SALSA, PICANTE, and VERDE
(spicy green salsa)
Machine Learning attacks on LWE with small sparse secrets

Kristin Lauter

Director, FAIR Labs North America

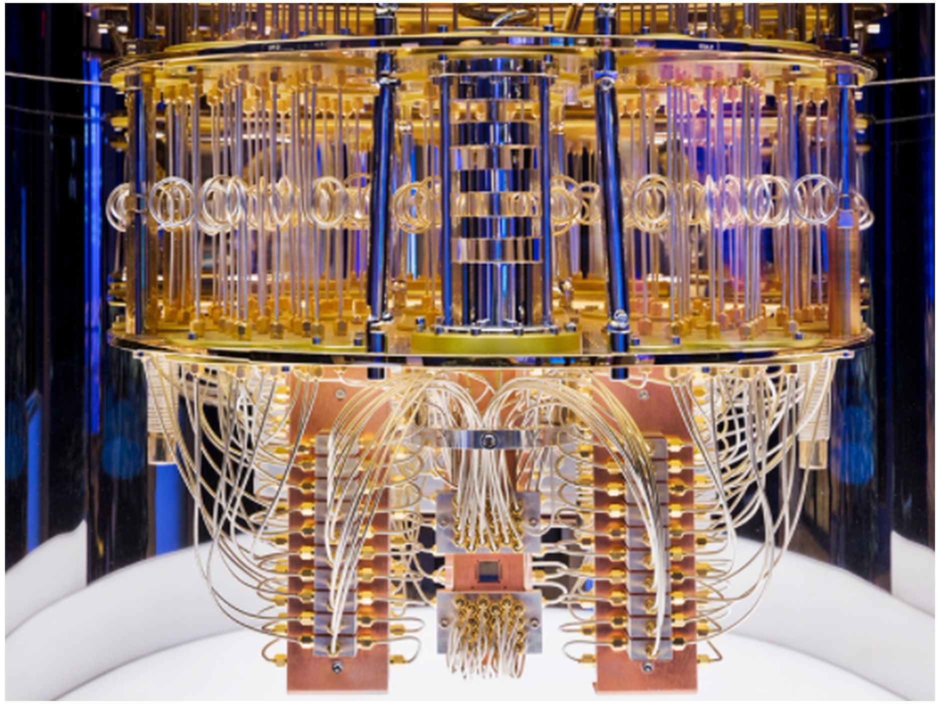
Joint work with: Francois Charton*, Mingjie Chen, Evrard Garcelon, Cathy Li, Mohamed Malhou, Jana Sotakova, Emily Wenger, Zeyuan Allen-Zhu

Meta AI

* combined author list from all papers, listed alphabetically

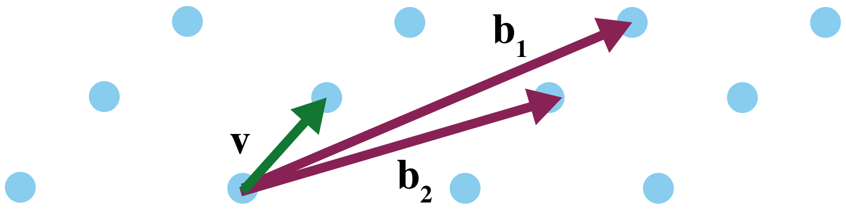
The race for post-quantum cryptography

- Full-scale quantum computers will break current public key encryption (RSA, ECC).
- NIST competition (2017-2022) standardized schemes for post-quantum cryptography.



The culprit: a quantum computer

Lattice cryptography: a leading post-quantum candidate



- **Lattice cryptography** schemes are believed to be quantum and classically secure.
- Lattice schemes rely on the Learning With Errors (LWE) [Regev 04] hardness assumption: $\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{s} + e \bmod q$
- Classical attacks use lattice reduction (e.g. LLL, BKZ)

Could we use a different attack paradigm?

LWE parameters

Hardness of LWE depends on selection of parameters:

n = dimension of lattice (e.g. n=256, 512, 1024)
q = modulus
e = error vector, sampled from Gaussian with std deviation σ
s = secret vector, sampled from secret distribution
m = # samples (a, b = a.s+e) attacker has access to

Examples where LWE is not hard:

- q is too large w.r.t. n: can use LLL polynomial time (O~(n^4)) algorithm to find s.

