OXFORD JOURNAL
OF STUDENT
SCHOLARSHIP

Towards a Lightweight Machine Learning Classifier for Detecting Timing-Based Side-Channel Attacks on AES

Aditya Dhote
dhoteaditya1234@gmail.com

ABSTRACT

AES (Advanced Encryption Standard) is a widely used cryptographic algorithm that has implementations prone to side-channel attacks (SCA). These implementations leak data, specifically timing data, that can be used to recover parts of or the whole cryptographic key. This paper studies whether or not an SVM (Support Vector Machine) classifier can run on constrained, lightweight hardware with a low CPU overhead and high accuracy to detect artificially injected timing anomalies in AES execution on said hardware. Lightweight detection is needed for constrained systems, where security software must operate alongside existing workloads without introducing significant overhead. For this experiment, we injected AES timing data with artificial timing anomalies, then processed 6 statistical features of this data. We trained an SVM classifier on these features and noted the accuracy as well as inference time per window. The SVM classifier had an internal accuracy of 96.26%, and an external accuracy of 94.15%, with a 1.800 ms average per window inference time. These results suggest that lightweight ML-based timing analysis may be a feasible approach for classifying the simulated timing anomalies used in this experiment with low inference latency. However, this experiment does not directly reproduce physical timing side-channel attacks or key-dependent cache leakage, which is a limitation and an area for future research.

INTRODUCTION

Cryptography protects sensitive data using encryption algorithms like AES. AES is a highly regarded symmetric encryption algorithm that is widely used, and was chosen to replace the old and aging encryption algorithm of Triple-DES. AES is a symmetric block cipher that uses a secret key to encrypt and decrypt plaintext securely. As a symmetric block cipher, AES uses key sizes of 128, 192, or 256 bits to encrypt a block of plaintext, which is 128 bits [2]. Modern 64 bit processors contain native instruction sets for AES, making it the standard for almost everyone [1,2]. The mathematical security of AES is considered to be strong, ensuring a uniform treatment of data by dividing it up into 4x4 columns, or 16 bytes, then undergoing byte substitution, shifting rows to introduce a layer of complexity, mixing data using a Hill cipher variant, then finally undergoing an XOR operation with the first round key, obtained from the initial key [1]. However, even though AES is mathematically secure, certain implementations are

more prone to attacks, especially AES-256, with the related-key attack. Lower complexity attacks (2^99.5) are also effective on simplistic or flawed implementations of AES [1]. Although these are effective attack vectors on AES, we will focus on side-channel attacks, which will be introduced next.

Encryption algorithms introduce other data and byproducts that, although seemingly unrelated to the average person, can be used to crack encryption on electronic devices. Byproducts include "power consumption, electromagnetic emanations, thermal signatures, and/or other phenomena and the encryptions taking place on the device" [3]. This data often has relationships to other similar data, and can be exploited to replicate encryption keys. This is called a side-channel attack (SCA). One example of a power consumption SCA is simple power analysis (SPA), where someone looks into electrical activity graphs of a device running cryptography to see information about the device and the encryption code it uses [3]. This research focuses specifically on timing-based leakage.

To explain timing-based leakage from encryption and how it occurs, we'll be looking at Kocher's work on this. The biggest thing to note is that timing side-channel attacks are based on cryptographic implementations that do not run in fixed time. Because they do not run in fixed time, we are able to deduce parts of the secret key from variations in timing data. These variations in timing data come from RAM cache hits and misses, where minute timing data details can be used to reveal sensitive keys [4]. For AES specifically, it combines user data, *k*, with secret key byte, *n*. The user, over numerous attempts, can record the time it takes for the encryption to finish the XOR operation k[0] XOR n[0]. By measuring the timing data for encryption, and using multiple different *k* values, the attacker can find out which table slots were accessed and get the secret key byte, *n* [5].

Existing research demonstrates that machine learning can be applied to IoT for intrusion detection, specifically an SVM, with a high accuracy and only 3 data features [7]. However, many approaches emphasize highly capable models, large datasets, or physical side-channel measurements such as power and electromagnetic traces [3]. This raises the question of whether a comparatively lightweight classifier can identify timing-based leakage using limited computational resources.

