $i$ 'th digit. The plaintext space defaults to  $2^r$ , as defined by `context.mod2r`.

### 7.152.3.10 genRecryptData()

```
long helib::SecKey::genRecryptData ( )
```

Generate bootstrapping data if needed, returns index of key.

### 7.152.3.11 GenSecKey()

```
long helib::SecKey::GenSecKey (
    long hwt = 0,
    long ptxtSpace = 0,
    long maxDegKswitch = 3 )
```

Key generation: This procedure generates a single secret key, pushes it onto the sKeys list using ImportSecKey from above.

### 7.152.3.12 ImportSecKey()

```
long helib::SecKey::ImportSecKey (
    const DoubleCRT & sKey,
    double bound,
    long ptxtSpace = 0,
    long maxDegKswitch = 3 )
```

We allow the calling application to choose a secret-key polynomial by itself, then insert it into the [SecKey](#) object, getting the index of that secret key in the sKeys list. If this is the first secret-key for this object then the procedure below also generates a corresponding public encryption key. It is assumed that the context already contains all parameters.

### 7.152.3.13 operator!=(())

```
bool helib::SecKey::operator!= (
    const SecKey & other ) const
```

### 7.152.3.14 operator==(())

```
bool helib::SecKey::operator== (
    const SecKey & other ) const
```

**7.152.3.15 skEncrypt() [1/2]**

```
long helib::SecKey::skEncrypt (
    Ctxt & ctxt,
    const NTL::ZZX & ptxt,
    long ptxtSpace,
    long skIdx ) const
```

Symmetric encryption using the secret key.

**7.152.3.16 skEncrypt() [2/2]**

```
long helib::SecKey::skEncrypt (
    Ctxt & ctxt,
    const zzX & ptxt,
    long ptxtSpace,
    long skIdx ) const
```

**7.152.3.17 void ::helib::readSecKeyBinary()**

```
helib::SecKey::void ::helib::readSecKeyBinary (
    std::istream & str,
    SecKey & sk )
```

**7.152.3.18 void ::helib::writeSecKeyBinary()**

```
helib::SecKey::void ::helib::writeSecKeyBinary (
    std::ostream & str,
    const SecKey & sk )
```

**7.152.4 Friends And Related Function Documentation****7.152.4.1 operator<<**

```
std::ostream& operator<< (
    std::ostream & str,
    const SecKey & sk ) [friend]
```

### 7.152.4.2 operator>>

```
std::istream& operator>> (
    std::istream & str,
    SecKey & sk ) [friend]
```

## 7.152.5 Member Data Documentation

### 7.152.5.1 sKeys

```
std::vector<DoubleCRT> helib::SecKey::sKeys
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[keys.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[keys.cpp](#)

## 7.153 helib::shallow\_clone< X > Class Template Reference

Shallow copy: initialize with copy constructor.

```
#include <clonedPtr.h>
```

### Static Public Member Functions

- static X \* [apply](#) (const X \*x)

### 7.153.1 Detailed Description

```
template<typename X>
class helib::shallow_clone< X >
```

Shallow copy: initialize with copy constructor.

#### Template Parameters

X	The class to which this points
---	--------------------------------

### 7.153.2 Member Function Documentation

### 7.153.2.1 apply()

```
template<typename X >
static X* helib::shallow_clone< X >::apply (
    const X * x ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/clonedPtr.h

## 7.154 helib::shift\_pa\_impl< type > Class Template Reference

### Static Public Member Functions

- static void `apply` (const `EncryptedArrayDerived`< type > &ea, `PlaintextArray` &pa, long k)

### 7.154.1 Member Function Documentation

#### 7.154.1.1 apply()

```
template<typename type >
static void helib::shift_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    long k ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EncryptedArray.cpp

## 7.155 helib::SKHandle Class Reference

A handle, describing the secret-key element that "matches" a part, of the form  $s^r(X^t)$ .

```
#include <Ctxt.h>
```

## Public Member Functions

- [SKHandle](#) (long newPowerOfS=0, long newPowerOfX=1, long newSecretKeyID=0)
- void [setBase](#) (long newSecretKeyID=-1)  
*Set powerOfS=powerOfX=1.*
- bool [isBase](#) (long ofKeyID=0) const  
*Is powerOfS==powerOfX==1?*
- void [setOne](#) (long newSecretKeyID=-1)  
*Set powerOfS=0, powerOfX=1.*
- bool [isOne](#) () const  
*Is powerOfS==0?*
- bool [operator==](#) (const [SKHandle](#) &other) const
- bool [operator!=](#) (const [SKHandle](#) &other) const
- long [getPowerOfS](#) () const
- long [getPowerOfX](#) () const
- long [getSecretKeyID](#) () const
- bool [mul](#) (const [SKHandle](#) &a, const [SKHandle](#) &b)  
*Computes the "product" of two handles.*
- void [read](#) (std::istream &str)
- void [write](#) (std::ostream &str) const

## Friends

- class [Ctxt](#)
- std::istream & [operator>>](#) (std::istream &s, [SKHandle](#) &handle)

### 7.155.1 Detailed Description

A handle, describing the secret-key element that "matches" a part, of the form  $s^r(X^t)$ .

### 7.155.2 Constructor & Destructor Documentation

#### 7.155.2.1 SKHandle()

```
helib::SKHandle::SKHandle (
    long newPowerOfS = 0,
    long newPowerOfX = 1,
    long newSecretKeyID = 0 ) [inline]
```

### 7.155.3 Member Function Documentation

**7.155.3.1 getPowerOfS()**

```
long helib::SKHandle::getPowerOfS ( ) const [inline]
```

**7.155.3.2 getPowerOfX()**

```
long helib::SKHandle::getPowerOfX ( ) const [inline]
```

**7.155.3.3 getSecretKeyID()**

```
long helib::SKHandle::getSecretKeyID ( ) const [inline]
```

**7.155.3.4 isBase()**

```
bool helib::SKHandle::isBase (
    long ofKeyID = 0 ) const [inline]
```

Is powerOfS==powerOfX==1?

**7.155.3.5 isOne()**

```
bool helib::SKHandle::isOne ( ) const [inline]
```

Is powerOfS==0?

**7.155.3.6 mul()**

```
bool helib::SKHandle::mul (
    const SKHandle & a,
    const SKHandle & b ) [inline]
```

Computes the "product" of two handles.

The key-ID's and powers of X must match, else an error state arises, which is represented using a key-ID of -1 and returning false. Also, note that inputs may alias outputs.

To determine if the resulting handle can be re-linearized using some key-switching matrices from the public key, use the method `pubKey.haveKeySWmatrix(handle,handle.secretKeyID)`, from the class [PubKey](#) in [keys.h](#)



### 7.155.3.7 operator"!="()

```
bool helib::SKHandle::operator!= (
    const SKHandle & other ) const [inline]
```

### 7.155.3.8 operator==()

```
bool helib::SKHandle::operator== (
    const SKHandle & other ) const [inline]
```

### 7.155.3.9 read()

```
void helib::SKHandle::read (
    std::istream & str )
```

### 7.155.3.10 setBase()

```
void helib::SKHandle::setBase (
    long newSecretKeyID = -1 ) [inline]
```

Set powerOfS=powerOfX=1.

### 7.155.3.11 setOne()

```
void helib::SKHandle::setOne (
    long newSecretKeyID = -1 ) [inline]
```

Set powerOfS=0, powerOfX=1.

### 7.155.3.12 write()

```
void helib::SKHandle::write (
    std::ostream & str ) const
```

## 7.155.4 Friends And Related Function Documentation

#### 7.155.4.1 Ctxt

```
friend class Ctxt [friend]
```

#### 7.155.4.2 operator>>

```
std::istream& operator>> (
    std::istream & s,
    SKHandle & handle ) [friend]
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/Ctxt.h
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Ctxt.cpp

### 7.156 StopReplicate Class Reference

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_Replicate.cpp

### 7.157 helib::sub\_pa\_impl< type > Class Template Reference

#### Static Public Member Functions

- static void [apply](#) (const [EncryptedArrayDerived](#)< type > &ea, [PlaintextArray](#) &pa, const [PlaintextArray](#) &other)

#### 7.157.1 Member Function Documentation

##### 7.157.1.1 apply()

```
template<typename type >
static void helib::sub_pa_impl< type >::apply (
    const EncryptedArrayDerived< type > & ea,
    PlaintextArray & pa,
    const PlaintextArray & other ) [inline], [static]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EncryptedArray.cpp

## 7.158 helib::SubDimension Class Reference

A node in a tree relative to some generator.

```
#include <permutations.h>
```

### Public Member Functions

- [SubDimension](#) (long sz=0, bool gd=false, long ee=0, const NTL::Vec< long > &bns1=dummyBenes, const NTL::Vec< long > &bns2=dummyBenes)
- long [totalLength](#) () const

### Public Attributes

- long [size](#)
- bool [good](#)
- long [e](#)
- NTL::Vec< long > [frstBenes](#)
- NTL::Vec< long > [scndBenes](#)

### Friends

- std::ostream & [operator<<](#) (std::ostream &s, const [SubDimension](#) &tree)

### 7.158.1 Detailed Description

A node in a tree relative to some generator.

### 7.158.2 Constructor & Destructor Documentation

#### 7.158.2.1 SubDimension()

```
helib::SubDimension::SubDimension (
    long sz = 0,
    bool gd = false,
    long ee = 0,
    const NTL::Vec< long > & bns1 = dummyBenes,
    const NTL::Vec< long > & bns2 = dummyBenes ) [inline], [explicit]
```

### 7.158.3 Member Function Documentation

### 7.158.3.1 totalLength()

```
long helib::SubDimension::totalLength ( ) const [inline]
```

## 7.158.4 Friends And Related Function Documentation

### 7.158.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & s,
    const SubDimension & tree ) [friend]
```

## 7.158.5 Member Data Documentation

### 7.158.5.1 e

```
long helib::SubDimension::e
```

### 7.158.5.2 frstBenes

```
NTL::Vec<long> helib::SubDimension::frstBenes
```

### 7.158.5.3 good

```
bool helib::SubDimension::good
```

### 7.158.5.4 scndBenes

```
NTL::Vec<long> helib::SubDimension::scndBenes
```

### 7.158.5.5 size

```
long helib::SubDimension::size
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[permutations.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[permutations.cpp](#)

## 7.159 helib::ThinEvalMap Class Reference

Class that provides the functionality for the linear transforms used in "thin" bootstrapping, where slots are assumed to contain constants. The interface is exactly the same as for [EvalMap](#), except that the constructor does not have a `normal_basis` parameter.

```
#include <EvalMap.h>
```

### Public Member Functions

- [ThinEvalMap](#) (const [EncryptedArray](#) &\_ea, bool minimal, const NTL::Vec< long > &mvec, bool \_invert, bool build\_cache)
- void [upgrade](#) ()
- void [apply](#) (Ctxt &ctxt) const

### 7.159.1 Detailed Description

Class that provides the functionality for the linear transforms used in "thin" bootstrapping, where slots are assumed to contain constants. The interface is exactly the same as for [EvalMap](#), except that the constructor does not have a `normal_basis` parameter.

### 7.159.2 Constructor & Destructor Documentation

#### 7.159.2.1 ThinEvalMap()

```
helib::ThinEvalMap::ThinEvalMap (
    const EncryptedArray & _ea,
    bool minimal,
    const NTL::Vec< long > & mvec,
    bool _invert,
    bool build_cache )
```

### 7.159.3 Member Function Documentation

### 7.159.3.1 apply()

```
void helib::ThinEvalMap::apply (
    Ctxt & ctxt ) const
```

### 7.159.3.2 upgrade()

```
void helib::ThinEvalMap::upgrade ( )
```

The documentation for this class was generated from the following files:

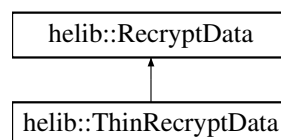
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[EvalMap.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[EvalMap.cpp](#)

## 7.160 helib::ThinRecryptData Class Reference

Same as above, but for "thin" bootstrapping, where the slots are assumed to contain constants.

```
#include <recryption.h>
```

Inheritance diagram for helib::ThinRecryptData:



### Public Member Functions

- void [init](#) (const [Context](#) &context, const NTL::Vec< long > &mvec\_, bool alsoThick, long t=0, bool [build\\_cache](#)=false, bool minimal=false)  
*Initialize the recryption data in the context.*

### Public Attributes

- std::shared\_ptr< const [ThinEvalMap](#) > [coeffToSlot](#)  
*linear maps*
- std::shared\_ptr< const [ThinEvalMap](#) > [slotToCoeff](#)

### Additional Inherited Members

#### 7.160.1 Detailed Description

Same as above, but for "thin" bootstrapping, where the slots are assumed to contain constants.

## 7.160.2 Member Function Documentation

### 7.160.2.1 init()

```
void helib::ThinRecryptData::init (
    const Context & context,
    const NTL::Vec< long > & mvec_,
    bool alsoThick,
    long t = 0,
    bool build_cache = false,
    bool minimal = false )
```

Initialize the recryption data in the context.

## 7.160.3 Member Data Documentation

### 7.160.3.1 coeffToSlot

```
std::shared_ptr<const ThinEvalMap> helib::ThinRecryptData::coeffToSlot
```

linear maps

### 7.160.3.2 slotToCoeff

```
std::shared_ptr<const ThinEvalMap> helib::ThinRecryptData::slotToCoeff
```

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[recryption.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[recryption.cpp](#)

## 7.161 TimingData Class Reference

### Public Member Functions

- [TimingData](#) (long \_m=-1)

## Public Attributes

- long [m](#)
- long [phim](#)
- long [nSlots](#)
- long [p](#)
- vector< [LowLvlTimingData](#) > [lowLvl](#)
- [HighLvlTimingData](#) [highLvl](#)
- [OtherTimingData](#) [other](#)

## 7.161.1 Constructor & Destructor Documentation

### 7.161.1.1 TimingData()

```
TimingData::TimingData (
    long _m = -1 )    [inline], [explicit]
```

## 7.161.2 Member Data Documentation

### 7.161.2.1 highLvl

```
HighLvlTimingData TimingData::highLvl
```

### 7.161.2.2 lowLvl

```
vector<LowLvlTimingData> TimingData::lowLvl
```

### 7.161.2.3 m

```
long TimingData::m
```

### 7.161.2.4 nSlots

```
long TimingData::nSlots
```



### 7.161.2.5 other

```
OtherTimingData TimingData::other
```

### 7.161.2.6 p

```
long TimingData::p
```

### 7.161.2.7 phim

```
long TimingData::phim
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[Test\\_Timing.cpp](#)

## 7.162 helib::TreeNode< T > Class Template Reference

A node in a full binary tree.

```
#include <permutations.h>
```

### Public Member Functions

- [TreeNode](#) ()
- [TreeNode](#) (const T &d)
- T & [getData](#) ()
- const T & [getData](#) () const
- long [getParent](#) () const
- long [getLeftChild](#) () const
- long [getRightChild](#) () const
- long [getPrev](#) () const
- long [getNext](#) () const

### Friends

- class [FullBinaryTree](#)< T >

### 7.162.1 Detailed Description

```
template<typename T>  
class helib::TreeNode< T >
```

A node in a full binary tree.

These nodes are in a `std::vector`, so we use indexes rather than pointers

### 7.162.2 Constructor & Destructor Documentation

#### 7.162.2.1 `TreeNode()` [1/2]

```
template<typename T >  
helib::TreeNode< T >::TreeNode ( ) [inline]
```

#### 7.162.2.2 `TreeNode()` [2/2]

```
template<typename T >  
helib::TreeNode< T >::TreeNode (   
    const T & d ) [inline], [explicit]
```

### 7.162.3 Member Function Documentation

#### 7.162.3.1 `getData()` [1/2]

```
template<typename T >  
T& helib::TreeNode< T >::getData ( ) [inline]
```

#### 7.162.3.2 `getData()` [2/2]

```
template<typename T >  
const T& helib::TreeNode< T >::getData ( ) const [inline]
```

### 7.162.3.3 getLeftChild()

```
template<typename T >
long helib::TreeNode< T >::getLeftChild ( ) const [inline]
```

### 7.162.3.4 getNext()

```
template<typename T >
long helib::TreeNode< T >::getNext ( ) const [inline]
```

### 7.162.3.5 getParent()

```
template<typename T >
long helib::TreeNode< T >::getParent ( ) const [inline]
```

### 7.162.3.6 getPrev()

```
template<typename T >
long helib::TreeNode< T >::getPrev ( ) const [inline]
```

### 7.162.3.7 getRightChild()

```
template<typename T >
long helib::TreeNode< T >::getRightChild ( ) const [inline]
```

## 7.162.4 Friends And Related Function Documentation

### 7.162.4.1 FullBinaryTree< T >

```
template<typename T >
friend class FullBinaryTree< T > [friend]
```

The documentation for this class was generated from the following file:

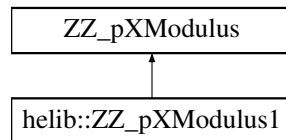
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[permutations.h](#)

## 7.163 helib::ZZ\_pXModulus1 Class Reference

placeholder for pXModulus ...no optimizations

```
#include <NumbTh.h>
```

Inheritance diagram for helib::ZZ\_pXModulus1:



### Public Member Functions

- [ZZ\\_pXModulus1](#) ([UNUSED](#) long \_m, const NTL::ZZ\_pX &\_f)
- const NTL::ZZ\_pXModulus & [upcast](#) () const

### 7.163.1 Detailed Description

placeholder for pXModulus ...no optimizations

### 7.163.2 Constructor & Destructor Documentation

#### 7.163.2.1 ZZ\_pXModulus1()

```
helib::ZZ_pXModulus1::ZZ_pXModulus1 (
    UNUSED long _m,
    const NTL::ZZ_pX & _f ) [inline]
```

### 7.163.3 Member Function Documentation

#### 7.163.3.1 upcast()

```
const NTL::ZZ_pXModulus& helib::ZZ_pXModulus1::upcast ( ) const [inline]
```

The documentation for this class was generated from the following file:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[NumbTh.h](#)

## 7.164 helib::zz\_pXModulus1 Class Reference

Auxiliary classes to facilitate faster reduction mod  $\Phi_m(X)$  when the input has degree less than  $m$ .

```
#include <NumbTh.h>
```

### Public Member Functions

- [zz\\_pXModulus1](#) (long  $_m$ , const NTL::zz\_pX &  $_f$ )
- const NTL::zz\_pXModulus & [upcast](#) () const

### Public Attributes

- long  $m$
- NTL::zz\_pX  $f$
- long  $n$
- bool [specialLogic](#)
- long  $k$
- long  $k1$
- NTL::fftRep [R0](#)
- NTL::fftRep [R1](#)
- NTL::zz\_pXModulus  $fm$

### 7.164.1 Detailed Description

Auxiliary classes to facilitate faster reduction mod  $\Phi_m(X)$  when the input has degree less than  $m$ .

### 7.164.2 Constructor & Destructor Documentation

#### 7.164.2.1 zz\_pXModulus1()

```
helib::zz_pXModulus1::zz_pXModulus1 (
    long  $_m$ ,
    const NTL::zz_pX &  $_f$  )
```

### 7.164.3 Member Function Documentation

#### 7.164.3.1 upcast()

```
const NTL::zz_pXModulus& helib::zz_pXModulus1::upcast ( ) const [inline]
```

## 7.164.4 Member Data Documentation

### 7.164.4.1 f

NTL::zz\_pX helib::zz\_pXModulus1::f

### 7.164.4.2 fm

NTL::zz\_pXModulus helib::zz\_pXModulus1::fm

### 7.164.4.3 k

long helib::zz\_pXModulus1::k

### 7.164.4.4 k1

long helib::zz\_pXModulus1::k1

### 7.164.4.5 m

long helib::zz\_pXModulus1::m

### 7.164.4.6 n

long helib::zz\_pXModulus1::n

### 7.164.4.7 R0

NTL::fftRep helib::zz\_pXModulus1::R0

### 7.164.4.8 R1

NTL::fftRep helib::zz\_pXModulus1::R1

### 7.164.4.9 specialLogic

bool helib::zz\_pXModulus1::specialLogic

The documentation for this class was generated from the following files:

- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/[NumbTh.h](#)
- /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/[NumbTh.cpp](#)

## Chapter 8

# File Documentation

### 8.1 mainpage.dox File Reference

### 8.2 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/[↩](#) HElib/include/helib/apiAttributes.h File Reference

#### Namespaces

- [helib](#)

#### Macros

- `#define` [UNUSED](#)

#### 8.2.1 Macro Definition Documentation

##### 8.2.1.1 UNUSED

```
#define UNUSED
```

### 8.3 [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/↵](#) HElib/include/helib/ArgMap.h File Reference

Easier arg parsing.

```
#include <iostream>
#include <iomanip>
#include <forward_list>
#include <initializer_list>
#include <set>
#include <unordered_map>
#include <algorithm>
#include <functional>
#include <string>
#include <sstream>
#include <fstream>
#include <cctype>
#include <memory>
#include <vector>
#include <tuple>
#include <type_traits>
#include <regex>
#include <helib/assertions.h>
```

#### Classes

- class [helib::ArgMap](#)  
*Basic class for arg parsing. Example use:*
- struct [helib::ArgMap::ArgProcessor](#)

#### Namespaces

- [helib](#)

#### 8.3.1 Detailed Description

Easier arg parsing.

### 8.4 [/Users/Projects/FHE\\_HElib\\_Repos\\_Public\\_staging/homenc\\_v1.0.2/↵](#) HElib/include/helib/assertions.h File Reference

Various assertion functions for use within HElib. These are meant as a replacement for C-style asserts, which (undesirably) exit the process without giving the opportunity to clean up and shut down gracefully. Instead, these functions will throw an exception if their conditions are violated.

```
#include <helib/exceptions.h>
```



## Namespaces

- [helib](#)

## Functions

- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>  
void helib::assertTrue (const T &value, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>  
void helib::assertFalse (T value, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>  
void helib::assertEq (const T &a, const T &b, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>  
void helib::assertNeq (const T &a, const T &b, const std::string &message)`
- `template<typename ExceptionTy = ::helib::LogicError, typename T = void>  
void helib::assertNotNull (const T &p, const std::string &message)`
- `template<typename ExceptionTy = ::helib::OutOfRangeError, typename T = void>  
void helib::assertInRange (const T &elem, const T &min, const T &max, const std::string &message, bool  
right_inclusive=false)`

### 8.4.1 Detailed Description

Various assertion functions for use within HElib. These are meant as a replacement for C-style asserts, which (undesirably) exit the process without giving the opportunity to clean up and shut down gracefully. Instead, these functions will throw an exception if their conditions are violated.

General usage is of the form:

```
helib::assertEq(my_variable, 51, "my_variable must equal 51!");
```

Most of these functions will result in an [helib::LogicError](#) being thrown if their conditions are violated, except for [assertInRange](#), which will throw an [helib::OutOfRange](#). However, if one wishes to throw some other helib exception, one can specify it as a template argument as follows. For example:

```
helib::assertTrue<helib::InvalidArgument>(some_var > 0, "some_var must be  
positive!");
```

The full set of helib exceptions can be found in [exceptions.h](#).

## 8.5 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/ HElib/include/helib/binaryArith.h File Reference

Implementing integer addition, multiplication in binary representation.

```
#include <helib/EncryptedArray.h>  
#include <helib/CtPtrs.h>
```

## Namespaces

- [helib](#)

## Functions

- `std::vector< long > helib::longToBitVector` (long num, long bitSize)  
*Returns a number as a vector of bits with LSB on the left.*
- `void helib::binaryCond` (CtPtrs &output, const Ctxt &cond, const CtPtrs &>trueValue, const CtPtrs &>falseValue)  
*Implementation of  $output = cond * trueValue + (1 - cond) * falseValue$ .*
- `void helib::binaryMask` (CtPtrs &binaryNums, const Ctxt &mask)  
*Zeroes the slots of *binaryNums* where the corresponding slot of *mask* is 0.*
- `void helib::concatBinaryNums` (CtPtrs &output, const CtPtrs &a, const CtPtrs &b)  
*Concatenates two binary numbers into a single CtPtrs object. E.g. If  $a=10111$ ,  $b=00101$  then  $output = 1011100101$ .*
- `void helib::splitBinaryNums` (CtPtrs &leftSplit, CtPtrs &rightSplit, const CtPtrs &input)  
*Splits a single binary number into two binary numbers *leftSplit* and *rightSplit*.*
- `void helib::leftBitwiseShift` (CtPtrs &output, const CtPtrs &input, const long shamt)  
*Left shift input by *shamt*.*
- `void helib::bitwiseRotate` (CtPtrs &output, const CtPtrs &input, long rotamt)  
*Rotate input by *rotamt*.*
- `void helib::bitwiseXOR` (CtPtrs &output, const CtPtrs &lhs, const CtPtrs &rhs)  
*Compute a bitwise XOR between *lhs* and *rhs*.*
- `void helib::bitwiseOr` (CtPtrs &output, const CtPtrs &lhs, const CtPtrs &rhs)  
*Compute a bitwise OR between *lhs* and *rhs*.*
- `void helib::bitwiseAnd` (CtPtrs &output, const CtPtrs &lhs, const CtPtrs &rhs)  
*Compute a bitwise AND between *lhs* and *rhs*.*
- `void helib::bitwiseAnd` (CtPtrs &output, const CtPtrs &input, const std::vector< long > mask)  
*Compute a bitwise AND between input and a `std::vector<long>`.*
- `void helib::bitwiseNot` (CtPtrs &output, const CtPtrs &input)  
*Compute a bitwise NOT of input.*
- `void helib::addTwoNumbers` (CtPtrs &sum, const CtPtrs &lhs, const CtPtrs &rhs, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Adds two numbers in binary representation where each ciphertext of the input vector contains a bit.*
- `void helib::negateBinary` (CtPtrs &negation, const CtPtrs &input)  
*Negates a number in binary 2's complement representation.*
- `void helib::subtractBinary` (CtPtrs &difference, const CtPtrs &lhs, const CtPtrs &rhs, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Subtracts *rhs* from *lhs* where *lhs*, *rhs* are in 2's complement.*
- `long helib::fifteenOrLess4Four` (const CtPtrs &out, const CtPtrs &in, long sizeLimit=4)  
*Add together up to fifteen {0,1} integers, producing a 4-bit counter.*
- `void helib::addManyNumbers` (CtPtrs &sum, CtPtrMat &numbers, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Sum an arbitrary amount of numbers in binary representation.*
- `void helib::multTwoNumbers` (CtPtrs &product, const CtPtrs &lhs, const CtPtrs &rhs, bool rhsTwosComplement=false, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Multiply two numbers in binary representation where each ciphertext of the input vector contains a bit.*
- `void helib::decryptBinaryNums` (std::vector< long > &pNums, const CtPtrs &eNums, const SecKey &sKey, const EncryptedArray &ea, bool twosComplement=false, bool allSlots=true)  
*Decrypt the binary numbers that are encrypted in *eNums*.*
- `void helib::packedRecrypt` (const CtPtrs &a, const CtPtrs &b, std::vector< zzX > \*unpackSlotEncoding)  
*Function for packed recryption to recrypt multiple numbers.*

### 8.5.1 Detailed Description

Implementing integer addition, multiplication in binary representation.

