Support Vector Machines as Adversaries for the Learning with Errors Decision Problem

Parker Szachta

December 20, 2024

Abstract

The learning with errors (LWE) search problem is thought to be a suitable framework for post-quantum cryptography, because time required to achieve an exact solution for it is not known to be less than exponential. There exists an LWE decision problem which can be reduced from the LWE search problem, so this paper applies a support vector machine (SVM) as an adversary for LWE decision. To deal with data that is not linearly separable, the non-linear RBF kernel was used. A grid search among hyperparameters was applied, leading to an accuracy confidence interval of $(0.51623915, 0.51735885)$ with a significance level of $\alpha = 0.99$, which represents an advantage of greater than 1.623%.

1 Names

The project was completed by Parker Szachta. Guidance was given from Prof. Sanjam Garg and Bhaskar Roberts.

2 Introduction

2.1 Learning with Errors (LWE)

Quantum computing is a rapidly evolving field, and research involving it has included its interactions with machine learning methods [16]. In 1999, Peter W. Shor announced an algorithm that uses a quantum computer to solve the discrete logarithm problem in polynomial time, a result that threatens to violate the Diffie-Hellman assumption [15]. Therefore, an ongoing problem has been to identify a new assumption that is more resistant to quantum adversaries.

One such candidate for a hardness assumption involves the learning with errors (LWE) search problem, which incorporates randomness and modular arithmetic. The LWE search problem involves a fixed dimension n, a fixed value q, and a fixed message space $\mathcal{M}$. An adversary is tasked with identifying a message $s \in \mathcal{M}$, given ordered pairs (a, b) such that $a \in \mathbb{Z}_q^n$ is uniformly random, e is an error term, and $b = (a \cdot s + e) \bmod q \in [0, q)$ [18]. Finding the exact message s is at least as hard as solving the shortest vector problem (SVP), which is not known to be faster than $O(n^{3.199n})$ [13]. In this sense, finding an exact solution for LWE search is not known to be computationally efficient.

2.2 Secret-Recovery Attacks on LWE Via Sequence to Sequence Models with Attention (SALSA)

However, the transformer, a machine learning architecture that processes how input tokens relate to each other in an attention-based mechanism, has successfully worked as part of an empirical adversary, named Secret-recovery Attacks on LWE via Sequence to sequence models with Attention (SALSA), against LWE search. For instance, the SALSA model was able to completely guess s when $n = 70$ and $\log_2(q) = 15$. The SALSA model works by training a transformer to discern between LWE search data and truly random data and then employing two algorithms to recover the message. Because an adversary may only be given a small amount of (a, b) pairs, the SALSA paper discusses how to combine training data to create more training data for the adversary [18]. Overall, SALSA demonstrates that creating, training, and testing a model can allow it to do the work of learning relationships between a and b to guess messages with non-negligible advantages.

One challenge with the SALSA model involves scaling. Let d be the proportion of bits in the binary representation of s that are 1. As d increases, the message becomes less sparse, and it becomes more difficult to uncover s. Moreover, the SALSA paper noted that expanding message recovery to non-binary messages is an open problem [18].

There have been improvements to the original SALSA model to be a more effective LWE adversary. For instance, SALSA PICANTE applied data preprocessing, resulting in a higher value of d [11]. Similarly, the SALSA VERDE paper applied data preprocessing, but it also applied a lower value of q. SALSA VERDE achieved the uncovering of ternary messages, as opposed to solely binary messages, as well as a speed increase relative to SALSA PICANTE [12]. The SALSA FRESCA paper incorporated pretraining [17]. All of these papers use the transformer architecture.

2.3 Support Vector Machines (SVMs)

The SALSA papers used a transformer for LWE, but there are other, more traditional machine learning models which could be used in attempts to be an LWE adversary. One option is a support vector machine (SVM), which is useful for binary classification. In particular, an SVM develops a hyperplane that divides a space into two regions: one per classification type.

There are two issues with applying a linear SVM to LWE search: it is not a binary classification task in the first place, and the data are not necessarily linearly separable.