This paper investigates whether a lightweight machine-learning classifier can distinguish normal AES execution from execution exhibiting artificially introduced timing leakage. Timing measurements are collected from AES operations and divided into fixed-size observation windows. Features extracted from these measurements are used to train and evaluate a lightweight classifier. Welch's t-test was used to establish the presence of statistically significant timing leakage across the dataset, while a Support Vector Machine (SVM) classifier was trained on six extracted features from 30,000 50-sample windows to identify individual windows exhibiting attack behavior. We then measured the CPU overhead while it was running to make sure it stayed within the threshold.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 describes our methodology, Section 4 presents results, and Section 5 discusses findings and limitations.

PRELIMINARY TERMINOLOGY

Symmetric cryptography, like AES, is one in which the key is used to do both encryption and decryption [9]. Timing based side-channel leakage is when the execution time of AES, if the timing is based on the secret key and not constant-time, is leaked to an attacker, and they are able to infer the secret key through the variations in execution time [2]. A support vector machine (SVM) is a supervised ML method used for classification, whose complexity depends on the input vector dimensions [7].

RELATED WORKS

2.1 Timing-Based Side-Channel Attacks

Timing information has long been recognized as a potential source of side-channel leakage in cryptographic implementations. Kocher demonstrated that secret-dependent variations in cryptographic execution time could be exploited to recover information about secret keys in implementations of RSA (Rivest–Shamir–Adleman), Diffie-Hellman, DSS (Digital Signature Standard), and related systems [4]. In experiments involving RSA implementations from the RSAREF (RSA Reference) toolkit on a Pentium-based system, repeated measurements of modular exponentiation were analyzed to distinguish timing distributions associated with different secret exponents. The reported results demonstrated that timing measurements could provide sufficient statistical information to identify the correct exponent with a high success rate. However, the effectiveness of this approach depends on obtaining sufficiently accurate timing measurements and a large number of observations, making environmental noise and measurement variability important practical considerations [4].

While [4] established the feasibility of exploiting execution-time differences to recover cryptographic information, subsequent work has considered the complementary problem of detecting anomalous timing behavior rather than exploiting it for key recovery. In embedded real-time systems, timing deviations have been investigated as indicators of malicious activity. One approach used Worst-Case Execution Time (WCET) estimates to identify abnormal execution in defined code regions on a LEON3 processor implementing the SPARC V8 instruction set [8]. The system successfully detected the simulated attacks evaluated in the study, with detection latency approaching the Maximum Attack Window for the tested applications. The approach also introduced only 3–22% area overhead, depending on the memory configuration, while avoiding runtime performance overhead [8]. However, the method relied primarily on WCET-based monitoring and did not incorporate Best-Case Execution Time (BCET), which the authors identified as a potential avenue for improving detection efficiency. Furthermore, the experiments were limited to a single-processor environment, leaving questions regarding applicability to more complex multiprocessor and multitasking systems [8].

These results demonstrate that execution-time behavior can be monitored as a means of identifying side-channel-related activity, but they do not establish whether machine-learning models can classify

timing behavior at a finer granularity. This motivates research into automated classification techniques capable of identifying patterns within measured execution data.

2.2 Machine Learning for Side-Channel Detection

Machine learning provides an alternative to manually defined timing thresholds by allowing a classifier to learn distinctions between benign and malicious execution behavior from observed measurements. This approach has been investigated for cache-based side-channel detection, including the detection of Spectre attacks. One real-time detection system monitored hardware performance counters associated with CPU cache behavior and used a neural network to classify potentially malicious activity [6]. Implemented on an Intel Core i7-7820HQ system running Linux, the system achieved an overall classification accuracy of 99.23% while introducing approximately 4–5% CPU overhead [6]. These results demonstrate that machine learning can provide effective real-time classification of side-channel activity while maintaining relatively low computational overhead.

However, the approach in [6] relies on hardware performance counters and specifically trained Spectre attack implementations. Consequently, its effectiveness against other forms of side-channel leakage, including timing leakage originating from cryptographic execution, is not established. Additionally, measurements were collected at 100 ms intervals, introducing a detection window during which an attack could potentially execute before malicious behavior was identified [6]. Thus, although the work demonstrates the feasibility of real-time ML-based side-channel detection, it leaves open the question of whether more frequent, fine-grained timing measurements can be classified with sufficiently low latency.
A related objective has been explored in lightweight intrusion detection for resource-constrained Internet-of-Things (IoT) devices. One proposed IDS (Intrusion Detection System) used a Support Vector Machine (SVM) with a small set of statistical features derived from network traffic, reducing the computational requirements associated with more complex detection approaches [7]. The proposed SVM achieved comparable or improved classification accuracy relative to the evaluated alternatives while substantially reducing training and testing time [7]. Although this work does not address side-channel attacks, its use of compact statistical feature representations and lightweight classification is relevant to the design of security systems for resource-constrained devices. However, the experiments focused on simulated network traffic rather than cryptographic execution measurements or real-time side-channel behavior, leaving the applicability of this methodology to timing-based side-channel detection unresolved.