## HElib/include/helib/binaryCompare.h File Reference

Implementing integer comparison in binary representation.

```
#include <helib/EncryptedArray.h>
#include <helib/CtPtrs.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::compareTwoNumbers](#) (CtPtrs &max, CtPtrs &min, Ctxt &mu, Ctxt &ni, const CtPtrs &a, const CtPtrs &b, bool twosComplement=false, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Compares two integers in binary  $a, b$ . Returns  $\max(a, b)$ ,  $\min(a, b)$  and indicator bits  $\mu=(a>b)$  and  $\text{ni}=(a<b)$*
- void [helib::compareTwoNumbers](#) (Ctxt &mu, Ctxt &ni, const CtPtrs &a, const CtPtrs &b, bool twosComplement=false, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Compares two integers in binary  $a, b$ . Returns only indicator bits  $\mu=(a>b)$  and  $\text{ni}=(a<b)$ .*

#### 8.6.1 Detailed Description

Implementing integer comparison in binary representation.

## 8.7 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/binio.h File Reference

```
#include <iostream>
#include <vector>
#include <type_traits>
#include <NTL/xdouble.h>
#include <NTL/vec_long.h>
```

### Namespaces

- [helib](#)

## Macros

- `#define BINIO_32BIT 4`
- `#define BINIO_48BIT 6`
- `#define BINIO_64BIT 8`
- `#define BINIO_EYE_SIZE 4`
- `#define BINIO_EYE_CONTEXTBASE_BEGIN "|BS|"`
- `#define BINIO_EYE_CONTEXTBASE_END "|BS|"`
- `#define BINIO_EYE_CONTEXT_BEGIN "|CN|"`
- `#define BINIO_EYE_CONTEXT_END "|CN|"`
- `#define BINIO_EYE_CTXT_BEGIN "|CX|"`
- `#define BINIO_EYE_CTXT_END "|CX|"`
- `#define BINIO_EYE_PK_BEGIN "|PK|"`
- `#define BINIO_EYE_PK_END "|PK|"`
- `#define BINIO_EYE_SK_BEGIN "|SK|"`
- `#define BINIO_EYE_SK_END "|SK|"`
- `#define BINIO_EYE_SKM_BEGIN "|KM|"`
- `#define BINIO_EYE_SKM_END "|KM|"`

## Functions

- `int helib::readEyeCatcher (std::istream &str, const char *expect)`
- `void helib::writeEyeCatcher (std::ostream &str, const char *eye)`
- `void helib::write_ntl_vec_long (std::ostream &str, const NTL::vec_long &vl, long intSize=BINIO_64BIT)`
- `void helib::read_ntl_vec_long (std::istream &str, NTL::vec_long &vl)`
- `long helib::read_raw_int (std::istream &str)`
- `int helib::read_raw_int32 (std::istream &str)`
- `void helib::write_raw_int (std::ostream &str, long num)`
- `void helib::write_raw_int32 (std::ostream &str, int num)`
- `void helib::write_raw_double (std::ostream &str, const double d)`
- `double helib::read_raw_double (std::istream &str)`
- `void helib::write_raw_xdouble (std::ostream &str, const NTL::xdouble xd)`
- `NTL::xdouble helib::read_raw_xdouble (std::istream &str)`
- `void helib::write_raw_ZZ (std::ostream &str, const NTL::ZZ &zz)`
- `void helib::read_raw_ZZ (std::istream &str, NTL::ZZ &zz)`
- `template<typename T >`  
`void helib::write_raw_vector (std::ostream &str, const std::vector< T > &v)`
- `template<> void helib::write_raw_vector< long > (std::ostream &str, const std::vector< long > &v)`
- `template<> void helib::write_raw_vector< double > (std::ostream &str, const std::vector< double > &v)`
- `template<typename T >`  
`void helib::read_raw_vector (std::istream &str, std::vector< T > &v, T &init)`
- `template<typename T >`  
`void helib::read_raw_vector (std::istream &str, std::vector< T > &v)`
- `template<> void helib::read_raw_vector< long > (std::istream &str, std::vector< long > &v)`
- `template<> void helib::read_raw_vector< double > (std::istream &str, std::vector< double > &v)`
- `template<typename T >`  
`void helib::read_raw_vector (std::istream &str, std::vector< T > &v, const Context &context)`

### 8.7.1 Macro Definition Documentation

#### 8.7.1.1 BINIO\_32BIT

```
#define BINIO_32BIT 4
```

#### 8.7.1.2 BINIO\_48BIT

```
#define BINIO_48BIT 6
```

#### 8.7.1.3 BINIO\_64BIT

```
#define BINIO_64BIT 8
```

#### 8.7.1.4 BINIO\_EYE\_CONTEXT\_BEGIN

```
#define BINIO_EYE_CONTEXT_BEGIN "|CN|"
```

#### 8.7.1.5 BINIO\_EYE\_CONTEXT\_END

```
#define BINIO_EYE_CONTEXT_END "|CN|"
```

#### 8.7.1.6 BINIO\_EYE\_CONTEXTBASE\_BEGIN

```
#define BINIO_EYE_CONTEXTBASE_BEGIN "|BS|"
```

#### 8.7.1.7 BINIO\_EYE\_CONTEXTBASE\_END

```
#define BINIO_EYE_CONTEXTBASE_END "|BS|"
```

#### 8.7.1.8 BINIO\_EYE\_CTXT\_BEGIN

```
#define BINIO_EYE_CTXT_BEGIN "|CX|"
```

#### 8.7.1.9 BINIO\_EYE\_CTXT\_END

```
#define BINIO_EYE_CTXT_END "]CX|"
```

#### 8.7.1.10 BINIO\_EYE\_PK\_BEGIN

```
#define BINIO_EYE_PK_BEGIN "|PK|"
```

#### 8.7.1.11 BINIO\_EYE\_PK\_END

```
#define BINIO_EYE_PK_END "]PK|"
```

#### 8.7.1.12 BINIO\_EYE\_SIZE

```
#define BINIO_EYE_SIZE 4
```

#### 8.7.1.13 BINIO\_EYE\_SK\_BEGIN

```
#define BINIO_EYE_SK_BEGIN "|SK|"
```

#### 8.7.1.14 BINIO\_EYE\_SK\_END

```
#define BINIO_EYE_SK_END "]SK|"
```

#### 8.7.1.15 BINIO\_EYE\_SKM\_BEGIN

```
#define BINIO_EYE_SKM_BEGIN "|KM|"
```

#### 8.7.1.16 BINIO\_EYE\_SKM\_END

```
#define BINIO_EYE_SKM_END "]KM|"
```

## 8.8 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/bluestein.h File Reference

declaration of BluesteinFFT(x, n, root, powers, powers\_aux, Rb):

```
#include <helib/NumbTh.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::BluesteinInit](#) (long n, const NTL::zz\_p &root, NTL::zz\_pX &powers, NTL::Vec< NTL::mulmod\_precon\_t > &powers\_aux, NTL::fftRep &Rb)  
*initialize bluestein*
- void [helib::BluesteinFFT](#) (NTL::zz\_pX &x, long n, const NTL::zz\_p &root, const NTL::zz\_pX &powers, const NTL::Vec< NTL::mulmod\_precon\_t > &powers\_aux, const NTL::fftRep &Rb)  
*apply bluestein*

#### 8.8.1 Detailed Description

declaration of BluesteinFFT(x, n, root, powers, powers\_aux, Rb):

Compute length-n FFT of the coefficient-vector of x (in place) If the degree of x is less than n then it treats the top coefficients as 0, if the degree of x is more than n then the extra coefficients are ignored. Similarly, if the top entries in x are zeros then x will have degree smaller than n. The argument root is a  $2n$ -th root of unity, namely  $\text{BluesteinFFT}(\dots, \text{root}, \dots) = \text{DFT}(\dots, \text{root}^2, \dots)$ .

The inverse-FFT is obtained just by calling  $\text{BluesteinFFT}(\dots, \text{root}^{-1})$ , but this procedure is *NOT SCALED*, so  $\text{BluesteinFFT}(x, n, \text{root}, \dots)$  and then  $\text{BluesteinFFT}(x, n, \text{root}^{-1}, \dots)$  will result in  $x = n * x_{\text{original}}$

The values powers, powers\_aux, and Rb must be precomputed by first calling  $\text{BluesteinInit}(n, \text{root}, \text{powers}, \text{powers\_aux}, \text{Rb})$ .

## 8.9 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/clonedPtr.h File Reference

Implementation of smart pointers with "deep cloning" semantics.

### Classes

- class [helib::deep\\_clone< X >](#)  
*Deep copy: initialize with clone.*
- class [helib::shallow\\_clone< X >](#)  
*Shallow copy: initialize with copy constructor.*

## Namespaces

- [helib](#)

## Macros

- #define [CLONED\\_PTR\\_TEMPLATE\\_MEMBERS](#)(CLONED\_PTR\_TYPE)
- #define [CLONED\\_PTR\\_DECLARE](#)(CLONED\_PTR\_TYPE, CLONED\_PTR\_INIT)

## Functions

- template<typename X , typename Cloner >  
void [helib::swap](#) (cloned\_ptr< X, Cloner > &x, cloned\_ptr< X, Cloner > &y)
- template<typename X , typename Cloner >  
void [helib::swap](#) (copied\_ptr< X, Cloner > &x, copied\_ptr< X, Cloner > &y)

### 8.9.1 Detailed Description

Implementation of smart pointers with "deep cloning" semantics.

Based (loosely) on code from

[http://github.com/yonat/smart\\_ptr/blob/master/cloned\\_ptr.h](http://github.com/yonat/smart_ptr/blob/master/cloned_ptr.h)

### 8.9.2 Macro Definition Documentation

#### 8.9.2.1 CLONED\_PTR\_DECLARE

```
#define CLONED_PTR_DECLARE(  
    CLONED_PTR_TYPE,  
    CLONED_PTR_INIT )
```

#### 8.9.2.2 CLONED\_PTR\_TEMPLATE\_MEMBERS

```
#define CLONED_PTR_TEMPLATE_MEMBERS(  
    CLONED_PTR_TYPE )
```

##### Value:

```
template <typename Y>  
CLONED_PTR_TYPE(const CLONED_PTR_TYPE<Y>& r)  
{  
    copy(r.ptr);  
}  
template <typename Y>  
CLONED_PTR_TYPE& operator=(const CLONED_PTR_TYPE<Y>& r)  
{  
    if (this != &r) {  
        delete ptr;  
        copy(r.ptr);  
    }  
    return *this;  
}
```



## 8.10 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/CModulus.h File Reference

Supports forward and backward length-m FFT transformations.

```
#include <helib/NumbTh.h>
#include <helib/PAlgebra.h>
#include <helib/bluestein.h>
#include <helib/clonedPtr.h>
```

### Classes

- class [helib::Cmodulus](#)

*Provides FFT and iFFT routines modulo a single-precision prime.*

### Namespaces

- [helib](#)

#### 8.10.1 Detailed Description

Supports forward and backward length-m FFT transformations.

This is a wrapper around the bluesteinFFT routines, for one modulus q.

## 8.11 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/Context.h File Reference

Keeps the parameters of an instance of the cryptosystem.

```
#include <helib/PAlgebra.h>
#include <helib/CModulus.h>
#include <helib/IndexSet.h>
#include <helib/recryption.h>
#include <helib/primeChain.h>
#include <helib/powerful.h>
#include <helib/apiAttributes.h>
#include <NTL/Lazy.h>
```

### Classes

- class [helib::Context](#)

*Maintaining the parameters.*

## Namespaces

- [helib](#)

## Functions

- long [helib::FindM](#) (long k, long nBits, long c, long p, long d, long s, long chosen\_m, bool [verbose](#)=false)  
*Returns smallest parameter m satisfying various constraints:*
- void [helib::writeContextBase](#) (std::ostream &s, const Context &context)  
*write [m p r gens ords] data*
- void [helib::readContextBase](#) (std::istream &s, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)  
*read [m p r gens ords] data, needed to construct context*
- std::unique\_ptr< Context > [helib::buildContextFromAscii](#) (std::istream &str)
- void [helib::writeContextBaseBinary](#) (std::ostream &str, const Context &context)  
*write [m p r gens ords] data*
- void [helib::writeContextBinary](#) (std::ostream &str, const Context &context)
- void [helib::readContextBaseBinary](#) (std::istream &s, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)  
*read [m p r gens ords] data, needed to construct context*
- std::unique\_ptr< Context > [helib::buildContextFromBinary](#) (std::istream &str)
- void [helib::readContextBinary](#) (std::istream &str, Context &context)
- void [helib::buildModChain](#) (Context &context, long nBits, long nDgts=3, bool willBeBootstrappable=false, long skHwt=0, long resolution=3, long bitsInSpecialPrimes=0)
- void [helib::endBuildModChain](#) (Context &context)

## Variables

- Context \* [helib::activeContext](#) = nullptr

### 8.11.1 Detailed Description

Keeps the parameters of an instance of the cryptosystem.

## 8.12 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/CtPtrs.h File Reference

Unified interface for vector of pointers to ciphertexts.

```
#include <initializer_list>
#include <helib/Ctxt.h>
#include <helib/PtrVector.h>
#include <helib/PtrMatrix.h>
```

## Namespaces

- [helib](#)

## Typedefs

- typedef PtrVector< Ctxt > [helib::CtPtrs](#)
- typedef PtrVector\_VecT< Ctxt > [helib::CtPtrs\\_VecCt](#)
- typedef PtrVector\_vectorT< Ctxt > [helib::CtPtrs\\_vectorCt](#)
- typedef PtrVector\_VecPt< Ctxt > [helib::CtPtrs\\_VecPt](#)
- typedef PtrVector\_vectorPt< Ctxt > [helib::CtPtrs\\_vectorPt](#)
- typedef PtrVector\_slice< Ctxt > [helib::CtPtrs\\_slice](#)
- typedef PtrMatrix< Ctxt > [helib::CtPtrMat](#)
- typedef PtrMatrix\_Vec< Ctxt > [helib::CtPtrMat\\_VecCt](#)
- typedef PtrMatrix\_vector< Ctxt > [helib::CtPtrMat\\_vectorCt](#)
- typedef PtrMatrix\_ptVec< Ctxt > [helib::CtPtrMat\\_ptVecCt](#)
- typedef PtrMatrix\_ptvector< Ctxt > [helib::CtPtrMat\\_ptvectorCt](#)

## Functions

- void [helib::packedDecrypt](#) (const CtPtrs &cPtrs, const std::vector< zzX > &unpackConsts, const Encrypted↔ Array &ea)
- void [helib::packedDecrypt](#) (const CtPtrs &array, const std::vector< zzX > &unpackConsts, const Encrypted↔ Array &ea, long belowLvl)
- void [helib::packedDecrypt](#) (const CtPtrMat &m, const std::vector< zzX > &unpackConsts, const Encrypted↔ Array &ea, long belowLvl=LONG\_MAX)
- long [helib::findMinBitCapacity](#) (const CtPtrs &v)
- long [helib::findMinBitCapacity](#) (const CtPtrMat &m)
- long [helib::findMinBitCapacity](#) (std::initializer\_list< const CtPtrs \* > list)
- void [helib::innerProduct](#) (Ctxt &result, const CtPtrs &v1, const CtPtrs &v2)
- Ctxt [helib::innerProduct](#) (const CtPtrs &v1, const CtPtrs &v2)

### 8.12.1 Detailed Description

Unified interface for vector of pointers to ciphertexts.

## 8.13 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/Ctxt.h File Reference

Declarations of a BGV-type ciphertext and key-switching matrices.

```
#include <cfloat>
#include <helib/DoubleCRT.h>
#include <helib/apiAttributes.h>
```

## Classes

- class [helib::Ptxt](#)  
An object that mimics the functionality of the [Ctxt](#) object, and acts as a convenient entry point for inputting/encoding data which is to be encrypted.
- class [helib::SKHandle](#)  
A handle, describing the secret-key element that "matches" a part, of the form  $s^{\wedge}r(X^{\wedge}t)$ .
- class [helib::CtxtPart](#)  
One entry in a ciphertext `std::vector`.
- class [helib::Ctxt](#)  
A [Ctxt](#) object holds a single ciphertext.

## Namespaces

- [helib](#)

## Functions

- `std::ostream & helib::operator<< (std::ostream &s, const SKHandle &handle)`
- `std::istream & helib::operator>> (std::istream &s, CtxtPart &p)`
- `std::ostream & helib::operator<< (std::ostream &s, const CtxtPart &p)`
- `void helib::totalProduct (Ctxt &out, const std::vector< Ctxt > &v)`
- `void helib::incrementalProduct (std::vector< Ctxt > &v)`
- `void helib::innerProduct (Ctxt &result, const std::vector< Ctxt > &v1, const std::vector< Ctxt > &v2)`
- `Ctxt helib::innerProduct (const std::vector< Ctxt > &v1, const std::vector< Ctxt > &v2)`
- `void helib::innerProduct (Ctxt &result, const std::vector< Ctxt > &v1, const std::vector< DoubleCRT > &v2)`  
Compute the inner product of a vectors of ciphertexts and a constant vector.
- `Ctxt helib::innerProduct (const std::vector< Ctxt > &v1, const std::vector< DoubleCRT > &v2)`
- `void helib::innerProduct (Ctxt &result, const std::vector< Ctxt > &v1, const std::vector< NTL::ZZX > &v2)`
- `Ctxt helib::innerProduct (const std::vector< Ctxt > &v1, const std::vector< NTL::ZZX > &v2)`
- `void helib::CheckCtxt (const Ctxt &c, const char *label)`  
print to cerr some info about ciphertext
- `void helib::extractDigits (std::vector< Ctxt > &digits, const Ctxt &c, long r=0)`  
Extract the mod-p digits of a mod-p^r ciphertext.
- `void helib::extractDigits (std::vector< Ctxt > &digits, const Ctxt &c, long r, bool shortCut)`
- `void helib::extendExtractDigits (std::vector< Ctxt > &digits, const Ctxt &c, long r, long e)`

### 8.13.1 Detailed Description

Declarations of a BGV-type ciphertext and key-switching matrices.

A ciphertext is a `std::vector` of "ciphertext parts", each part consists of a polynomial (element of polynomial ring  $R_Q$ ) and a "handle" describing the secret-key polynomial that this part multiplies during decryption. For example:

- A "canonical" ciphertext has two parts, the first part multiplies 1 and the second multiplies the "base" secret key  $s$ .
- When you multiply two canonical ciphertexts you get a 3-part ciphertext, with parts corresponding to 1,  $s$ , and  $s^{\wedge}2$ .
- When you apply automorphism  $X \rightarrow X^{\wedge}t$  to a generic ciphertext, then

- the part corresponding to 1 still remains wrt 1
- every other part corresponding to some  $s'$  will now be corresponding to the polynomial  $s'(X^t) \bmod \Phi_m(X)$

This type of representation lets you in principle add ciphertexts that are defined with respect to different keys:

- For parts of the two ciphertexts that point to the same secret-key polynomial, you just add the two Double-CRT polynomials
- Parts in one ciphertext that do not have counter-part in the other ciphertext will just be included in the result intact. For example, you have the ciphertexts  $C1 = (a \text{ relative to } 1, b \text{ relative to } s)$   $C2 = (u \text{ relative to } 1, v \text{ relative to } s(X^3))$  Then their sum will be  $C1+C2 = (a+u \text{ relative to } 1, b \text{ relative to } s, v \text{ relative to } s(X^3))$

Similarly, in principle you can also multiply arbitrary ciphertexts, even ones that are defined with respect to different keys, and the result will be defined with respect to the tensor product of the two keys.

The current implementation is more restrictive, however. It requires that a ciphertext has one part wrt 1, that for every  $r \geq 1$  there is at most one part wrt to  $s^r(X^t)$  (for some  $t$ ), and that the  $r$ 's are consecutive. For example you cannot have parts wrt  $(1, s, s^3)$  without having a part wrt  $s^2$ .

It follows that you can only add/multiply ciphertexts if one of the two lists of handles is a prefix of the other. For example, one can add a ciphertext wrt  $(1, s(X^2))$  to another wrt  $(1, s(X^2), s^2(X^2))$ , but not to another ciphertext wrt  $(1, s)$ .

## 8.14 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/debugging.h File Reference

debugging utilities

```
#include <iostream>
#include <string>
#include <NTL/ZZX.h>
#include <helib/NumbTh.h>
```

### Namespaces

- [helib](#)

### Macros

- `#define FLAG_PRINT_ZZX 1`
- `#define FLAG_PRINT_POLY 2`
- `#define FLAG_PRINT_VEC 4 /* decode to ZZx */`
- `#define FLAG_PRINT_DVEC 8 /* decode to double float */`
- `#define FLAG_PRINT_XVEC 16 /* decode to complex numbers */`

## Functions

- void [helib::setupDebugGlobals](#) (SecKey \*debug\_key, const std::shared\_ptr< const EncryptedArray > &debug\_ea, NTL::ZZX debug\_ptxt=NTL::ZZX{})  
*Setup function for setting up the global debug variables.*
- void [helib::cleanupDebugGlobals](#) ()  
*Cleanup function for clearing the global debug variables.*
- void [helib::decryptAndPrint](#) (std::ostream &s, const Ctxt &ctxt, const SecKey &sk, const EncryptedArray &ea, long flags=0)
- NTL::xdouble [helib::embeddingLargestCoeff](#) (const Ctxt &ctxt, const SecKey &sk)
- double [helib::realToEstimatedNoise](#) (const Ctxt &ctxt, const SecKey &sk)
- void [helib::checkNoise](#) (const Ctxt &ctxt, const SecKey &sk, const std::string &msg, double thresh=10.0)
- bool [helib::decryptAndCompare](#) (const Ctxt &ctxt, const SecKey &sk, const EncryptedArray &ea, const PlaintextArray &pa)
- void [helib::rawDecrypt](#) (NTL::ZZX &plaintext, const std::vector< NTL::ZZX > &zzParts, const DoubleCRT &s←Key, long q=0)
- template<typename VEC >  
std::ostream & [helib::printVec](#) (std::ostream &s, const VEC &v, long nCoeffs=40)
- std::ostream & [helib::printZZX](#) (std::ostream &s, const NTL::ZZX &poly, long nCoeffs=40)

## Variables

- SecKey \* [helib::dbgKey](#) = nullptr
- std::shared\_ptr< const EncryptedArray > [helib::dbgEa](#) = nullptr
- NTL::ZZX [helib::dbg\\_ptxt](#)

### 8.14.1 Detailed Description

debugging utilities

### 8.14.2 Macro Definition Documentation

#### 8.14.2.1 FLAG\_PRINT\_DVEC

```
#define FLAG_PRINT_DVEC 8 /* decode to double float */
```

#### 8.14.2.2 FLAG\_PRINT\_POLY

```
#define FLAG_PRINT_POLY 2
```

### 8.14.2.3 FLAG\_PRINT\_VEC

```
#define FLAG_PRINT_VEC 4 /* decode to ZZX */
```

### 8.14.2.4 FLAG\_PRINT\_XVEC

```
#define FLAG_PRINT_XVEC 16 /* decode to complex numbers */
```

### 8.14.2.5 FLAG\_PRINT\_ZZX

```
#define FLAG_PRINT_ZZX 1
```

## 8.15 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/DoubleCRT.h File Reference

Integer polynomials (elements in the ring  $R_Q$ ) in double-CRT form.

```
#include <helib/zzX.h>
#include <helib/NumbTh.h>
#include <helib/IndexMap.h>
#include <helib/timing.h>
```

## Classes

- class [helib::DoubleCRTHelper](#)  
*A helper class to enforce consistency within an [DoubleCRTHelper](#) object.*
- class [helib::DoubleCRT](#)  
*Implementing polynomials (elements in the ring  $R_Q$ ) in double-CRT form.*

## Namespaces

- [helib](#)

## Typedefs

- typedef std::shared\_ptr< DoubleCRT > [helib::DCRTptr](#)
- typedef std::shared\_ptr< NTL::ZZX > [helib::ZZXptr](#)

## Functions

- void [helib::conv](#) (DoubleCRT &d, const NTL::ZZX &p)
- void [helib::conv](#) (NTL::ZZX &p, const DoubleCRT &d)
- NTL::ZZX [helib::to\\_ZZX](#) (const DoubleCRT &d)

### 8.15.1 Detailed Description

Integer polynomials (elements in the ring  $R_Q$ ) in double-CRT form.

## 8.16 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/EncryptedArray.h File Reference

Data-movement operations on encrypted arrays of slots.

```
#include <exception>
#include <cmath>
#include <complex>
#include <NTL/Lazy.h>
#include <NTL/pair.h>
#include <NTL/SmartPtr.h>
#include <helib/DoubleCRT.h>
#include <helib/Context.h>
#include <helib/Ctxt.h>
#include <helib/keys.h>
```

## Classes

- class [helib::EncryptedArrayBase](#)  
*virtual class for data-movement operations on arrays of slots*
- class [helib::EncryptedArrayDerived< type >](#)  
*Derived concrete implementation of [EncryptedArrayBase](#).*
- class [helib::EncryptedArrayCx](#)  
*A different derived class to be used for the approximate-numbers scheme.*
- class [helib::EncryptedArray](#)  
*A simple wrapper for a smart pointer to an [EncryptedArrayBase](#). This is the interface that higher-level code should use.*
- class [helib::PlaintextArrayBase](#)
- class [helib::PlaintextArrayDerived< type >](#)
- class [helib::PlaintextArray](#)

## Namespaces

- [helib](#)



**Macros**

- #define [HELIB\\_MORE\\_UNWRAPARGS](#)(n) NTL\_SEPARATOR\_##n NTL\_REPEATER\_##n(NTL\_UNWRAPARGS(n))
- #define [PA\\_BOILER](#)
- #define [CPA\\_BOILER](#)

**Typedefs**

- typedef std::complex< double > [helib::cx\\_double](#)

**Functions**

- template<typename RX , typename RXModulus >  
void [helib::plaintextAutomorph](#) (RX &bb, const RX &a, long k, long m, const RXModulus &PhimX)
- template<typename RX , typename type >  
void [helib::plaintextAutomorph](#) (RX &b, const RX &a, long i, long j, const EncryptedArrayDerived< type > &ea)
- EncryptedArrayBase \* [helib::buildEncryptedArray](#) (const Context &context, const PAlgebraMod &alMod, const NTL::ZZX &G=NTL::ZZX::zero())  
*A "factory" for building EncryptedArrays.*
- std::ostream & [helib::operator<<](#) (std::ostream &s, const PlaintextArray &pa)
- void [helib::rotate](#) (const EncryptedArray &ea, PlaintextArray &pa, long k)
- void [helib::shift](#) (const EncryptedArray &ea, PlaintextArray &pa, long k)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, const std::vector< long > &array)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, const std::vector< NTL::ZZX > &array)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, long val)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, const NTL::ZZX &val)
- void [helib::random](#) (const EncryptedArray &ea, PlaintextArray &pa)
- void [helib::decode](#) (const EncryptedArray &ea, std::vector< long > &array, const PlaintextArray &pa)
- void [helib::decode](#) (const EncryptedArray &ea, std::vector< NTL::ZZX > &array, const PlaintextArray &pa)
- bool [helib::equals](#) (const EncryptedArray &ea, const PlaintextArray &pa, const PlaintextArray &other)
- bool [helib::equals](#) (const EncryptedArray &ea, const PlaintextArray &pa, const std::vector< long > &other)
- bool [helib::equals](#) (const EncryptedArray &ea, const PlaintextArray &pa, const std::vector< NTL::ZZX > &other)
- void [helib::add](#) (const EncryptedArray &ea, PlaintextArray &pa, const PlaintextArray &other)
- void [helib::sub](#) (const EncryptedArray &ea, PlaintextArray &pa, const PlaintextArray &other)
- void [helib::mul](#) (const EncryptedArray &ea, PlaintextArray &pa, const PlaintextArray &other)
- void [helib::negate](#) (const EncryptedArray &ea, PlaintextArray &pa)
- void [helib::frobeniusAutomorph](#) (const EncryptedArray &ea, PlaintextArray &pa, long j)
- void [helib::frobeniusAutomorph](#) (const EncryptedArray &ea, PlaintextArray &pa, const NTL::Vec< long > &vec)
- void [helib::applyPerm](#) (const EncryptedArray &ea, PlaintextArray &pa, const NTL::Vec< long > &pi)
- void [helib::power](#) (const EncryptedArray &ea, PlaintextArray &pa, long e)
- void [helib::runningSums](#) (const EncryptedArray &ea, Ctxt &ctxt)

*A ctxt that encrypts  $(x_1, \dots, x_n)$  is replaced by an encryption of  $(y_1, \dots, y_n)$ , where  $y_i = \sum_{j \leq i} x_j$ .*

**8.16.1 Detailed Description**

Data-movement operations on encrypted arrays of slots.

## 8.16.2 Macro Definition Documentation

### 8.16.2.1 CPA\_BOILER

```
#define CPA_BOILER
```

#### Value:

```
const PAlgebraModDerived<type>& tab = ea.getTab();
UNUSED const RX& G = ea.getG();
UNUSED long n = ea.size();
UNUSED long d = ea.getDegree();
const std::vector<RX>& data = pa.getData<type>();
RBak bak;
bak.save();
tab.restoreContext();
```

```

/
/
/
/
/
/
/

```

### 8.16.2.2 HELIB\_MORE\_UNWRAPARGS

```
#define HELIB_MORE_UNWRAPARGS(
    n ) NTL_SEPARATOR_##n NTL_REPEATER_##n (NTL_UNWRAPARG)
```

### 8.16.2.3 PA\_BOILER

```
#define PA_BOILER
```

#### Value:

```
const PAlgebraModDerived<type>& tab = ea.getTab();
UNUSED const RX& G = ea.getG();
UNUSED long n = ea.size();
UNUSED long d = ea.getDegree();
std::vector<RX>& data = pa.getData<type>();
RBak bak;
bak.save();
tab.restoreContext();
```

```

/
/
/
/
/
/
/

```

## 8.17 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/EvalMap.h File Reference

Implementing the recryption linear transformations.

```
#include <helib/EncryptedArray.h>
#include <helib/matmul.h>
```

## Classes

- class [helib::EvalMap](#)  
*Class that provides the functionality for the linear transforms used in bootstrapping. The constructor is invoked with three arguments:*
- class [helib::ThinEvalMap](#)  
*Class that provides the functionality for the linear transforms used in "thin" bootstrapping, where slots are assumed to contain constants. The interface is exactly the same as for [EvalMap](#), except that the constructor does not have a `normal_basis` parameter.*

## Namespaces

- [helib](#)

### 8.17.1 Detailed Description

Implementing the reryption linear transformations.

## 8.18 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/exceptions.h File Reference

Various HElib-specific exception types.

```
#include <exception>
#include <stdexcept>
#include <sstream>
```

## Classes

- class [helib::Exception](#)  
*Base class that other HElib exception classes inherit from.*
- class [helib::LogicError](#)  
*Inherits from [Exception](#) and `std::logic_error`.*
- class [helib::OutOfRangeError](#)  
*Inherits from [Exception](#) and `std::out_of_range`.*
- class [helib::RuntimeError](#)  
*Inherits from [Exception](#) and `std::runtime_error`.*
- class [helib::InvalidArgument](#)  
*Inherits from [Exception](#) and `std::invalid_argument`.*

## Namespaces

- [helib](#)

### 8.18.1 Detailed Description

Various HELib-specific exception types.