The first issue is that SVMs are designed for a fundamentally different task than LWE is suited for. Therefore, for an SVM to be useful for LWE, an SVM needs to be applied in such a way that s can be extracted from the result. An SVM is useful for binary classification, but the goal of LWE is to identify a message $s \in \mathcal{M}$. This can be resolved with a similar problem to LWE search: LWE decision, which involves determining whether data points $(a, b) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$ are generated uniformly at random or are from the LWE process [18]. The LWE search problem can be reduced to the LWE decision problem [14], so this paper focuses on the LWE decision problem. To construct coordinates and classifications for the SVM, consider a space in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. The points in this space are of the form (a, b), where a and b are from the LWE problem or are created uniformly at random. If the former is true, let $(a, b) \in \mathcal{A}$. If the latter is true, let $(a, b) \in \mathcal{B}$. The binary classification problem is, for any

given data point, to determine whether it is in $\mathcal{A}$ or $\mathcal{B}$.

The second issue is due to the modulus q, because the values in b 'wrap around' q. As a toy example, assume $n = 1$, $s = 2$, $e = 0$, and $q = 7$. Possible ordered pairs of (a, b) could include $\{(1, 2), (2, 4), (3, 6), (4, 1)\}$. When a increases, so does b, until $as \geq q$, because b 'wraps around' q. Therefore, a hyperplane cannot linearly separate data between data points, since no such linear relationship exists. Fortunately, this can be resolved by using a non-linear kernel, such as a radial basis function (RBF), which maps data to a higher-dimensional space. In such a space, a linear hyperplane could be used.

Another solution to the second issue is to use a soft margin SVM. A soft margin SVM is very similar to a hard margin SVM, with the main difference being that soft margin SVMs allow for a hyperplane that does not linearly separate the data. A soft margin SVM has the potential to be quite useful for the LWE problem, because it is unlikely that the many high-dimensional points involved in the LWE problem are linearly separable.

This project aims to generate LWE decision data and assess the performance of soft margin SVMs as LWE decision adversaries. Tables were created in a similar format to those in SALSA, although SALSA was for the LWE search problem, not the LWE decision problem.

Even though the SVM mechanism has been used for cryptography in the past [10], the SALSA authors did not investigate SVMs for LWE. Therefore, as far as this project's author knows, the specific formulation of using SVM as an LWE decision adversary is novel. If SVMs are indeed effective at being an adversary for LWE decision, then this project is useful, because researchers would be able to know a limitation of LWE's effectiveness, before bad actors do.

An SVM has promise for being an LWE decision adversary, because the SALSA mechanism struggled with increased dimensionality n [18], but SVMs are even more effective with high dimensionality. Also, SVMs are relatively effective in applying their findings from training data to testing data. This is especially true for soft-margin SVMs [5].

3 Results

Below are the confidence intervals for the Accuracy, Precision, Recall, and F1 Score given 50 trials, $h = 3$, $n = 30$, and $q = 845813581$. Significance levels are all values $\alpha \in \{0.9, 0.95, 0.99\}$. For each confidence interval, let L be its lower bound and U be its upper bound. A green cell indicates that $L > 0.51$, a yellow cell indicates that $L < 0.51 < U$, and a red cell indicates that $U < 0.51$. The context for these results are explained in the Technical Overview section below.

$\log_{10} C$	$\log_{10} \gamma$	**Accuracy**	**F1 Score**	**Precision**	**Recall**
0	**0.5**	(0.516203, 0.5169694)	(0.51750141, 0.51830729)	(0.516113, 0.51688145)	(0.51879159, 0.51985401)
0	**1**	(0.51433462, 0.51504698)	(0.50846094, 0.51049346)	(0.51464509, 0.51537763)	(0.5021886, 0.506069)
0	**1.5**	(0.5076205, 0.5082295)	(0.32730996, 0.35878022)	(0.51546879, 0.51733958)	(0.24409533, 0.28890547)
1	**0.5**	(0.51643297, 0.51714863)	(0.51620364, 0.51698205)	(0.51644621, 0.51716323)	(0.51590024, 0.51686616)
{1, . . . , 9}	**1**	(0.51444374, 0.51514826)	(0.5105768, 0.51219571)	(0.51464723, 0.51536551)	(0.5063426, 0.5093718)
{1, . . . , 9}	**1.5**	(0.50830789, 0.50901811)	(0.38749745, 0.41650079)	(0.51307148, 0.51497857)	(0.316683, 0.3649642)
{2, . . . , 9}	**0.5**	(0.51644149, 0.51715651)	(0.51621281, 0.51698988)	(0.51645471, 0.51717112)	(0.51591031, 0.51687369)