Examples of LWE in practice:

- NIST 2022 PQC standard:
  - Kyber 512, 768. RLWE with module structure, $n=256$, $k = 2,3$.
  - Log $q = 12$
  - Binomial secret distribution (for $k=2$, $s_i = -2,-1,0,1$, or 2)
  - Small secret, but not sparse (e.g. ~37% zero bits)

Homomorphic Encryption Standard (HomomorphicEncryption.org 2018)

- Large dimension, $n=1024, 2048, \ldots 2^{15}, 2^{16}$
- Log $q > 30$ …
- Binary, ternary, Gaussian, random secret distributions
- Small error, $\sigma \sim 3.2$
- Some small secret distributions are standardized
- Sparse secrets *not* in standard, but used in practice ($h=64$)

Classical attacks on LWE

- Standards set by estimating classical lattice reduction attacks
  - LLL, BKZ, fplll, BZK 2.0, … [LLL, Schnorr, Stehle, Chen-Nguyen, …]
  - Using LWE Estimator [Albrecht-Player-Scott 2015 ++]
- Concrete secret-recovery attacks: uSVP, decoding, dual
  - All work by using lattice reduction to find *the shortest vector* or a “short enough” vector
- BKZ improves LLL, increases *blocksize*, but exponential in *blocksize*

Could we use a different attack paradigm?

Use Machine Learning (ML) to attack LWE?

Learning with errors (LWE)

$$(\boldsymbol{a} \cdot \boldsymbol{s} + e) \text{ mod } q = b$$

LWE attack goal

Given LWE samples $\{(\boldsymbol{a}, b)\}$, recover $\boldsymbol{s}$.

Key attack intuitions

1. LWE assumes learning from noisy data is hard;

   but ML models are good at learning from noisy data!
2. LWE is like linear regression *but modulo q!*
3. ML models can do other math[1], *but not good at modular arithmetic!*

[1] e.g. *Linear Algebra with Transformers*, Charton, 2020

Our initial work: SALSA [WCCL NeurIPS 2022]

- ML-based attack on LWE with *sparse binary secrets*, e.g. $\boldsymbol{s} \in [0,1]^N$
  - Uses transformer models
  - Models trained on LWE samples $(\boldsymbol{a}, \boldsymbol{b})$ to predict $\boldsymbol{b}$ from $\boldsymbol{a}$
  - Develop cryptographic distinguishers which use the models as oracle
- SALSA (2022) recovers sparse binary secrets for small size LWE problems

Secret-recovery Attacks on LWE via Sequence-to-sequence models with Attention

SALSA: Attacking Lattice Cryptography with Transformers, Wenger et al, NeurIPS 2022