This is largely a mirror image of the standard library exceptions, with the added ancestor of `Exception`. This allows one to distinguish between general exceptions and those specifically thrown by HELib. For example:

```
try {
    // Some code including calls to HELib
}
catch(const Exception& err) {
    // HELib error handling
}
catch(const std::exception& err) {
    // Generic error handling
}
```

To make sure that this is a pattern that can be used, we should only throw exceptions derived from `Exception` wherever possible.

## 8.19 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HELib/include/helib/FHE.h File Reference

Deprecated entry point header for HELib (legacy alias of [helib.h](#))

```
#include <helib/helib.h>
```

### 8.19.1 Detailed Description

Deprecated entry point header for HELib (legacy alias of [helib.h](#))

## 8.20 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HELib/include/helib/fhe\_stats.h File Reference

```
#include <vector>
#include <iostream>
```

### Classes

- struct [helib::fhe\\_stats\\_record](#)

### Namespaces

- [helib](#)

## Macros

- #define [HELIB\\_STATS\\_UPDATE](#)(name, val)
- #define [HELIB\\_STATS\\_SAVE](#)(name, val)

## Functions

- void [helib::print\\_stats](#) (std::ostream &s)
- const std::vector< double > \* [helib::fetch\\_saved\\_values](#) (const char \*)

## Variables

- bool [helib::fhe\\_stats](#) = false

### 8.20.1 Macro Definition Documentation

#### 8.20.1.1 HELIB\_STATS\_SAVE

```
#define HELIB_STATS_SAVE(  
    name,  
    val )
```

##### Value:

```
do {  
    if (fhe_stats) {  
        static fhe_stats_record _local_stats_record(name);  
        _local_stats_record.save(val);  
    }  
} while (0)
```

```
//  
//  
//  
//
```

#### 8.20.1.2 HELIB\_STATS\_UPDATE

```
#define HELIB_STATS_UPDATE(  
    name,  
    val )
```

##### Value:

```
do {  
    if (fhe_stats) {  
        static fhe_stats_record _local_stats_record(name);  
        _local_stats_record.update(val);  
    }  
} while (0)
```

```
//  
//  
//  
//
```

## 8.21 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/helib.h File Reference

Entry point header for HElib.

```
#include <helib/DoubleCRT.h>
#include <helib/Context.h>
#include <helib/Ctxt.h>
#include <helib/keySwitching.h>
#include <helib/keys.h>
#include <helib/EncryptedArray.h>
#include <helib/Ptxt.h>
```

### 8.21.1 Detailed Description

Entry point header for HElib.

## 8.22 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/hypercube.h File Reference

Hypercubes and their slices.

```
#include <helib/NumbTh.h>
```

### Classes

- class [helib::CubeSignature](#)  
*Holds a vector of dimensions for a hypercube and some additional data.*
- class [helib::HyperCube< T >](#)  
*A multi-dimensional cube.*
- class [helib::ConstCubeSlice< T >](#)  
*A constant lower-dimension slice of a hypercube.*
- class [helib::CubeSlice< T >](#)  
*A lower-dimension slice of a hypercube.*

### Namespaces

- [helib](#)

## Functions

- `std::ostream & helib::operator<<` (`std::ostream &s`, `const CubeSignature &sig`)
- `template<typename T >`  
`void helib::getHyperColumn` (`NTL::Vec< T > &v`, `const ConstCubeSlice< T > &s`, `long pos`)
- `template<typename T >`  
`void helib::setHyperColumn` (`const NTL::Vec< T > &v`, `const CubeSlice< T > &s`, `long pos`)
- `template<typename T >`  
`void helib::setHyperColumn` (`const NTL::Vec< T > &v`, `const CubeSlice< T > &s`, `long pos`, `const T &val`)
- `template<typename T >`  
`void helib::print3D` (`const HyperCube< T > &c`)

### 8.22.1 Detailed Description

Hypercubes and their slices.

## 8.23 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/IndexMap.h File Reference

Implementation of a map indexed by a dynamic set of integers.

```
#include <helib/IndexSet.h>
#include <helib/clonedPtr.h>
```

## Classes

- class `helib::IndexMapInit< T >`  
*Initializing elements in an `IndexMap`.*
- class `helib::IndexMap< T >`  
*`IndexMap<T>` implements a generic map indexed by a dynamic index set.*

## Namespaces

- `helib`

## Functions

- `template<typename T >`  
`bool helib::operator==` (`const IndexMap< T > &map1`, `const IndexMap< T > &map2`)  
*Comparing maps, by comparing all the elements.*
- `template<typename T >`  
`bool helib::operator!=` (`const IndexMap< T > &map1`, `const IndexMap< T > &map2`)

### 8.23.1 Detailed Description

Implementation of a map indexed by a dynamic set of integers.

## 8.24 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↔ v1.0.2/HElib/include/helib/IndexSet.h File Reference

A dynamic set of integers.

```
#include <helib/NumbTh.h>
```

### Classes

- class [helib::IndexSet](#)  
*A dynamic set of non-negative integers.*
- class [helib::IndexSet::iterator](#)

### Namespaces

- [helib](#)

### Functions

- IndexSet [helib::operator|](#) (const IndexSet &s, const IndexSet &t)  
*union*
- IndexSet [helib::operator&](#) (const IndexSet &s, const IndexSet &t)  
*intersection*
- IndexSet [helib::operator^](#) (const IndexSet &s, const IndexSet &t)  
*exclusive-or*
- IndexSet [helib::operator/](#) (const IndexSet &s, const IndexSet &t)  
*set minus*
- std::ostream & [helib::operator<<](#) (std::ostream &str, const IndexSet &set)
- std::istream & [helib::operator>>](#) (std::istream &str, IndexSet &set)
- long [helib::card](#) (const IndexSet &s)  
*Functional cardinality.*
- bool [helib::empty](#) (const IndexSet &s)
- bool [helib::operator<=](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s1 subset or equal to s2.*
- bool [helib::operator<](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s1 strict subset of s2.*
- bool [helib::operator>=](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s2 subset or equal to s2.*
- bool [helib::operator>](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s2 strict subset of s1.*
- bool [helib::disjoint](#) (const IndexSet &s1, const IndexSet &s2)  
*Functional disjoint.*



### 8.24.1 Detailed Description

A dynamic set of integers.

## 8.25 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/intraSlot.h File Reference

Packing/unpacking of mod-p integers in  $GF(p^d)$  slots.

```
#include <NTL/BasicThreadPool.h>
#include <helib/Context.h>
#include <helib/EncryptedArray.h>
#include <helib/Ctxt.h>
#include <helib/CtPtrs.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::buildUnpackSlotEncoding](#) (std::vector< zzX > &unpackSlotEncoding, const EncryptedArray &ea)
- void [helib::unpack](#) (const CtPtrs &unpacked, const Ctxt &packed, const EncryptedArray &ea, const std::vector< zzX > &unpackSlotEncoding)
- long [helib::unpack](#) (const CtPtrs &unpacked, const CtPtrs &packed, const EncryptedArray &ea, const std::vector< zzX > &unpackSlotEncoding)
- void [helib::repack](#) (Ctxt &packed, const CtPtrs &unpacked, const EncryptedArray &ea)
- long [helib::repack](#) (const CtPtrs &packed, const CtPtrs &unpacked, const EncryptedArray &ea)
- void [helib::unpackSlots](#) (std::vector< unsigned long > &value, PlaintextArray &pa, const EncryptedArray &ea)
- void [helib::unpackSlots](#) (std::vector< unsigned long > &value, NTL::ZZX &pa, const EncryptedArray &ea)
- void [helib::packConstant](#) (zzX &result, unsigned long data, long nbits, const EncryptedArray &ea)
- void [helib::packConstants](#) (zzX &result, const std::vector< unsigned long > &data, long nbits, const EncryptedArray &ea)

### 8.25.1 Detailed Description

Packing/unpacking of mod-p integers in  $GF(p^d)$  slots.

## 8.26 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/keys.h File Reference

- Declaration of public key

```
#include <helib/keySwitching.h>
```

## Classes

- class [helib::PubKey](#)  
*The public key.*
- class [helib::SecKey](#)  
*The secret key.*

## Namespaces

- [helib](#)

## Macros

- #define [HELIB\\_KSS\\_UNKNOWN](#) (0)
- #define [HELIB\\_KSS\\_FULL](#) (1)
- #define [HELIB\\_KSS\\_BSGS](#) (2)
- #define [HELIB\\_KSS\\_MIN](#) (3)

## Functions

- void [helib::writePubKeyBinary](#) (std::ostream &str, const PubKey &pk)
- void [helib::readPubKeyBinary](#) (std::istream &str, PubKey &pk)
- void [helib::writeSecKeyBinary](#) (std::ostream &str, const SecKey &sk)
- void [helib::readSecKeyBinary](#) (std::istream &str, SecKey &sk)
- double [helib::RLWE](#) (DoubleCRT &c0, DoubleCRT &c1, const DoubleCRT &s, long p, NTL::ZZ \*prg↵  
Seed=nullptr)
- double [helib::RLWE1](#) (DoubleCRT &c0, const DoubleCRT &c1, const DoubleCRT &s, long p)  
*Same as RLWE, but assumes that c1 is already chosen by the caller.*

### 8.26.1 Detailed Description

- Declaration of public key
- Declaration of secret key

Copyright IBM Corporation 2019 All rights reserved.

### 8.26.2 Macro Definition Documentation

#### 8.26.2.1 HELIB\_KSS\_BSGS

```
#define HELIB_KSS_BSGS (2)
```

### 8.26.2.2 HELIB\_KSS\_FULL

```
#define HELIB_KSS_FULL (1)
```

### 8.26.2.3 HELIB\_KSS\_MIN

```
#define HELIB_KSS_MIN (3)
```

### 8.26.2.4 HELIB\_KSS\_UNKNOWN

```
#define HELIB_KSS_UNKNOWN (0)
```

## 8.27 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/keySwitching.h File Reference

- Declaration of key switching matrices

```
#include <climits>
#include <helib/DoubleCRT.h>
#include <helib/Context.h>
#include <helib/Ctxt.h>
```

### Classes

- class [helib::KeySwitch](#)  
*Key-switching matrices.*

### Namespaces

- [helib](#)

### Functions

- `std::ostream & helib::operator<< (std::ostream &str, const KeySwitch &matrix)`

## Strategies for generating key-switching matrices

These functions are implemented in KeySwitching.cpp

- `#define HELIB_KEYSWITCH_THRESH (50)`  
*Constant defining threshold above which a baby-set/giant-step strategy is used.*
- `#define HELIB_KEYSWITCH_MIN_THRESH (8)`  
*Constant defining threshold above which a single giant step matrix is added even in HELIB\_KSS\_MIN mode. This helps in the matmul routines.*
- `long helib::KSGiantStepSize (long D)`  
*Function that returns number of baby steps. Used to keep this and matmul routines "in sync".*
- `void helib::addAllMatrices (SecKey &sKey, long keyID=0)`  
*Maximalistic approach: generate matrices  $s(X^e) \rightarrow s(X)$  for all  $e$  in  $Zm^*$ .*
- `void helib::addFewMatrices (SecKey &sKey, long keyID=0)`  
*Generate matrices so every  $s(X^e)$  can be reLinearized in at most two steps.*
- `void helib::addSome1DMatrices (SecKey &sKey, long bound=HELIB_KEYSWITCH_THRESH, long keyID=0)`  
*Generate some matrices of the form  $s(X^{g^i}) \rightarrow s(X)$ , but not all. For a generator  $g$  whose order is larger than bound, generate only enough matrices for the giant-step/baby-step procedures ( $2 \cdot \sqrt{\text{ord}(g)}$ ) of them.*
- `void helib::add1DMatrices (SecKey &sKey, long keyID=0)`  
*Generate all matrices  $s(X^{g^i}) \rightarrow s(X)$  for generators  $g$  of  $Zm^* / (p)$  and  $i < \text{ord}(g)$ . If  $g$  has different orders in  $Zm^*$  and  $Zm^* / (p)$  then generate also matrices of the form  $s(X^{g^{-i}}) \rightarrow s(X)$*
- `void helib::addBSGS1DMatrices (SecKey &sKey, long keyID=0)`
- `void helib::addSomeFrbMatrices (SecKey &sKey, long bound=HELIB_KEYSWITCH_THRESH, long keyID=0)`  
*Generate all/some Frobenius matrices of the form  $s(X^{p^i}) \rightarrow s(X)$*
- `void helib::addFrbMatrices (SecKey &sKey, long keyID=0)`
- `void helib::addBSGSFrbMatrices (SecKey &sKey, long keyID=0)`
- `void helib::addMinimal1DMatrices (SecKey &sKey, long keyID=0)`  
*These routines just add a single matrix (or two, for bad dimensions)*
- `void helib::addMinimalFrbMatrices (SecKey &sKey, long keyID=0)`
- `void helib::addMatrices4Network (SecKey &sKey, const PermNetwork &net, long keyID=0)`
- `void helib::addTheseMatrices (SecKey &sKey, const std::set< long > &automVals, long keyID=0)`  
*Generate specific key-switching matrices, described by the given set.*

### 8.27.1 Detailed Description

- Declaration of key switching matrices
- Other key switching-related free functions

Copyright IBM Corporation 2019 All rights reserved.

### 8.27.2 Macro Definition Documentation

### 8.27.2.1 HELIB\_KEYSWITCH\_MIN\_THRESH

```
#define HELIB_KEYSWITCH_MIN_THRESH (8)
```

Constant defining threshold above which a single giant step matrix is added even in HELIB\_KSS\_MIN mode. This helps in the matmul routines.

### 8.27.2.2 HELIB\_KEYSWITCH\_THRESH

```
#define HELIB_KEYSWITCH_THRESH (50)
```

Constant defining threshold above which a baby-set/giant-step strategy is used.

## 8.28 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/matching.h File Reference

Classes and functions for max-flow in a generic graph.

```
#include <helib/NumbTh.h>
```

### Classes

- class [helib::FlowEdge](#)  
*An edge in a flow graph.*
- class [helib::LabeledEdge](#)  
*A generic directed edge in a graph with some labels.*
- class [helib::LabeledVertex](#)  
*A generic node in a graph with some labels.*
- class [helib::BipartiteGraph](#)  
*A bipartite flow graph.*

### Namespaces

- [helib](#)

### Typedefs

- typedef std::unordered\_map< long, FlowEdge > [helib::FNeighborList](#)
- typedef std::vector< FNeighborList > [helib::FlowGraph](#)
- typedef std::unordered\_multimap< long, LabeledEdge > [helib::LNeighborList](#)

## Functions

- long [helib::maximum\\_flow](#) (FlowGraph &fg, long src, long sink)

### 8.28.1 Detailed Description

Classes and functions for max-flow in a generic graph.

## 8.29 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/matmul.h File Reference

some matrix / linear algenra stuff

```
#include <helib/EncryptedArray.h>
```

## Classes

- class [helib::MatMulFull](#)
- class [helib::MatMulFull\\_derived< type >](#)
- class [helib::BlockMatMulFull](#)
- class [helib::BlockMatMulFull\\_derived< type >](#)
- class [helib::MatMul1D](#)
- class [helib::MatMul1D\\_partial< type >](#)
- class [helib::MatMul1D\\_derived< type >](#)
- class [helib::BlockMatMul1D](#)
- class [helib::BlockMatMul1D\\_partial< type >](#)
- class [helib::BlockMatMul1D\\_derived< type >](#)
- struct [helib::ConstMultiplierCache](#)
- class [helib::MatMulExecBase](#)
- class [helib::MatMul1DExec](#)
- class [helib::BlockMatMul1DExec](#)
- class [helib::MatMulFullExec](#)
- class [helib::BlockMatMulFullExec](#)

## Namespaces

- [helib](#)

## Functions

- void [helib::traceMap](#) (Ctxt &ctxt)
- void [helib::mul](#) (PlaintextArray &pa, const MatMul1D &mat)
- void [helib::mul](#) (PlaintextArray &pa, const BlockMatMul1D &mat)
- void [helib::mul](#) (PlaintextArray &pa, const MatMulFull &mat)
- void [helib::mul](#) (PlaintextArray &pa, const BlockMatMulFull &mat)

## Variables

- int [helib::fhe\\_test\\_force\\_bsgs](#) = 0
- int [helib::fhe\\_test\\_force\\_hoist](#) = 0

### 8.29.1 Detailed Description

some matrix / linear algenra stuff

## 8.30 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/multicore.h File Reference

Support for multi-threaded implementations.

## Namespaces

- [helib](#)

## Macros

- #define [HELIB\\_atomic\\_long](#) long
- #define [HELIB\\_atomic\\_ulong](#) unsigned long
- #define [HELIB\\_MUTEX\\_TYPE](#) int
- #define [HELIB\\_MUTEX\\_GUARD](#)(mx) ((void)mx)

### 8.30.1 Detailed Description

Support for multi-threaded implementations.

### 8.30.2 Macro Definition Documentation

#### 8.30.2.1 HELIB\_atomic\_long

```
#define HELIB_atomic_long long
```

### 8.30.2.2 HELIB\_atomic\_ulong

```
#define HELIB_atomic_ulong unsigned long
```

### 8.30.2.3 HELIB\_MUTEX\_GUARD

```
#define HELIB_MUTEX_GUARD(  
    mx ) ((void)mx)
```

### 8.30.2.4 HELIB\_MUTEX\_TYPE

```
#define HELIB_MUTEX_TYPE int
```

## 8.31 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/norms.h File Reference

```
#include <vector>  
#include <complex>  
#include <NTL/ZZX.h>  
#include <NTL/xdouble.h>  
#include <helib/zzX.h>
```

## Namespaces

- [helib](#)

## Functions

- long [helib::sumOfCoeffs](#) (const zzX &f)
- NTL::ZZ [helib::sumOfCoeffs](#) (const NTL::ZZX &f)
- NTL::ZZ [helib::sumOfCoeffs](#) (const DoubleCRT &f)
- template<typename T >  
double [helib::largestCoeff](#) (const NTL::Vec< T > &f)  
*The L-infinity norm of an element (in coefficient representation)*
- template<typename T >  
double [helib::largestCoeff](#) (const std::vector< T > &f)
- NTL::ZZ [helib::largestCoeff](#) (const NTL::ZZX &f)
- NTL::ZZ [helib::largestCoeff](#) (const NTL::Vec< NTL::ZZ > &f)
- NTL::ZZ [helib::largestCoeff](#) (const DoubleCRT &f)
- double [helib::coeffsL2NormSquared](#) (const zzX &f)  
*The L2-norm of an element (in coefficient representation)*



- NTL::xdouble [helib::coeffsL2NormSquared](#) (const NTL::ZZX &f)
- NTL::xdouble [helib::coeffsL2NormSquared](#) (const DoubleCRT &f)
- double [helib::coeffsL2Norm](#) (const zzX &f)
- NTL::xdouble [helib::coeffsL2Norm](#) (const NTL::ZZX &f)
- NTL::xdouble [helib::coeffsL2Norm](#) (const DoubleCRT &f)
- double [helib::embeddingLargestCoeff](#) (const zzX &f, const PAlgebra &palg)
- double [helib::embeddingLargestCoeff](#) (const std::vector< double > &f, const PAlgebra &palg)
- void [helib::embeddingLargestCoeff\\_x2](#) (double &norm1, double &norm2, const std::vector< double > &f1, const std::vector< double > &f2, const PAlgebra &palg)
- NTL::xdouble [helib::embeddingLargestCoeff](#) (const NTL::ZZX &f, const PAlgebra &palg)
- void [helib::CKKS\\_canonicalEmbedding](#) (std::vector< cx\_double > &v, const zzX &f, const PAlgebra &palg)
- void [helib::CKKS\\_canonicalEmbedding](#) (std::vector< cx\_double > &v, const NTL::ZZX &f, const PAlgebra &palg)
- void [helib::CKKS\\_canonicalEmbedding](#) (std::vector< cx\_double > &v, const std::vector< double > &f, const PAlgebra &palg)
- void [helib::CKKS\\_embedInSlots](#) (zzX &f, const std::vector< cx\_double > &v, const PAlgebra &palg, double scaling)

### 8.31.1 Detailed Description

- computing various norms of ring elements

## 8.32 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/NumbTh.h File Reference

Miscellaneous utility functions.

```
#include <vector>
#include <set>
#include <cmath>
#include <complex>
#include <string>
#include <climits>
#include <iostream>
#include <fstream>
#include <istream>
#include <sstream>
#include <ctime>
#include <memory>
#include <unordered_map>
#include <NTL/version.h>
#include <NTL/ZZ.h>
#include <NTL/ZZX.h>
#include <NTL/ZZ_p.h>
#include <NTL/ZZ_pX.h>
#include <NTL/xdouble.h>
#include <NTL/mat_GF2.h>
#include <NTL/mat_GF2E.h>
#include <NTL/GF2XFactoring.h>
#include <NTL/mat_lzz_p.h>
#include <NTL/mat_lzz_pE.h>
```

```
#include <NTL/lzz_pXFactoring.h>
#include <NTL/GF2EX.h>
#include <NTL/lzz_pEX.h>
#include <NTL/FFT.h>
#include <helib/range.h>
#include <helib/assertions.h>
#include <helib/apiAttributes.h>
```

## Classes

- class [helib::RandomState](#)  
*Facility for "restoring" the [NTL](#) PRG state.*
- class [helib::zz\\_pXModulus1](#)  
*Auxiliary classes to facilitate faster reduction mod  $\Phi_m(X)$  when the input has degree less than  $m$ .*
- class [helib::ZZ\\_pXModulus1](#)  
*placeholder for pXModulus ...no optimizations*

## Namespaces

- [helib](#)
- [helib::FHEglobals](#)

## Macros

- `#define ERFC\_INVERSE\_SIZE (long(sizeof(erfc_inverse) / sizeof(erfc_inverse[0])))`

## Typedefs

- typedef long [helib::LONG](#)

## Functions

- bool [helib::setDryRun](#) (bool toWhat=true)
- bool [helib::isDryRun](#) ()
- void [helib::setAutomorphVals](#) (std::set< long > \*aVals)
- bool [helib::isSetAutomorphVals](#) ()
- void [helib::recordAutomorphVal](#) (long k)
- void [helib::setAutomorphVals2](#) (std::set< long > \*aVals)
- bool [helib::isSetAutomorphVals2](#) ()
- void [helib::recordAutomorphVal2](#) (long k)
- long [helib::bitSetToLong](#) (long bits, long bitSize)  
*Considers *bits* as a vector of bits and returns the value it represents when interpreted as a  $n$ -bit 2's complement number, where  $n$  is given by *bitSize*.*
- long [helib::mcMod](#) (long a, long b)  
*Routines for computing mathematically correct mod and div.*
- long [helib::mcDiv](#) (long a, long b)
- long [helib::balRem](#) (long a, long q)
- double [helib::fsquare](#) (double x)

- Return the square of a number as a double.*

  - long [helib::multOrd](#) (long p, long m)
- Return multiplicative order of p modulo m, or 0 if  $\text{GCD}(p, m) \neq 1$ .*

  - void [helib::ppsolve](#) (NTL::vec\_zz\_pE &x, const NTL::mat\_zz\_pE &A, const NTL::vec\_zz\_pE &b, long p, long r)
- Prime power solver.*

  - void [helib::ppsolve](#) (NTL::vec\_GF2E &x, const NTL::mat\_GF2E &A, const NTL::vec\_GF2E &b, long p, long r)

*A version for GF2: must have  $p == 2$  and  $r == 1$ .*

  - void [helib::pplInvert](#) (NTL::mat\_zz\_p &X, const NTL::mat\_zz\_p &A, long p, long r)

*Compute the inverse mod  $p^\wedge r$  of an  $n \times n$  matrix.*

  - void [helib::pplInvert](#) (NTL::mat\_zz\_pE &X, const NTL::mat\_zz\_pE &A, long p, long r)
  - void [helib::pplInvert](#) (NTL::mat\_GF2 &X, const NTL::mat\_GF2 &A, [UNUSED](#) long p, [UNUSED](#) long r)
  - void [helib::pplInvert](#) (NTL::mat\_GF2E &X, const NTL::mat\_GF2E &A, [UNUSED](#) long p, [UNUSED](#) long r)
  - void [helib::buildLinPolyMatrix](#) (NTL::mat\_zz\_pE &M, long p)
  - void [helib::buildLinPolyMatrix](#) (NTL::mat\_GF2E &M, long p)
  - void [helib::buildLinPolyCoeffs](#) (NTL::vec\_zz\_pE &C, const NTL::vec\_zz\_pE &L, long p, long r)

*Combination of buildLinPolyMatrix and ppsolve.*

  - void [helib::buildLinPolyCoeffs](#) (NTL::vec\_GF2E &C, const NTL::vec\_GF2E &L, long p, long r)

*A version for GF2: must be called with  $p == 2$  and  $r == 1$ .*

  - void [helib::applyLinPoly](#) (NTL::zz\_pE &beta, const NTL::vec\_zz\_pE &C, const NTL::zz\_pE &alpha, long p)

*Apply a linearized polynomial with coefficient vector C.*

  - void [helib::applyLinPoly](#) (NTL::GF2E &beta, const NTL::vec\_GF2E &C, const NTL::GF2E &alpha, long p)

*A version for GF2: must be called with  $p == 2$  and  $r == 1$ .*

  - double [helib::log2](#) (const NTL::xdouble &x)

*Base-2 logarithm.*

  - void [helib::factorize](#) (std::vector< long > &factors, long N)

*Factoring by trial division, only works for  $N < 2^{\{60\}}$ , only the primes are recorded, not their multiplicity.*

  - void [helib::factorize](#) (std::vector< NTL::ZZ > &factors, const NTL::ZZ &N)
  - void [helib::factorize](#) (NTL::Vec< NTL::Pair< long, long > > &factors, long N)

*Factoring by trial division, only works for  $N < 2^{\{60\}}$  primes and multiplicities are recorded.*

  - void [helib::pp\\_factorize](#) (std::vector< long > &factors, long N)

*Prime-power factorization.*

  - void [helib::phiN](#) (long &phiN, std::vector< long > &facts, long N)

*Compute Phi(N) and also factorize N.*

  - void [helib::phiN](#) (NTL::ZZ &phiN, std::vector< NTL::ZZ > &facts, const NTL::ZZ &N)
  - long [helib::phi\\_N](#) (long N)

*Compute Phi(N).*

  - long [helib::findGenerators](#) (std::vector< long > &gens, std::vector< long > &ords, long m, long p, const std::vector< long > &candidates=std::vector< long >())
  - void [helib::FindPrimitiveRoot](#) (NTL::zz\_p &r, unsigned long e)

*Find e-th root of unity modulo the current modulus.*

  - void [helib::FindPrimitiveRoot](#) (NTL::ZZ\_p &r, unsigned long e)
  - long [helib::mobius](#) (long n)

*Compute mobius function (naive method as n is small).*

  - NTL::ZZX [helib::Cyclotomic](#) (long N)

*Compute cyclotomic polynomial.*

  - NTL::ZZX [helib::makeIrredPoly](#) (long p, long d)

*Return a degree-d irreducible polynomial mod p.*

  - long [helib::primroot](#) (long N, long phiN)

*Find a primitive root modulo N.*

  - long [helib::ord](#) (long N, long p)

*Compute the highest power of p that divides N.*

- bool [helib::is2power](#) (long m)
- NTL::ZZX [helib::RandPoly](#) (long n, const NTL::ZZ &p)
- void [helib::MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q, bool abs)
- void [helib::MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q)
- NTL::ZZX [helib::MulMod](#) (const NTL::ZZX &f, long a, long q, bool abs)
- NTL::ZZX [helib::MulMod](#) (const NTL::ZZX &f, long a, long q)
- void [helib::balanced\\_MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q)
- template<typename T1 , typename T2 >  
void [helib::convert](#) (T1 &x1, const T2 &x2)  
*A generic template that resolves to NTL's conv routine.*
- template<typename T1 , typename T2 >  
void [helib::convert](#) (std::vector< T1 > &v1, const std::vector< T2 > &v2)  
*generic vector conversion routines*
- template<typename T1 , typename T2 >  
void [helib::convert](#) (std::vector< T1 > &v1, const NTL::Vec< T2 > &v2)
- template<typename T1 , typename T2 >  
void [helib::convert](#) (NTL::Vec< T1 > &v1, const std::vector< T2 > &v2)
- template<typename T >  
void [helib::convert](#) (std::vector< T > &v1, const std::vector< T > &v2)  
*Trivial type conversion, useful for generic code.*
- template<typename T1 , typename T2 >  
T1 [helib::convert](#) (const T2 &v2)
- template<typename T >  
std::vector< T > [helib::vector\\_replicate](#) (const T &a, long n)
- template<typename T >  
std::vector< T > [helib::Vec\\_replicate](#) (const T &a, long n)
- long [helib::computeProd](#) (const NTL::Vec< long > &vec)  
*returns \prod\_d vec[d]*
- long [helib::computeProd](#) (const std::vector< long > &vec)
- void [helib::mul](#) (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, long b)
- void [helib::div](#) (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, long b)
- void [helib::add](#) (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, const std::vector< NTL::ZZX > &b)
- long [helib::is\\_in](#) (long x, int \*X, long sz)  
*Finds whether x is an element of the set X of size sz, Returns -1 if not and the location if true.*
- long [helib::CRTcoeff](#) (long p, long q, bool symmetric=false)  
*Returns a CRT coefficient:  $x = (0 \bmod p, 1 \bmod q)$ . If symmetric is set then  $x \in [-pq/2, pq/2)$ , else  $x \in [0, pq)$*
- template<class zzvec >  
bool [helib::intVecCRT](#) (NTL::vec\_ZZ &vp, const NTL::ZZ &p, const zzvec &vq, long q)  
*Incremental integer CRT for vectors.*
- template<typename T , bool maxFlag>  
long [helib::argminmax](#) (std::vector< T > &v)  
*Find the index of the (first) largest/smallest element.*
- template<typename T >  
long [helib::argmax](#) (std::vector< T > &v)
- template<typename T >  
long [helib::argmin](#) (std::vector< T > &v)
- long [helib::argmax](#) (std::vector< long > &v, bool(\*moreThan)(long, long))  
*A variant with a specialized comparison function (\*moreThan)(a,b) returns the comparison  $a > b$ .*
- bool [helib::closeToOne](#) (const NTL::xdouble &x, long p)
- std::pair< long, long > [helib::rationalApprox](#) (double x, long denomBound=0)
- std::pair< NTL::ZZ, NTL::ZZ > [helib::rationalApprox](#) (NTL::xdouble x, NTL::xdouble denomBound=NTL::xdouble(0.0))
- void [helib::seekPastChar](#) (std::istream &str, int cc)

*Advance the input stream beyond white spaces and a single instance of the char cc.*

- template<typename T >  
void [helib::reverse](#) (NTL::Vec< T > &v, long lo, long hi)

*Reverse a vector in place.*

- template<typename T >  
void [helib::rotate](#) (NTL::Vec< T > &v, long k)

*Rotate a vector in place using swaps.*

- template<typename T >  
long [helib::lsize](#) (const std::vector< T > &v)

*Size of STL vector as a long (rather than unsigned long)*

- template<typename T >  
void [helib::killVec](#) (std::vector< T > &vec)

*NTL/std compatibility.*

- template<typename T >  
void [helib::killVec](#) (NTL::Vec< T > &vec)

- template<typename T >  
void [helib::setLengthZero](#) (std::vector< T > &vec)

- template<typename T >  
void [helib::setLengthZero](#) (NTL::Vec< T > &vec)

- template<typename T >  
long [helib::lsize](#) (const NTL::Vec< T > &v)

- template<typename T >  
void [helib::resize](#) (NTL::Vec< T > &v, long sz, const T &val)

- template<typename T >  
void [helib::resize](#) (std::vector< T > &v, long sz, const T &val)

- template<typename T >  
void [helib::resize](#) (NTL::Vec< T > &v, long sz)

- template<typename T >  
void [helib::resize](#) (std::vector< T > &v, long sz)

- template<typename T1 , typename T2 >  
bool [helib::sameObject](#) (const T1 \*p1, const T2 \*p2)

*Testing if two vectors point to the same object.*

- void [helib::ModComp](#) (NTL::ZZX &res, const NTL::ZZX &g, const NTL::ZZX &h, const NTL::ZZX &f)

*Modular composition of polynomials:  $res = g(h) \bmod f$ .*

- long [helib::polyEvalMod](#) (const NTL::ZZX &poly, long x, long p)

*Evaluates a modular integer polynomial, returns  $poly(x) \bmod p$ .*

- void [helib::interpolateMod](#) (NTL::ZZX &poly, const NTL::vec\_long &x, const NTL::vec\_long &y, long p, long e=1)

*Interpolate polynomial such that  $poly(x[i] \bmod p) = y[i] \bmod p^e$ . It is assumed that the points  $x[i]$  are all distinct modulo  $p$ .*

- long [helib::divc](#) (long a, long b)

*returns ceiling( $a/b$ ); assumes  $a \geq 0$ ,  $b > 0$ ,  $a+b \leq MAX\_LONG$*

- void [helib::rem](#) (NTL::zz\_pX &r, const NTL::zz\_pX &a, const zz\_pXModulus1 &ff)

- template<typename T >  
std::ostream & [helib::operator<<](#) (std::ostream &s, std::vector< T > v)

- template<typename T >  
std::istream & [helib::operator>>](#) (std::istream &s, std::vector< T > &v)

- template<typename T >  
std::string [helib::vecToStr](#) (const std::vector< T > &v)

- template<typename T >  
NTL::Vec< T > [helib::atoVec](#) (const char \*a)

- template<typename T >  
std::vector< T > [helib::atovector](#) (const char \*a)

- void [helib::TofftRep\\_trunc](#) (NTL::fftRep &y, const NTL::zz\_pX &x, long k, [UNUSED](#) long len, long lo, long hi)

- void `helib::TofftRep_trunc` (NTL::fftRep &y, const NTL::zz\_pX &x, long k, long len)
- template<typename T, typename P, typename... Args>  
void `helib::make_lazy` (const NTL::Lazy< T, P > &obj, Args &&... args)
- template<typename T, typename P, typename F, typename... Args>  
void `helib::make_lazy_with_fun` (const NTL::Lazy< T, P > &obj, F f, Args &&... args)
- void `helib::Warning` (const char \*msg)
- void `helib::Warning` (const std::string &msg)

## Variables

- const long double `helib::PI`
- bool `helib::FHEglobals::dryRun` = false  
*A dry-run flag The dry-run option disables most operations, to save time. This lets us quickly go over the evaluation of a circuit and estimate the resulting noise magnitude, without having to actually compute anything.*
- std::set< long > \* `helib::FHEglobals::automorphVals` = nullptr  
*A list of required automorphisms When non-nullptr, causes Ctxt::smartAutomorphism to just record the requested automorphism rather than actually performing it. This can be used to get a list of needed automorphisms for certain operations and then generate all these key-switching matrices. Should only be used in conjunction with dryRun=true.*
- std::set< long > \* `helib::FHEglobals::automorphVals2` = nullptr
- const double `helib::erfc_inverse` []
- void `helib::PolyRed` (NTL::ZZX &out, const NTL::ZZX &in, long q, bool abs=false)  
*Reduce all the coefficients of a polynomial modulo q.*
- void `helib::PolyRed` (NTL::ZZX &out, const NTL::ZZX &in, const NTL::ZZ &q, bool abs=false)
- void `helib::PolyRed` (NTL::ZZX &F, long q, bool abs=false)
- void `helib::PolyRed` (NTL::ZZX &F, const NTL::ZZ &q, bool abs=false)
- void `helib::vecRed` (NTL::Vec< NTL::ZZ > &out, const NTL::Vec< NTL::ZZ > &in, long q, bool abs)
- void `helib::vecRed` (NTL::Vec< NTL::ZZ > &out, const NTL::Vec< NTL::ZZ > &in, const NTL::ZZ &q, bool abs)

## Some enhanced conversion routines

- void `helib::convert` (long &x1, const NTL::GF2X &x2)
- void `helib::convert` (long &x1, const NTL::zz\_pX &x2)
- void `helib::convert` (NTL::vec\_zz\_pE &X, const std::vector< NTL::ZZX > &A)
- void `helib::convert` (NTL::mat\_zz\_pE &X, const std::vector< std::vector< NTL::ZZX >> &A)
- void `helib::convert` (std::vector< NTL::ZZX > &X, const NTL::vec\_zz\_pE &A)
- void `helib::convert` (std::vector< std::vector< NTL::ZZX >> &X, const NTL::mat\_zz\_pE &A)
- void `helib::convert` (NTL::Vec< long > &out, const NTL::ZZX &in)
- void `helib::convert` (NTL::Vec< long > &out, const NTL::zz\_pX &in, bool symmetric=true)
- void `helib::convert` (NTL::Vec< long > &out, const NTL::GF2X &in)
- void `helib::convert` (NTL::ZZX &out, const NTL::Vec< long > &in)
- void `helib::convert` (NTL::GF2X &out, const NTL::Vec< long > &in)

### 8.32.1 Detailed Description

Miscellaneous utility functions.

## 8.32.2 Macro Definition Documentation

### 8.32.2.1 ERFC\_INVERSE\_SIZE

