A Appendix

A.1 Parameters

Table 9. LWE, *preprocessing, and training parameters*. *For the adaptive increase of preprocessing parameters, we start with blocksize β_1 and LLL-delta δ_{LLL1}, and upgrade to β_2 and δ_{LLL2} at a later stage. Parameters base B and bucket size r are used to tokenize the numbers for transformer training.*

n	$\log_2 q$	q	β_1	δ_{LLL1}	β_2	δ_{LLL2}	base B	bucket size r
256	12	3329	35	0.99	40	1	417	1
	14	11197	35	0.99	40	1	1400	1
	16	42899	35	0.99	40	1	5363	4
	18	222553	35	0.99	40	0.99	27820	16
	20	842779	35	0.99	40	0.99	105348	64
350	21	1489513	30	0.96	40	0.99	186190	128
	27	94056013	30	0.96	40	0.99	5878501	4096
512	41	2199023255531	18	0.93	22	0.96	137438953471	134217728

VERDE runs data preprocessing with the parameters shown in Table 9, in parallel using multiple CPUs. We fully parallelize when the time to process one matrix is greater than 24 hours, e.g., for $n = 350, \log_2 q = 27$, we used 5000 CPUs. Otherwise, we parallelize on fewer CPUs depending on the attack time allowed, e.g., preprocessing is completed on 270 CPUs in less than 4 days for $n = 256, \log_2 q = 14$, and on 990 CPUs in less than 3 days for $n = 256, \log_2 q = 18$.

In VERDE, the tokenization used by the transformer mirrors the strategy in §4 of [42], but uses smaller bases B and larger bucket sizes r for better performance (Table 9, 10 of [42]). Decreasing B is further supported by Table 10, evaluated on a low modulus data using VERDE's preprocessing ($n = 256, \log_2 q = 16$). Full ternary secret recovery and partial Gaussian secret recovery both improve with smaller $B = 5363$.

Table 10. *Secret recovery for different bases*. *$n = 256, \log_2 q = 16$. We show full ternary secret recovery and partial Gaussian secret recovery when using $B = 7150$ and 5363 on the same datasets and secrets.*

	ternary					Gaussian			
h	8	9	10	11	12	3	4	5	6
$B = 7150$	2/10	0/10	1/10	0/10	1/10	5/10	6/10	1/10	1/10
$B = 5363$	2/10	0/10	2/10	0/10	2/10	5/10	6/10	1/10	3/10

A.2 More seeds for initialization

The ML attacks benefit from running multiple times with different seeds, or *initializations*, as was demonstrated in PICANTE [42, Section 6.4]. More seeds improve both the success probability and the number of epochs required. Table 11 shows how binary secret recovery improves with more seeds, for different Hamming weights h when $n = 256$ and $\log_2 q = 12$.

Table 11. *Secret recovery with 1 vs 5 seeds*. *$n = 256$, $\log_2 q = 12$, binary secrets. For 1 seed, epoch is the epoch of secret recovery; For 5 seeds (ran on the same secrets as the 1 seed experiments), epoch is the lowest epoch of secret recovery among the 5 initializations for each secret.*

h	3	4	5	6
recovery, 1 seed	5/10	1/10	1/10	1/10
epoch	0,0,0,8,17	0	17	13
recovery, 5 seeds	7/10	3/10	1/10	2/10
epoch	0,0,0,7,7,8,17	0,5,5	7	4,7

Table 12 shows the results for ternary secrets on $n = 512$, $\log_2 q = 41$, where we run 5 initializations for each secret. While most initializations partially recovered the secret, only a few got full recovery within 20 epochs. Full recovery benefits from more initializations, especially for high h.