2.3 Research Gap

Collectively, prior research establishes three important capabilities. First, timing variations can expose information about cryptographic execution and can be exploited as a side channel [4]. Second, execution-time monitoring can be used to identify anomalous behavior in embedded systems [8]. Third, machine-learning classifiers can detect side-channel activity while maintaining relatively low computational overhead [6], and lightweight feature representations can reduce the computational requirements of security classifiers on resource-constrained devices [7].

However, these approaches address different aspects of the problem. Timing-attack research has primarily

focused on exploiting timing measurements to recover secret information, while timing-monitoring approaches have generally relied on predefined execution-time bounds rather than learned classification. Existing ML-based side-channel detection has demonstrated strong performance for cache-based attacks, but has not directly established the feasibility of classifying fine-grained AES execution-time measurements. Similarly, lightweight SVM-based security detection has been demonstrated for IoT network traffic, but not for cryptographic timing leakage [7].

This work addresses this intersection by investigating a lightweight machine-learning classifier for detecting timing-based side-channel leakage in AES execution. AES timing measurements are divided into fixed-size observation windows, from which a compact set of statistical features is extracted before classification. The resulting approach is evaluated in terms of classification performance and statistical evidence of timing leakage, with particular emphasis on whether a lightweight model can identify small timing deviations while remaining suitable for low-latency, resource-constrained deployment.

METHODOLOGY

3.1 Data Creation

To make the training and testing data, we used a python program to make two AES datasets. One is a constant time baseline that uses PyCryptodome and the other is a simulated version that introduces a random, generated delay between 5 and 50 ns, and introduces it into ~8% of samples. We chose an 8% number because it introduces a measurable timing difference without making the distributions trivially separable. Furthermore, the limited number of modified samples is a conservative test of the classifier, because timing-based side-channel attacks rely on detecting subtle variations rather than large and obvious changes. A vastly higher injection rate would make the attack signal too strong and make classification too simple for the classifier, misrepresenting the motivations of this experiment.

How we specifically generated the data is as follows: We made a python script to generate 30,000 windows each with 50 AES timing measurements. The script uses a fixed 128-bit AES key generated using *get_random_bytes(16)*, using AES in ECB (Electronic Codebook) mode. The script generates a new, random 16-byte plaintext for each encryption, then measures each encryption with *time.perf_counter_ns()*. The baseline data is collected using AES encryption with no artificial delay, and attack data is generated with an artificial delay, with each generated value having an 8% probability of receiving an artificial delay. This artificial delay is randomly selected from a range of 5-50 ns, and implemented as a busy-wait loop using *time.perf_counter_ns()*.

3.2 Data Collection

To collect the timing data, we used the *time.perf_counter_ns()* function for nanosecond resolution and collected 30,000 windows per class, with 50 samples per window. This resulted in 1.5 million timing samples per class. The raw timing data is written to a .CSV file as *window*, *sample*, and *time_ns*. The data on which the classifier was trained on is referred to as ‘internal’ and the data the classifier is tested on is

referred to as 'external'. Hence, internal accuracy refers to the accuracy of the classifier on the data it was trained on, and external accuracy is the accuracy of the classifier on unseen data. Both internal and external datasets are generated under the same conditions.

3.3 Feature Extraction of Timing Data

Regarding the data, we used a python program to collect 6 features per window. This made the data much easier to sort through for the SVM classifier instead of relying on raw timing data itself. We collected mean_ns (mean time per window), std_ns (standard deviation per window), max_ns (max time per window), min_ns (minimum time per window), range_ns (range of time per window), and spike_count. For the spike_count feature, we considered a spike anything greater than two standard deviations away from the mean. These features were selected because they are computationally lightweight and capture the shape of the timing distribution without requiring floating-point arithmetic beyond basic statistics.