```
#define ERFC_INVERSE_SIZE (long(sizeof(erfc_inverse) / sizeof(erfc_inverse[0])))
```

## 8.33 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/PAlgebra.h File Reference

Declarations of the classes PAlgebra.

```
#include <exception>
#include <utility>
#include <vector>
#include <complex>
#include <helib/NumbTh.h>
#include <helib/zzX.h>
#include <helib/hypercube.h>
#include <helib/PGFFT.h>
#include <helib/clonedPtr.h>
#include <helib/apiAttributes.h>
```

## Classes

- struct [helib::half\\_FFT](#)
- struct [helib::quarter\\_FFT](#)
- class [helib::PAlgebra](#)
  - The structure of  $(\mathbb{Z}/m\mathbb{Z})^* / (p)$*
- class [helib::PAlgebraModBase](#)
  - Virtual base class for [PAlgebraMod](#).*
- class [helib::PAlgebraModDerived< type >](#)
  - A concrete instantiation of the virtual class.*
- class [helib::MappingData< type >](#)
  - Auxiliary structure to support encoding/decoding slots.*
- class [helib::PAlgebraModDerived< type >](#)
  - A concrete instantiation of the virtual class.*
- class [helib::PAlgebraModCx](#)
- class [helib::PAlgebraMod](#)
  - The structure of  $\mathbb{Z}[X]/(\text{Phi}_m(X), p)$*

## Namespaces

- [helib](#)





## 8.34 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/permutations.h File Reference

Permutations over Hypercubes and their slices.

```
#include <helib/PAlgebra.h>
#include <helib/matching.h>
#include <helib/hypercube.h>
#include <helib/apiAttributes.h>
```

### Classes

- class [helib::ColPerm](#)  
*Permuting a single dimension (column) of a hypercube.*
- class [helib::GeneralBenesNetwork](#)  
*Implementation of generalized Benes Permutation Network.*
- class [helib::FullBinaryTree](#)  
*A simple implementation of full binary trees (each non-leaf has 2 children)*
- class [helib::TreeNode< T >](#)  
*A node in a full binary tree.*
- class [helib::FullBinaryTree](#)  
*A simple implementation of full binary trees (each non-leaf has 2 children)*
- class [helib::GenDescriptor](#)  
*A minimal description of a generator for the purpose of building tree.*
- class [helib::SubDimension](#)  
*A node in a tree relative to some generator.*
- class [helib::GeneratorTrees](#)  
*A std::vector of generator trees, one per generator in  $Z_{m*}(p)$*
- class [helib::PermNetLayer](#)  
*The information needed to apply one layer of a permutation network.*
- class [helib::PermNetwork](#)  
*A full permutation network.*

### Namespaces

- [helib](#)

### Typedefs

- typedef NTL::Vec< long > [helib::Permut](#)  
*A simple permutation is just a vector with  $p[i]=\pi_i$ .*
- typedef FullBinaryTree< SubDimension > [helib::OneGeneratorTree](#)

## Functions

- template<typename T >  
void [helib::applyPermToVec](#) (NTL::Vec< T > &out, const NTL::Vec< T > &in, const Permut &p1)  
*Apply a permutation to a std::vector, out[i]=in[p1[i]] (NOT in-place)*
- template<typename T >  
void [helib::applyPermToVec](#) (std::vector< T > &out, const std::vector< T > &in, const Permut &p1)
- template<typename T >  
void [helib::applyPermsToVec](#) (NTL::Vec< T > &out, const NTL::Vec< T > &in, const Permut &p2, const Permut &p1)  
*Apply two permutations to a std::vector out[i]=in[p2[p1[i]]] (NOT in-place)*
- template<typename T >  
void [helib::applyPermsToVec](#) (std::vector< T > &out, const std::vector< T > &in, const Permut &p2, const Permut &p1)
- void [helib::randomPerm](#) (Permut &perm, long n)  
*A random size-n permutation.*
- std::ostream & [helib::operator<<](#) (std::ostream &s, const ColPerm &p)
- void [helib::breakPermByDim](#) (std::vector< ColPerm > &out, const Permut &pi, const CubeSignature &sig)  
*Takes a permutation pi over m-dimensional cube  $C=Z_{\{n1\}} \times \dots \times Z_{\{nm\}}$  and expresses pi as a product  $pi = \rho_{i \leftarrow \{2m-1\}} \circ \dots \circ \rho_{i \leftarrow 2} \circ \rho_{i \leftarrow 1}$  where each  $\rho_{i \leftarrow j}$  is a column permutation along one dimension. Specifically for  $i < m$ , the permutations  $\rho_{i \leftarrow j}$  and  $\rho_{i \leftarrow \{2(m-1)-j\}}$  permute the i'th dimension.*

### 8.34.1 Detailed Description

Permutations over Hypercubes and their slices.

## 8.35 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/PGFFT.h File Reference

```
#include <vector>
#include <complex>
```

## Classes

- class [helib::PGFFT](#)
- class [helib::PGFFT::aligned\\_allocator< T >](#)

## Namespaces

- [helib](#)

## 8.36 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/polyEval.h File Reference

Homomorphic Polynomial Evaluation.

```
#include <helib/Context.h>
#include <helib/Ctxt.h>
```

### Classes

- class [helib::DynamicCtxtPowers](#)  
*Store powers of X, compute them dynamically as needed.*

### Namespaces

- [helib](#)

### Functions

- void [helib::polyEval](#) (Ctxt &ret, NTL::ZZX poly, const Ctxt &x, long k=0)  
*Evaluate a cleartext polynomial on an encrypted input.*
- void [helib::polyEval](#) (Ctxt &ret, const NTL::Vec< Ctxt > &poly, const Ctxt &x)  
*Evaluate an encrypted polynomial on an encrypted input.*

#### 8.36.1 Detailed Description

Homomorphic Polynomial Evaluation.

## 8.37 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/PolyMod.h File Reference

Underlying slot type structure of BGV ptxts.

```
#include <NTL/ZZX.h>
#include <vector>
#include <memory>
#include <helib/PolyModRing.h>
```

## Classes

- class [helib::PolyMod](#)

*An object that contains an  $NTL : \mathbb{Z}ZX$  polynomial along with a coefficient modulus  $p2r$  and a polynomial modulus  $G$ .*

## Namespaces

- [helib](#)

### 8.37.1 Detailed Description

Underlying slot type structure of BGV ptxts.

## 8.38 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/PolyModRing.h File Reference

Definition of the plaintext slot algebraic ring.

```
#include <NTL/ZZX.h>
#include <NTL/ZZ_p.h>
#include <vector>
```

## Classes

- struct [helib::PolyModRing](#)

*Lightweight type for describing the structure of a single slot of the plaintext space.*

## Namespaces

- [helib](#)

### 8.38.1 Detailed Description

Definition of the plaintext slot algebraic ring.

## 8.39 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/powerful.h File Reference

The "powerful basis" of a cyclotomic ring.

```
#include <helib/NumbTh.h>
#include <helib/clonedPtr.h>
#include <helib/bluestein.h>
#include <helib/hypercube.h>
#include <helib/DoubleCRT.h>
#include <helib/Context.h>
```

### Classes

- class [helib::PowerfulTranslationIndexes](#)  
*Holds index tables for translation between powerful and  $zz\_pX$ .*
- class [helib::PowerfulConversion](#)  
*Conversion between powerful representation in  $R_m/(q)$  and  $zz\_pX$ .*
- class [helib::PowerfulDCRT](#)  
*Conversion between powerful representation, [DoubleCRT](#), and  $ZZX$ .*

### Namespaces

- [helib](#)

#### 8.39.1 Detailed Description

The "powerful basis" of a cyclotomic ring.

## 8.40 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/primeChain.h File Reference

handling the chain of moduli

```
#include <vector>
#include <helib/IndexSet.h>
```

### Classes

- class [helib::ModuliSizes](#)  
*A helper class to map required modulo-sizes to  $primeSets$ .*

## Namespaces

- [helib](#)

## Functions

- `std::ostream & helib::operator<< (std::ostream &s, const ModuliSizes::Entry &e)`
- `std::istream & helib::operator>> (std::istream &s, ModuliSizes::Entry &e)`
- `void helib::write (std::ostream &s, const ModuliSizes::Entry &e)`
- `void helib::read (std::istream &s, ModuliSizes::Entry &e)`

### 8.40.1 Detailed Description

handling the chain of moduli

## 8.41 `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_↵ v1.0.2/HElib/include/helib/PtrMatrix.h` File Reference

Convenience class templates providing a unified interface for a matrix of objects, returning pointers to these objects.

```
#include <initializer_list>
#include <helib/PtrVector.h>
```

## Classes

- struct [helib::PtrMatrix< T >](#)  
*An abstract class for an array of PtrVectors.*
- struct [helib::PtrMatrix\\_Vec< T >](#)  
*An implementation of [PtrMatrix](#) using `Vec< Vec< T> >`*
- struct [helib::PtrMatrix\\_ptVec< T >](#)  
*An implementation of [PtrMatrix](#) using `Vec< Vec< T>* >`*
- struct [helib::PtrMatrix\\_vector< T >](#)  
*An implementation of [PtrMatrix](#) using `vector< vector< T> >`*
- struct [helib::PtrMatrix\\_ptvector< T >](#)  
*An implementation of [PtrMatrix](#) using `vector< vector< T>* >`*

## Namespaces

- [helib](#)

## Functions

- template<typename T >  
long [helib::lsize](#) (const PtrMatrix< T > &v)
- template<typename T >  
void [helib::resize](#) (PtrMatrix< T > &v, long newSize)
- template<typename T >  
void [helib::setLengthZero](#) (PtrMatrix< T > &v)
- template<typename T >  
const T \* [helib::ptr2nonNull](#) (std::initializer\_list< const PtrVector< T > \* > list)

### 8.41.1 Detailed Description

Convenience class templates providing a unified interface for a matrix of objects, returning pointers to these objects.

## 8.42 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/PtrVector.h File Reference

Convenience class templates providing a unified interface for a collection of objects, returning pointers to these objects.

```
#include <stdexcept>
#include <climits>
#include <vector>
#include <NTL/vector.h>
#include <helib/assertions.h>
#include <helib/apiAttributes.h>
```

## Classes

- struct [helib::PtrVector< T >](#)  
*Abstract class for an array of objects.*
- struct [helib::PtrVector\\_VecPt< T >](#)  
*An implementation of [PtrVector](#) using [Vec< T\\*>](#)*
- struct [helib::PtrVector\\_vectorPt< T >](#)  
*An implementation of [PtrVector](#) using [vector< T\\*>](#)*
- struct [helib::PtrVector\\_VecT< T >](#)  
*An implementation of [PtrVector](#) using [Vec< T>](#)*
- struct [helib::PtrVector\\_vectorT< T >](#)  
*An implementation of [PtrVector](#) using [vector< T>](#)*
- struct [helib::PtrVector\\_slice< T >](#)  
*An implementation of [PtrVector](#) as a slice of another [PtrVector](#).*
- struct [helib::PtrVector\\_Singleton< T >](#)  
*An implementation of [PtrVector](#) from a single [T](#) object.*

## Namespaces

- [helib](#)

## Functions

- `template<typename T >`  
`long helib::lsize (const PtrVector< T > &v)`
- `template<typename T >`  
`void helib::setLengthZero (PtrVector< T > &v)`
- `template<typename T >`  
`void helib::resize (PtrVector< T > &v, long newSize, const T &val)`
- `template<typename T >`  
`void helib::resize (PtrVector< T > &v, long newSize, const T *val)`
- `template<typename V1 , typename V2 >`  
`void helib::vecCopy (V1 &v1, const V2 &v2, long sizeLimit=0)`
- `template<typename V , typename T >`  
`void helib::vecCopy (V &v1, const PtrVector< T > &v2, long sizeLimit=0)`
- `template<typename V , typename T >`  
`void helib::vecCopy (PtrVector< T > &v1, const V &v2, long sizeLimit=0)`
- `template<typename T >`  
`void helib::vecCopy (PtrVector< T > &v1, const PtrVector< T > &v2, long sizeLimit=0)`

### 8.42.1 Detailed Description

Convenience class templates providing a unified interface for a collection of objects, returning pointers to these objects.

## 8.43 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/Ptxt.h File Reference

Plaintext object parameterised on CKKS and BGV schemes. Also contains definition of CKKS and BGV structs.

```
#include <type_traits>
#include <vector>
#include <algorithm>
#include <numeric>
#include <iomanip>
#include <helib/Context.h>
#include <helib/EncryptedArray.h>
#include <helib/assertions.h>
#include <helib/PolyMod.h>
```

## Classes

- struct [helib::CKKS](#)  
*Type for CKKS scheme, to be used as template parameter.*
- struct [helib::BGV](#)  
*Type for BGV scheme, to be used as template parameter.*
- class [helib::Ptxt](#)  
*An object that mimics the functionality of the Ctxt object, and acts as a convenient entry point for inputting/encoding data which is to be encrypted.*



**Namespaces**

- [helib](#)

**Functions**

- `template<typename From , typename Scheme >`  
`std::vector< typename Scheme::SlotType > helib::convertDataToSlotVector (const std::vector< From > &data, const Context &context)`  
*Converts `std::vector<From>` to `std::vector<Scheme::SlotType>`.*
- `template<typename Scheme >`  
`void helib::innerProduct (Ptxt< Scheme > &result, const std::vector< Ptxt< Scheme >> &first_vec, const std::vector< Ptxt< Scheme >> &second_vec)`  
*Free function that computes the inner product of two vectors of [Ptxt](#).*

**8.43.1 Detailed Description**

Plaintext object parameterised on CKKS and BGV schemes. Also contains definition of CKKS and BGV structs.

## 8.44 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/randomMatrices.h File Reference

implementation of random matrices of various forms, used for testing

```
#include <helib/matmul.h>
#include <NTL/BasicThreadPool.h>
```

**Classes**

- class [helib::RandomMatrix< type >](#)
- class [helib::RandomMultiMatrix< type >](#)
- class [helib::RandomBlockMatrix< type >](#)
- class [helib::RandomMultiBlockMatrix< type >](#)
- class [helib::RandomFullMatrix< type >](#)
- class [helib::RandomFullBlockMatrix< type >](#)

**Namespaces**

- [helib](#)

**Functions**

- `MatMul1D * helib::buildRandomMatrix (const EncryptedArray &ea, long dim)`
- `MatMul1D * helib::buildRandomMultiMatrix (const EncryptedArray &ea, long dim)`
- `BlockMatMul1D * helib::buildRandomBlockMatrix (const EncryptedArray &ea, long dim)`
- `BlockMatMul1D * helib::buildRandomMultiBlockMatrix (const EncryptedArray &ea, long dim)`
- `MatMulFull * helib::buildRandomFullMatrix (const EncryptedArray &ea)`
- `BlockMatMulFull * helib::buildRandomFullBlockMatrix (const EncryptedArray &ea)`

### 8.44.1 Detailed Description

implementation of random matrices of various forms, used for testing

## 8.45 `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_↵ v1.0.2/HElib/include/helib/range.h` File Reference

### Classes

- class [helib::general\\_range< T >](#)
- class [helib::general\\_range< T >::iterator](#)

### Namespaces

- [helib](#)

### Functions

- `general_range< long >` [helib::range](#) (long n)
- `general_range< long >` [helib::range](#) (long m, long n)

## 8.46 `/Users/Projects/FHE_HElib_Repos_Public_staging/homenc_↵ v1.0.2/HElib/include/helib/recryption.h` File Reference

Define some data structures to hold recryption data.

```
#include <helib/NumbTh.h>
```

### Classes

- class [helib::RecryptData](#)  
*A structure to hold recryption-related data inside the [Context](#).*
- class [helib::ThinRecryptData](#)  
*Same as above, but for "thin" bootstrapping, where the slots are assumed to contain constants.*

### Namespaces

- [helib](#)

## Macros

- #define [HELIB\\_MIN\\_CAP\\_FRAC](#) (2.0 / 3.0)

## Variables

- long [helib::thinRecrypt\\_initial\\_level](#)
- long [helib::fhe\\_force\\_chen\\_han](#) = 0
- long [helib::printFlag](#)

### 8.46.1 Detailed Description

Define some data structures to hold recryption data.

### 8.46.2 Macro Definition Documentation

#### 8.46.2.1 HELIB\_MIN\_CAP\_FRAC

```
#define HELIB_MIN_CAP_FRAC (2.0 / 3.0)
```

## 8.47 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/replicate.h File Reference

Procedures for replicating a ciphertext slot across a full ciphertext.

```
#include <helib/EncryptedArray.h>
#include <helib/Ptxt.h>
```

## Classes

- class [helib::ReplicateHandler](#)  
*An abstract class to handle call-backs to get the output of replicate.*

## Namespaces

- [helib](#)

## Functions

- void `helib::replicate` (const EncryptedArray &ea, Ctxt &ctx, long pos)  
*The value in slot #pos is replicated in all other slots. On an n-slot ciphertext, this algorithm performs  $O(\log n)$  1D rotations.*
- void `helib::replicate0` (const EncryptedArray &ea, Ctxt &ctx, long pos)  
*A lower-level routine. Same as replicate, but assumes all slots are zero except slot #pos.*
- void `helib::replicateAll` (const EncryptedArray &ea, const Ctxt &ctx, ReplicateHandler \*handler, long rec↵ Bound=64, RepAuxDim \*repAuxPtr=nullptr)
- void `helib::replicateAll` (std::vector< Ctxt > &v, const EncryptedArray &ea, const Ctxt &ctx, long rec↵ Bound=64, RepAuxDim \*repAuxPtr=nullptr)
- template<typename Scheme >  
void `helib::replicateAll` (std::vector< Ptxt< Scheme >> &v, const EncryptedArray &, const Ptxt< Scheme > &ptxt)  
*Generate a vector of plaintexts with each slot replicated in each plaintext.*
- void `helib::replicateAllOrig` (const EncryptedArray &ea, const Ctxt &ctx, ReplicateHandler \*handler, RepAux \*repAuxPtr=nullptr)
- void `helib::replicate` (const EncryptedArray &ea, PlaintextArray &pa, long i)
- template<typename Scheme >  
void `helib::replicate` (const EncryptedArray &, Ptxt< Scheme > &ptxt, long i)  
*Replicate single slot of a Ptxt object across all of its slots.*

## Variables

- NTL\_THREAD\_LOCAL bool `helib::replicateVerboseFlag` = false

### 8.47.1 Detailed Description

Procedures for replicating a ciphertext slot across a full ciphertext.

This module implements a recursive,  $O(1)$ -amortized algorithm for replications. On an input ciphertext that encrypts  $(x_1, \dots, x_n)$ , we generate the  $n$  encrypted std::vectors  $(x_1, \dots, x_1), \dots, (x_n, \dots, x_n)$ , in that order.

To process the output std::vectors, a "call back" mechanism is used (so that we do not need to generate them all, and instead can return them one by one). For this purpose, the caller should pass a pointer to a class derived from the purely abstract class ReplicateHandler.

The replication procedures are meant to be used for linear algebra operation where a matrix-std::vector multiplication can be implemented for example by replicating each entry of the std::vector as a stand-alone ciphertext, then use the SIMD operations on these ciphertexts.

## 8.48 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/include/helib/sample.h File Reference

```
#include <vector>
#include <NTL/xdouble.h>
#include <NTL/ZZX.h>
#include <NTL/ZZ_pX.h>
#include <helib/zzX.h>
```

## Namespaces

- [helib](#)

## Functions

- void [helib::sampleSmall](#) (zzX &poly, long n, double prob=0.5)
- void [helib::sampleSmall](#) (NTL::ZZX &poly, long n, double prob=0.5)
- void [helib::sampleHWt](#) (zzX &poly, long n, long Hwt=100)  
*Sample a degree-(n-1) poly as above, with only Hwt nonzero coefficients.*
- void [helib::sampleHWt](#) (NTL::ZZX &poly, long n, long Hwt=100)
- void [helib::sampleGaussian](#) (zzX &poly, long n, double stdev)  
*Sample polynomials with Gaussian coefficients.*
- void [helib::sampleGaussian](#) (NTL::ZZX &poly, long n, double stdev)
- void [helib::sampleUniform](#) (zzX &poly, long n, long B=100)  
*Sample a degree-(n-1) ZZX, with coefficients uniform in [-B,B].*
- void [helib::sampleUniform](#) (NTL::ZZX &poly, long n, const NTL::ZZ &B=NTL::ZZ(100L))
- void [helib::sampleGaussian](#) (std::vector< double > &dvec, long n, double stdev)  
*Choose a vector of continuous Gaussians.*
- double [helib::sampleHWt](#) (zzX &poly, const Context &context, long Hwt=100)
- double [helib::sampleHWtBounded](#) (zzX &poly, const Context &context, long Hwt=100)
- double [helib::sampleHWtBoundedEffectiveBound](#) (const Context &context, long Hwt=100)
- double [helib::sampleSmall](#) (zzX &poly, const Context &context)
- double [helib::sampleSmallBounded](#) (zzX &poly, const Context &context)
- double [helib::sampleGaussian](#) (zzX &poly, const Context &context, double stdev)
- double [helib::sampleGaussianBounded](#) (zzX &poly, const Context &context, double stdev)
- double [helib::sampleUniform](#) (zzX &poly, const Context &context, long B=100)
- NTL::xdouble [helib::sampleUniform](#) (NTL::ZZX &poly, const Context &context, const NTL::ZZ &B=NTL::ZZ(100L))
- void [helib::reduceModPhimX](#) (zzX &poly, const PALgebra &palg)
- const NTL::zz\_pXModulus & [helib::getPhimXMod](#) (const PALgebra &palg)
  
- double [helib::boundFreshNoise](#) (long m, long phim, double sigma, double epsilon=9e-13)
- double [helib::boundRoundingNoise](#) (long m, long phim, long p2r, double epsilon=9e-13)

### 8.48.1 Detailed Description

- implementing various sampling routines

## 8.49 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/tableLookup.h File Reference

Code for homomorphic table lookup and fixed-point functions.

