CYBER APOCALYPSE 2026 · CRYPTO

Hidden Secret

Category: Crypto **Primitive:** hidden subset sum with geometric weights

Technique: sample products + LLL on a 210-dim lattice **Solve time:** 5 min 49 s

---

FLAG

HTB{too_few_samples_too_much_lattice_42874424b5e32fabd174491e1ba6cc85}

> *The oldest vow-stones were carved with patterns few can still read. Priests claimed they described more than history, they concealed the shape of a secret known only to the first kings. Recover what lies beneath the symbols, and another piece of the realm's foundation will reveal itself.*

1. Files provided

File	Description
`chall.py`	The generator
`output.txt`	`pub = (p, arr)` with a 8192-bit prime and 7 residues, plus the AES-ECB ciphertext

```
from Crypto.Util.number import getPrime
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import hashlib
import secrets

bits = 8192
samp = 7
deg = 40

def geninstance():
    p = getPrime(bits)
    x = secrets.randbelow(p)
    a = secrets.randbelow(p)
    arr = []
    for _ in range(samp):
        tot = 0
        for j in range(deg):
            tot += a*pow(x, j, p)*(1 - secrets.randbelow(3))
        tot %= p
        arr.append(tot)
    return (p, arr), x

pub, x = geninstance()

key = hashlib.sha256(str(x).encode()).digest()
flag = open("flag.txt", "rb").read()
cipher = AES.new(key, AES.MODE_ECB)
with open("output.txt", "w") as f:
    print(f"{pub = }", file= f)
    print(f"encflag = bytes.fromhex('{cipher.encrypt(pad(flag, 16)).hex()}')", file= f)
```

`1 - secrets.randbelow(3)` is a uniform *trit* in `{-1, 0, 1}`, so each published sample is

$$\texttt{arr[i]} = a \cdot P_i(x) \bmod p\,, \quad P_i(X) = \Sigma_{j=0..39}\, c_{ij}\, X^j\,, \quad c_{ij} \in \{-1,0,1\}$$

with one secret multiplier `a`, one secret evaluation point `x`, and 7 secret ternary polynomials of degree < 40. The AES key is `sha256(str(x))`, so **`x` must be recovered exactly** — nothing less will do.

2. Why the obvious attacks all stall

Write `u_j = a·x^j`. Then `arr[i] = <c_i, u>`: seven inner products of the unknown weight vector `u ∈ Z_p^40` with unknown ternary vectors. That is a **hidden subset sum problem** with 7 samples and 40 hidden weights, and every published algorithm for it (Nguyen–Stern, Coron–Gini, …) needs *more* samples than weights. Here the ratio is 7 : 40.

Everything else runs into the same wall — a chicken and egg:

- **Knowing `x` makes it trivial.** With `x` in hand, `L_u = {v : <v,u> ≡ 0 mod p}` has minimum `≈ p^(1/40) = 2^205`, so a ternary vector of norm 5 is absurdly short and LLL recovers each `c_i` instantly.
- **Knowing the `c_i` makes it trivial.** `arr[0]·P_i(X) − arr[i]·P_0(X)` vanishes at `x` for every `i`, so a polynomial `gcd` mod `p` hands over `X − x`.
- But every lattice one can *write down* needs one of the two. Dividing out `a` (`r_i = arr[i]/arr[0] = P_i(x)/P_0(x)`) removes `a`, but the remaining conditions `Σ_j x^j (arr[0]c_ij − arr[i]c_0j) ≡ 0` are **bilinear**: the coefficients multiplying the unknown trits are the unknown monomials `x^j`. Eliminating `x` by a resultant makes the system degree 78 in the trits.

Information-theoretically three samples already pin `x` down uniquely, so the four extra samples are not there for redundancy — they are there because *the intended attack needs them*.