3.4 SVM Training and Results Evaluation

To train the SVM classifier, we made a python script that loads both the attack and baseline feature datasets. It labels the attack data with a 1 and the baseline data with a 0, then combines them and removes the *window* identifier from the features. After that, it performs an 80/20 stratified train/test split, using *random_state=42*. It fits *StandardScaler* on only the training data then transforms the test data. It also trains the RBF kernel SVM using *C=10*, and *gamma="scale"*. To evaluate the robustness of the classifier, it was trained and evaluated on the same datasets, with the rest of the parameters unchanged, except for the varying *random_state*. These parameters were selected to provide flexible nonlinear classification while balancing classifier complexity, generalization, and computational efficiency. We saved the ~~model~~ classifier and the scaler with joblib for deployment. Furthermore, we trained a Decision Tree and Logistic Regression model on identical features for distribution. The 93% accuracy threshold was chosen because of our 92%-8% baseline and attack split. A classifier with a 92% accuracy would simply predict the baseline class and not predict any attack samples, making it a majority-class dummy classifier. Therefore, 93% was selected as the minimum target beyond the baseline sample percentage. The reported precision, recall, F1-score, FNR, and FPR also provide additional evidence of whether the classifier is detecting the attack class.

3.5 Visualization of Steps

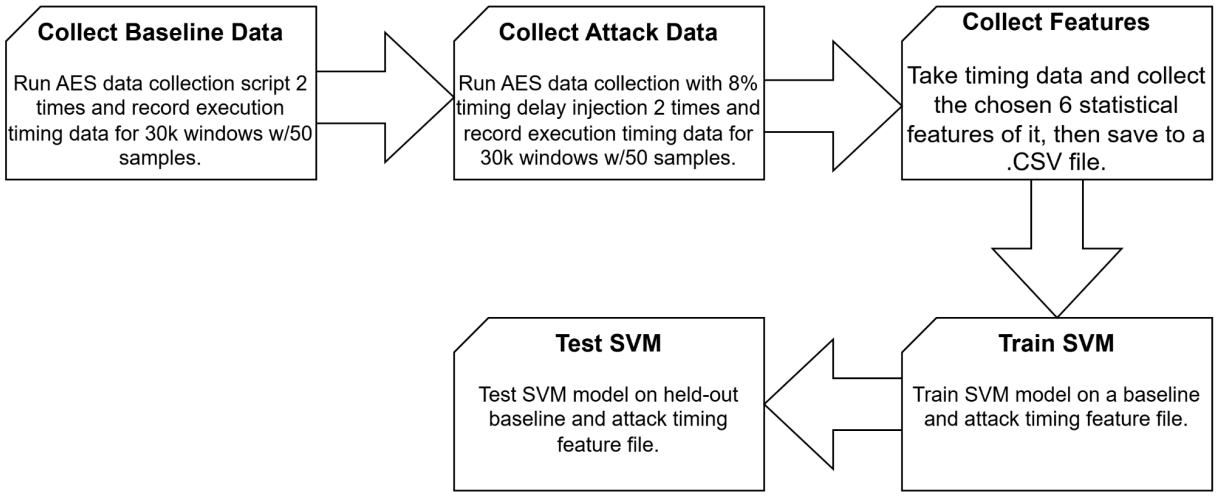


Figure 1: Flowchart diagram of data collection and SVM training process

3.6 Setup Details

For this paper, we used a Raspberry Pi 5 with 8 gigabytes of RAM, running Ubuntu 24.04.4 LTS (Noble Numbat). We also used python libraries numpy (2.4.4), pillow (12.3.0), pycryptodome (3.23.0), scikit-learn (1.8.0), and scipy (1.17.1).

To measure the CPU usage data, we captured the CPU utilization data while the SVM classifier python script ran as a background process. We collected the process identifier (PID) and monitored the CPU utilization with the Linux *pidstat* utility with the *-u* option. The measurement procedure was repeated 5 times with each run lasting 20 seconds. The resulting CPU usage measurements were recorded for comparison and analysis.

RESULTS

Classifier	Internal Accuracy	External Accuracy	Inference Per Window
SVM	96.26%	94.15%	1.800 ms
Decision Tree	95.90%	93.34%	4.243 ms
Logistic Regression	91.66%	87.20%	1.020 ms

Table 1: SVM achieves the best generalization on external test data, with Decision Tree only slightly behind it. Training time is also a one time offline cost.