```
#include <functional>
#include <helib/EncryptedArray.h>
#include <helib/CtPtrs.h>
```

## Namespaces

- [helib](#)

## Functions

- void [helib::computeAllProducts](#) (CtPtrs &products, const CtPtrs &array, std::vector< zzX > \*unpackSlotEncoding=nullptr)
- void [helib::tableLookup](#) (Ctxt &out, const std::vector< zzX > &table, const CtPtrs &idx, std::vector< zzX > \*unpackSlotEncoding=nullptr)
- void [helib::tableWriteIn](#) (const CtPtrs &table, const CtPtrs &idx, std::vector< zzX > \*unpackSlotEncoding=nullptr)
- void [helib::buildLookupTable](#) (std::vector< zzX > &T, std::function< double(double)> f, long nbits\_in, long scale\_in, long sign\_in, long nbits\_out, long scale\_out, long sign\_out, const EncryptedArray &ea)

*Built a table-lookup for a function in fixed-point representation.*

### 8.49.1 Detailed Description

Code for homomorphic table lookup and fixed-point functions.

## 8.50 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/timing.h File Reference

Utility functions for measuring time.

```
#include <helib/NumbTh.h>
#include <helib/multicore.h>
```

## Classes

- class [helib::FHEtimer](#)  
*A simple class to accumulate time.*

## Namespaces

- [helib](#)

## Macros

- #define [HELIB\\_STRINGIFY](#)(x) #x
- #define [HELIB\\_TOSTRING](#)(x) [HELIB\\_STRINGIFY](#)(x)
- #define [HELIB\\_AT](#) \_\_FILE\_\_ ":" [HELIB\\_TOSTRING](#)(\_\_LINE\_\_)
- #define [HELIB\\_stringify\\_aux](#)(s) #s
- #define [HELIB\\_stringify](#)(s) [HELIB\\_stringify\\_aux](#)(s)
- #define [HELIB\\_TIMER\\_START](#)
- #define [HELIB\\_TIMER\\_STOP](#) \_local\_auto\_timer.stop()
- #define [HELIB\\_NTIMER\\_START](#)(n)
- #define [HELIB\\_NTIMER\\_STOP](#)(n) \_named\_local\_auto\_timer##n.stop();

## Functions

- void `helib::registerTimer` (FHEtimer \*timer)
- unsigned long `helib::GetTimerClock` ()
- void `helib::setTimersOn` ()
- void `helib::setTimersOff` ()
- bool `helib::areTimersOn` ()
- const FHEtimer \* `helib::getTimerByName` (const char \*name)
- void `helib::resetAllTimers` ()
- void `helib::printAllTimers` (std::ostream &str=std::cerr)  
*Print the value of all timers to stream.*
- bool `helib::printNamedTimer` (std::ostream &str, const char \*name)

### 8.50.1 Detailed Description

Utility functions for measuring time.

This module contains some utility functions for measuring the time that various methods take to execute. To use it, we insert the macro `HELIB_TIMER_START` at the beginning of the method(s) that we want to time and `HELIB_TIMER_STOP` at the end, then the main program needs to call the function `setTimersOn()` to activate the timers and `setTimersOff()` to pause them. To obtain the value of a given timer (in seconds), the application can use the function `getTime4func(const char *fncName)`, and the function `printAllTimers()` prints the values of all timers to an output stream.

Using this method we can have at most one timer per method/function, and the timer is called by the same name as the function itself (using the built-in macro `__func__`). We can also use the "lower level" methods `startFHEtimer(name)`, `stopFHEtimer(name)`, and `resetFHEtimer(name)` to add timers with arbitrary names (not necessarily associated with functions).

### 8.50.2 Macro Definition Documentation

#### 8.50.2.1 HELIB\_AT

```
#define HELIB_AT __FILE__ ":" HELIB_TOSTRING(__LINE__)
```

#### 8.50.2.2 HELIB\_NTIMER\_START

```
#define HELIB_NTIMER_START(  
    n )
```

#### Value:

```
static helib::FHEtimer _named_local_timer##n(#n, HELIB_AT);  
helib::auto_timer _named_local_auto_timer##n(&_named_local_timer##n) \
```

### 8.50.2.3 HELIB\_NTIMER\_STOP

```
#define HELIB_NTIMER_STOP(  
    n ) _named_local_auto_timer##n.stop();
```

### 8.50.2.4 HELIB\_stringify

```
#define HELIB_stringify(  
    s ) HELIB_stringify_aux(s)
```

### 8.50.2.5 HELIB\_STRINGIFY

```
#define HELIB_STRINGIFY(  
    x ) #x
```

### 8.50.2.6 HELIB\_stringify\_aux

```
#define HELIB_stringify_aux(  
    s ) #s
```

### 8.50.2.7 HELIB\_TIMER\_START

```
#define HELIB_TIMER_START
```

#### Value:

```
static helib::FHEtimer _local_timer(__func__, HELIB_AT);  
helib::auto_timer _local_auto_timer(&_local_timer)
```

\

### 8.50.2.8 HELIB\_TIMER\_STOP

```
#define HELIB_TIMER_STOP _local_auto_timer.stop()
```

### 8.50.2.9 HELIB\_TOSTRING

```
#define HELIB_TOSTRING(  
    x ) HELIB_STRINGIFY(x)
```



## 8.51 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/include/helib/zzX.h File Reference

```
#include <NTL/vector.h>
#include <NTL/lzz_pX.h>
#include <NTL/GF2X.h>
```

### Namespaces

- [helib](#)

### Typedefs

- typedef NTL::Vec< long > [helib::zzX](#)

### Functions

- bool [helib::IsZero](#) (const zzX &a)
- void [helib::clear](#) (zzX &a)
- void [helib::convert](#) (NTL::zz\_pX &x, const zzX &a)
- void [helib::add](#) (zzX &res, const zzX &a, const zzX &b)
- zzX [helib::operator+](#) (const zzX &a, const zzX &b)
- zzX & [helib::operator+=](#) (zzX &a, const zzX &b)
- void [helib::div](#) (zzX &res, const zzX &a, long b)
- zzX [helib::operator/](#) (const zzX &a, long b)
- zzX & [helib::operator/=](#) (zzX &a, long b)
- void [helib::mul](#) (zzX &res, const zzX &a, long b)
- zzX [helib::operator\\*](#) (const zzX &a, long b)
- zzX & [helib::operator\\*="](#) (zzX &a, long b)
- void [helib::normalize](#) (zzX &f)
- const NTL::zz\_pXModulus & [helib::getPhimXMod](#) (const PAlgebra &palg)
- void [helib::reduceModPhimX](#) (zzX &poly, const PAlgebra &palg)
- void [helib::MulMod](#) (zzX &res, const zzX &a, const zzX &b, const PAlgebra &palg)
- zzX [helib::MulMod](#) (const zzX &a, const zzX &b, const PAlgebra &palg)
- zzX [helib::balanced\\_zzX](#) (const NTL::zz\_pX &f)
- zzX [helib::balanced\\_zzX](#) (const NTL::GF2X &f)

#### 8.51.1 Detailed Description

- manipulating polynomials with single-precision coefficient It is assumed that the result is also single-precision

## 8.52 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/BenesNetwork.cpp File Reference

```
#include <NTL/lzz_pXFactoring.h>
#include <helib/EncryptedArray.h>
#include <cstdlib>
#include <list>
#include <sstream>
#include <memory>
#include <NTL/vector.h>
#include <helib/NumbTh.h>
#include <helib/permutations.h>
```

### Namespaces

- [helib](#)

## 8.53 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/binaryArith.cpp File Reference

Implementing integer addition, multiplication in binary representation.

```
#include <numeric>
#include <climits>
#include <map>
#include <algorithm>
#include <stdexcept>
#include <atomic>
#include <mutex>
#include <NTL/BasicThreadPool.h>
#include <helib/binaryArith.h>
```

### Classes

- class [helib::DAGnode](#)  
*A node in an addition-DAG structure.*
- class [helib::ScratchCell](#)  
*A class to help manage the allocation of temporary [Ctxt](#) objects.*
- class [helib::AddDAG](#)  
*A class representing the logic of the order of bit products when adding two integers.*
- struct [helib::PtrMatrix\\_PtPtrVector< T >](#)  
*An implementation of [PtrMatrix](#) using `vector< PtrVector<T>*>`*

## Namespaces

- [helib](#)

## Macros

- `#define BPL_ESTIMATE` (30)

## Typedefs

- `typedef std::pair< long, long >` [helib::Nodeldx](#)

## Functions

- `long` [helib::defaultPmiddle](#) (long delta)
- `long` [helib::defaultQmiddle](#) (long delta)
- `void` [helib::packedRecrypt](#) (const CPtrs &a, const CPtrs &b, std::vector< zzX > \*unpackSlotEncoding)  
*Function for packed recryption to recrypt multiple numbers.*
- `std::vector< long >` [helib::longToBitVector](#) (long num, long bitSize)  
*Returns a number as a vector of bits with LSB on the left.*
- `void` [helib::binaryMask](#) (CPtrs &binaryNums, const Ctxt &mask)  
*Zeroes the slots of *binaryNums* where the corresponding slot of *mask* is 0.*
- `void` [helib::binaryCond](#) (CPtrs &output, const Ctxt &cond, const CPtrs &>trueValue, const CPtrs &>falseValue)  
*Implementation of  $output = cond * trueValue + (1 - cond) * falseValue$ .*
- `void` [helib::concatBinaryNums](#) (CPtrs &output, const CPtrs &a, const CPtrs &b)  
*Concatenates two binary numbers into a single CPtrs object. E.g. If  $a=10111$ ,  $b=00101$  then  $output = 1011100101$ .*
- `void` [helib::splitBinaryNums](#) (CPtrs &leftSplit, CPtrs &rightSplit, const CPtrs &input)  
*Splits a single binary number into two binary numbers *leftSplit* and *rightSplit*.*
- `void` [helib::leftBitwiseShift](#) (CPtrs &output, const CPtrs &input, const long shamt)  
*Left shift input by *shamt*.*
- `void` [helib::bitwiseRotate](#) (CPtrs &output, const CPtrs &input, long rotamt)  
*Rotate input by *rotamt*.*
- `void` [helib::bitwiseXOR](#) (CPtrs &output, const CPtrs &lhs, const CPtrs &rhs)  
*Compute a bitwise XOR between *lhs* and *rhs*.*
- `void` [helib::bitwiseOr](#) (CPtrs &output, const CPtrs &lhs, const CPtrs &rhs)  
*Compute a bitwise OR between *lhs* and *rhs*.*
- `void` [helib::bitwiseAnd](#) (CPtrs &output, const CPtrs &lhs, const CPtrs &rhs)  
*Compute a bitwise AND between *lhs* and *rhs*.*
- `void` [helib::bitwiseAnd](#) (CPtrs &output, const CPtrs &input, const std::vector< long > mask)  
*Compute a bitwise AND between *input* and a `std::vector<long>`.*
- `void` [helib::bitwiseNot](#) (CPtrs &output, const CPtrs &input)  
*Compute a bitwise NOT of *input*.*
- `void` [helib::addTwoNumbers](#) (CPtrs &sum, const CPtrs &lhs, const CPtrs &rhs, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)  
*Adds two numbers in binary representation where each ciphertext of the input vector contains a bit.*
- `void` [helib::negateBinary](#) (CPtrs &negation, const CPtrs &input)  
*Negates a number in binary 2's complement representation.*
- `void` [helib::subtractBinary](#) (CPtrs &difference, const CPtrs &lhs, const CPtrs &rhs, std::vector< zzX > \*unpackSlotEncoding=nullptr)

*Subtracts  $rhs$  from  $lhs$  where  $lhs, rhs$  are in 2's complement.*

- void [helib::addManyNumbers](#) (CtPtrs &sum, CtPtrMat &numbers, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)

*Sum an arbitrary amount of numbers in binary representation.*

- void [helib::multTwoNumbers](#) (CtPtrs &product, const CtPtrs &lhs, const CtPtrs &rhs, bool rhsTwosComplement=false, long sizeLimit=0, std::vector< zzX > \*unpackSlotEncoding=nullptr)

*Multiply two numbers in binary representation where each ciphertext of the input vector contains a bit.*

- long [helib::fifteenOrLess4Four](#) (const CtPtrs &out, const CtPtrs &in, long sizeLimit=4)

*Add together up to fifteen  $\{0,1\}$  integers, producing a 4-bit counter.*

- void [helib::decryptBinaryNums](#) (std::vector< long > &pNums, const CtPtrs &eNums, const SecKey &sKey, const EncryptedArray &ea, bool twosComplement=false, bool allSlots=true)

*Decrypt the binary numbers that are encrypted in  $eNums$ .*

### 8.53.1 Detailed Description

Implementing integer addition, multiplication in binary representation.

### 8.53.2 Macro Definition Documentation

#### 8.53.2.1 BPL\_ESTIMATE

```
#define BPL_ESTIMATE (30)
```

## 8.54 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/binaryCompare.cpp File Reference

Implementing integer comparison in binary representation.

```
#include <algorithm>
#include <NTL/BasicThreadPool.h>
#include <helib/binaryArith.h>
```

### Namespaces

- [helib](#)

### Macros

- #define [BPL\\_ESTIMATE](#) (30)

## Functions

- void [helib::runningSums](#) (CtPtrs &v)
- void [helib::compareTwoNumbersImplementation](#) (CtPtrs &max, CtPtrs &min, Ctxt &mu, Ctxt &ni, const CtPtrs &aa, const CtPtrs &bb, bool twosComplement, std::vector< zzX > \*unpackSlotEncoding, bool cmp\_only)
- void [helib::compareTwoNumbers](#) (CtPtrs &max, CtPtrs &min, Ctxt &mu, Ctxt &ni, const CtPtrs &a, const CtPtrs &b, bool twosComplement=false, std::vector< zzX > \*unpackSlotEncoding=nullptr)
 

*Compares two integers in binary  $a, b$ . Returns  $\max(a, b)$ ,  $\min(a, b)$  and indicator bits  $\mu=(a>b)$  and  $\text{ni}=(a<b)$*
- void [helib::compareTwoNumbers](#) (Ctxt &mu, Ctxt &ni, const CtPtrs &a, const CtPtrs &b, bool twosComplement=false, std::vector< zzX > \*unpackSlotEncoding=nullptr)
 

*Compares two integers in binary  $a, b$ . Returns only indicator bits  $\mu=(a>b)$  and  $\text{ni}=(a<b)$ .*

### 8.54.1 Detailed Description

Implementing integer comparison in binary representation.

### 8.54.2 Macro Definition Documentation

#### 8.54.2.1 BPL\_ESTIMATE

```
#define BPL_ESTIMATE (30)
```

## 8.55 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/binio.cpp File Reference

```
#include <helib/binio.h>
#include <helib/assertions.h>
#include <cstring>
#include <sys/types.h>
```

## Namespaces

- [helib](#)

## Functions

- int [helib::readEyeCatcher](#) (std::istream &str, const char \*expect)
- void [helib::writeEyeCatcher](#) (std::ostream &str, const char \*eye)
- long [helib::read\\_raw\\_int](#) (std::istream &str)
- int [helib::read\\_raw\\_int32](#) (std::istream &str)
- void [helib::write\\_raw\\_int](#) (std::ostream &str, long num)
- void [helib::write\\_raw\\_int32](#) (std::ostream &str, int num)
- void [helib::write\\_ntl\\_vec\\_long](#) (std::ostream &str, const NTL::vec\_long &vl, long intSize=[BINIO\\_64BIT](#))
- void [helib::read\\_ntl\\_vec\\_long](#) (std::istream &str, NTL::vec\_long &vl)
- void [helib::write\\_raw\\_double](#) (std::ostream &str, const double d)
- double [helib::read\\_raw\\_double](#) (std::istream &str)
- void [helib::write\\_raw\\_xdouble](#) (std::ostream &str, const NTL::xdouble xd)
- NTL::xdouble [helib::read\\_raw\\_xdouble](#) (std::istream &str)
- void [helib::write\\_raw\\_ZZ](#) (std::ostream &str, const NTL::ZZ &zz)
- void [helib::read\\_raw\\_ZZ](#) (std::istream &str, NTL::ZZ &zz)
- template<> void [helib::read\\_raw\\_vector< long >](#) (std::istream &str, std::vector< long > &v)
- template<> void [helib::write\\_raw\\_vector< long >](#) (std::ostream &str, const std::vector< long > &v)
- template<> void [helib::read\\_raw\\_vector< double >](#) (std::istream &str, std::vector< double > &v)
- template<> void [helib::write\\_raw\\_vector< double >](#) (std::ostream &str, const std::vector< double > &v)

## 8.56 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/bluestein.cpp File Reference

```
#include <helib/bluestein.h>
#include <helib/timing.h>
#include <helib/CModulus.h>
#include <helib/apiAttributes.h>
```

## Namespaces

- [helib](#)

## Macros

- [#define NEW\\_BLUE](#) (1)

## Functions

- void [helib::BluesteinInit](#) (long n, const NTL::zz\_p &root, NTL::zz\_pX &powers, NTL::Vec< NTL::mulmod\_↵  
precon\_t > &powers\_aux, NTL::fftRep &Rb)  
*initialize bluestein*
- void [helib::BluesteinFFT](#) (NTL::zz\_pX &x, long n, [UNUSED](#) const NTL::zz\_p &root, const NTL::zz\_pX &powers, const NTL::Vec< NTL::mulmod\_precon\_t > &powers\_aux, const NTL::fftRep &Rb)

## 8.56.1 Macro Definition Documentation

### 8.56.1.1 NEW\_BLUE

```
#define NEW_BLUE (1)
```

## 8.57 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/CModulus.cpp File Reference

```
#include <helib/CModulus.h>
#include <helib/timing.h>
```

### Namespaces

- [helib](#)

### Functions

- NTL::zz\_pContext [helib::BuildContext](#) (long p, long maxroot)

## 8.58 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Context.cpp File Reference

```
#include <cstring>
#include <algorithm>
#include <helib/Context.h>
#include <helib/EvalMap.h>
#include <helib/powerful.h>
#include <helib/binio.h>
#include <helib/sample.h>
#include <helib/EncryptedArray.h>
#include <helib/PolyModRing.h>
```

### Namespaces

- [helib](#)

## Functions

- long [helib::FindM](#) (long k, long nBits, long c, long p, long d, long s, long chosen\_m, bool [verbose](#)=false)  
*Returns smallest parameter m satisfying various constraints:*
- void [helib::writeContextBaseBinary](#) (std::ostream &str, const Context &context)  
*write [m p r gens ords] data*
- void [helib::readContextBaseBinary](#) (std::istream &s, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)  
*read [m p r gens ords] data, needed to construct context*
- std::unique\_ptr< Context > [helib::buildContextFromBinary](#) (std::istream &str)
- void [helib::writeContextBinary](#) (std::ostream &str, const Context &context)
- void [helib::readContextBinary](#) (std::istream &str, Context &context)
- void [helib::writeContextBase](#) (std::ostream &s, const Context &context)  
*write [m p r gens ords] data*
- std::ostream & [helib::operator<<](#) (std::ostream &str, const Context &context)
- void [helib::readContextBase](#) (std::istream &s, unsigned long &m, unsigned long &p, unsigned long &r, std::vector< long > &gens, std::vector< long > &ords)  
*read [m p r gens ords] data, needed to construct context*
- std::unique\_ptr< Context > [helib::buildContextFromAscii](#) (std::istream &str)
- std::istream & [helib::operator>>](#) (std::istream &str, Context &context)
- NTL::ZZX [helib::getG](#) (const EncryptedArray &ea)

## 8.59 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_ v1.0.2/HElib/src/Ctxt.cpp File Reference

```
#include <NTL/BasicThreadPool.h>
#include <helib/binio.h>
#include <helib/timing.h>
#include <helib/Context.h>
#include <helib/Ctxt.h>
#include <helib/keySwitching.h>
#include <helib/CtPtrs.h>
#include <helib/EncryptedArray.h>
#include <helib/Ptxt.h>
#include <helib/debugging.h>
#include <helib/norms.h>
#include <helib/fhe_stats.h>
#include <helib/powerful.h>
```

## Namespaces

- [helib](#)



## Functions

- void [helib::addSomePrimes](#) (Ctxt &c)
  - void [helib::computeIntervalForMul](#) (double &lo, double &hi, const Ctxt &ctxt1, const Ctxt &ctxt2)
  - void [helib::computeIntervalForSqr](#) (double &lo, double &hi, const Ctxt &ctxt)
  - std::istream & [helib::operator>>](#) (std::istream &str, SKHandle &handle)
  - std::ostream & [helib::operator<<](#) (std::ostream &s, const CtxtPart &p)
  - std::istream & [helib::operator>>](#) (std::istream &s, CtxtPart &p)
  - std::ostream & [helib::operator<<](#) (std::ostream &str, const Ctxt &ctxt)
  - std::istream & [helib::operator>>](#) (std::istream &str, Ctxt &ctxt)
  - void [helib::incrementalProduct](#) (std::vector< Ctxt > &v)
  - void [helib::totalProduct](#) (Ctxt &out, const std::vector< Ctxt > &v)
  - void [helib::innerProduct](#) (Ctxt &result, const CtPtrs &v1, const CtPtrs &v2)
  - void [helib::innerProduct](#) (Ctxt &result, const std::vector< Ctxt > &v1, const std::vector< Ctxt > &v2)
  - void [helib::innerProduct](#) (Ctxt &result, const std::vector< Ctxt > &v1, const std::vector< DoubleCRT > &v2)
- Compute the inner product of a vectors of ciphertexts and a constant vector.*
- void [helib::innerProduct](#) (Ctxt &result, const std::vector< Ctxt > &v1, const std::vector< NTL::ZZX > &v2)

## Variables

- int [helib::fhe\\_watcher](#) = 0

## 8.60 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/debugging.cpp File Reference

```
#include <NTL/xdouble.h>
#include <helib/debugging.h>
#include <helib/norms.h>
#include <helib/Context.h>
#include <helib/Ctxt.h>
#include <helib/EncryptedArray.h>
```

## Namespaces

- [helib](#)

## Functions

- double [helib::realToEstimatedNoise](#) (const Ctxt &ctxt, const SecKey &sk)
  - double [helib::log2\\_realToEstimatedNoise](#) (const Ctxt &ctxt, const SecKey &sk)
  - void [helib::checkNoise](#) (const Ctxt &ctxt, const SecKey &sk, const std::string &msg, double thresh=10.0)
  - NTL::xdouble [helib::embeddingLargestCoeff](#) (const Ctxt &ctxt, const SecKey &sk)
  - void [helib::decryptAndPrint](#) (std::ostream &s, const Ctxt &ctxt, const SecKey &sk, const EncryptedArray &ea, long flags=0)
  - bool [helib::decryptAndCompare](#) (const Ctxt &ctxt, const SecKey &sk, const EncryptedArray &ea, const PlaintextArray &pa)
  - void [helib::rawDecrypt](#) (NTL::ZZX &plaintext, const std::vector< NTL::ZZX > &zzParts, const DoubleCRT &s, Key, long q=0)
  - void [helib::CheckCtxt](#) (const Ctxt &c, const char \*label)
- print to cerr some info about ciphertext*

## 8.61 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/DoubleCRT.cpp File Reference

```
#include <NTL/ZZVec.h>
#include <NTL/BasicThreadPool.h>
#include <helib/timing.h>
#include <helib/binio.h>
#include <helib/sample.h>
#include <helib/DoubleCRT.h>
#include <helib/Context.h>
#include <helib/norms.h>
#include <helib/fhe_stats.h>
```

### Namespaces

- [helib](#)

### Functions

- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::AddFun >](#) (const DoubleCRT &other, AddFun fun, bool matchIndexSets)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::SubFun >](#) (const DoubleCRT &other, SubFun fun, bool matchIndexSets)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::MulFun >](#) (const NTL::ZZ &num, MulFun fun)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::AddFun >](#) (const NTL::ZZ &num, AddFun fun)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::SubFun >](#) (const NTL::ZZ &num, SubFun fun)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::MulFun >](#) (const NTL::ZZX &poly, MulFun fun)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::AddFun >](#) (const NTL::ZZX &poly, AddFun fun)
- template DoubleCRT & [helib::DoubleCRT::Op< DoubleCRT::SubFun >](#) (const NTL::ZZX &poly, SubFun fun)
- std::ostream & [helib::operator<<](#) (std::ostream &str, const DoubleCRT &d)
- std::istream & [helib::operator>>](#) (std::istream &str, DoubleCRT &d)

## 8.62 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/EaCx.cpp File Reference

```
#include <algorithm>
#include <type_traits>
#include <helib/zzX.h>
#include <helib/EncryptedArray.h>
#include <helib/timing.h>
#include <helib/clonedPtr.h>
#include <helib/norms.h>
#include <helib/debugging.h>
#include <helib/apiAttributes.h>
```

## Namespaces

- [helib](#)

## 8.63 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EncryptedArray.cpp File Reference

```
#include <algorithm>
#include <helib/zzX.h>
#include <helib/EncryptedArray.h>
#include <helib/timing.h>
#include <helib/clonedPtr.h>
#include <helib/norms.h>
```

## Classes

- class [helib::rotate\\_pa\\_impl< type >](#)
- class [helib::shift\\_pa\\_impl< type >](#)
- class [helib::encode\\_pa\\_impl< type >](#)
- class [helib::random\\_pa\\_impl< type >](#)
- class [helib::decode\\_pa\\_impl< type >](#)
- class [helib::equals\\_pa\\_impl< type >](#)
- class [helib::add\\_pa\\_impl< type >](#)
- class [helib::sub\\_pa\\_impl< type >](#)
- class [helib::mul\\_pa\\_impl< type >](#)
- class [helib::negate\\_pa\\_impl< type >](#)
- class [helib::frobeniusAutomorph\\_pa\\_impl< type >](#)
- class [helib::applyPerm\\_pa\\_impl< type >](#)
- class [helib::print\\_pa\\_impl< type >](#)

## Namespaces

- [helib](#)

## Functions

- EncryptedArrayBase \* [helib::buildEncryptedArray](#) (const Context &context, const PAIgebraMod &alMod, const NTL::ZZX &G=NTL::ZZX::zero())  
A "factory" for building EncryptedArrays.
- void [helib::runningSums](#) (const EncryptedArray &ea, Ctxt &ctxt)  
A ctxt that encrypts  $(x_1, \dots, x_n)$  is replaced by an encryption of  $(y_1, \dots, y_n)$ , where  $y_i = \sum_{j \leq i} x_j$ .
- void [helib::totalSums](#) (const EncryptedArray &ea, Ctxt &ctxt)
- void [helib::applyLinPoly1](#) (const EncryptedArray &ea, Ctxt &ctxt, const std::vector< NTL::ZZX > &C)
- void [helib::applyLinPolyMany](#) (const EncryptedArray &ea, Ctxt &ctxt, const std::vector< std::vector< NTL::ZZX >> &Cvec)
- template<typename P >  
void [helib::applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< P > &encodedC, long d)

- template void [helib::applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< zzX > &encodedC, long d)
- template void [helib::applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< NTL::ZZX > &encodedC, long d)
- template void [helib::applyLinPolyLL](#) (Ctxt &ctxt, const std::vector< DoubleCRT > &encodedC, long d)
- void [helib::rotate](#) (const EncryptedArray &ea, PlaintextArray &pa, long k)
- void [helib::shift](#) (const EncryptedArray &ea, PlaintextArray &pa, long k)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, const std::vector< long > &array)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, const std::vector< NTL::ZZX > &array)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, long val)
- void [helib::encode](#) (const EncryptedArray &ea, PlaintextArray &pa, const NTL::ZZX &val)
- void [helib::random](#) (const EncryptedArray &ea, PlaintextArray &pa)
- void [helib::decode](#) (const EncryptedArray &ea, std::vector< long > &array, const PlaintextArray &pa)
- void [helib::decode](#) (const EncryptedArray &ea, std::vector< NTL::ZZX > &array, const PlaintextArray &pa)
- bool [helib::equals](#) (const EncryptedArray &ea, const PlaintextArray &pa, const PlaintextArray &other)
- bool [helib::equals](#) (const EncryptedArray &ea, const PlaintextArray &pa, const std::vector< long > &other)
- bool [helib::equals](#) (const EncryptedArray &ea, const PlaintextArray &pa, const std::vector< NTL::ZZX > &other)
- void [helib::add](#) (const EncryptedArray &ea, PlaintextArray &pa, const PlaintextArray &other)
- void [helib::sub](#) (const EncryptedArray &ea, PlaintextArray &pa, const PlaintextArray &other)
- void [helib::mul](#) (const EncryptedArray &ea, PlaintextArray &pa, const PlaintextArray &other)
- void [helib::negate](#) (const EncryptedArray &ea, PlaintextArray &pa)
- void [helib::frobeniusAutomorph](#) (const EncryptedArray &ea, PlaintextArray &pa, long j)
- void [helib::frobeniusAutomorph](#) (const EncryptedArray &ea, PlaintextArray &pa, const NTL::Vec< long > &vec)
- void [helib::power](#) (const EncryptedArray &ea, PlaintextArray &pa, long e)
- void [helib::applyPerm](#) (const EncryptedArray &ea, PlaintextArray &pa, const NTL::Vec< long > &pi)
- void [helib::print](#) (const EncryptedArray &ea, std::ostream &s, const PlaintextArray &pa)

## 8.64 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/eqtesting.cpp File Reference

Useful functions for equality testing...