SALSA ingredients

🧅 **Transformer model** → train model on LWE samples

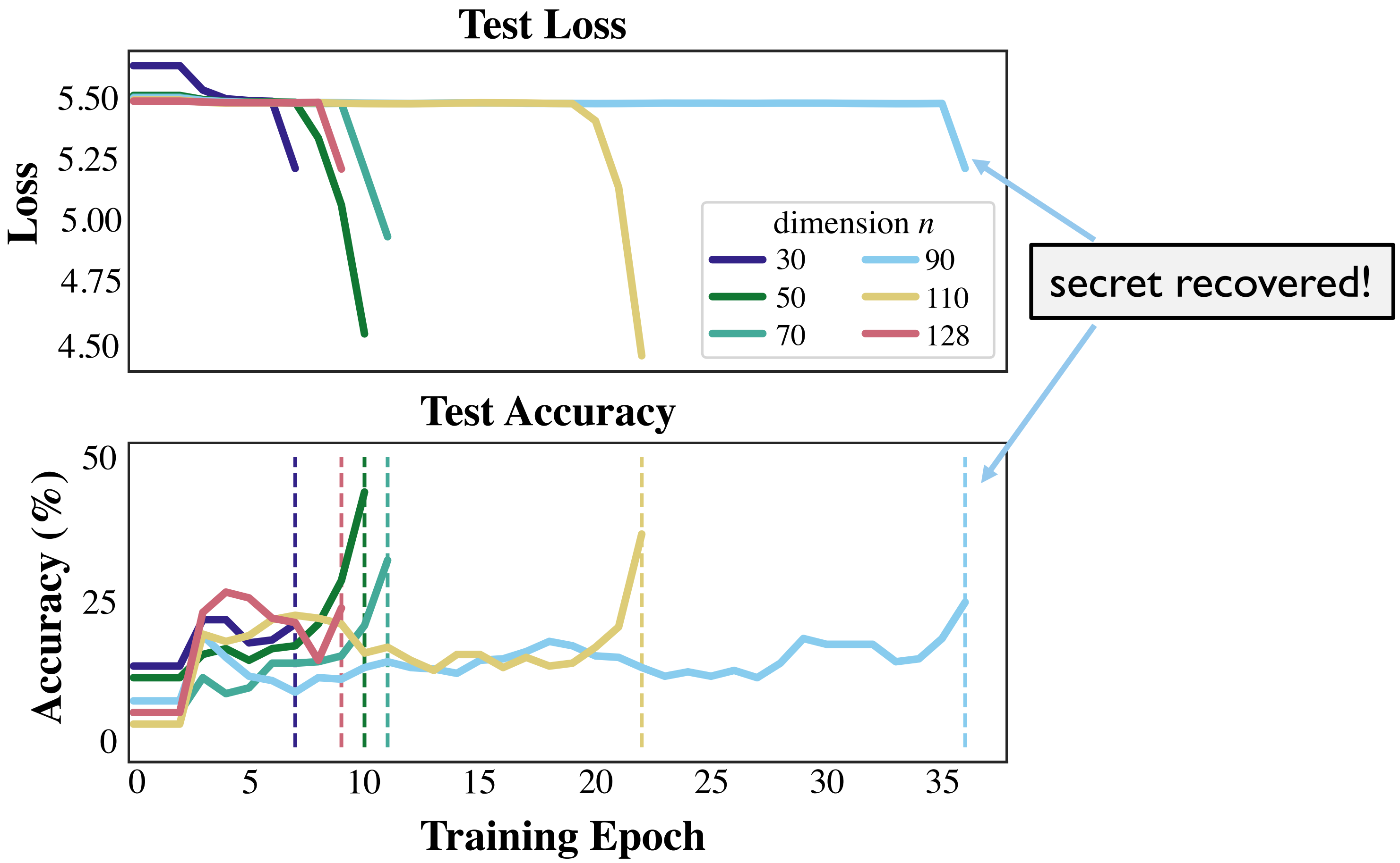
🫑 **Secret recovery** 🌶️ → extract secret prediction from model

🍅 **Secret verification** → check if secret is correct

SALSA performance

SALSA can successfully recover sparse, binary secrets for small LWE problems

- SALSA recovers secrets when model starts to learn
- High accuracy not needed for secret recovery



Model accuracy/loss on (R)LWE problems with Hamming weight 3

Secret Distinguishers

3 *distinguishers* enable secret recovery from trained model F.

- Direct
- Distinguisher
- Cross-attention [Picante]

<u>High level distinguisher idea:</u>

Let s_i be a bit in secret $\boldsymbol{s}$ and a_i corresponding coordinate of input $\boldsymbol{a}$.

Let $\mathbf{a_c}$ be a with constant c added to entry a_i

If $s_i = 0$ then $F(\mathbf{a_c}) \approx F(\boldsymbol{a})$,

where $\boldsymbol{F}$ is the model and $\boldsymbol{c}$ is a constant.

Key limitations of SALSA

- Significant data requirements (4 million LWE samples)
- Small dimension (best n=128)
- Low Hamming weight (best h=5)
- Binary secrets only

Our subsequent work, PICANTE and VERDE, addresses these limitations.
Now, SALSA-like attacks are closer to attacking real-world systems.