Leak Probability	Internal SVM Accuracy	External SVM Accuracy
5%	96.27%	76.71%
8%	96.26%	94.15%
20%	98.08%	98.09%

Table 2: SVM accuracy across leak probability settings. Reliable generalization requires a minimum leakage rate - at 5%, the model classifier overfits to training data and external accuracy drops substantially.

True Negatives - 27856	False Positives - 2144
False Negatives - 1364	True Positives - 28636

Table 3: A representative confusion matrix with random_state of 42, with the model run on external data.

Random State	Avg. Precision	Avg. Recall	Avg. F1 Score	FNR	FPR
42	0.94	0.94	0.94	0.045	0.071
123	0.94	0.94	0.94	0.046	0.071
456	0.94	0.94	0.94	0.045	0.071
789	0.94	0.94	0.94	0.044	0.073
2026	0.94	0.94	0.94	0.046	0.071

Table 4: To ensure that the results were independent of a specific random condition (random_state), we tested the model on 4 other random_state, and can see clearly that the model achieves an average precision of 94% on external test data, as well as 94% average recall and F1 score. We can look at the variations across the random_state through the FNR (False Negative Rate) and FPR (False Positive Rate), where slight variations throughout the runs show that the classification performance between runs was not particularly dependent on the particular internal test-train split.

Other Results

To verify that the simulated timing leakage in our simulation is statistically real, we applied TVLA (Test Vector Leakage Assessment) using Welch's t-test. The test yielded $|t| = 62.56$ and $p < 2.2 \times 10^{-308}$, confirming that the attack and baseline distributions are statistically distinct with overwhelming

confidence. However, TVLA is a global test - it indicates whether leakage exists anywhere in the dataset but cannot identify which individual windows contain an attack. Our SVM classifier addresses this gap by making an independent classification decision on each 50-sample window, achieving 96.26% accuracy on held-out data and 94.15% on a fully separate external dataset. The SVM classifier also has an AUC (Area Under the Curve) score of 0.9815, indicating a high ability to discriminate between baseline and attack.

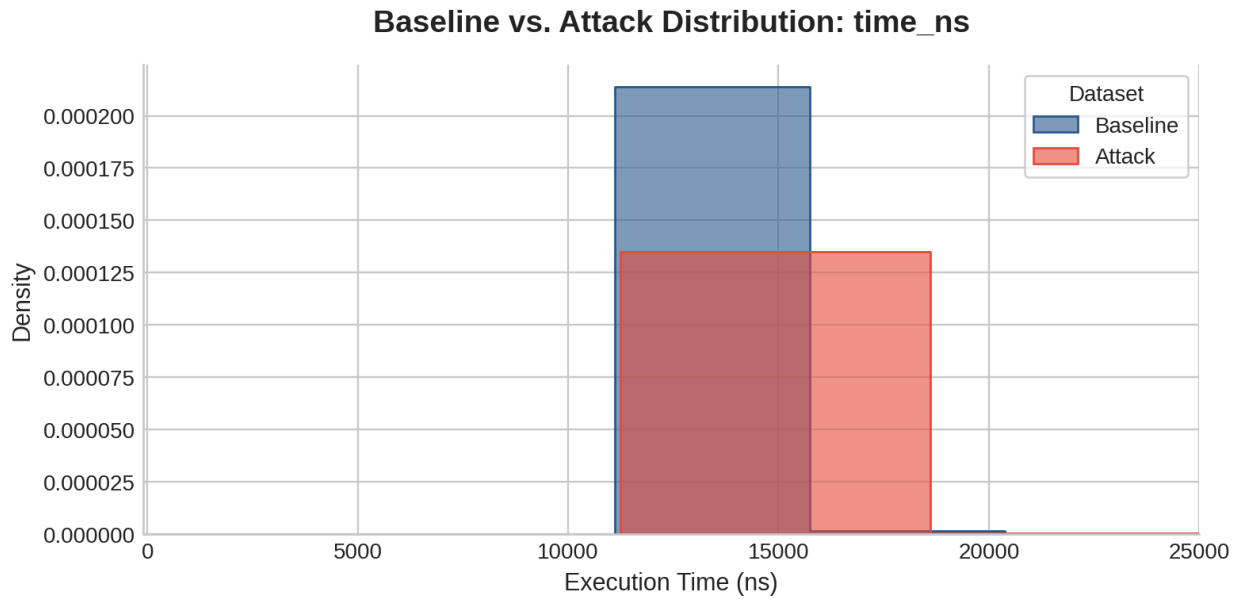


Figure 2: Attack and baseline timing data distribution visualization. Attack data shifts the baseline distribution, adding a wider range with a lower peak density, indicating class separability.