```
#include <NTL/lzz_pXFactoring.h>
#include <helib/timing.h>
#include <helib/EncryptedArray.h>
#include <helib/Ptxt.h>
#include <cstdio>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::mapTo01](#) (const EncryptedArray &ea, Ctxt &ctxt)
- template<typename Scheme >  
void [helib::mapTo01](#) (const EncryptedArray &, Ptxt< Scheme > &ptxt)
- template void [helib::mapTo01](#) (const EncryptedArray &, Ptxt< BGV > &ptxt)
- template void [helib::mapTo01](#) (const EncryptedArray &, Ptxt< CKKS > &ptxt)
- void [helib::fastPower](#) (Ctxt &ctxt, long d)
- void [helib::incrementalZeroTest](#) (Ctxt \*res[], const EncryptedArray &ea, const Ctxt &ctxt, long n)

### 8.64.1 Detailed Description

Useful functions for equality testing...

## 8.65 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/EvalMap.cpp File Reference

```
#include <helib/EvalMap.h>
#include <helib/apiAttributes.h>
#include <NTL/lzz_pXFactoring.h>
#include <NTL/GF2XFactoring.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::RelaxedInv](#) (NTL::Mat< NTL::zz\_p > &x, const NTL::Mat< NTL::zz\_p > &a)
- void [helib::RelaxedInv](#) (NTL::Mat< NTL::GF2 > &x, const NTL::Mat< NTL::GF2 > &a)
- void [helib::TraceMap](#) (NTL::GF2X &w, const NTL::GF2X &a, long d, const NTL::GF2XModulus &F, const NTL::GF2X &b)

## 8.66 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/extractDigits.cpp File Reference

```
#include <NTL/ZZ.h>
#include <NTL/ZZ_p.h>
#include <helib/EncryptedArray.h>
#include <helib/polyEval.h>
#include <helib/debugging.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::extractDigits](#) (std::vector< Ctxt > &digits, const Ctxt &c, long r=0)  
*Extract the mod-p digits of a mod-p<sup>r</sup> ciphertext.*
- void [helib::extendExtractDigits](#) (std::vector< Ctxt > &digits, const Ctxt &c, long r, long e)

## 8.67 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/fhe\_stats.cpp File Reference

```
#include <helib/fhe_stats.h>
#include <helib/multicore.h>
#include <algorithm>
#include <utility>
#include <cstring>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::print\\_stats](#) (std::ostream &s)
- const std::vector< double > \* [helib::fetch\\_saved\\_values](#) (const char \*)

## 8.68 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/hypercube.cpp File Reference

```
#include <helib/hypercube.h>
#include <iomanip>
#include <NTL/lzz_p.h>
```

### Namespaces

- [helib](#)

### Functions

- template<typename T >  
void [helib::getHyperColumn](#) (NTL::Vec< T > &v, const ConstCubeSlice< T > &s, long pos)
- template<typename T >  
void [helib::setHyperColumn](#) (const NTL::Vec< T > &v, const CubeSlice< T > &s, long pos)
- template<typename T >  
void [helib::setHyperColumn](#) (const NTL::Vec< T > &v, const CubeSlice< T > &s, long pos, const T &val)
- template<typename T >  
void [helib::print3D](#) (const HyperCube< T > &c)
- template void [helib::getHyperColumn](#) (NTL::Vec< long > &v, const ConstCubeSlice< long > &s, long pos)
- template void [helib::setHyperColumn](#) (const NTL::Vec< long > &v, const CubeSlice< long > &s, long pos)
- template void [helib::setHyperColumn](#) (const NTL::Vec< long > &v, const CubeSlice< long > &s, long pos, const long &val)

- template void [helib::print3D](#) (const HyperCube< long > &c)
- template void [helib::getHyperColumn](#) (NTL::Vec< NTL::zz\_p > &v, const ConstCubeSlice< NTL::zz\_p > &s, long pos)
- template void [helib::setHyperColumn](#) (const NTL::Vec< NTL::zz\_p > &v, const CubeSlice< NTL::zz\_p > &s, long pos)
- template void [helib::setHyperColumn](#) (const NTL::Vec< NTL::zz\_p > &v, const CubeSlice< NTL::zz\_p > &s, long pos, const NTL::zz\_p &val)
- template void [helib::print3D](#) (const HyperCube< NTL::zz\_p > &c)

## 8.69 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/IndexSet.cpp File Reference

```
#include <helib/IndexSet.h>
#include <helib/binio.h>
```

### Namespaces

- [helib](#)

### Functions

- IndexSet [helib::operator|](#) (const IndexSet &s, const IndexSet &t)  
*union*
- IndexSet [helib::operator&](#) (const IndexSet &s, const IndexSet &t)  
*intersection*
- IndexSet [helib::operator^](#) (const IndexSet &s, const IndexSet &t)  
*exclusive-or*
- IndexSet [helib::operator/](#) (const IndexSet &s, const IndexSet &t)  
*set minus*
- long [helib::card](#) (const IndexSet &s)  
*Functional cardinality.*
- bool [helib::operator<=](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s1 subset or equal to s2.*
- bool [helib::operator<](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s1 strict subset of s2.*
- bool [helib::operator>=](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s2 subset or equal to s2.*
- bool [helib::operator>](#) (const IndexSet &s1, const IndexSet &s2)  
*Is s2 strict subset of s1.*
- std::ostream & [helib::operator<<](#) (std::ostream &str, const IndexSet &set)
- std::istream & [helib::operator>>](#) (std::istream &str, IndexSet &set)

## 8.70 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/intraSlot.cpp File Reference

```
#include <memory>
#include <helib/replicate.h>
#include <helib/intraSlot.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::unpack](#) (const CtPtrs &unpacked, const Ctxt &packed, const EncryptedArray &ea, const std::vector< zzX > &unpackSlotEncoding)
- long [helib::unpack](#) (const CtPtrs &unpacked, const CtPtrs &packed, const EncryptedArray &ea, const std::vector< zzX > &unpackSlotEncoding)
- void [helib::repack](#) (Ctxt &packed, const CtPtrs &unpacked, const EncryptedArray &ea)
- long [helib::repack](#) (const CtPtrs &packed, const CtPtrs &unpacked, const EncryptedArray &ea)
- void [helib::packConstant](#) (zzX &result, unsigned long data, long nbits, const EncryptedArray &ea)
- void [helib::packConstants](#) (zzX &result, const std::vector< unsigned long > &data, long nbits, const EncryptedArray &ea)
- void [helib::unpackSlots](#) (std::vector< unsigned long > &value, PlaintextArray &pa, const EncryptedArray &ea)

## 8.71 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/keys.cpp File Reference

```
#include <queue>
#include <helib/keys.h>
#include <helib/timing.h>
#include <helib/EncryptedArray.h>
#include <helib/Ptxt.h>
#include <helib/binio.h>
#include <helib/sample.h>
#include <helib/norms.h>
#include <helib/apiAttributes.h>
#include <helib/fhe_stats.h>
```

### Namespaces

- [helib](#)



## Macros

- #define [DECRYPT\\_ON\\_PWFL\\_BASIS](#)

## Functions

- double [helib::RLWE1](#) (DoubleCRT &c0, const DoubleCRT &c1, const DoubleCRT &s, long p)  
*Same as RLWE, but assumes that c1 is already chosen by the caller.*
- double [helib::RLWE](#) (DoubleCRT &c0, DoubleCRT &c1, const DoubleCRT &s, long p, NTL::ZZ \*prg↵  
Seed=nullptr)
- std::ostream & [helib::operator<<](#) (std::ostream &str, const PubKey &pk)
- std::istream & [helib::operator>>](#) (std::istream &str, PubKey &pk)
- void [helib::writePubKeyBinary](#) (std::ostream &str, const PubKey &pk)
- void [helib::readPubKeyBinary](#) (std::istream &str, PubKey &pk)
- std::ostream & [helib::operator<<](#) (std::ostream &str, const SecKey &sk)
- std::istream & [helib::operator>>](#) (std::istream &str, SecKey &sk)
- void [helib::writeSecKeyBinary](#) (std::ostream &str, const SecKey &sk)
- void [helib::readSecKeyBinary](#) (std::istream &str, SecKey &sk)

### 8.71.1 Macro Definition Documentation

#### 8.71.1.1 DECRYPT\_ON\_PWFL\_BASIS

```
#define DECRYPT_ON_PWFL_BASIS
```

## 8.72 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/keySwitching.cpp File Reference

A few strategies for generating key-switching matrices.

```
#include <unordered_set>
#include <NTL/ZZ.h>
#include <helib/permutations.h>
#include <helib/binio.h>
#include <helib/keySwitching.h>
#include <helib/keys.h>
#include <helib/apiAttributes.h>
```

## Namespaces

- [helib](#)

## Macros

- #define `computeParams`(context, m, i)

## Functions

- `std::ostream & helib::operator<<` (`std::ostream &str`, `const KeySwitch &matrix`)

### Strategies for generating key-switching matrices

These functions are implemented in `KeySwitching.cpp`

- long `helib::KSGiantStepSize` (long D)  
Function that returns number of baby steps. Used to keep this and `matmul` routines "in sync".
- void `helib::addAllMatrices` (SecKey &sKey, long keyID=0)  
Maximalistic approach: generate matrices  $s(X^e) \rightarrow s(X)$  for all  $e$  in  $Zm^*$ .
- void `helib::addSome1DMatrices` (SecKey &sKey, long bound=`HELIB_KEYSWITCH_THRESH`, long keyID=0)  
Generate some matrices of the form  $s(X^{g^i}) \rightarrow s(X)$ , but not all. For a generator  $g$  whose order is larger than bound, generate only enough matrices for the giant-step/baby-step procedures ( $2 \cdot \sqrt{\text{ord}(g)}$  of them).
- void `helib::add1DMatrices` (SecKey &sKey, long keyID=0)  
Generate all matrices  $s(X^{g^i}) \rightarrow s(X)$  for generators  $g$  of  $Zm^*/(p)$  and  $i < \text{ord}(g)$ . If  $g$  has different orders in  $Zm^*$  and  $Zm^*/(p)$  then generate also matrices of the form  $s(X^{g^{-i}}) \rightarrow s(X)$
- void `helib::addBSGS1DMatrices` (SecKey &sKey, long keyID=0)
- void `helib::addSomeFrbMatrices` (SecKey &sKey, long bound=`HELIB_KEYSWITCH_THRESH`, long keyID=0)  
Generate all/some Frobenius matrices of the form  $s(X^{p^i}) \rightarrow s(X)$
- void `helib::addFrbMatrices` (SecKey &sKey, long keyID=0)
- void `helib::addBSGSFrbMatrices` (SecKey &sKey, long keyID=0)
- void `helib::addMinimal1DMatrices` (SecKey &sKey, long keyID=0)  
These routines just add a single matrix (or two, for bad dimensions)
- void `helib::addMinimalFrbMatrices` (SecKey &sKey, long keyID=0)
- void `helib::addMatrices4Network` (SecKey &sKey, const PermNetwork &net, long keyID=0)
- void `helib::addTheseMatrices` (SecKey &sKey, const std::set< long > &automVals, long keyID=0)  
Generate specific key-switching matrices, described by the given set.

### 8.72.1 Detailed Description

A few strategies for generating key-switching matrices.

Copyright IBM Corporation 2012 All rights reserved.

### 8.72.2 Macro Definition Documentation

## 8.72.2.1 computeParams

```
#define computeParams(
    context,
    m,
    i )
```

## Value:

```
bool native;
long ord, gi, g2md;
NTL::mulmod_precon_t g2mdminv;
if (i == context.zMStar.numOfGens()) { /* Frobenius matrices */
    ord = context.zMStar.getOrdP();
    gi = context.zMStar.getP();
    native = true;
} else { /* one of the "regular" dimensions */
    ord = context.zMStar.OrderOf(i);
    gi = context.zMStar.ZmStarGen(i);
    native = context.zMStar.SameOrd(i);
    if (!native) {
        g2md = PowerMod(gi, -ord, m); /* g^{-ord} mod m */
        g2mdminv = PrepMulModPrecon(g2md, m);
    }
}
NTL::mulmod_precon_t gminv = PrepMulModPrecon(gi, m);
```

## 8.73 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matching.cpp File Reference

```
#include <helib/matching.h>
#include <queue>
```

## Namespaces

- [helib](#)

## Functions

- void [helib::printFlow](#) (FlowGraph &fg)
- long [helib::maximum\\_flow](#) (FlowGraph &fg, long src, long sink)

## 8.74 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/matmul.cpp File Reference

```
#include <cstdint>
#include <tuple>
#include <algorithm>
#include <NTL/BasicThreadPool.h>
#include <helib/matmul.h>
#include <helib/norms.h>
#include <helib/fhe_stats.h>
#include <helib/apiAttributes.h>
```

## Classes

- class [helib::BasicAutomorphPrecon](#)  
*Pre-computation to speed many automorphism on the same ciphertext.*
- class [helib::GeneralAutomorphPrecon](#)
- class [helib::GeneralAutomorphPrecon\\_UNKNOWN](#)
- class [helib::GeneralAutomorphPrecon\\_FULL](#)
- class [helib::GeneralAutomorphPrecon\\_BSGS](#)
- struct [helib::ConstMultiplier](#)
- struct [helib::ConstMultiplier\\_DoubleCRT](#)
- struct [helib::ConstMultiplier\\_zzX](#)
- struct [helib::MatMul1D\\_derived\\_impl< type >](#)
- struct [helib::MatMul1DExec\\_construct< type >](#)
- struct [helib::BlockMatMul1D\\_derived\\_impl< type >](#)
- struct [helib::BlockMatMul1DExec\\_construct< type >](#)
- class [helib::MatMulFullHelper< type >](#)
- struct [helib::MatMulFullExec\\_construct< type >](#)
- struct [helib::MatMulFullExec\\_construct< type >::MatMulDimComp](#)
- class [helib::BlockMatMulFullHelper< type >](#)
- struct [helib::BlockMatMulFullExec\\_construct< type >](#)
- struct [helib::BlockMatMulFullExec\\_construct< type >::BlockMatMulDimComp](#)
- struct [helib::mul\\_MatMul1D\\_impl< type >](#)
- struct [helib::mul\\_BlockMatMul1D\\_impl< type >](#)
- struct [helib::mul\\_MatMulFull\\_impl< type >](#)
- struct [helib::mul\\_BlockMatMulFull\\_impl< type >](#)

## Namespaces

- [helib](#)

## Macros

- `#define` [ALT\\_MATMUL](#) (1)
- `#define` [HELIB\\_BSGS\\_MUL\\_THRESH](#) [HELIB\\_KEYSWITCH\\_THRESH](#)
- `#define` [HELIB\\_TRACE\\_THRESH](#) (50)

## Functions

- `std::shared_ptr< GeneralAutomorphPrecon >` [helib::buildGeneralAutomorphPrecon](#) (const Ctxt &ctxt, long dim, const EncryptedArray &ea)
- `template<typename RX >`  
`std::shared_ptr< ConstMultiplier >` [helib::build\\_ConstMultiplier](#) (const RX &poly)
- `template<typename RX , typename type >`  
`std::shared_ptr< ConstMultiplier >` [helib::build\\_ConstMultiplier](#) (const RX &poly, long dim, long amt, const EncryptedArrayDerived< type > &ea)
- void [helib::MulAdd](#) (Ctxt &x, const std::shared\_ptr< ConstMultiplier > &a, const Ctxt &b)
- void [helib::DestMulAdd](#) (Ctxt &x, const std::shared\_ptr< ConstMultiplier > &a, Ctxt &b)
- void [helib::GenBabySteps](#) (std::vector< std::shared\_ptr< Ctxt >> &v, const Ctxt &ctxt, long dim, bool clean)
- void [helib::mul](#) (PlaintextArray &pa, const MatMul1D &mat)
- void [helib::mul](#) (PlaintextArray &pa, const BlockMatMul1D &mat)
- void [helib::mul](#) (PlaintextArray &pa, const MatMulFull &mat)
- void [helib::mul](#) (PlaintextArray &pa, const BlockMatMulFull &mat)
- void [helib::traceMap](#) (Ctxt &ctxt)

## 8.74.1 Macro Definition Documentation

### 8.74.1.1 ALT\_MATMUL

```
#define ALT_MATMUL (1)
```

### 8.74.1.2 HELIB\_BSGS\_MUL\_THRESH

```
#define HELIB_BSGS_MUL_THRESH HELIB_KEYSWITCH_THRESH
```

### 8.74.1.3 HELIB\_TRACE\_THRESH

```
#define HELIB_TRACE_THRESH (50)
```

## 8.75 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/norms.cpp File Reference

```
#include <complex>
#include <cmath>
#include <algorithm>
#include <helib/NumbTh.h>
#include <helib/DoubleCRT.h>
#include <helib/norms.h>
#include <helib/PAlgebra.h>
#include <helib/fhe_stats.h>
```

## Namespaces

- [helib](#)

## Macros

- [#define USE\\_HALF\\_FFT](#) (1)
- [#define USE\\_QUARTER\\_FFT](#) (1)
- [#define USE\\_TWO\\_QUARTERS](#) (0)

## Functions

- long [helib::sumOfCoeffs](#) (const zzX &f)
- NTL::ZZ [helib::sumOfCoeffs](#) (const NTL::ZZX &f)
- NTL::ZZ [helib::sumOfCoeffs](#) (const DoubleCRT &f)
- NTL::ZZ [helib::largestCoeff](#) (const NTL::ZZX &f)
- NTL::ZZ [helib::largestCoeff](#) (const NTL::Vec< NTL::ZZ > &f)
- NTL::ZZ [helib::largestCoeff](#) (const DoubleCRT &f)
- double [helib::coeffsL2NormSquared](#) (const zzX &f)  
*The L2-norm of an element (in coefficient representation)*
- NTL::xdouble [helib::coeffsL2NormSquared](#) (const NTL::ZZX &f)
- NTL::xdouble [helib::coeffsL2NormSquared](#) (const DoubleCRT &f)
- double [helib::embeddingLargestCoeff](#) (const std::vector< double > &f, const PAlgebra &palg)
- void [helib::embeddingLargestCoeff\\_x2](#) (double &norm1, double &norm2, const std::vector< double > &f1, const std::vector< double > &f2, const PAlgebra &palg)
- double [helib::embeddingLargestCoeff](#) (const zzX &f, const PAlgebra &palg)
- NTL::xdouble [helib::embeddingLargestCoeff](#) (const NTL::ZZX &f, const PAlgebra &palg)
- void [helib::CKKS\\_canonicalEmbedding](#) (std::vector< cx\_double > &v, const std::vector< double > &f, const PAlgebra &palg)
- void [helib::CKKS\\_canonicalEmbedding](#) (std::vector< cx\_double > &v, const zzX &f, const PAlgebra &palg)
- void [helib::CKKS\\_canonicalEmbedding](#) (std::vector< cx\_double > &v, const NTL::ZZX &f, const PAlgebra &palg)
- void [helib::CKKS\\_embedInSlots](#) (zzX &f, const std::vector< cx\_double > &v, const PAlgebra &palg, double scaling)

### 8.75.1 Detailed Description

- computing various norms of ring elements

### 8.75.2 Macro Definition Documentation

#### 8.75.2.1 USE\_HALF\_FFT

```
#define USE_HALF_FFT (1)
```

#### 8.75.2.2 USE\_QUARTER\_FFT

```
#define USE_QUARTER_FFT (1)
```

#### 8.75.2.3 USE\_TWO\_QUARTERS

```
#define USE_TWO_QUARTERS (0)
```

## 8.76 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/NumbTh.cpp File Reference

```
#include <helib/NumbTh.h>
#include <helib/timing.h>
#include <fstream>
#include <cctype>
#include <algorithm>
```

### Namespaces

- [helib](#)

### Functions

- long [helib::bitSetToLong](#) (long bits, long bitSize)  
*Considers `bits` as a vector of bits and returns the value it represents when interpreted as a  $n$ -bit 2's complement number, where  $n$  is given by `bitSize`.*
- long [helib::mcMod](#) (long a, long b)  
*Routines for computing mathematically correct mod and div.*
- long [helib::mcDiv](#) (long a, long b)
- long [helib::multOrd](#) (long p, long m)  
*Return multiplicative order of  $p$  modulo  $m$ , or 0 if  $\text{GCD}(p, m) \neq 1$ .*
- long [helib::computeProd](#) (const NTL::Vec< long > &vec)  
*returns `\prod_d vec[d]`*
- long [helib::computeProd](#) (const std::vector< long > &vec)
- NTL::ZZX [helib::makeIrredPoly](#) (long p, long d)  
*Return a degree- $d$  irreducible polynomial mod  $p$ .*
- void [helib::factorize](#) (std::vector< long > &factors, long N)  
*Factoring by trial division, only works for  $N < 2^{60}$ , only the primes are recorded, not their multiplicity.*
- void [helib::factorize](#) (std::vector< NTL::ZZ > &factors, const NTL::ZZ &N)
- void [helib::factorize](#) (NTL::Vec< NTL::Pair< long, long > > &factors, long N)  
*Factoring by trial division, only works for  $N < 2^{60}$  primes and multiplicities are recorded.*
- void [helib::pp\\_factorize](#) (std::vector< long > &factors, long N)  
*Prime-power factorization.*
- void [helib::phiN](#) (long &phiN, std::vector< long > &facts, long N)  
*Compute  $\Phi(N)$  and also factorize  $N$ .*
- void [helib::phiN](#) (NTL::ZZ &phiN, std::vector< NTL::ZZ > &facts, const NTL::ZZ &N)
- long [helib::phi\\_N](#) (long N)  
*Compute  $\Phi(N)$ .*
- long [helib::findGenerators](#) (std::vector< long > &gens, std::vector< long > &ords, long m, long p, const std::vector< long > &candidates=std::vector< long >())
- template<typename zp, typename zz >  
void [helib::FindPrimRootT](#) (zp &root, unsigned long e)
- void [helib::FindPrimitiveRoot](#) (NTL::zz\_p &r, unsigned long e)  
*Find  $e$ -th root of unity modulo the current modulus.*
- void [helib::FindPrimitiveRoot](#) (NTL::ZZ\_p &r, unsigned long e)
- long [helib::mobius](#) (long n)

- Compute mobius function (naive method as n is small).*

  - NTL::ZZX [helib::Cyclotomic](#) (long N)

*Compute cyclotomic polynomial.*

  - long [helib::primroot](#) (long N, long phiN)

*Find a primitive root modulo N.*

  - long [helib::ord](#) (long N, long p)

*Compute the highest power of p that divides N.*

  - NTL::ZZX [helib::RandPoly](#) (long n, const NTL::ZZ &p)
  - void [helib::MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q, bool abs)
  - void [helib::balanced\\_MulMod](#) (NTL::ZZX &out, const NTL::ZZX &f, long a, long q)
  - long [helib::is\\_in](#) (long x, int \*X, long sz)

*Finds whether x is an element of the set X of size sz, Returns -1 if not and the location if true.*

  - template<class zzvec >  
bool [helib::intVecCRT](#) (NTL::vec\_ZZ &vp, const NTL::ZZ &p, const zzvec &vq, long q)

*Incremental integer CRT for vectors.*

  - template bool [helib::intVecCRT](#) (NTL::vec\_ZZ &, const NTL::ZZ &, const NTL::vec\_ZZ &, long)
  - template bool [helib::intVecCRT](#) (NTL::vec\_ZZ &, const NTL::ZZ &, const NTL::vec\_long &, long)
  - template bool [helib::intVecCRT](#) (NTL::vec\_ZZ &, const NTL::ZZ &, const NTL::Vec< NTL::zz\_p > &, long)
  - void [helib::ModComp](#) (NTL::ZZX &res, const NTL::ZZX &g, const NTL::ZZX &h, const NTL::ZZX &f)

*Modular composition of polynomials: res = g(h) mod f.*

  - long [helib::polyEvalMod](#) (const NTL::ZZX &poly, long x, long p)

*Evaluates a modular integer polynomial, returns poly(x) mod p.*

  - void [helib::interpolateMod](#) (NTL::ZZX &poly, const NTL::vec\_long &x, const NTL::vec\_long &y, long p, long e=1)

*Interpolate polynomial such that  $\text{poly}(x[i] \bmod p) = y[i] \bmod p^e$ . It is assumed that the points  $x[i]$  are all distinct modulo p.*

  - void [helib::seekPastChar](#) (std::istream &str, int cc)

*Advance the input stream beyond white spaces and a single instance of the char cc.*

  - void [helib::buildLinPolyMatrix](#) (NTL::mat\_zz\_pE &M, long p)
  - void [helib::buildLinPolyMatrix](#) (NTL::mat\_GF2E &M, long p)
  - void [helib::mul](#) (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, long b)
  - void [helib::div](#) (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, long b)
  - void [helib::add](#) (std::vector< NTL::ZZX > &x, const std::vector< NTL::ZZX > &a, const std::vector< NTL::ZZX > &b)
  - void [helib::ppsolve](#) (NTL::vec\_zz\_pE &x, const NTL::mat\_zz\_pE &A, const NTL::vec\_zz\_pE &b, long p, long r)

*Prime power solver.*

  - void [helib::ppsolve](#) (NTL::vec\_GF2E &x, const NTL::mat\_GF2E &A, const NTL::vec\_GF2E &b, long p, long r)

*A version for GF2: must have p == 2 and r == 1.*

  - void [helib::ppInvert](#) (NTL::mat\_zz\_pE &X, const NTL::mat\_zz\_pE &A, long p, long r)
  - void [helib::ppInvert](#) (NTL::mat\_zz\_p &X, const NTL::mat\_zz\_p &A, long p, long r)

*Compute the inverse mod  $p^r$  of an  $n \times n$  matrix.*

  - void [helib::buildLinPolyCoeffs](#) (NTL::vec\_zz\_pE &C, const NTL::vec\_zz\_pE &L, long p, long r)

*Combination of buildLinPolyMatrix and ppsolve.*

  - void [helib::buildLinPolyCoeffs](#) (NTL::vec\_GF2E &C, const NTL::vec\_GF2E &L, long p, long r)

*A version for GF2: must be called with p == 2 and r == 1.*

  - void [helib::applyLinPoly](#) (NTL::zz\_pE &beta, const NTL::vec\_zz\_pE &C, const NTL::zz\_pE &alpha, long p)

*Apply a linearized polynomial with coefficient vector C.*

  - void [helib::applyLinPoly](#) (NTL::GF2E &beta, const NTL::vec\_GF2E &C, const NTL::GF2E &alpha, long p)

*A version for GF2: must be called with p == 2 and r == 1.*

  - std::pair< long, long > [helib::rationalApprox](#) (double x, long denomBound=0)
  - std::pair< NTL::ZZ, NTL::ZZ > [helib::rationalApprox](#) (NTL::xdouble x, NTL::xdouble denomBound=NTL::xdouble(0.0))



- void [helib::rem](#) (NTL::zz\_pX &r, const NTL::zz\_pX &a, const zz\_pXModulus1 &ff)
- void [helib::PolyRed](#) (NTL::ZZX &out, const NTL::ZZX &in, const NTL::ZZ &q, bool abs=false)
- void [helib::PolyRed](#) (NTL::ZZX &out, const NTL::ZZX &in, long q, bool abs=false)  
*Reduce all the coefficients of a polynomial modulo q.*
- void [helib::vecRed](#) (NTL::Vec< NTL::ZZ > &out, const NTL::Vec< NTL::ZZ > &in, long q, bool abs)
- void [helib::vecRed](#) (NTL::Vec< NTL::ZZ > &out, const NTL::Vec< NTL::ZZ > &in, const NTL::ZZ &q, bool abs)

### Some enhanced conversion routines

- void [helib::convert](#) (NTL::vec\_zz\_pE &X, const std::vector< NTL::ZZX > &A)
- void [helib::convert](#) (NTL::mat\_zz\_pE &X, const std::vector< std::vector< NTL::ZZX > > &A)
- void [helib::convert](#) (std::vector< NTL::ZZX > &X, const NTL::vec\_zz\_pE &A)
- void [helib::convert](#) (std::vector< std::vector< NTL::ZZX > > &X, const NTL::mat\_zz\_pE &A)
- void [helib::convert](#) (NTL::Vec< long > &out, const NTL::ZZX &in)
- void [helib::convert](#) (NTL::Vec< long > &out, const NTL::zz\_pX &in, bool symmetric=true)
- void [helib::convert](#) (NTL::Vec< long > &out, const NTL::GF2X &in)
- void [helib::convert](#) (NTL::ZZX &out, const NTL::Vec< long > &in)
- void [helib::convert](#) (NTL::GF2X &out, const NTL::Vec< long > &in)