Table 12. ***Ternary secret recovery with 5 initializations***. $n = 512$. *'-': recovery did not occur in* ≤ 20 *epochs.*

h	45	46	50	55	58	60
epoch of partial recovery	4,4,6,6,7	1,1,1,1,1	3,4,5,7,-	5,6,6,6,7	6,6,8,10	5,8,8,8,12
epoch of full recovery	10,-,-,-,-	5,7,-,-,-	4,10,14,17,-	16,-,-,-,-	11,11,-,-,-	-,-,-,-,-

A.3 Comparison with PICANTE

To demonstrate the power of the new preprocessing in SALSA VERDE, we run a set of experiments on $n = 256$, $\log_2 q = 23$, where PICANTE also showed success. Using blocksize $\beta = 40$, each matrix is processed by VERDE in about 1.5 days; PICANTE took 2.2 days/matrix. VERDE achieves a reduction factor of 0.25, compared to 0.33 in PICANTE. As shown in Table 1, the difference in the preprocessing step is even more striking for lower q.

The highest h recovered by PICANTE was $h = 31$ in 4 out of 20 experiments; VERDE recovered $h = 43$. In Table 13, we see that for $h = 26 - 31$, VERDE significantly outperforms PICANTE in both the success rate and the number of epochs required. In other words, better preprocessing results in lower training time and better secret recovery.

Table 13. Epochs of secret recovery for PICANTE vs. VERDE. $n = 256$ *and* $\log_2 q = 23$. *'-' means secret not recovered. 5 secrets per* h*, except for* PICANTE $h = 31$ *(20 secrets).*

h	26	27	28	29	30	31
PICANTE	2,3,4,7,-	10,-,-,-,-	5,-,-,-,-	5,9,11,-,-	17,20,32,-,-	6,12,26,27 (out of 20 secrets)
VERDE	0,0,2,2,7	2,6,-,-,-	0,0,0,1,2	0,1,1,1,-	0,1,2,3,-	1,2,3,4,-

A.4 Distinguisher tested on reduced data outperforms random data

We compare the performance of running the distinguisher on the preprocessed data that were held out from the training set (Dist_{BKZ}) with running on random vectors ($\text{Dist}_{\text{rand}}$). We run both set of experiments with the same model initialization seeds, and record a success for the method(s) that recovers the secret at the earliest epoch. Table 14 indicates that Dist_{BKZ} performs better.

Table 14. ***Secret recovery by running the distinguisher on the random vectors and bkz preprocessed data***. $n = 256, \log_2 q = 23$, *on data processed using* PICANTE*'s approach.*

h	27	28	29	30	31
Dist_{BKZ}	2/5	1/5	1/5	0/5	0/5
$\text{Dist}_{\text{rand}}$	1/5	0/5	0/5	0/5	0/5

A.5 Dimension reduction techniques

Most of the entries in a sparse secret are zero, so prior work [5] has suggested the idea of randomly assuming a subset of the entries to be zero and removing them to reduce the dimension of the lattice problem. The assumption will be correct with some probability depending on the secret's sparsity. Here we explore an improvement on this strategy: we use the partially trained model to glean signal on which entries should be kicked out. We can either try to kick out zeros, which we call *dimension reduction*, or in the binary case, kick out 1s, which we call *Hamming reduction*, or combined.

This technique will be better than random when the model has begun to learn information about the bits, reflected in their relative rankings. Specifically, the ranking strategies described in [42, Section 4.3] are used to compute scores which estimate the likelihood of secret bits being 1. Once the model has started to learn, we can assume that the highest ranked bits will correspond to secret bits which are equal to 1, and the lowest ranked bits will correspond to zeros. So we use this information to reduce the dimension of the problem by kicking out low-ranked bits which we guess to be zero or high-ranked bits which we guess to be 1. Then, we retrain a model on the smaller dimensional samples and hope to recover the secret. If the original kicked out bits were correct and the model recovers the secret of the smaller dimensional problem, then we find the original secret.

Dimension reduction. Since there are many more zeros than 1s in sparse secrets, we can potentially reduce the dimension significantly. Once we remove the bits with low scores, we can simply re-run training on the dataset with $(\mathbf{a}', b)$ where $\mathbf{a}'$ are the samples with the corresponding bits removed. If the indices have been identified incorrectly, then the reduction will fail. For $n = 256, \log_2 q = 14$, VERDE attempted 10 binary secrets with $h = 10$ and did not recover the secret. Then we tried dimension reduction on these experiments and recovered one secret.