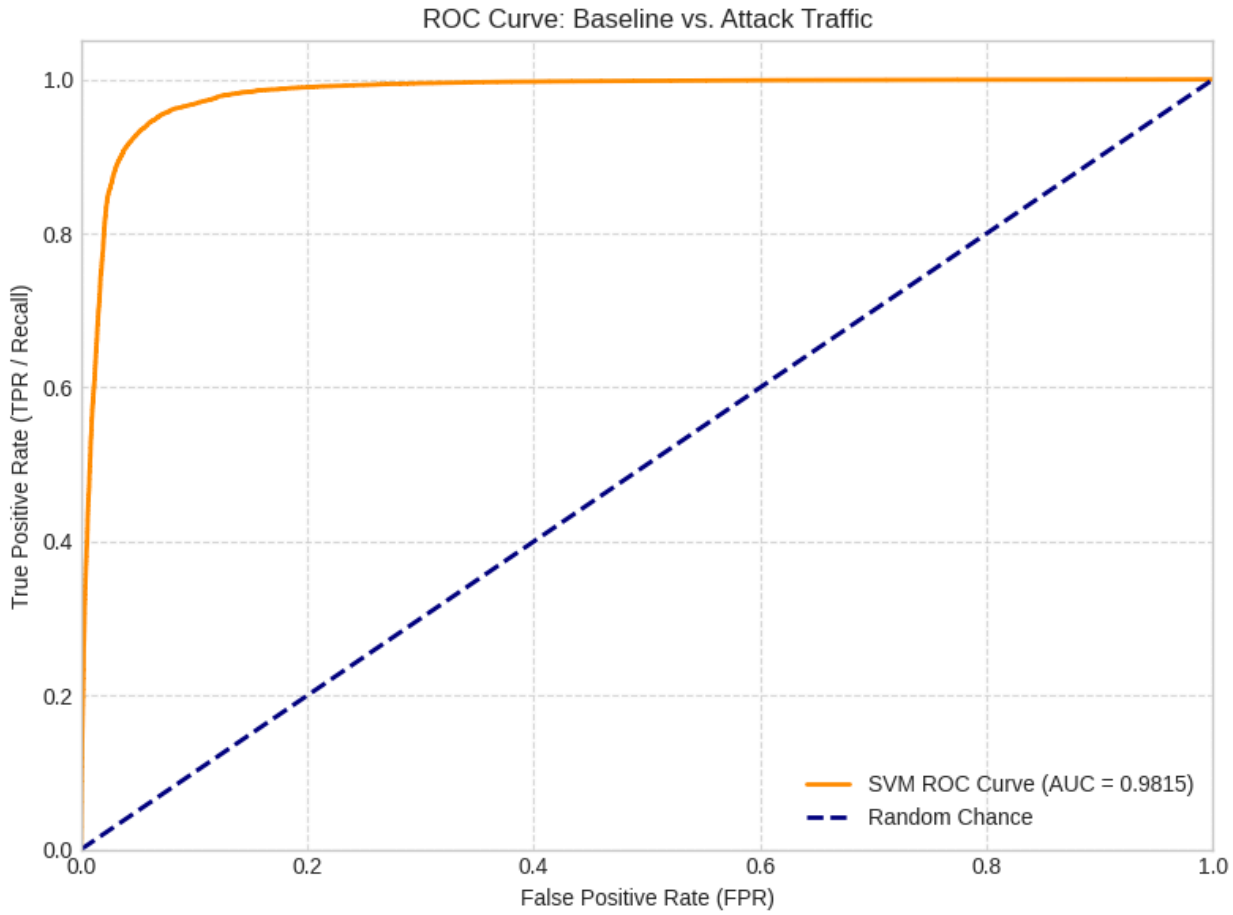


Figure 3: ROC curve with AUC score = 0.9815 for SVM classifier. The model significantly outperformed random chance (dashed line) and reached a high true positive rate, as shown by the curve ascending sharply.