SALSA, PICANTE, VERDE

Attack version	LWE samples required	Attackable h/density	Attackable dimension	Secret types recovered
SALSA [1]	4,000,000	$h \leq 5, d \leq 0.05$	$n \leq 128$	Binary
PICANTE [2]	4n	$h \leq 60, d \approx 0.2$	$n \leq 350$	Binary
VERDE [3]	4n	$h \leq 63, d \approx 0.1$	$n \leq 512$	Binary, Ternary, Gaussian*

[1] *SALSA: Attacking Lattice Cryptography with Transformers*, Wenger et al, NeurIPS 2022
[2] *SALSA PICANTE: a machine learning attack on LWE with binary secrets*, Li et al, 2023, under review
[3] *SALSA VERDE: a machine learning attack on Learning with Errors with sparse small secrets*, Li et al, 2023, under review

From 4,000,000 to 4n samples

PICANTE and VERDE run a novel *preprocessing* step to improve data efficiency.

$$\begin{matrix}(\mathbf{a_1}, b_1)\\(\mathbf{a_2}, b_2)\\ \vdots \\(\mathbf{a_{4n}}, b_{4n})\end{matrix} \Rightarrow \begin{bmatrix}\mathbf{a}_2\\ \mathbf{a}_{3n-1}\\ \mathbf{a}_{2n+2}\\ \mathbf{a}_{n-1}\\ \cdots \\ \mathbf{a}_{4n-3}\end{bmatrix} = \mathbf{A_i} \Rightarrow \mathrm{R_{BKZ}}\mathbf{A_i}$$

1 4n LWE samples

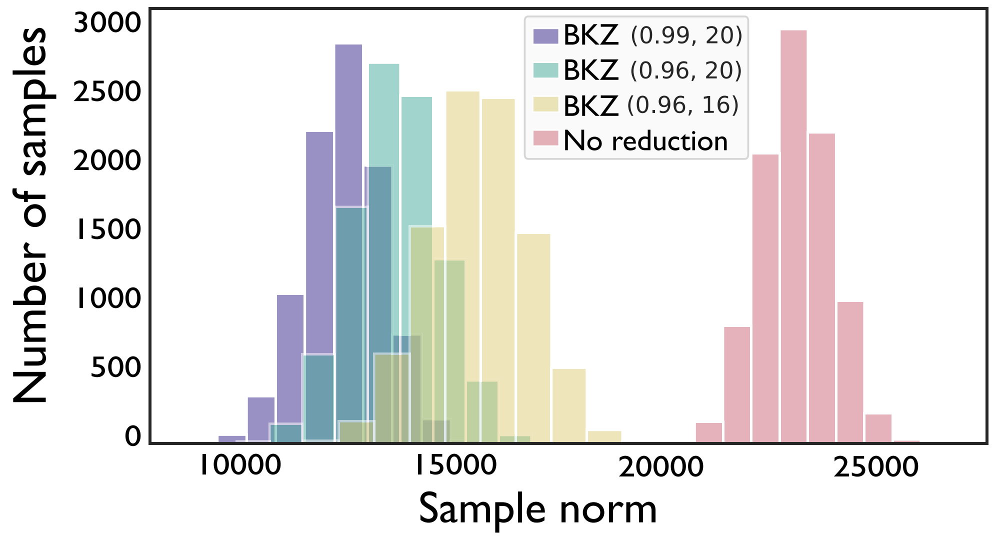
2 Resample $\mathbf{a}s$ to make $\binom{4n}{n}$ matrices

3 Reduce with BKZ

Reducing Standard deviation

Post-preprocessing, $\boldsymbol{a}$ coordinates have standard deviation smaller than uniform random.

Running BKZ with increasing strength reduces the standard deviation α of entries of vectors, making learning easier.



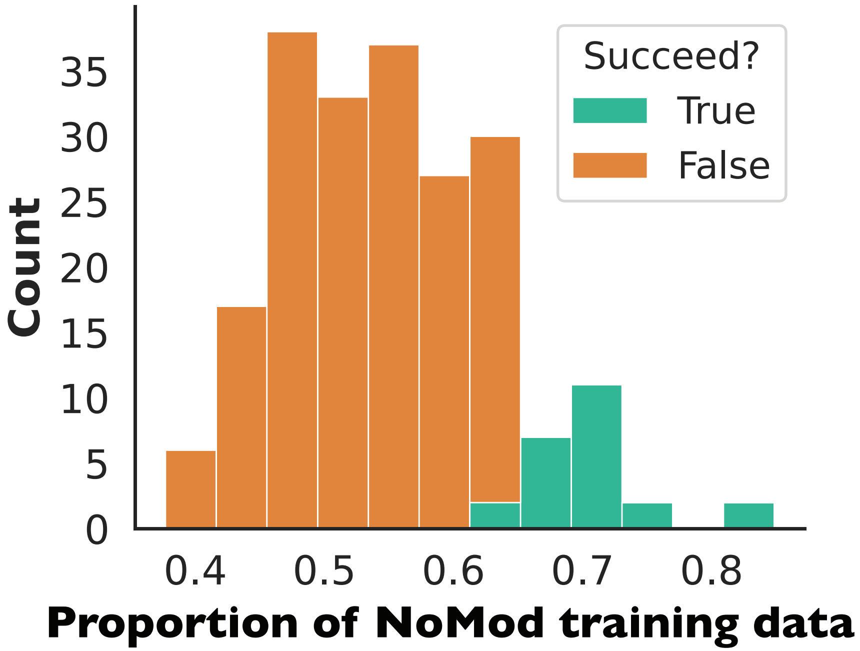
legend = BKZ (delta, block size)