Hamming reduction. Kicking out $1s$ from the secret is particularly valuable, given the theoretical analysis of the VERDE in Section 6. If nonzero bits are indeed ranked at the top by the model, a straightforward approach of kicking out the top-ranked bits and retraining on the smaller dimension and Hamming weight will likely yield improved secret recovery.

But in case some of the top-ranked bits are not equal to 1, we propose the following strategy. Suppose S is a small set of indices for bits with the highest scores. We construct the following problem: let $\mathbf{s}'$ be $\mathbf{s}$ with bits in S flipped, and $\mathbf{a}'$ be $\mathbf{a}$ with a_i negated for $i \in S$. Equivalently, for $i \in S$, $a'_i = -a_i$ and $s'_i = 1 - s_i$. Then, the corresponding

$$b' = \mathbf{a}' \cdot \mathbf{s}' = \sum_{i \notin S} a_i s_i + \sum_{i \in S} a'_i s'_i = \sum_{i \notin S} a_i s_i + \sum_{i \in S} -a_i(1 - s_i) = b - \sum_{i \in S} a_i.$$

If more than half of the indices in S are 1, then $\mathbf{s}'$ has a smaller Hamming weight, hence the instance $(\mathbf{a}', b' = b - \sum_{i \in S} a_i)$ is likely easier. If exactly half of the indices in S are 1, then $\mathbf{s}'$ has the same h as $\mathbf{s}$, but the new instance $(\mathbf{a}', b')$ will have a different **NoMod** and may be recoverable. This strategy can be applied to the top-ranked bits, and will succeed in reducing the Hamming weight if more than half of those bits are 1.

A.6 Attacking sparse secrets on larger dimensions

For sparse secrets on even larger dimensions, we can apply our attack after using combinatorial techniques to exploit the sparsity of the secret. The approach would be to combine VERDE with the techniques from [5, 20] as follows: Randomly kick out k entries of the secret, assuming they are zero, which will be true with some probability. This reduces the LWE problem to a smaller dimension where VERDE can recover the secret. The expected cost of the attack would be VERDE's cost multiplied by $1/p$, where $p = (\frac{n-h}{n})^k$ is the probability that the assumption that the k entries are 0 is correct.

A.7 Comparison with lattice reduction/uSVP attacks

In this section we compare VERDE with state-of-the-art classical lattice reduction attacks in two ways. The LWE Estimator [6] gives heuristic predicted running times for the best known lattice reduction attacks. But even the authors of the LWE Estimator claim that the estimates are often wrong and unreliable. So we compare VERDE to the Estimator results but we also compare to concrete running times achieved by running lattice attacks ourselves, on the same machines where we run our ML-based attacks. High-level comparison for binary secrets, $n = 256$, is in Table 15.

Table 15. Comparison of VERDE's and uSVP attack performance on LWE problems with $n = 256$, binary secrets, varying q and h. *VERDE's total attack time is the sum of preprocessing and training time (with recovery included). Preprocessing time assumes full parallelization, and training time is the number of epochs to recovery multiplied by epoch time (1.5 hours/epoch). N/A means no successful secret recovery.*

LWE parameters		VERDE attack time			uSVP attack time (hrs)
$\log_2 q$	h	*Preprocessing (hrs)*	*Training*	*Total (hrs)*	
12	8	1.5	2 epochs	4.5	N/A
14	12	2.5	2-5 epochs	5.5-10	N/A
16	14	8.0	2 epochs	11	N/A
18	18	7.0	3 epochs	11.5	558
18	20	7.0	1-8 epochs	8.5-19	259
20	22	7.5	5 epochs	15	135-459
20	23	7.5	3-4 epochs	12-15	167-330
20	24	7.5	4 epochs	13.5	567
20	25	7.5	5 epochs	15	76 - 401

To summarize the comparison, VERDE outperforms existing classical attacks in two senses: 1) VERDE fully recovers sparse binary and ternary secrets for n and q where existing classical attacks do not succeed in several weeks or months using *fplll* BKZ 2.0 [19] with the required block size; and 2) in cases where we improve the implementation enough to run them with the required large block size, we find that in all cases, VERDE recovers the secrets much faster (see Table 15).