For the single-window inference time, we collected a mean of 1.800 ms on 1000 windows. CPU usage was an 8.26% average across 5 runs for 20 seconds each, which was just under the 10% usage mark. We also chose to include a 50 ms inter-window delay to the detection to keep CPU usage below 10%, a necessity for the classifier to be lightweight. Without this delay, CPU overhead was kept at 50-60%, a significant amount for a classifier.

Please note that the training time for the SVM classifier, around 15.5 seconds on average, does not affect real-time performance.

DISCUSSION

Looking at the data in Table 2, SVM has the highest external accuracy amongst all the tested models. Although Decision Tree had a less than 1% difference from SVMs internal and external accuracy, it has a much larger window inference time of ~4.243 ms, as compared to SVMs 1.800 ms time. On the other hand, Logistic Regression was able to achieve a lower inference latency of 1.020 ms, but at the cost of a lower external accuracy of 87.20%. This shows that SVM was able to reliably classify the artificially injecting timing anomalies, with a relatively low inference latency. However, this should not be interpreted as SVM could reliably detect physical timing-based side-channel attacks or key-dependent cache leakage, which this experiment did not evaluate.

To assess feature contributions to the SVM classifier, we applied permutation feature importance, which measures the reduction in classifier accuracy when each feature is randomly shuffled independently. Because the RBF kernel maps data into a high-dimensional space, direct weight extraction is not possible, making permutation importance the appropriate method. Results showed that mean_ns had the largest influence (importance = 0.255), followed by min_ns (0.148) and std_ns (0.140), with range_ns, max_ns, and spike_count each contributing below 0.06. Notably, this distribution differs from a parallel Decision Tree importance analysis, which assigned 71.6% of its weight to min_ns alone. This discrepancy reflects a fundamental difference in how the two classifiers operate: the Decision Tree identifies a single threshold on minimum execution time that separates the majority of windows, while the SVM exploits a richer, non-linear combination of timing features. The broader feature reliance of the SVM suggests it is detecting a more generalizable signal rather than a single implementation artifact, which is consistent with its stronger performance on the external test dataset.