Increasing attackable h

NoMod results suggest range of attackable h based on training data property.

NoMod = % of training data for which
$|x| = a \cdot s - b < q/2$
% training data not wrapped around modulus

When NoMod > 67%, success is almost guaranteed. Increasing h decreases NoMod, since more $\boldsymbol{a}$ elements are used to compute b.



But note, distribution does not need to be centered at 0, it just needs to be concentrated!

Increasing attackable h

Theory result: σ of training data determines recoverable h.

- If $|\mathrm{x}| = \mathrm{a} \cdot \mathrm{s} \ - \mathrm{b} \ < \mathrm{q}/2$ is a normally distributed random variable, 68% of its values will be within one $\boldsymbol{\sigma}$ of the mean. This is also the observed NoMod success threshold. Thus, we want $\boldsymbol{\sigma_x} < q/2$.
- If $\boldsymbol{s}$ is binary with Hamming weight $\boldsymbol{h}$ and entries of $\boldsymbol{a}$ have stdev $\boldsymbol{\sigma_a}$, then $\boldsymbol{\sigma_x} = \sqrt{h}\, \sigma_a + \sigma_e \approx \sqrt{h}\, \sigma_a$ (since $\boldsymbol{\sigma_e}$ is negligible).
- Therefore, s is recoverable if $\sqrt{h}\, \sigma_a < q/2$ or $\sigma_a = \frac{q}{2\sqrt{h}}$.
- This result highlights the importance of preprocessing: when $\boldsymbol{\sigma_a}$ is reduced by factor $\boldsymbol{\alpha}$, recoverable h increases by a factor of $\boldsymbol{\alpha^2}$!

Recovering ternary secrets

Our novel *two-bit distinguisher* enables recovery of more complex secrets.

<u>High level idea:</u> Let s_i, s_j be bits in secret $\boldsymbol{s}$ and a_i, a_j be corresponding coordinates of input $\boldsymbol{a}$.

If $s_i = s_j$ **then** $F(a_i + c) \approx F(a_j + c)$,

where F is the model and c is a constant.

Recovering ternary secrets

Detailed two-bit distinguisher method:

- First, identify nonzero secret bits using binary distinguisher.
- Compare nonzero bits pairwise using the $F(a_i + c)$ intuition.
- Partition nonzero bits into two cliques based on similarities/differences in observed $F(a_i + c)$ vs. $F(a_j + c)$.
- Set bits in one clique to 1 and the others to -1, check secret correctness.
- If correct, secret recovered! If not, continue training.

Scaling up dimension

We improve our choice of encoding base to attack larger dimensions.

- As modulus q increases, required transformer vocabulary size also increases.
- Transformers can't learn vocabularies with millions/billions of characters! Too complex!
- To reduce vocab size, we encode integers on two tokens, using base $B \geq \sqrt{q}$.
- When q is very large, we round the second bit using rounding token r, chosen so that vocabulary size $\frac{B}{r} < 10{,}000$.

Comparison with classical attacks

PICANTE and VERDE run faster than classical attacks but require more compute.

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

Comparison of Verde with concrete uSVP attack

n=256, binary secrets

Verde's preprocessing time assumes full parallelization.

Future directions

- Generalize to general (non-sparse) small secret distributions.
- Scale to smaller q

How?

- Improve models' ability to learn modular arithmetic.
- Decrease preprocessing compute/time requirements.
- Concentration methods for distribution

Thank you!

Questions?