Summarizing classical lattice reduction attacks. Table 16 gives the estimated heuristic cost and specifies the block size for attacking sparse binary and ternary secrets for $n = 256, 350, 512$ and various q with the best known classical lattice reduction attack.

Table 16. ***Best classical attack from the* LWE *Estimator***. *On binary and ternary secrets. For* VERDE*'s highest h, we run the estimator and report the best attack and their cost. β is the estimated block size of the attack.*

n	$\log_2 q$	binary secret				ternary secret			
		h	best attack	rop	β	h	best attack	rop	β
256	12	8	dual_mitm_hybrid	$2^{43.0}$	40	9	dual_mitm_hybrid	$2^{43.8}$	40
	14	12	dual_hybrid	$2^{47.2}$	40	13	dual_hybrid	$2^{47.7}$	41
	16	14	dual_hybrid	$2^{46.8}$	40	16	dual_hybrid	$2^{47.4}$	41
	18	23	bdd_hybrid	$2^{47.1}$	45	23	dual_hybrid	$2^{47.4}$	41
	20	36	bdd	$2^{44.0}$	45	33	bdd	$2^{44.2}$	46
350	21	12	dual_mitm_hybrid, bdd_mitm_hybrid	$2^{46.4}$	40	13	dual_mitm_hybrid	$2^{46.9}$	54
	27	36	bdd	$2^{44.9}$	47	38	bdd, bdd_hybrid	$2^{44.1}$	43
512	41	63	usvp/bdd	$2^{42.9}$	40	60	usvp/bdd	$2^{42.9}$	40

Table 17 gives the same information for the uSVP classical lattice reduction attack, focusing on $n = 256$ for some of the larger q and h where VERDE succeeds.

Table 17. ***Estimated cost and block sizes of uSVP***. *$n = 256, \log_2 q = 16, 18, 20$, for binary, ternary, and Gaussian secrets with h nonzero entries.*

$\log_2 q$	binary			ternary			Gaussian		
	h	rop	β	h	rop	β	h	rop	β
16	12	$2^{54.7}$	86	12	$2^{54.9}$	87	6	$2^{69.3}$	138
18	18	$2^{49.5}$	67	19	$2^{49.8}$	68	7	$2^{59.5}$	102
20	25	$2^{45.4}$	52	24	$2^{45.4}$	52	7	$2^{53.5}$	80

uSVP attack performance, binary secrets. For $n = 256$ and $\log_2 q = 20$, we run the concrete uSVP attack using *fplll* BKZ 2.0 with Kannan's embedding and parameters ([37], [16]). With block size 50 and 55, secrets are not recovered in 25 days, and block size 60 or larger cannot finish the first BKZ loop in 3 days. We propose two improvements to get these attacks to run faster and better: 1) we rearrange the rows of the uSVP matrix as in VERDE; 2) we use the adaptive float type upgrade as in VERDE. In addition, after each BKZ loop, we also run the secret validation obtained from the shortest vector found so far, and terminate if we get a secret match.

With rearranging the rows but without the adaptive float type upgrade, we have to use higher precision because otherwise the attack fails due to low precision after running for 23 hours. The attacks run quite slowly with high precision and did not recover secrets after 25 days, except in one case where a binary secret with $h = 22$ was recovered with block size 55 in 414 hours, roughly 17 days.

With rearranging the rows and the adaptive float type upgrade, we ran $n = 256$ and $\log_2 q = 20$ with block size $50 - 55$ for binary/ternary secrets, and $n = 256$ and $\log_2 q = 18$ with block size 65 for binary/ternary secrets. See Table 18 for running times and h. For example, for $\log_2 q = 20$, a ternary secret with $h = 25$ was recovered in 280 hours, roughly 12 days, and for $\log_2 q = 18$, a binary secret with $h = 20$ was found in ≈ 11 days. When the block size used is lower than predicted by the estimator (50 instead of 52 for $\log_2 q = 20$ and 65 instead of 67 for $\log_2 q = 18$, see Table 17, 18), the uSVP attack succeeds in recovering only a few secrets out of many which were tried.