## 8.77 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/OptimizePermutations.cpp File Reference

```
#include <cstdlib>
#include <list>
#include <sstream>
#include <memory>
#include <NTL/vector.h>
#include <helib/NumbTh.h>
#include <helib/EncryptedArray.h>
#include <helib/permutations.h>
#include <helib/apiAttributes.h>
```

### Namespaces

- [helib](#)

## Functions

- void [helib::removeDups](#) (std::list< long > &x, bool \*aux)
- void [helib::addOffset](#) (std::list< long > &x, long offset, long n, bool \*aux, [UNUSED](#) bool good=false)
- long [helib::reducedCount](#) (const std::list< long > &x, long n, bool \*aux)
- void [helib::buildBenesCostTable](#) (long n, long k, bool good, NTL::Vec< NTL::Vec< long >> &tab)
- std::ostream & [helib::operator<<](#) (std::ostream &s, LongNodePtr p)
- BenesMemoEntry [helib::optimalBenesAux](#) (long i, long budget, long nlev, const NTL::Vec< NTL::Vec< long >> &costTab, BenesMemoTable &memoTab)
- void [helib::optimalBenes](#) (long n, long budget, bool good, long &cost, LongNodePtr &solution)
- void [helib::print](#) (std::ostream &s, SplitNodePtr p, bool first)
- std::ostream & [helib::operator<<](#) (std::ostream &s, SplitNodePtr p)
- long [helib::length](#) (GenNodePtr ptr)
- std::ostream & [helib::operator<<](#) (std::ostream &s, GenNodePtr p)
- LowerMemoEntry [helib::optimalLower](#) (long order, bool good, long budget, long mid, LowerMemoTable &lowerMemoTable)
- UpperMemoEntry [helib::optimalUpperAux](#) (const NTL::Vec< GenDescriptor > &vec, long i, long budget, long mid, UpperMemoTable &upperMemoTable, LowerMemoTable &lowerMemoTable)

## 8.78 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/PAlgebra.cpp File Reference

```
#include <algorithm>
#include <cmath>
#include <helib/PAlgebra.h>
#include <helib/hypercube.h>
#include <helib/timing.h>
#include <NTL/ZZXFactoring.h>
#include <NTL/GF2EXFactoring.h>
#include <NTL/lzz_pEXFactoring.h>
#include <NTL/BasicThreadPool.h>
#include <mutex>
```

## Namespaces

- [helib](#)

## Functions

- template<typename RX >  
bool [helib::poly\\_comp](#) (const RX &a, const RX &b)
- bool [helib::less\\_than](#) (NTL::GF2 a, NTL::GF2 b)
- bool [helib::less\\_than](#) (NTL::zz\_p a, NTL::zz\_p b)
- bool [helib::less\\_than](#) (const NTL::GF2X &a, const NTL::GF2X &b)
- bool [helib::less\\_than](#) (const NTL::zz\_pX &a, const NTL::zz\_pX &b)
- bool [helib::less\\_than](#) (const NTL::GF2E &a, const NTL::GF2E &b)
- bool [helib::less\\_than](#) (const NTL::zz\_pE &a, const NTL::zz\_pE &b)
- bool [helib::less\\_than](#) (const NTL::GF2EX &a, const NTL::GF2EX &b)
- bool [helib::less\\_than](#) (const NTL::zz\_pEX &a, const NTL::zz\_pEX &b)

- double [helib::calcPolyNormBnd](#) (long m)
- bool [helib::comparePAlgebra](#) (const PAlgebra &palg, unsigned long m, unsigned long p, [UNUSED](#) unsigned long r, const std::vector< long > &gens, const std::vector< long > &ords)
- PAlgebraModBase \* [helib::buildPAlgebraMod](#) (const PAlgebra &zMStar, long r)  
*Builds a table, of type PA\_GF2 if p == 2 and r == 1, and PA\_zz\_p otherwise.*
- template<typename T>  
void [helib::PAlgebraLift](#) (const NTL::ZZX &phimx, const T &lfactors, T &factors, T &crtc, long r)
- void [helib::EDF](#) (NTL::vec\_zz\_pX &v, const NTL::zz\_pX &f, long d)
- NTL::zz\_pEX [helib::FrobeniusMap](#) (const NTL::zz\_pEXModulus &F)
- void [helib::InvModpr](#) (NTL::zz\_pX &S, const NTL::zz\_pX &F, const NTL::zz\_pX &G, long p, long r)
- template<> void [helib::PAlgebraLift](#) (const NTL::ZZX &phimx, const NTL::vec\_zz\_pX &lfactors, NTL::vec\_zz\_pX &factors, NTL::vec\_zz\_pX &crtc, long r)

## 8.79 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/PermNetwork.cpp File Reference

```
#include <NTL/ZZ.h>
#include <helib/Context.h>
#include <helib/Ctxt.h>
#include <helib/permutations.h>
#include <helib/EncryptedArray.h>
```

### Namespaces

- [helib](#)

### Functions

- std::ostream & [helib::operator<<](#) (std::ostream &s, const PermNetwork &net)

## 8.80 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/permutations.cpp File Reference

```
#include <helib/permutations.h>
```

### Namespaces

- [helib](#)

## Functions

- `template<typename T >`  
`void helib::applyPermToVec` (NTL::Vec< T > &out, const NTL::Vec< T > &in, const Permut &p1)  
*Apply a permutation to a std::vector, out[i]=in[p1[i]] (NOT in-place)*
- `template<typename T >`  
`void helib::applyPermToVec` (std::vector< T > &out, const std::vector< T > &in, const Permut &p1)
- `template<typename T >`  
`void helib::applyPermsToVec` (NTL::Vec< T > &out, const NTL::Vec< T > &in, const Permut &p2, const Permut &p1)  
*Apply two permutations to a std::vector out[i]=in[p2[p1[i]]] (NOT in-place)*
- `template<typename T >`  
`void helib::applyPermsToVec` (std::vector< T > &out, const std::vector< T > &in, const Permut &p2, const Permut &p1)
- `template void helib::applyPermToVec< long >` (NTL::Vec< long > &out, const NTL::Vec< long > &in, const Permut &p1)
- `template void helib::applyPermToVec< long >` (std::vector< long > &out, const std::vector< long > &in, const Permut &p1)
- `template void helib::applyPermToVec< NTL::ZZX >` (std::vector< NTL::ZZX > &out, const std::vector< NTL::ZZX > &in, const Permut &p1)
- `template void helib::applyPermsToVec< long >` (NTL::Vec< long > &out, const NTL::Vec< long > &in, const Permut &p2, const Permut &p1)
- `template void helib::applyPermsToVec< long >` (std::vector< long > &out, const std::vector< long > &in, const Permut &p2, const Permut &p1)
- `void helib::randomPerm` (Permut &perm, long n)  
*A random size-n permutation.*
- `void helib::breakPermTo3` (const HyperCube< long > &pi, long dim, ColPerm &rho1, HyperCube< long > &rho2, ColPerm &rho3)
- `void helib::breakPermByDim` (std::vector< ColPerm > &out, const Permut &pi, const CubeSignature &sig)  
*Takes a permutation pi over m-dimensional cube  $C = \mathbb{Z}_{\{n1\}} \times \dots \times \mathbb{Z}_{\{nm\}}$  and expresses pi as a product  $pi = \rho_{i \leftarrow \{2m-1\}} \circ \dots \circ \rho_{i \leftarrow 2} \circ \rho_{i \leftarrow 1}$  where each  $\rho_{i \leftarrow j}$  is a column permutation along one dimension. Specifically for  $i < m$ , the permutations  $\rho_{i \leftarrow j}$  and  $\rho_{i \leftarrow \{2(m-1)-j\}}$  permute the i'th dimension.*
- `void helib::ComputeOneGenMapping` (Permut &genMap, const OneGeneratorTree &T)  
*to a single generator tree*
- `std::ostream & helib::operator<<` (std::ostream &s, const ColPerm &p)
- `std::ostream & helib::operator<<` (std::ostream &s, const SubDimension &sd)
- `std::ostream & helib::operator<<` (std::ostream &s, const GeneratorTrees &trees)

## 8.81 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/PGFFT.cpp File Reference

```
#include <helib/PGFFT.h>
#include <cassert>
#include <cstdlib>
#include <limits>
```

## Namespaces

- [helib](#)

## Macros

- `#define PGFFT_USE_TRUNCATED_BLUE` (1)
- `#define PGFFT_USE_EXPLICIT_MUL` (1)
- `#define RESTRICT`
- `#define PGFFT_FFT_RDUP` (4)
- `#define fwd_butterfly(xx0, xx1, w)`
- `#define fwd_butterfly0(xx0, xx1)`
- `#define inv_butterfly0(xx0, xx1)`
- `#define inv_butterfly(xx0, xx1, w)`
- `#define PGFFT_NEW_FFT_THRESH` (10)
- `#define PGFFT_BRC_THRESH` (11)
- `#define PGFFT_BRC_Q` (5)
- `#define PGFFT_STRATEGY_NULL` (0)
- `#define PGFFT_STRATEGY_POW2` (1)
- `#define PGFFT_STRATEGY_BLUE` (2)
- `#define PGFFT_STRATEGY_TBLUE` (3)

## Typedefs

- `template<class T >`  
  using `helib::aligned_vector` = `PGFFT::aligned_vector< T >`
- `typedef complex< double > helib::cmplx_t`
- `typedef long double helib::ldbl`

### 8.81.1 Macro Definition Documentation

#### 8.81.1.1 fwd\_butterfly

```
#define fwd_butterfly(  
    xx0,  
    xx1,  
    w )
```

##### Value:

```
do \  
{ \  
    cmplx_t x0_ = xx0; \  
    cmplx_t x1_ = xx1; \  
    cmplx_t t_ = x0_ - x1_; \  
    xx0 = x0_ + x1_; \  
    xx1 = MUL(t_, w); \  
} \  
while (0)
```

### 8.81.1.2 fwd\_butterfly0

```
#define fwd_butterfly0(
    xx0,
    xx1 )
```

#### Value:

```
do \
{ \
    cmplx_t x0_ = xx0; \
    cmplx_t x1_ = xx1; \
    xx0 = x0_ + x1_; \
    xx1 = x0_ - x1_; \
} \
while (0)
```

### 8.81.1.3 inv\_butterfly

```
#define inv_butterfly(
    xx0,
    xx1,
    w )
```

#### Value:

```
do \
{ \
    cmplx_t x0_ = xx0; \
    cmplx_t x1_ = xx1; \
    cmplx_t t_ = CMUL(x1_, w); \
    xx0 = x0_ + t_; \
    xx1 = x0_ - t_; \
} while (0)
```

### 8.81.1.4 inv\_butterfly0

```
#define inv_butterfly0(
    xx0,
    xx1 )
```

#### Value:

```
do \
{ \
    cmplx_t x0_ = xx0; \
    cmplx_t x1_ = xx1; \
    xx0 = x0_ + x1_; \
    xx1 = x0_ - x1_; \
} while (0)
```

### 8.81.1.5 PGFFT\_BRC\_Q

```
#define PGFFT_BRC_Q (5)
```

#### 8.81.1.6 PGFFT\_BRC\_THRESH

```
#define PGFFT_BRC_THRESH (11)
```

#### 8.81.1.7 PGFFT\_FFT\_RDUP

```
#define PGFFT_FFT_RDUP (4)
```

#### 8.81.1.8 PGFFT\_NEW\_FFT\_THRESH

```
#define PGFFT_NEW_FFT_THRESH (10)
```

#### 8.81.1.9 PGFFT\_STRATEGY\_BLUE

```
#define PGFFT_STRATEGY_BLUE (2)
```

#### 8.81.1.10 PGFFT\_STRATEGY\_NULL

```
#define PGFFT_STRATEGY_NULL (0)
```

#### 8.81.1.11 PGFFT\_STRATEGY\_POW2

```
#define PGFFT_STRATEGY_POW2 (1)
```

#### 8.81.1.12 PGFFT\_STRATEGY\_TBLUE

```
#define PGFFT_STRATEGY_TBLUE (3)
```

#### 8.81.1.13 PGFFT\_USE\_EXPLICIT\_MUL

```
#define PGFFT_USE_EXPLICIT_MUL (1)
```

#### 8.81.1.14 PGFFT\_USE\_TRUNCATED\_BLUE

```
#define PGFFT_USE_TRUNCATED_BLUE (1)
```

#### 8.81.1.15 RESTRICT

```
#define RESTRICT
```

### 8.82 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/polyEval.cpp File Reference

```
#include <helib/Context.h>
#include <helib/polyEval.h>
```

#### Namespaces

- [helib](#)

#### Functions

- void [helib::polyEval](#) (Ctxt &ret, const NTL::Vec< Ctxt > &poly, const Ctxt &x)  
*Evaluate an encrypted polynomial on an encrypted input.*
- void [helib::polyEval](#) (Ctxt &ret, NTL::ZZX poly, const Ctxt &x, long k=0)  
*Evaluate a cleartext polynomial on an encrypted input.*

### 8.83 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/PolyMod.cpp File Reference

```
#include <helib/PolyMod.h>
#include <helib/exceptions.h>
#include <NTL/ZZX.h>
#include <NTL/ZZ_p.h>
#include <vector>
```

#### Namespaces

- [helib](#)



## Functions

- `std::istream & helib::operator>>` (`std::istream &is`, `PolyMod &poly`)
- `std::ostream & helib::operator<<` (`std::ostream &os`, `const PolyMod &poly`)

## 8.84 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/PolyModRing.cpp File Reference

```
#include <helib/PolyModRing.h>
```

## Namespaces

- [helib](#)

## Functions

- `std::ostream & helib::operator<<` (`std::ostream &os`, `const PolyModRing &ring`)

## 8.85 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/powerful.cpp File Reference

```
#include <helib/powerful.h>
```

## Namespaces

- [helib](#)

## Functions

- `void helib::computeDivVec` (`NTL::Vec< long > &divVec`, `long m`, `const NTL::Vec< long > &powVec`)
- `void helib::computeInvVec` (`NTL::Vec< long > &invVec`, `const NTL::Vec< long > &divVec`, `const NTL::Vec< long > &powVec`)

## 8.86 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_ v1.0.2/HElib/src/primeChain.cpp File Reference

handling the chain of moduli

```
#include <climits>
#include <cmath>
#include <algorithm>
#include <helib/primeChain.h>
#include <helib/Context.h>
#include <helib/sample.h>
#include <helib/binio.h>
```

### Classes

- struct [helib::PrimeGenerator](#)

### Namespaces

- [helib](#)

### Functions

- bool [helib::operator>](#) (const ModuliSizes::Entry &a, const ModuliSizes::Entry &b)
- std::ostream & [helib::operator<<](#) (std::ostream &s, const ModuliSizes::Entry &e)
- std::istream & [helib::operator>>](#) (std::istream &s, ModuliSizes::Entry &e)
- void [helib::write](#) (std::ostream &s, const ModuliSizes::Entry &e)
- void [helib::read](#) (std::istream &s, ModuliSizes::Entry &e)
- std::ostream & [helib::operator<<](#) (std::ostream &s, const ModuliSizes &szs)
- std::istream & [helib::operator>>](#) (std::istream &s, ModuliSizes &szs)
- void [helib::addSmallPrimes](#) (Context &context, long resolution, long cpSize)  
*Add small primes to get target resolution.*
- long [helib::ctxtPrimeSize](#) (long nBits)
- void [helib::addCtxtPrimes](#) (Context &context, long nBits, long targetSize)
- void [helib::endBuildModChain](#) (Context &context)
- void [helib::buildModChain](#) (Context &context, long nBits, long nDgts=3, bool willBeBootstrappable=false, long skHwt=0, long resolution=3, long bitsInSpecialPrimes=0)

#### 8.86.1 Detailed Description

handling the chain of moduli

## 8.87 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Ptxt.cpp File Reference

```
#include <random>
#include <helib/Ptxt.h>
#include <helib/apiAttributes.h>
```

### Namespaces

- [helib](#)

### Functions

- `template<typename Scheme >`  
`Scheme::SlotType helib::randomSlot (const Context &context)`
- `template<> BGV::SlotType helib::randomSlot< BGV > (const Context &context)`
- `template<> CKKS::SlotType helib::randomSlot< CKKS > (UNUSED const Context &context)`

## 8.88 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/randomMatrices.cpp File Reference

implementation of random matrices of various forms build functions, used for testing

```
#include <helib/randomMatrices.h>
```

### Namespaces

- [helib](#)

### Functions

- `MatMul1D * helib::buildRandomMatrix (const EncryptedArray &ea, long dim)`
- `MatMul1D * helib::buildRandomMultiMatrix (const EncryptedArray &ea, long dim)`
- `BlockMatMul1D * helib::buildRandomBlockMatrix (const EncryptedArray &ea, long dim)`
- `BlockMatMul1D * helib::buildRandomMultiBlockMatrix (const EncryptedArray &ea, long dim)`
- `MatMulFull * helib::buildRandomFullMatrix (const EncryptedArray &ea)`
- `BlockMatMulFull * helib::buildRandomFullBlockMatrix (const EncryptedArray &ea)`

### 8.88.1 Detailed Description

implementation of random matrices of various forms build functions, used for testing

## 8.89 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/recryption.cpp File Reference

```
#include <NTL/BasicThreadPool.h>
#include <helib/recryption.h>
#include <helib/EncryptedArray.h>
#include <helib/EvalMap.h>
#include <helib/powerful.h>
#include <helib/CtPtrs.h>
#include <helib/intraSlot.h>
#include <helib/norms.h>
#include <helib/sample.h>
#include <helib/debugging.h>
#include <helib/fhe_stats.h>
```

### Classes

- struct [helib::PubKeyHack](#)

### Namespaces

- [helib](#)

### Macros

- #define [DROP\\_BEFORE\\_THIN\\_RECRYPT](#)
- #define [THIN\\_RECRYPT\\_NLEVELS](#) (3)

### Functions

- void [helib::extractDigitsPacked](#) (Ctxt &ctxt, long botHigh, long r, long ePrime, const std::vector< NTL::ZZX > &unpackSlotEncoding)
- void [helib::extractDigitsThin](#) (Ctxt &ctxt, long botHigh, long r, long ePrime)
- void [helib::packedRecrypt](#) (const CtPtrs &cPtrs, const std::vector< zzX > &unpackConsts, const Encrypted↵  
Array &ea)
- void [helib::packedRecrypt](#) (const CtPtrs &array, const std::vector< zzX > &unpackConsts, const Encrypted↵  
Array &ea, long belowLvl)
- void [helib::packedRecrypt](#) (const CtPtrMat &m, const std::vector< zzX > &unpackConsts, const Encrypted↵  
Array &ea, long belowLvl=LONG\_MAX)

#### 8.89.1 Macro Definition Documentation

### 8.89.1.1 DROP\_BEFORE\_THIN\_RECRYPT

```
#define DROP_BEFORE_THIN_RECRYPT
```

### 8.89.1.2 THIN\_RECRYPT\_NLEVELS

```
#define THIN_RECRYPT_NLEVELS (3)
```

## 8.90 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/replicate.cpp File Reference

```
#include <helib/replicate.h>
#include <helib/timing.h>
#include <helib/clonedPtr.h>
```

### Classes

- class [helib::ExplicitReplicator](#)  
An implementation of [ReplicateHandler](#) that explicitly returns all the replicated ciphertexts in one big vector.
- class [helib::replicate\\_pa\\_impl< type >](#)

### Namespaces

- [helib](#)

### Functions

- void [helib::replicate](#) (const EncryptedArray &ea, Ctxt &ctx, long pos)  
The value in slot #pos is replicated in all other slots. On an n-slot ciphertext, this algorithm performs  $O(\log n)$  1D rotations.
- void [helib::replicate0](#) (const EncryptedArray &ea, Ctxt &ctx, long pos)  
A lower-level routine. Same as replicate, but assumes all slots are zero except slot #pos.
- void [helib::replicateAllOrig](#) (const EncryptedArray &ea, const Ctxt &ctx, ReplicateHandler \*handler, RepAux \*repAuxPtr=nullptr)
- void [helib::replicateAll](#) (const EncryptedArray &ea, const Ctxt &ctx, ReplicateHandler \*handler, long rec↔ Bound=64, RepAuxDim \*repAuxPtr=nullptr)
- void [helib::replicateAll](#) (std::vector< Ctxt > &v, const EncryptedArray &ea, const Ctxt &ctx, long rec↔ Bound=64, RepAuxDim \*repAuxPtr=nullptr)
- void [helib::replicate](#) (const EncryptedArray &ea, PlaintextArray &pa, long i)

## 8.91 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/sample.cpp File Reference

```
#include <vector>
#include <NTL/ZZX.h>
#include <NTL/ZZ_pX.h>
#include <NTL/BasicThreadPool.h>
#include <helib/NumbTh.h>
#include <helib/Context.h>
#include <helib/sample.h>
#include <helib/norms.h>
#include <helib/apiAttributes.h>
#include <helib/powerful.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::sampleHWt](#) (zzX &poly, long n, long Hwt=100)  
*Sample a degree-(n-1) poly as above, with only Hwt nonzero coefficients.*
- void [helib::sampleHWt](#) (NTL::ZZX &poly, long n, long Hwt=100)
- void [helib::sampleSmall](#) (zzX &poly, long n, double prob=0.5)
- void [helib::sampleSmall](#) (NTL::ZZX &poly, long n, double prob=0.5)
- void [helib::sampleGaussian](#) (std::vector< double > &dvec, long n, double stdev)  
*Choose a vector of continuous Gaussians.*
- void [helib::sampleGaussian](#) (zzX &poly, long n, double stdev)  
*Sample polynomials with Gaussian coefficients.*
- void [helib::sampleGaussian](#) (NTL::ZZX &poly, long n, double stdev)
- void [helib::sampleUniform](#) (zzX &poly, long n, long B=100)  
*Sample a degree-(n-1) ZZX, with coefficients uniform in [-B,B].*
- void [helib::sampleUniform](#) (NTL::ZZX &poly, long n, const NTL::ZZ &B=NTL::ZZ(100L))
- double [helib::sampleHWt](#) (zzX &poly, const Context &context, long Hwt=100)
- double [helib::sampleHWtBoundedEffectiveBound](#) (const Context &context, long Hwt=100)
- double [helib::sampleHWtBounded](#) (zzX &poly, const Context &context, long Hwt=100)
- double [helib::sampleSmall](#) (zzX &poly, const Context &context)
- double [helib::sampleSmallBounded](#) (zzX &poly, const Context &context)
- double [helib::sampleGaussian](#) (zzX &poly, const Context &context, double stdev)
- double [helib::sampleGaussianBounded](#) (zzX &poly, const Context &context, double stdev)
- double [helib::sampleUniform](#) (zzX &poly, const Context &context, long B=100)
- NTL::xdouble [helib::sampleUniform](#) (NTL::ZZX &poly, const Context &context, const NTL::ZZ &B=NTL::ZZ(100L))↵
- double [helib::boundRoundingNoise](#) (UNUSED long m, long phim, long p2r, double epsilon)
- double [helib::boundFreshNoise](#) (long m, long phim, double sigma, double epsilon=9e-13)

## 8.92 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/tableLookup.cpp File Reference

Code for homomorphic table lookup and fixed-point functions.

```
#include <limits>
#include <cmath>
#include <cstdlib>
#include <stdexcept>
#include <NTL/BasicThreadPool.h>
#include <helib/intraSlot.h>
#include <helib/tableLookup.h>
```

### Namespaces

- [helib](#)

### Functions

- void [helib::computeAllProducts](#) (CtPtrs &products, const CtPtrs &array, std::vector< zzX > \*unpackSlot, Encoding=nullptr)
- void [helib::tableLookup](#) (Ctxt &out, const std::vector< zzX > &table, const CtPtrs &idx, std::vector< zzX > \*unpackSlot, Encoding=nullptr)
- void [helib::tableWriteIn](#) (const CtPtrs &table, const CtPtrs &idx, std::vector< zzX > \*unpackSlot, Encoding=nullptr)
- void [helib::buildLookupTable](#) (std::vector< zzX > &T, std::function< double(double)> f, long nbits\_in, long scale\_in, long sign\_in, long nbits\_out, long scale\_out, long sign\_out, const EncryptedArray &ea)

*Built a table-lookup for a function in fixed-point representation.*

#### 8.92.1 Detailed Description

Code for homomorphic table lookup and fixed-point functions.

## 8.93 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_approxNums.cpp File Reference

```
#include <NTL/ZZ.h>
#include <algorithm>
#include <complex>
#include <helib/norms.h>
#include <helib/helib.h>
#include <helib/debugging.h>
#include <helib/ArgMap.h>
```

## Functions

- double `calcMaxDiff` (const vector< `cx_double` > &v1, const vector< `cx_double` > &v2)
- double `calcMaxRelDiff` (const vector< `cx_double` > &v1, const vector< `cx_double` > &v2)
- bool `cx_equals` (const vector< `cx_double` > &v1, const vector< `cx_double` > &v2, double epsilon)
- void `testBasicArith` (const `PubKey` &publicKey, const `SecKey` &secretKey, const `EncryptedArrayCx` &ea, double epsilon)
- void `testComplexArith` (const `PubKey` &publicKey, const `SecKey` &secretKey, const `EncryptedArrayCx` &ea, double epsilon)
- void `testRotsNShifts` (const `PubKey` &publicKey, const `SecKey` &secretKey, const `EncryptedArrayCx` &ea, double epsilon)
- void `debugCompare` (const `EncryptedArrayCx` &ea, const `SecKey` &sk, vector< `cx_double` > &p, const `Ctxt` &c, double epsilon)
- void `negateVec` (vector< `cx_double` > &p1)
- void `add` (vector< `cx_double` > &to, const vector< `cx_double` > &from)
- void `sub` (vector< `cx_double` > &to, const vector< `cx_double` > &from)
- void `mul` (vector< `cx_double` > &to, const vector< `cx_double` > &from)
- void `rotate` (vector< `cx_double` > &p, long amt)
- void `testGeneralOps` (const `PubKey` &publicKey, const `SecKey` &secretKey, const `EncryptedArrayCx` &ea, double epsilon, long nRounds)
- int `main` (int argc, char \*argv[])

## Variables

- bool `verbose` =false

### 8.93.1 Function Documentation

#### 8.93.1.1 `add()`

```
void add (
    vector< cx_double > & to,
    const vector< cx_double > & from )
```

#### 8.93.1.2 `calcMaxDiff()`

```
double calcMaxDiff (
    const vector< cx_double > & v1,
    const vector< cx_double > & v2 )
```



#### 8.93.1.3 calcMaxRelDiff()

```
double calcMaxRelDiff (
    const vector< cx_double > & v1,
    const vector< cx_double > & v2 )
```

#### 8.93.1.4 cx\_equals()

```
bool cx_equals (
    const vector< cx_double > & v1,
    const vector< cx_double > & v2,
    double epsilon ) [inline]
```

#### 8.93.1.5 debugCompare()

```
void debugCompare (
    const EncryptedArrayCx & ea,
    const SecKey & sk,
    vector< cx_double > & p,
    const Ctxt & c,
    double epsilon )
```

#### 8.93.1.6 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.93.1.7 mul()

```
void mul (
    vector< cx_double > & to,
    const vector< cx_double > & from )
```

#### 8.93.1.8 negateVec()

```
void negateVec (
    vector< cx_double > & p1 )
```

#### 8.93.1.9 rotate()

```
void rotate (
    vector< cx_double > & p,
    long amt )
```

#### 8.93.1.10 sub()

```
void sub (
    vector< cx_double > & to,
    const vector< cx_double > & from )
```

#### 8.93.1.11 testBasicArith()

```
void testBasicArith (
    const PubKey & publicKey,
    const SecKey & secretKey,
    const EncryptedArrayCx & ea,
    double epsilon )
```

#### 8.93.1.12 testComplexArith()

```
void testComplexArith (
    const PubKey & publicKey,
    const SecKey & secretKey,
    const EncryptedArrayCx & ea,
    double epsilon )
```

#### 8.93.1.13 testGeneralOps()

```
void testGeneralOps (
    const PubKey & publicKey,
    const SecKey & secretKey,
    const EncryptedArrayCx & ea,
    double epsilon,
    long nRounds )
```

### 8.93.1.14 testRotsNShifts()

```
void testRotsNShifts (
    const PubKey & publicKey,
    const SecKey & secretKey,
    const EncryptedArrayCx & ea,
    double epsilon )
```

## 8.93.2 Variable Documentation

### 8.93.2.1 verbose

```
bool verbose =false
```

## 8.94 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_Bin\_IO.cpp File Reference

```
#include <cassert>
#include <cstring>
#include <fstream>
#include <utility>
#include <unistd.h>
#include <NTL/ZZX.h>
#include <NTL/vector.h>
#include <helib/helib.h>
#include <helib/ArgMap.h>
#include <helib/debugging.h>
```

## Functions

- bool [isLittleEndian](#) ()
- void [cleanupFiles](#) (const char \*file)
- template<class... Files>  
void [cleanupFiles](#) (const char \*file, Files... files)
- long [compareFiles](#) (string filename1, string filename2)
- int [main](#) (int argc, char \*argv[ ])

### 8.94.1 Function Documentation

**8.94.1.1 cleanupFiles() [1/2]**