3. The key idea: products of samples are new samples

The only computable objects are polynomial expressions in the seven published residues. Products are the interesting ones, because they keep the same hidden geometric weights:

```
arr[i]*arr[k]          = a^2 * (P_i P_k)(x)
arr[i]*arr[k]*arr[l]   = a^3 * (P_i P_k P_l)(x)
...
```

A product of `D` samples is `a^D · F(x)` where `F` is a product of `D` ternary polynomials of degree ≤ 39 — so `deg F ≤ 39D`, i.e. `F` lives in a space of dimension `39D + 1`. The number of distinct degree-`D` products of 7 samples is `C(D+6, 6)`. Compare:

`D`	**products `C(D+6,6)`**	**dimension `39D+1`**	**linear relations?**
1	7	40	no
2	28	79	no
3	84	118	no
4	**210**	**157**	**yes — 53 of them**

At `D = 4` there are more products than dimensions, so there exist integer vectors `λ ∈ Z^210` with

$$\Sigma_\alpha \lambda_\alpha \cdot F_\alpha(X) = 0 \quad \text{(identically, as a polynomial)}$$

and each such λ immediately gives a *computable* congruence, because the identity holds at $X = x$ as well:

$$\Sigma_\alpha \lambda_\alpha \cdot w_\alpha \equiv 0 \pmod p, \quad w_\alpha = \text{product of the 4 samples}$$

That is the whole trick. $\mathtt{samp = 7}$ is exactly what makes $D = 4$ the first working degree (with 6 samples one would need $D = 5$ and 252 products; with 8 samples $D = 3$ and only 120).

Why LLL finds them

Consider the computable lattice $\Lambda_w = \{\lambda \in \mathbb{Z}^{210} : \Sigma\, \lambda_\alpha\, w_\alpha \equiv 0 \bmod p\}$: dimension 210, determinant p, so a *random* lattice of that shape has minimum

$$\text{sqrt}(210 / 2\pi e) \cdot p^{1/210} = 3.5 \cdot 2^{39.0} = 2^{40.8}$$

The 53 structural relations, on the other hand, are integer relations among 210 vectors of length 157 whose entries are only a few hundred in size — a quick simulation with random ternary polynomials shows they have norm about $\mathbf{2^{25}}$. They are ~2^{15} shorter than anything a random 210-dimensional lattice of determinant p contains, and they span a whole 53-dimensional sublattice, so plain LLL rakes them in.

This is also what fixes $\mathtt{bits = 8192}$: the attack needs $p^{(1/210)} \gg (\text{relation norm}) \approx 2^{25}$. With a 4096-bit prime $p^{(1/210)} = 2^{19.5}$ and the structural relations would be *longer* than generic lattice vectors — invisible to LLL. 8192 bits is chosen just large enough for the separation to exist.

Running `fplll -a lll` on the 210-dimensional basis (≈ 5 minutes, 8192-bit entries) gives exactly the predicted split:

```
53 basis vectors of log2(norm) ~ 25.0 - 25.6     <-- the structural relations
157 basis vectors of log2(norm) ~ 49             <-- generic vectors
```

and each short one verifies $\Sigma\, \lambda_\alpha\, w_\alpha \equiv 0 \pmod p$.

4. From the relations back to the polynomials

Order the 210 multisets α. The relations say that the vector

$$\mathrm{col}_k[\alpha] = [X^k]\ F_\alpha \qquad (k = 0 \ldots 156)$$

is orthogonal to every `λ` — the 53 relations are precisely the orthogonal complement of the lattice spanned by these 157 "coefficient columns". Now note that **`col_k` depends only on the trits of level ≤ k**, and splitting off the terms where a single factor contributes degree `k`:

```
[X^k] F_alpha  =  known(c_{*,0..k-1})  +  sum_i A_alpha,i * c_i,k ,
A_alpha,i      =  sum over occurrences of i in alpha of the product of the other three constants
```

which is **affine in the seven level-`k` trits**. So each level gives a linear system `B·c = V` with `B = KER·A` (53 × 7) — recover the polynomials one coefficient level at a time. Level 0 is quartic rather than affine, so it is brute-forced over the `3^7 = 2187` ternary choices.

The degeneracy, and how to prune it

`B` turns out to have rank 5, not 7, because the relation lattice is invariant under three symmetries:

- a global scalar `P_i → t·P_i`;
- a **common polynomial factor** `P_i → h·P_i` (relations survive: `Σ λ_α F_α h^4 = 0`);
- a substitution `X → sX`.