Table 1: Confidence Intervals for Metrics with $\alpha = 0.9$

$\log_{10} C$	$\log_{10} \gamma$	**Accuracy**	**F1 Score**	**Precision**	**Recall**
0	**0.5**	(0.51612959, 0.51704281)	(0.51742421, 0.51838448)	(0.51603939, 0.51695506)	(0.51868983, 0.51995577)
0	**1**	(0.51426639, 0.51511521)	(0.50826626, 0.51068814)	(0.51457492, 0.5154478)	(0.50181691, 0.50644069)
0	**1.5**	(0.50756216, 0.50828784)	(0.32429553, 0.36179466)	(0.5152896, 0.51751877)	(0.23980311, 0.29319769)
1	**0.5**	(0.51636442, 0.51721718)	(0.51612908, 0.51705661)	(0.51637753, 0.51723192)	(0.51580772, 0.51695868)
{1, . . . , 9}	**1**	(0.51437625, 0.51521575)	(0.51042173, 0.51235078)	(0.51457843, 0.51543432)	(0.50605245, 0.50966195)
{1, . . . , 9}	**1.5**	(0.50823986, 0.50908614)	(0.38471931, 0.41927892)	(0.51288881, 0.51516124)	(0.3120583, 0.3695889)
{2, . . . , 9}	**0.5**	(0.516373, 0.517225)	(0.51613838, 0.51706431)	(0.51638608, 0.51723974)	(0.51581803, 0.51696597)

Table 2: Confidence Intervals for Metrics with $\alpha = 0.95$

$\log_{10} C$	$\log_{10} \gamma$	**Accuracy**	**F1 Score**	**Precision**	**Recall**
0	**0.5**	(0.51598612, 0.51718628)	(0.51727335, 0.51853535)	(0.51589553, 0.51709892)	(0.51849094, 0.52015466)
0	**1**	(0.51413303, 0.51524857)	(0.50788575, 0.51106865)	(0.51443778, 0.51558494)	(0.50109047, 0.50716713)
0	**1.5**	(0.50744815, 0.50840185)	(0.31840398, 0.3676862)	(0.51493937, 0.517869)	(0.23141422, 0.30158658)
1	**0.5**	(0.51623045, 0.51735115)	(0.51598335, 0.51720234)	(0.51624329, 0.51736615)	(0.5156269, 0.5171395)
{1, . . . , 9}	**1**	(0.51424436, 0.51534764)	(0.51011865, 0.51265386)	(0.51444396, 0.51556879)	(0.50548535, 0.51022905)
{1, . . . , 9}	**1.5**	(0.50810691, 0.50921909)	(0.37928961, 0.42470863)	(0.51253179, 0.51551827)	(0.30301959, 0.37862761)
{2, . . . , 9}	**0.5**	(0.51623915, 0.51735885)	(0.5159929, 0.51720979)	(0.51625197, 0.51737386)	(0.51563767, 0.51714633)

Table 3: Confidence Intervals for Metrics with $\alpha = 0.99$

4 Technical Overview

4.1 SVMs as Optimization Problems

A hard margin SVM takes as input n data points (x_i, y_i) such that $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$, for all $i \in [n]$, where p is the dimension of the space that each x_i exists in. It then develops a hyperplane $f(x) = w \cdot x + b$, where $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$, such that for all $i \in [n]$, if $w \cdot x_i + b > 0$, then $y_i = 1$, and if $w \cdot x_i + b < 0$, then $y_i = -1$. The minimal distance of any point to the hyperplane is maximized. This results in the optimization problem $\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} ||w||^2$ subject to the constraints that for all $i \in [n]$, if $w \cdot x_i + b > 1$, then $y_i = 1$, and if $w \cdot x_i + b < -1$, then $y_i = -1$ [20].