```
void cleanupFiles (
    const char * file )
```

**8.94.1.2 cleanupFiles() [2/2]**

```
template<class... Files>
void cleanupFiles (
    const char * file,
    Files... files )
```

**8.94.1.3 compareFiles()**

```
long compareFiles (
    string filename1,
    string filename2 )
```

**8.94.1.4 isLittleEndian()**

```
bool isLittleEndian ( )
```

**8.94.1.5 main()**

```
int main (
    int argc,
    char * argv[] )
```

**8.95 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵  
v1.0.2/HElib/src/Test\_binaryArith.cpp File  
Reference**

```
#include <iostream>
#include <cassert>
#include <fstream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <NTL/BasicThreadPool.h>
#include <helib/helib.h>
#include <helib/intraSlot.h>
#include <helib/binaryArith.h>
#include <helib/ArgMap.h>
```

## Functions

- void [test15for4](#) ([SecKey](#) &secKey)
- void [testProduct](#) ([SecKey](#) &secKey, long bitSize1, long bitSize2, long outSize, bool bootstrap=false)
- void [testAdd](#) ([SecKey](#) &secKey, long bitSize1, long bitSize2, long outSize, bool bootstrap=false)
- int [main](#) (int argc, char \*argv[])

### 8.95.1 Function Documentation

#### 8.95.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.95.1.2 test15for4()

```
void test15for4 (
    SecKey & secKey )
```

#### 8.95.1.3 testAdd()

```
void testAdd (
    SecKey & secKey,
    long bitSize1,
    long bitSize2,
    long outSize,
    bool bootstrap = false )
```

#### 8.95.1.4 testProduct()

```
void testProduct (
    SecKey & secKey,
    long bitSize1,
    long bitSize2,
    long outSize,
    bool bootstrap = false )
```

## 8.96 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_ v1.0.2/HElib/src/Test\_binaryCompare.cpp File Reference

```
#include <iostream>
#include <cassert>
#include <fstream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <NTL/BasicThreadPool.h>
#include <helib/helib.h>
#include <helib/intraSlot.h>
#include <helib/binaryArith.h>
#include <helib/binaryCompare.h>
#include <helib/ArgMap.h>
```

### Functions

- void [testCompare](#) ([SecKey](#) &secKey, long bitSize, bool bootstrap=false)
- int [main](#) (int argc, char \*argv[])

### 8.96.1 Function Documentation

#### 8.96.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.96.1.2 testCompare()

```
void testCompare (
    SecKey & secKey,
    long bitSize,
    bool bootstrap = false )
```

## 8.97 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_bootstrapping.cpp File Reference

```
#include <NTL/ZZ.h>
#include <NTL/fileio.h>
#include <NTL/BasicThreadPool.h>
#include <cassert>
#include <helib/EncryptedArray.h>
#include <helib/EvalMap.h>
#include <helib/powerful.h>
#include <helib/matmul.h>
#include <helib/ArgMap.h>
#include <helib/debugging.h>
```

### Macros

- #define `num_mValues` (sizeof(mValues)/(14\*sizeof(long)))
- #define `OUTER_REP` (1)
- #define `INNER_REP` (1)

### Functions

- void `TestIt` (long idx, long p, long r, long L, long c, long skHwt, int build\_cache=0)
- int `main` (int argc, char \*argv[])

### 8.97.1 Macro Definition Documentation

#### 8.97.1.1 INNER\_REP

```
#define INNER_REP (1)
```

#### 8.97.1.2 num\_mValues

```
#define num_mValues (sizeof(mValues)/(14*sizeof(long)))
```

#### 8.97.1.3 OUTER\_REP

```
#define OUTER_REP (1)
```

## 8.97.2 Function Documentation

### 8.97.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

### 8.97.2.2 TestIt()

```
void TestIt (
    long idx,
    long p,
    long r,
    long L,
    long c,
    long skHwt,
    int build_cache = 0 )
```

## 8.98 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_EaCx.cpp File Reference

```
#include <NTL/ZZ.h>
#include <cassert>
#include <helib/norms.h>
#include <helib/EncryptedArray.h>
#include <helib/ArgMap.h>
```

### Functions

- int [main](#) (int argc, char \*argv[])

### Variables

- bool [noPrint](#) = true

## 8.98.1 Function Documentation



### 8.98.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

## 8.98.2 Variable Documentation

### 8.98.2.1 noPrint

```
bool noPrint = true
```

## 8.99 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_EvalMap.cpp File Reference

```
#include <NTL/BasicThreadPool.h>
#include <cassert>
#include <helib/EvalMap.h>
#include <helib/hypercube.h>
#include <helib/powerful.h>
#include <helib/ArgMap.h>
```

## Namespaces

- [std](#)
- [NTL](#)

## Functions

- void [TestIt](#) (long p, long r, long c, long \_k, long L, Vec< long > &mvec, Vec< long > &gens, Vec< long > &ords, long useCache)
- int [main](#) (int argc, char \*argv[])

### 8.99.1 Function Documentation

### 8.99.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

### 8.99.1.2 TestIt()

```
void TestIt (
    long p,
    long r,
    long c,
    long _k,
    long L,
    Vec< long > & mvec,
    Vec< long > & gens,
    Vec< long > & ords,
    long useCache )
```

## 8.100 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_extractDigits.cpp File Reference

```
#include <NTL/ZZ.h>
#include <helib/EncryptedArray.h>
#include <helib/polyEval.h>
#include <helib/debugging.h>
#include <helib/ArgMap.h>
```

## Functions

- int [main](#) (int argc, char \*argv[])

### 8.100.1 Function Documentation

#### 8.100.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

## 8.101 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_fatboot.cpp File Reference

```
#include <NTL/ZZ.h>
#include <NTL/fileio.h>
#include <NTL/BasicThreadPool.h>
#include <cassert>
#include <helib/EncryptedArray.h>
#include <helib/EvalMap.h>
#include <helib/powerful.h>
#include <helib/matmul.h>
#include <helib/debugging.h>
#include <helib/fhe_stats.h>
#include <helib/ArgMap.h>
```

### Macros

- `#define OUTER_REP` (1)
- `#define INNER_REP` (1)

### Functions

- void `TestIt` (long p, long r, long L, long c, long skHwt, int build\_cache=0)
- int `main` (int argc, char \*argv[])

### Variables

- long `printFlag`

#### 8.101.1 Macro Definition Documentation

##### 8.101.1.1 INNER\_REP

```
#define INNER_REP (1)
```

##### 8.101.1.2 OUTER\_REP

```
#define OUTER_REP (1)
```

## 8.101.2 Function Documentation

### 8.101.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

### 8.101.2.2 TestIt()

```
void TestIt (
    long p,
    long r,
    long L,
    long c,
    long skHwt,
    int build_cache = 0 )
```

## 8.101.3 Variable Documentation

### 8.101.3.1 printFlag

```
long printFlag
```

## 8.102 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_General.cpp File Reference

```
#include <NTL/ZZ.h>
#include <NTL/BasicThreadPool.h>
#include <helib/helib.h>
#include <NTL/lzz_pXFactoring.h>
#include <cassert>
#include <cstdio>
#include <helib/ArgMap.h>
#include <helib/fhe_stats.h>
```

## Macros

- #define [debugCompare](#)(ea, sk, p, c)

## Functions

- void [TestIt](#) (long R, long p, long r, long d, long c, long k, long w, long L, long m, const Vec< long > &gens, const Vec< long > &ords)
- int [main](#) (int argc, char \*\*argv)

### 8.102.1 Macro Definition Documentation

#### 8.102.1.1 debugCompare

```
#define debugCompare(  
    ea,  
    sk,  
    p,  
    c )
```

### 8.102.2 Function Documentation

#### 8.102.2.1 main()

```
int main (  
    int argc,  
    char ** argv )
```

#### 8.102.2.2 TestIt()

```
void TestIt (  
    long R,  
    long p,  
    long r,  
    long d,  
    long c,  
    long k,  
    long w,  
    long L,  
    long m,  
    const Vec< long > & gens,  
    const Vec< long > & ords )
```

## 8.103 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_intraSlot.cpp File Reference

```
#include <helib/intraSlot.h>
#include <helib/ArgMap.h>
```

### Functions

- int [main](#) (int argc, char \*\*argv)

### Variables

- bool [verbose](#) = false

## 8.103.1 Function Documentation

### 8.103.1.1 main()

```
int main (
    int argc,
    char ** argv )
```

## 8.103.2 Variable Documentation

### 8.103.2.1 verbose

```
bool verbose = false
```

## 8.104 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_IO.cpp File Reference

```
#include <cassert>
#include <fstream>
#include <unistd.h>
#include <NTL/ZZX.h>
#include <NTL/vector.h>
#include <helib/helib.h>
#include <helib/ArgMap.h>
```

## Macros

- #define `N_TESTS` 3

## Functions

- void `checkCiphertext` (const `Ctxt` &ctxt, const `ZZX` &ptxt, const `SecKey` &sk)
- int `main` (int argc, char \*argv[ ])

### 8.104.1 Macro Definition Documentation

#### 8.104.1.1 `N_TESTS`

```
#define N_TESTS 3
```

### 8.104.2 Function Documentation

#### 8.104.2.1 `checkCiphertext()`

```
void checkCiphertext (
    const Ctxt & ctxt,
    const ZZX & ptxt,
    const SecKey & sk )
```

#### 8.104.2.2 `main()`

```
int main (
    int argc,
    char * argv[ ] )
```

## 8.105 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_matmul.cpp File Reference

```
#include <helib/matmul.h>
#include <NTL/BasicThreadPool.h>
#include <helib/fhe_stats.h>
#include <helib/randomMatrices.h>
#include <helib/ArgMap.h>
```

## Functions

- `template<class Matrix >`  
`bool DoTest (const Matrix &mat, const EncryptedArray &ea, const SecKey &secretKey, bool minimal, bool verbose)`
- `void TestIt (Context &context, long dim, bool verbose, long full, long block)`
- `int main (int argc, char *argv[])`

## Variables

- `int ks_strategy = 0`

### 8.105.1 Function Documentation

#### 8.105.1.1 DoTest()

```
template<class Matrix >
bool DoTest (
    const Matrix & mat,
    const EncryptedArray & ea,
    const SecKey & secretKey,
    bool minimal,
    bool verbose )
```

#### 8.105.1.2 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.105.1.3 TestIt()

```
void TestIt (
    Context & context,
    long dim,
    bool verbose,
    long full,
    long block )
```

### 8.105.2 Variable Documentation



### 8.105.2.1 ks\_strategy

```
int ks_strategy = 0
```

## 8.106 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_PAAlgebra.cpp File Reference

```
#include <cassert>
#include <string>
#include <sstream>
#include <NTL/ZZ.h>
#include <helib/NumbTh.h>
#include <helib/Context.h>
#include <helib/ArgMap.h>
```

### Functions

- int [main](#) (int argc, char \*argv[])

### 8.106.1 Function Documentation

#### 8.106.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

## 8.107 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_Permutations.cpp File Reference

```
#include <NTL/ZZ.h>
#include <helib/NumbTh.h>
#include <helib/timing.h>
#include <helib/permutations.h>
#include <helib/EncryptedArray.h>
#include <helib/ArgMap.h>
```

## Functions

- void `testCtxt` (long `m`, long `p`, long `widthBound=0`, long `L=0`, long `r=1`)
- void `testCube` (Vec< `GenDescriptor` > &`vec`, long `widthBound`)
- int `main` (int `argc`, char \*`argv[]`)

### 8.107.1 Function Documentation

#### 8.107.1.1 `main()`

```
int main (
    int argc,
    char * argv[] )
```

#### 8.107.1.2 `testCtxt()`

```
void testCtxt (
    long m,
    long p,
    long widthBound = 0,
    long L = 0,
    long r = 1 )
```

#### 8.107.1.3 `testCube()`

```
void testCube (
    Vec< GenDescriptor > & vec,
    long widthBound )
```

## 8.108 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_PGFFT.cpp File Reference

```
#include <helib/PGFFT.h>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <cmath>
#include <cstdint>
#include <stdexcept>
#include <limits>
```

## Namespaces

- [Fft](#)

## Typedefs

- typedef long double [ldbl](#)
- typedef std::complex< [ldbl](#) > [lcx](#)
- typedef complex< double > [cmplx\\_t](#)

## Functions

- void [Fft::transform](#) (std::vector< [lcx](#) > &vec)
- void [Fft::inverseTransform](#) (std::vector< [lcx](#) > &vec)
- void [Fft::transformRadix2](#) (std::vector< [lcx](#) > &vec)
- void [Fft::transformBluestein](#) (std::vector< [lcx](#) > &vec)
- void [Fft::convolve](#) (const std::vector< [lcx](#) > &vecx, const std::vector< [lcx](#) > &vecy, std::vector< [lcx](#) > &vecout)
- int [main](#) ()

## 8.108.1 Typedef Documentation

### 8.108.1.1 [cmplx\\_t](#)

```
typedef complex<double> cmplx\_t
```

### 8.108.1.2 [lcx](#)

```
typedef std::complex<ldbl> lcx
```

### 8.108.1.3 [ldbl](#)

```
typedef long double ldbl
```

## 8.108.2 Function Documentation

### 8.108.2.1 main()

```
int main ( )
```

## 8.109 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_PolyEval.cpp File Reference

```
#include <NTL/ZZ.h>  
#include <helib/polyEval.h>  
#include <helib/EncryptedArray.h>  
#include <helib/ArgMap.h>  
#include <helib/debugging.h>
```

### Functions

- bool [testEncrypted](#) (long d, const [EncryptedArray](#) &ea, const [SecKey](#) &secretKey)
- void [testIt](#) (long d, long k, long p, long r, long m, long L, bool isMonic=false)
- int [main](#) (int argc, char \*argv[])

## 8.109.1 Function Documentation

### 8.109.1.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

### 8.109.1.2 testEncrypted()

```
bool testEncrypted (  
    long d,  
    const EncryptedArray & ea,  
    const SecKey & secretKey )
```

### 8.109.1.3 testIt()

```
void testIt (
    long d,
    long k,
    long p,
    long r,
    long m,
    long L,
    bool isMonic = false )
```

## 8.110 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_Powerful.cpp File Reference

```
#include <helib/hypercube.h>
#include <helib/powerful.h>
#include <helib/Context.h>
#include <helib/ArgMap.h>
```

### Functions

- void [testSimpleConversion](#) (const Vec< long > &mvec)
- void [testHighLvlConversion](#) (const [Context](#) &context, const Vec< long > &mvec)
- int [main](#) (int argc, char \*argv[])

### 8.110.1 Function Documentation

#### 8.110.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.110.1.2 testHighLvlConversion()

```
void testHighLvlConversion (
    const Context & context,
    const Vec< long > & mvec )
```

### 8.110.1.3 testSimpleConversion()

```
void testSimpleConversion (
    const Vec< long > & mvec )
```

## 8.111 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_PtrVector.cpp File Reference

```
#include <cassert>
#include <iostream>
#include <NTL/tools.h>
#include <helib/NumbTh.h>
#include <helib/PtrVector.h>
#include <helib/PtrMatrix.h>
```

### Classes

- class [MyClass](#)

### Typedefs

- typedef [PtrVector](#)< [MyClass](#) > [MyPtrVec](#)
- typedef [PtrVector\\_VecT](#)< [MyClass](#) > [MyPtrVec\\_Vec](#)
- typedef [PtrVector\\_VecPt](#)< [MyClass](#) > [MyPtrVec\\_VecPt](#)
- typedef [PtrVector\\_vectorT](#)< [MyClass](#) > [MyPtrVec\\_vector](#)
- typedef [PtrVector\\_vectorPt](#)< [MyClass](#) > [MyPtrVec\\_vectorPt](#)
- typedef [PtrVector\\_slice](#)< [MyClass](#) > [MyPtrVec\\_slice](#)
- typedef [PtrMatrix](#)< [MyClass](#) > [MyPtrMat](#)
- typedef [PtrMatrix\\_Vec](#)< [MyClass](#) > [MyPtrMat\\_Vec](#)
- typedef [PtrMatrix\\_ptVec](#)< [MyClass](#) > [MyPtrMat\\_ptVec](#)
- typedef [PtrMatrix\\_vector](#)< [MyClass](#) > [MyPtrMat\\_vector](#)
- typedef [PtrMatrix\\_ptvector](#)< [MyClass](#) > [MyPtrMat\\_ptvector](#)

### Functions

- template<typename T2 >  
bool [compPointers](#) (const [MyPtrVec](#) &a, T2 &b)
- void [test1](#) ([MyClass](#) array[], int length, const [MyPtrVec](#) &ptrs)
- void [test2](#) ([MyClass](#) \*array[], int length, const [MyPtrVec](#) &ptrs)
- void [printPtrVector](#) (const [MyPtrVec](#) &ptrs)
- void [test3](#) ([MyPtrVec](#) &ptrs)
- template<typename T >  
void [test4](#) (const [MyPtrMat](#) &mat, const T &array)
- template<typename T >  
void [test5](#) (const [MyPtrMat](#) &mat, const T &array)
- int [main](#) ()

## Variables

- bool `verbose` = false

### 8.111.1 Typedef Documentation

#### 8.111.1.1 MyPtrMat

```
typedef PtrMatrix<MyClass> MyPtrMat
```

#### 8.111.1.2 MyPtrMat\_ptVec

```
typedef PtrMatrix_ptVec<MyClass> MyPtrMat_ptVec
```

#### 8.111.1.3 MyPtrMat\_ptvector

```
typedef PtrMatrix_ptvector<MyClass> MyPtrMat_ptvector
```

#### 8.111.1.4 MyPtrMat\_Vec

```
typedef PtrMatrix_Vec<MyClass> MyPtrMat_Vec
```

#### 8.111.1.5 MyPtrMat\_vector

```
typedef PtrMatrix_vector<MyClass> MyPtrMat_vector
```

#### 8.111.1.6 MyPtrVec

```
typedef PtrVector<MyClass> MyPtrVec
```

#### 8.111.1.7 MyPtrVec\_slice

```
typedef PtrVector_slice<MyClass> MyPtrVec_slice
```

#### 8.111.1.8 MyPtrVec\_Vec

```
typedef PtrVector_VecT<MyClass> MyPtrVec_Vec
```

#### 8.111.1.9 MyPtrVec\_VecPt

```
typedef PtrVector_VecPt<MyClass> MyPtrVec_VecPt
```

#### 8.111.1.10 MyPtrVec\_vector

```
typedef PtrVector_vectorT<MyClass> MyPtrVec_vector
```

#### 8.111.1.11 MyPtrVec\_vectorPt

```
typedef PtrVector_vectorPt<MyClass> MyPtrVec_vectorPt
```

### 8.111.2 Function Documentation

#### 8.111.2.1 compPointers()

```
template<typename T2 >
bool compPointers (
    const MyPtrVec & a,
    T2 & b )
```

#### 8.111.2.2 main()

```
int main ( )
```



#### 8.111.2.3 printPtrVector()

```
void printPtrVector (
    const MyPtrVec & ptrs )
```

#### 8.111.2.4 test1()

```
void test1 (
    MyClass array[],
    int length,
    const MyPtrVec & ptrs )
```

#### 8.111.2.5 test2()

```
void test2 (
    MyClass * array[],
    int length,
    const MyPtrVec & ptrs )
```

#### 8.111.2.6 test3()

```
void test3 (
    MyPtrVec & ptrs )
```

#### 8.111.2.7 test4()

```
template<typename T >
void test4 (
    const MyPtrMat & mat,
    const T & array )
```

#### 8.111.2.8 test5()

```
template<typename T >
void test5 (
    const MyPtrMat & mat,
    const T & array )
```

### 8.111.3 Variable Documentation

#### 8.111.3.1 verbose

```
bool verbose = false
```

## 8.112 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_ v1.0.2/HElib/src/Test\_Replicate.cpp File Reference

```
#include <cassert>
#include <NTL/lzz_pXFactoring.h>
#include <helib/helib.h>
#include <helib/replicate.h>
#include <helib/ArgMap.h>
```

### Classes

- class [StopReplicate](#)
- class [ReplicateTester](#)

### Functions

- void [TestIt](#) (long m, long p, long r, long d, long L, long bnd, long B)
- int [main](#) (int argc, char \*argv[])

### 8.112.1 Function Documentation

#### 8.112.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.112.1.2 TestIt()

```
void TestIt (
    long m,
    long p,
    long r,
    long d,
    long L,
    long bnd,
    long B )
```

### 8.113 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_tableLookup.cpp File Reference

```
#include <iostream>
#include <cassert>
#include <NTL/BasicThreadPool.h>
#include <helib/intraSlot.h>
#include <helib/tableLookup.h>
#include <helib/ArgMap.h>
```

#### Functions

- void [testLookup](#) (const [SecKey](#) &sKey, long insize, long outsize)
- void [testWritein](#) (const [SecKey](#) &sKey, long insize, long nTests)
- int [main](#) (int argc, char \*argv[])

#### 8.113.1 Function Documentation

##### 8.113.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

##### 8.113.1.2 testLookup()

```
void testLookup (
    const SecKey & sKey,
    long insize,
    long outsize )
```

### 8.113.1.3 testWritein()

```
void testWritein (
    const SecKey & sKey,
    long insize,
    long nTests )
```

## 8.114 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_thinboot.cpp File Reference

```
#include <NTL/BasicThreadPool.h>
#include <helib/helib.h>
#include <helib/matmul.h>
#include <helib/debugging.h>
#include <helib/fhe_stats.h>
#include <helib/ArgMap.h>
#include <algorithm>
#include <cmath>
#include <string>
```

### Functions

- void [TestIt](#) (long p, long r, long L, long c, long skHwt, int build\_cache=0)
- int [main](#) (int argc, char \*argv[])

### 8.114.1 Function Documentation

#### 8.114.1.1 main()

```
int main (
    int argc,
    char * argv[ ] )
```

#### 8.114.1.2 TestIt()

```
void TestIt (
    long p,
    long r,
    long L,
    long c,
    long skHwt,
    int build_cache = 0 )
```

8.115

/Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_ThinBootstrapping.cpp

File Reference

755

## 8.115 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_ThinBootstrapping.cpp File Reference

```
#include <NTL/BasicThreadPool.h>
#include <cassert>
#include <helib/helib.h>
#include <helib/matmul.h>
#include <helib/debugging.h>
#include <helib/ArgMap.h>
```

### Macros

- #define `num_mValues` (sizeof(mValues)/(14\*sizeof(long)))
- #define `OUTER_REP` (1)

### Functions

- void `TestIt` (long idx, long p, long r, long L, long c, long skHwt, int build\_cache=0)
- int `main` (int argc, char \*argv[])

## 8.115.1 Macro Definition Documentation

### 8.115.1.1 `num_mValues`

```
#define num_mValues (sizeof(mValues)/(14*sizeof(long)))
```

### 8.115.1.2 `OUTER_REP`

```
#define OUTER_REP (1)
```

## 8.115.2 Function Documentation

### 8.115.2.1 `main()`

```
int main (
    int argc,
    char * argv[] )
```

### 8.115.2.2 TestIt()

```
void TestIt (
    long idx,
    long p,
    long r,
    long L,
    long c,
    long skHwt,
    int build_cache = 0 )
```

## 8.116 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/Test\_ThinEvalMap.cpp File Reference

```
#include <cassert>
#include <helib/helib.h>
#include <helib/EvalMap.h>
#include <NTL/BasicThreadPool.h>
#include <helib/ArgMap.h>
```

### Functions

- void [TestIt](#) (long p, long r, long c, long \_k, long w, long L, Vec< long > &mvec, Vec< long > &gens, Vec< long > &ords, long useCache)
- int [main](#) (int argc, char \*argv[])

### 8.116.1 Function Documentation

#### 8.116.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

#### 8.116.1.2 TestIt()

```
void TestIt (
    long p,
    long r,
    long c,
    long _k,
    long w,
    long L,
    Vec< long > & mvec,
    Vec< long > & gens,
    Vec< long > & ords,
    long useCache )
```

## 8.117 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/Test\_Timing.cpp File Reference

```
#include <cassert>
#include <cstdio>
#include <memory>
#include <NTL/ZZ.h>
#include <NTL/BasicThreadPool.h>
#include <helib/helib.h>
#include <helib/matmul.h>
#include <helib/replicate.h>
#include <helib/permutations.h>
#include <helib/ArgMap.h>
#include <helib/debugging.h>
#include <helib/randomMatrices.h>
```

### Classes

- class [LowLvTimingData](#)
- class [HighLvTimingData](#)
- class [OtherTimingData](#)
- class [TimingData](#)
- class [ReplicateDummy](#)

### Macros

- `#define numTests 11`

### Functions

- void [timeInit](#) (long m, long p, long r, long d, long L, long nTests)
- long [rotationAmount](#) (const [EncryptedArray](#) &ea, const [PubKey](#) &publicKey, bool onlyWithMatrix)
- void [timeOps](#) (const [EncryptedArray](#) &ea, const [PubKey](#) &publicKey, [Ctxt](#) &ret, const vector< [Ctxt](#) > &c, ZZx &p, long nTests, [LowLvTimingData](#) &td)
- void [timeHighLv](#) (const [EncryptedArray](#) &ea, const [PubKey](#) &publicKey, [Ctxt](#) &ret, const vector< [Ctxt](#) > &c, [GeneratorTrees](#) &trees, long nTests, [HighLvTimingData](#) &td)
- void [TimeIt](#) (long m, long p, [TimingData](#) &data, bool high=false)
- void [printTimeData](#) ([TimingData](#) &td)
- int [main](#) (int argc, char \*argv[])

### 8.117.1 Macro Definition Documentation

#### 8.117.1.1 numTests

```
#define numTests 11
```

## 8.117.2 Function Documentation

### 8.117.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

### 8.117.2.2 printTimeData()

```
void printTimeData (
    TimingData & td )
```

### 8.117.2.3 rotationAmount()

```
long rotationAmount (
    const EncryptedArray & ea,
    const PubKey & publicKey,
    bool onlyWithMatrix )
```

### 8.117.2.4 timeHighLvl()

```
void timeHighLvl (
    const EncryptedArray & ea,
    const PubKey & publicKey,
    Ctxt & ret,
    const vector< Ctxt > & c,
    GeneratorTrees & trees,
    long nTests,
    HighLvlTimingData & td )
```

### 8.117.2.5 timeInit()

```
void timeInit (
    long m,
    long p,
    long r,
    long d,
    long L,
    long nTests )
```



### 8.117.2.6 TimeIt()

```
void TimeIt (
    long m,
    long p,
    TimingData & data,
    bool high = false )
```

cannot test high-level routines at level <8

### 8.117.2.7 timeOps()

```
void timeOps (
    const EncryptedArray & ea,
    const PubKey & publicKey,
    Ctxt & ret,
    const vector< Ctxt > & c,
    ZZx & p,
    long nTests,
    LowLvlTimingData & td )
```

## 8.118 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_v1.0.2/HElib/src/timing.cpp File Reference

```
#include <algorithm>
#include <utility>
#include <cstring>
#include <ctime>
#include <helib/timing.h>
```

## Namespaces

- [helib](#)

## Functions

- unsigned long [helib::GetTimerClock](#) ()
- bool [helib::timer\\_compare](#) (const FHEtimer \*a, const FHEtimer \*b)
- void [helib::registerTimer](#) (FHEtimer \*timer)
- void [helib::resetAllTimers](#) ()
- void [helib::printAllTimers](#) (std::ostream &str=std::cerr)  
*Print the value of all timers to stream.*
- const FHEtimer \* [helib::getTimerByName](#) (const char \*name)
- bool [helib::printNamedTimer](#) (std::ostream &str, const char \*name)

## Variables

- const unsigned long [helib::CLOCK\\_SCALE](#) = (unsigned long)CLOCKS\_PER\_SEC

### 8.119 /Users/Projects/FHE\_HElib\_Repos\_Public\_staging/homenc\_↵ v1.0.2/HElib/src/zzX.cpp File Reference

```
#include <mutex>
#include <map>
#include <helib/PAlgebra.h>
#include <helib/timing.h>
#include <helib/zzX.h>
```

## Namespaces

- [helib](#)

## Functions

- void [helib::MulMod](#) (zzX &res, const zzX &a, const zzX &b, const PAlgebra &palg)
- void [helib::add](#) (zzX &res, const zzX &a, const zzX &b)
- void [helib::mul](#) (zzX &res, const zzX &a, long b)
- void [helib::div](#) (zzX &res, const zzX &a, long b)
- void [helib::normalize](#) (zzX &f)
- const NTL::zz\_pXModulus & [helib::getPhimXMod](#) (const PAlgebra &palg)
- void [helib::reduceModPhimX](#) (zzX &poly, const PAlgebra &palg)
- zzX [helib::balanced\\_zzX](#) (const NTL::zz\_pX &f)
- zzX [helib::balanced\\_zzX](#) (const NTL::GF2X &f)

#### 8.119.1 Detailed Description

- manipulating polynomials with single-precision coefficient It is assumed that the result is also single-precision