Worse, there is a family of junk solutions: any tuple whose columns are *zero* from some level on (e.g. "all polynomials are constants") satisfies every relation trivially. Naive level-by-level search explodes (5 → 2207 → 4.8 M candidates by level 2), and scoring by smallest column norm walks straight into the all-constants branch.

The clean fix uses a property the real solution has and the junk does not: **the 157 true columns are linearly independent.** So keep a running row-echelon form of the columns recovered so far and discard any branch whose new column is linearly dependent on them. A degenerate continuation contributes `t·col_0` (or `0`) and dies immediately; a `h`-multiplied continuation needs `|c_ij ± t·c_i,j−m| ≤ 1` at all 280 positions and dies within a few levels. With that single rule the search stays a couple of dozen branches wide for all 40 levels:

```
level0 survivors: 4
level 1 survivors 12      level 15 survivors 28
level 2 survivors 20      level 20 survivors 52
level 5 survivors  8      level 39 survivors 32
```

32 candidate tuples of ternary polynomials survive — cheap to test individually.

5. Recovering `x`

For a surviving tuple `Q`, `arr[0]·Q_i(X) − arr[i]·Q_0(X)` vanishes at `x` for every `i` (it equals `a·(Q_0 Q_i − Q_i Q_0)(x) = 0` when `Q = P`, and picks up only a harmless extra factor when `Q = h·P`). Taking a `gcd` of three of them over `F_p` leaves a linear factor, whose root is a candidate `x`. Verify it by `a = arr[0]/Q_0(x)` and checking all seven samples.

One subtlety: the recovery is blind to two more symmetries of the *sample data* itself:

- `Q_i(X) = P_i(−X)` is also ternary and reproduces the samples at `−x`;
- the reversal `Q_i(X) = X^39 P_i(1/X)` is ternary too and reproduces them at `1/x`.

So the consistency check is passed by `±x` and `±1/x` alike (this bit me first: the first consistent root decrypted to garbage). Just try all four and keep the one whose AES-ECB decryption starts with `HTB{`.

```
for X in {x, -x % p, pow(x,-1,p), -pow(x,-1,p) % p}:
    flag = AES.new(sha256(str(X).encode()).digest(), AES.MODE_ECB).decrypt(encflag)
```

6. Result

`writeups/solve.py` runs the whole chain (5 min 49 s wall clock, of which ~5 min is the 210-dimensional LLL):

```
[*] LLL on a 210-dimensional lattice with 8192-bit entries ...
[+] 53 structural relations, log2(norm) ~ 25.5
[+] 32 candidate polynomial tuples
[+] x = 837235077365728253163680683821287291080584843054252636562178871900253887687378 4...
[+] FLAG: HTB{too_few_samples_too_much_lattice_42874424b5e32fabd174491e1ba6cc85}
```

The flag itself confirms the intended route — with only 7 samples the direct hidden-subset-sum machinery is out of reach, so you buy extra samples by multiplying the ones you have, and pay for them with a much bigger lattice.

FLAG

HTB{too_few_samples_too_much_lattice_42874424b5e32fabd174491e1ba6cc85}

7. Takeaways

- A "too few samples" hidden-subset-sum instance can be inflated: products of samples are samples for the same geometric weight sequence, at the cost of a higher-degree polynomial space. Relations appear as soon as `C(D+m−1, m−1) > (deg−1)·D + 1`.
- Structural relations that hold *identically* also hold *mod p*, and they are far shorter than generic vectors of the modular relation lattice — a dense sublattice LLL will find.
- Modulus size is a design parameter for the *attack*, not just for security: the whole thing only separates because `p^(1/210)` sits comfortably above the relation norm.
- When a reconstruction has symmetries (global factor, common polynomial factor, `X → sX`, coefficient reversal), expect a handful of equally valid answers and let the ciphertext arbitrate.

Cyber Apocalypse 2026 · Crypto · Hidden Secret — solved with fplll + a 40-level ternary reconstruction.