Table 18. *uSVP concrete attack time,* $n = 256$***.*** *Upper:* $\log_2 q = 20$*, lower:* $\log_2 q = 18$*. For each* h*, block size and secret distribution, we run 5 experiments on different secrets and show the secret recovery time (in hours). '-' means no success in 25 days.*

$\mathbf{n = 256}, \log_2 \mathbf{q = 20}$

blocksize	secret	$h = 22$	$h = 23$	$h = 24$	$h = 25$
50	binary	-	-	-	451, 531
	ternary	261, 367	-	-	-
55	binary	135, 161, 459	167, 323, 330	567	76, 280, 401
	ternary	43, 241, 432	234, 296	226, 257, 337	280

$\mathbf{n = 256}, \log_2 \mathbf{q = 18}$

blocksize	secret	$h = 11$	$h = 12$	$h = 17$	$h = 18$	$h = 19$	$h = 20$
65	binary	32	-	-	558	-	259
	ternary	324, 553	109	594	-	-	-

The concrete experiments allow us to validate to some extent the predictions of the Estimator in many cases, giving confidence in the comparison with VERDE. Running the uSVP attack with block size 70 or larger didn't finish the first BKZ loop in 3 days, so we expect longer attack time for Gaussian secrets and for binary and ternary secrets with lower q.

Gaussian secrets. VERDE achieves partial Gaussian secret recovery for small h, reducing the secret recovery to a lattice problem in tiny dimension (h). Because the preprocessing time and per epoch time does not vary with secret distribution, VERDE's attack time on Gaussian secrets is comparable to on binary secrets (see Table 15, 19). We note that preprocessing takes longer for $n = 350$ and $\log_2 q = 27$ due to precision issues. This is an important observation because lattice problems are supposed to be harder for smaller q. In contrast, with classical attacks, the Estimator predicts significantly larger block sizes required, and longer running times (Table 20), than for binary secrets. VERDE compares favorably to classical attacks on Gaussian secrets, especially on small q (where the problem is harder), e.g., VERDE recovers Gaussian secrets with $h = 5 - 6$ in $4.5 - 15$ hours for $n = 256, \log_2 q = 12$, where the cost of best classical attacks is predicted to be 2^{91} rop.

Table 19. VERDE's performance on LWE problems with $n = 256$ **and** 350**, Gaussian secrets, varying** q **and** h**.** *Preprocessing time: hours to process one matrix. Total attack time: sum of preprocessing time (assuming full parallelization) and training time (number of epochs multiplied by hours per epoch, see §2).*

LWE parameters			VERDE attack time			
n	$\log_2 q$	h	*Preprocessing (hrs)*	*Training*	*hrs/epoch*	*Total (hrs)*
256	12	5-6	1.5	2-9 epochs	1.5	4.5-15
	14	5-6	2.5	2-21 epochs		5.5-34
	16	9	8.0	2 epochs		11
	18	9	7.0	3 epochs		11.5
	20	10	7.5	5 epochs		15
350	21	5	16	1-5 epochs	1.6	18-24
	27	10	216	2-13 epochs		219-237

Table 20. LWE *Estimator: best classical attack on Gaussian secrets on various* q*. $n = 256$ *and* 350*. For* VERDE*'s highest* h*, we show the best classical attack, the attack cost (rop), and predicted block size* β *from the* LWE *Estimator.*

n	$\log_2 q$	h	best attack	rop	β
256	12	6	bdd / bdd_hybrid	$2^{91.0}$	214
	14	6	bdd / bdd_hybrid	$2^{77.7}$	166
	16	9	bdd / bdd_hybrid	$2^{67.1}$	128
	18	9	bdd	$2^{57.7}$	93
	20	10	bdd / bdd_hybrid	$2^{51.9}$	72
350	21	5	bdd	$2^{65.2}$	120
	27	10	bdd / bdd_hybrid	$2^{50.2}$	68