On the other hand, soft margin SVM employs a loss function and allows for some forgiveness when a point is misclassified. A soft margin SVM operates under the optimization problem $\min_{w \in \mathbb{R}^p,\ b \in \mathbb{R}} \frac{1}{2} C||w||^2 + \frac{1}{n} \Sigma_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i + b)\}$, where $C > 0$ is a hyperparameter [20]. The value C can be controlled to adjust for the amount of forgiveness given to misclassified points [3]. Therefore, a soft margin SVM was used for this project.

To allow the SVMs to understand a more rich understanding of the inputted data, an RBF kernel was used. In other words, the SVMs were allowed to learn nonlinear decision boundaries. A kernel is a method of determining how similar two data points are to each other. Changing the kernel allows for more rich representations of similarity besides the standard dot product. The RBF kernel uses the function $K(x_1, x_2) = e^{-\gamma' \cdot ||x_1 - x_2||^2}$, where γ' is a hyperparameter that determines the strength of each individual data point's affect on the decision boundary [2]. Therefore, in `scikit-learn`, the package that was used for the SVM, the default value of $\gamma' = \frac{1}{p \text{Var}(\{x_i \mid i \in [n]\})}$ [3], where Var is the variance function. The grid search involved values of γ, where $\gamma' = \frac{\gamma}{p \text{Var}(\{x_i \mid i \in [n]\})}$.

4.2 Dataset Generation and Hyperparameter Choices

Various datasets, of which there were 50, were generated to be trained on an SVM with an RBF kernel. Each dataset contains the same distributions of amounts of data points, in Table 4 below. Since exactly 50% of the data was generated via the LWE distribution, a fully naive classifier would achieve 50% accuracy on the LWE decision problem.

Data	**LWE**	**Uniform**
Train	250000	250000
Test	50000	50000

Table 4: Dataset Split Distribution

Each LWE instance in the same dataset was based on the same binary message, $s \in \{0,1\}^n$, with $n = 30$. However, different datasets could have different messages, since each dataset's message was sampled identically and independently at random. Each value of s had exactly $h = 3$ bits that are 1, resulting in the density being $d = \frac{h}{n} = \frac{3}{30} = 0.1 \in [0.002, 0.15]$, in line with the set provided by the SALSA paper [18]. The remaining bits were 0. For each data point, the value a was generated uniformly at random from $\mathbb{Z}_q^n$ with $q = 845813581$, aligning with one of the q values used for SALSA [18]. If the data point was to be an LWE point, the error e was generated by first sampling from a Gaussian distribution with mean 0 and standard deviation $q/1000$ and then rounding it to an integer. The value b was then computed via the equation $b = (a \cdot s + e) \bmod q$, and the data point was the concatenation of a and b, again aligning with SALSA [19]. If the data point was to be a uniformly at random point, then the data point was the concatenation of $n + 1$ integers sampled identically and independently from $\mathbb{Z}_q$.

A grid search for the regularization hyperparameters, C and γ, was conducted over $C \in \{10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9\}$ and $\gamma \in \{10^{0.5}, 10^1, 10^{1.5}\}$. These values were hand-picked after manual evaluation of hyperparameters. For each dataset and for each value of (C, γ), an SVM with an RBF kernel was then fit to the training data. The features were the data points, the labels were 1 for LWE points, and the labels were 0 for uniformly random points.

4.3 Metrics

For each dataset and each value of (C, γ), the accuracy, F1 Score, precision, and recall of the SVM were collected. Confidence intervals for each calculated metric for each value of (C, γ) were calculated with $\alpha \in \{0.9, 0.95, 0.99\}$.