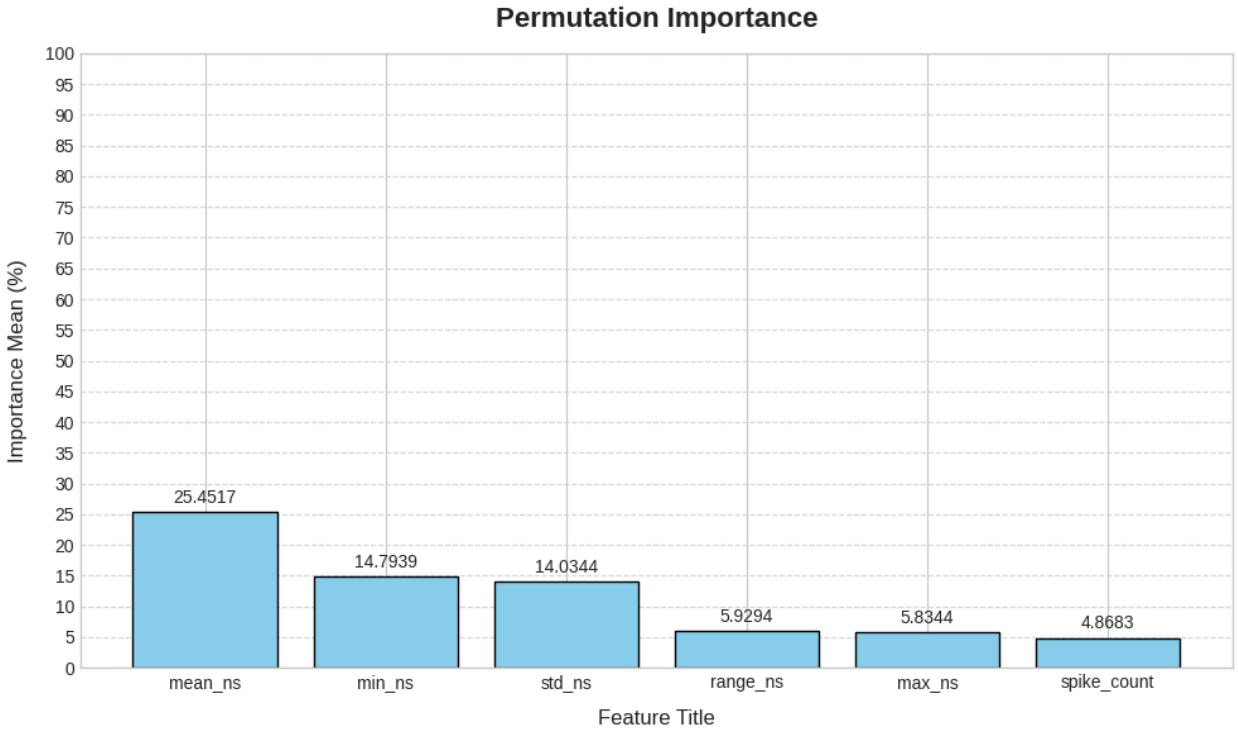


Figure 4: SVM permutation importance visualization. We can see that the most important feature is mean_ns at ~25%, showing that the model has importance spread across several different features.

Regarding the low external accuracy in Table 3 for SVM at ~5% leakage, this is because the classifier, at that point, will start overfitting on the data, leading to inaccurate predictions on the timing data. Regarding the practical application of this, if real-world cache-miss rates are closer to 5%, the classifier as trained may not generalize to those environments.

More importantly, ~~The~~ the simulation uses a software-injected delay to create timing anomalies rather than physical cache eviction, which limits direct applicability to production attacks. The probabilistic delay model approximates but does not fully replicate key-dependent T-table access patterns or establish detection of physical timing-based side-channel attacks. Future work should validate on physical hardware using Linux perf counters and extend detection to other ciphers. We also suggest that future work focus on creating a live detection and training script so that the SVM is trained on data collected from the specific machine and time of deployment, as background noise can vary the data, causing pretrained models to be ineffective on even the same device under different conditions. Furthermore, future work should look into physical timing-based side-channel attacks, as that remains a limitation of the current experiment.

CONCLUSION

This paper presented a lightweight machine learning based detection system that classified software-injected, artificial timing-based anomalies in AES on a Raspberry Pi 5. Using an SVM classifier trained on six statistical features extracted from 50-sample AES timing windows, the system achieved 96.26% accuracy on internal test data and 94.15% on an independent external dataset, surpassing the 93% target while maintaining per-window inference latency of 1.800 ms and CPU overhead below 10%. TVLA analysis confirmed that the simulated attack and baseline distributions were statistically distinct ($|t| = 62.56$ and $p < 2.2 \times 10^{-308}$). Future work will extend this approach to physical cache profiling using hardware performance counters and evaluate if the classifier generalizes to real side-channel behavior and additional cryptographic workloads.

REFERENCES

[1] B. Sarkar, A. Saha, D. Dutta, G. De Sarkar, and K. Karmakar, “A Survey on the Advanced Encryption Standard (AES): A Pillar of Modern Cryptography,” *International Journal of Computer Science and Mobile Computing*, vol. 13, no. 4, pp. 68--87, 2024.

[2] N. Mouha and N. Mouha, *Review of the Advanced Encryption Standard*. US Department of Commerce, National Institute of Standards and Technology, 2021.

[3] M. Randolph and W. Diehl, "Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman," *Cryptography*, vol. 4, no. 2, p. 15, 2020.

[4] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18--22, 1996, Proceedings*, Springer, 1996, pp. 104–114.

[5] D. J. Bernstein, "Cache-Timing Attacks on AES," 2005.

[6] J. Depoix and P. Altmeyer, "Detecting Spectre Attacks by Identifying Cache Side-Channel Attacks Using Machine Learning," *Advanced Microkernel Operating Systems*, vol. 75, p. 48, 2018.

[7] S. U. Jan, S. Ahmed, V. Shakhov, and I. Koo, "Toward a Lightweight Intrusion Detection System for the Internet of Things," *IEEE access*, vol. 7, pp. 42450--42471, 2019.

[8] N. Bellec, S. Rokicki, and I. Puaut, "Attack Detection Through Monitoring of Timing Deviations in Embedded Real-Time Systems," in *ECRTS 2020-32nd Euromicro Conference on Real-Time Systems*, 2020, pp. 1--22.

[9] S. E. Frankel, K. Kent, R. Lewkowski, A. D Orebaugh, R. W. Ritchey, and S. R. Sharma, "SP 800-77. Guide to IPsec VPNs," National Institute of Standards \& Technology, 2005.