Consider a data point to be positive if it is an LWE instance and negative otherwise. Let T_P be the number of True Positive points, T_N be the number of True Negative points, F_P be the number of False Positive points, and F_N be the number of False Negative points. Using these representations, metrics were calculated in accordance with the following calculations:

$$\text{Accuracy} = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} [6]$$

$$\text{Precision} = \frac{T_P}{T_P + F_P} [7]$$

$$\text{Recall} = \frac{T_P}{T_P + F_N}[8]$$

$$\text{F1 Score} = \frac{2T_P}{2T_P + F_N + F_P}[9]$$

Note that F1 Score $= \frac{2T_P}{2T_P+F_N+F_P} = \frac{1}{\frac{2T_P+F_N+F_P}{2T_P}} = \frac{1}{\frac{T_P+F_P}{2T_P}+\frac{T_P+F_N}{2T_P}} = \frac{2}{\frac{T_P+F_P}{T_P}+\frac{T_P+F_N}{T_P}} = \frac{2}{\frac{1}{\frac{T_P}{T_P+F_P}}+\frac{1}{\frac{T_P}{T_P+F_N}}} = \frac{2}{\frac{1}{\text{Precision}}+\frac{1}{\text{Recall}}}$.[9]

4.4 Confidence Interval Conditions

This experiment has met most of the relevant conditions for confidence intervals. First of all, the data points are all independently sampled at random, assuming the "random" Python package does so. Also, the number of trials is $t = 50$, which is at least the recommended amount of 30, and the sample size is less than 0.1 times the population size. Additionally, one requirement is that if p is the expected probability from a confidence interval, $pt \geq 10$ and $(1-p)t \geq 10$. The lowest lower bound of a confidence interval in the results is 0.23141422, so $p \geq 0.23141422$. Also, the greatest upper bound of a confidence interval in the results is 0.52015466, so $p \geq 0.52015466$, so $1 - p \geq 1 - 0.52015466$. Therefore, $pt \geq 0.23141422 \cdot 50 \geq 10$ and $(1-p)t \geq (1 - 0.52015466) \cdot 50 \geq 10$ [4].

4.5 Adversarial Advantages

For each $\alpha \in \{0.9, 0.95, 0.99\}$, the confidence interval for accuracy with the greatest lower bound occurred when $\log_{10} C \in \{2, \ldots, 9\}$ and $\log_{10} \gamma = 0.5$. These confidence intervals were $(0.51644149, 0.51715651)$, $(0.516373, 0.517225)$, and $(0.51623915, 0.51735885)$, respectively.

A fully naive adversary for the LWE decision problem would have an expected accuracy of $\frac{1}{2} = 0.5$. Therefore, the SVM adversary's advantage is $A - 0.5$, where A is the SVM adversary's accuracy. The following table demonstrates the SVM adversary's advantages.

α	**Accuracy**	**Advantage**
0.9	$(0.51644149, 0.51715651)$	$(0.01644149, 0.01715651)$
0.95	$(0.516373, 0.517225)$	$(0.016373, 0.017225)$
0.99	$(0.51623915, 0.51735885)$	$(0.01623915, 0.01735885)$

Table 5: Confidence Intervals for Accuracy and Advantages with $\log_{10} C \in \{2, \ldots, 9\}$ and $\log_{10} \gamma = 0.5$

4.6 Limitations

One limitation is that not all of the functions used for randomness in the experiments are cryptographically pseudorandom. The `random` Python package was used for this project for choosing which elements of s were 1 and for selecting values of e. This Python package is deterministic and not cryptographically secure [1], which could lead to the SVM models exploiting patterns in the data that would not be present in truly random data. In other

words, the data that was used could exhibit patterns that would not be present by an adversary that only has access to a truly random oracle.

Another limitation is that the reported results are for one specific choice of (h, q, n), namely $(3, 845813581, 30)$. These parameters are noted in the SALSA paper [18], but it is possible that any cryptographic advantage would disappear when selecting other parameters. In such a scenario, a challenger could simply select one of these different parameter sequences and defend against the LWE adversary.

4.7 Conclusion

This project demonstrates that there are limits on using the LWE decision problem for security purposes, regardless of whether such use is in the post-quantum landscape. With the specified choices of parameters and hyperparameters, SVM adversaries were able to reach an advantage of greater than 1.623%, which is not negligible. Further research should investigate how the advantage changes as the hyperparameters change as well as whether additional compute has a meaningful affect on an SVM adversary's advantage.

The program that contains the code for the research, `generation.py`, was run using an NVIDIA A100 GPU through Google Colab Pro+. Code was optimized to run on a GPU so that it would run faster than on a CPU. Code and results data for this project are available in the GitHub repository at `https://github.com/parkszachta/lwe`. Specifically, the Python code is in the `generation.py` file, and results data is in the `results.db` file.

References

[1] random — generate pseudo-random numbers. `https://docs.python.org/3/library/random.html`.

[2] Rbf. `https://scikit-learn.org/1.5/auto_examples/svm/plot_svm_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py`.

[3] Svc. `https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html`.

[4] Z. Bobbitt. The 6 confidence interval assumptions to check. `https://www.statology.org/confidence-interval-assumptions/`, 06 2021.

[5] goelaparna1520. Support vector machine in machine learning. `https://www.geeksforgeeks.org/support-vector-machine-in-machine-learning/`.

[6] A. Gramfort, M. Blondel, O. Grisel, A. Joly, J. Wersdorfer, L. Buitinck, J. Nothman, N. Dawe, J. Shah, S. Jha, B. Stein, S. Yao, and M. Karbownik. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`.

[7] A. Gramfort, M. Blondel, O. Grisel, A. Joly, J. Wersdorfer, L. Buitinck, J. Nothman, N. Dawe, J. Shah, S. Jha, B. Stein, S. Yao, and M. Karbownik. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html`.

[8] A. Gramfort, M. Blondel, O. Grisel, A. Joly, J. Wersdorfer, L. Buitinck, J. Nothman, N. Dawe, J. Shah, S. Jha, B. Stein, S. Yao, and M. Karbownik. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html`.

[9] A. Gramfort, M. Blondel, O. Grisel, A. Joly, J. Wersdorfer, L. Buitinck, J. Nothman, N. Dawe, J. Shah, S. Jha, B. Stein, S. Yao, and M. Karbownik. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`.

[10] K. H. Han, W.-K. Lee, A. Karmakar, J. M. B. Mera, and S. O. Hwang. cufe: High performance privacy preserving support vector machine with inner-product functional encryption. *IEEE Transactions on Emerging Topics in Computing*, 12(1):328–343, 2024.

[11] C. Li, J. Sotáková, E. Wenger, M. Malhou, E. Garcelon, F. Charton, and K. Lauter. SALSA PICANTE: a machine learning attack on LWE with binary secrets. 10 2023.

[12] C. Y. Li, E. Wenger, Z. Allen-Zhu, F. Charton, and K. Lauter. SALSA VERDE: a machine learning attack on Learning With Errors with sparse small secrets. 10 2023.

[13] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. pages 1468–1480, 1 2010.

[14] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[15] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. 41(2), 1999.

[16] V. Sood and R. P. Chauhan. Archives of Quantum Computing: Research Progress and Challenges. 31:73–91, 2024.

[17] S. Stevens, E. Wenger, C. Li, N. Nolte, E. Saxena, F. Charton, and K. Lauter. SALSA FRESCA: Angular Embeddings and Pre-Training for ML Attacks on Learning With Errors. 2 2024.

[18] E. Wenger, M. Chen, F. Charton, and K. Lauter. SALSA: Attacking Lattice Cryptography with Transformers. 2023.

[19] E. Wenger, E. Saxena, M. Malhou, E. Thieu, and K. Lauter. Benchmarking attacks on learning with errors. Cryptology ePrint Archive, Paper 2024/1229, 2024. `https://eprint.iacr.org/2024/1229`.

[20] X. Zhang. *Support Vector Machines*, pages 1214–1220. Springer US, Boston, MA, 2017.