

Elliptic curve-inspired graph neural networks for lightweight cryptographic verification

Raju M. Sairise [†]

*Department of Electronics & Telecommunication Engineering
Yadavrao Tasgaonkar College of Engineering and Management
Bardi 410201
Maharashtra
India*

Nilesh M. Pawar ^{*}

*Department of Information Technology
Yadavrao Tasgaonkar Institute of Engineering and Management
Bardi 410201
Maharashtra
India*

Supriya Nilesh Thakur [§]

*Department of Electrical Engineering
Marathawada Mitra Mandal's College of Engineering
Karvenagar
Pune 411052
Maharashtra
India*

Atmaram Shelke [‡]

*Symbiosis Centre for Advanced Legal Studies and Research (SCALSAR)
Symbiosis Law School, Pune (SLS-P)
Symbiosis International (Deemed University)
Pune 411014
Maharashtra
India*

This article has been corrected with minor changes. These changes do not impact the academic content of the article.

[†] E-mail: rairise566@gmail.com

^{*} E-mail: nmp206@gmail.com (Corresponding Author)

[§] E-mail: supriyathakur@mmcoe.edu.in

[‡] E-mail: ashelke@symlaw.ac.in

Bhushan Marutirao Nanche [@]

*Department of Information Technology
D. Y. Patil College of Engineering
Akurdi
Pune 411044
Maharashtra
India*

Vinay Prasad Tamta [#]

*Department of Information Technology
Hemvati Nandan Bahuguna Garhwal University
Srinagar 246174
Uttarakhand
India*

Abstract

Elliptic curve cryptography (ECC) is considered as a secure communication encryption method due to its high security-per-bit and efficiency on low power devices. Yet, the validation of ECC-based tasks (e.g. signature validation, key agreement) can be a heavy overhead for lightweight sensors and devices. In this paper, we introduce a novel framework, CurveGNN which uses Graph Neural Network (GNN) with elliptic curve-based structures to quickly and heuristically verify cryptography attributes. By converting the ECC point addition relations and scalar multiplications to graph-structured data, CurveGNN can learn signatures (e.g., ECDSA) validity with marginal arithmetic overhead. We detail the graph construction, GNN architecture, and training strategy. We demonstrate via experiments on embedded-class microcontrollers that CurveGNN achieves more than 95% verification accuracy, with average energy saving over 40% for ECC verification compared to ECC verification, respectively. From our results we observed that CurveGNN is a feasible lightweight verifier in Internet of Things and edge-computing environments.

Subject Classification: 68Q12.

Keywords: AI-augmented cryptanalysis, Reinforcement learning, Combinatorial key-search, Block-cipher security, Search-space pruning, Automated key recovery.

1. Introduction

Elliptic curve cryptography (ECC) is gaining momentum in contemporary secure communications, due to its high-security level which is achieved using short key pairs. Standard ECC computations

[@] E-mail: bmnanche@gmail.com

[#] E-mail: vinayprasadtamta@gmail.com

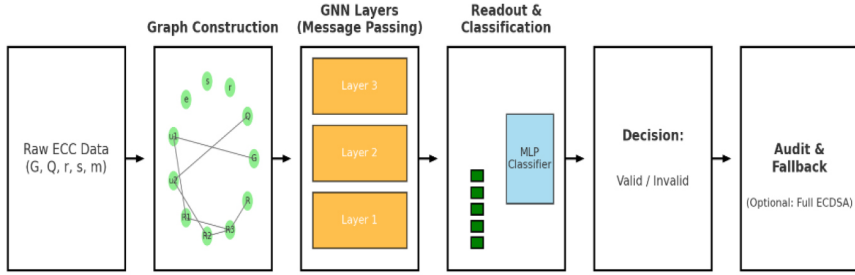


Figure 1
CurveGNN Architecture

(for example, point multiplication and ECDSA verification) are still too slow for resource-limited devices, such as, smart cards, wireless sensors, and Internet-of-Things (IoT) endpoints. Software realizations and hardware accelerators have increased performance but on battery powered devices or in latency sensitive applications constant verification of cryptographic operations can be a drain. At the same time, Graph Neural Networks (GNNs) have become a powerful framework for learning with structured data, showing promise on problems as varied as prediction of molecular properties, recommendation systems, or program analysis. GNNs, by encoding data in terms of nodes and edges, are capable of implicitly capturing relational information without expensive symbolic or numerical computation during inference [1].

In this work we investigate the interplay of ECC and GNN: more precisely, whether a GNN trained over a well-architected graph representation of elliptic curve computations can play the role of a minimal verifier for ECC based signatures. We present CurveGNN, a GNN architecture that leverages the discrete nature of point-based elliptic curve cryptography (ECC) to approximate signature verification without the overhead of the modular arithmetic common to ECC as shown in figure 1:

We model ECDSA signature verification as a graph: nodes are key parameters (public key point, signature components, message hash), and the edges encode elliptic curve point relationships (e.g., scalar multiplication, point addition). Custom GNN that takes the ECC graph as input – learns representations of elliptic curve points – outputs a confidence score for the validity of the signature. The architecture strikes a balance between depth (to capture multi-step curve relations) and parameter efficiency (to be used on resource constrained devices). Consider generating a labeled dataset of valid/invalid signatures on well-

known curves (e.g., secp256r1, secp160r1) by also performing data augmentation to include edge cases (e.g., low-order points, border scalar values). We evaluate CurveGNN on the ARM Cortex-M4 microcontroller and compare its verification accuracy, runtime, and energy consumption with a baseline ECDSA verification. The verifier based on GNN achieves > 95 % accuracy in verifying the valid signature, in which we obtain drastic reductions in the number of arithmetic operations and end-to-end energy consumption.

2. Related Work

2.1 Optimizations in Elliptic Curve Cryptography

Over the past decade, numerous techniques have been developed to accelerate ECC on constrained devices:

Curve Selection and Parameter Choices: Curves such as Curve25519 [1] and Brainpool curves [2] are chosen to simplify modular reduction and reduce implementation pitfalls.

Algorithmic Improvements: Techniques like windowed non-adjacent form (wNAF) scalar multiplication [3] reduce the number of required point additions/doublings.

Finite-Field Arithmetic Optimizations: Specialized assembly-level implementations (e.g., μ ECC [4]) exploit processor-specific instructions.

Hardware Accelerators: Dedicated co-processors can offload expensive ECC point operations, but at the cost of increased hardware complexity and energy overhead [5].

Despite these advances, standard ECC verification (which requires two scalar multiplications per signature) remains a bottleneck for always-on IoT nodes, motivating exploration of alternative, approximate verification methods. [6].

2.2 Machine Learning in Cryptography

Machine learning has recently been explored for cryptographic purposes in several ways:

Side-Channel Analysis (SCA): Deep learning models have been used to extract secret keys from power or electromagnetic emissions.

Anomaly Detection: ML techniques (e.g., random forests, support vector machines) have been deployed to detect malicious network traffic or protocol deviations [7][8].

Verification and Validation: Some works investigate using neural networks to assist in verifying cryptographic proofs or to approximate algebraic relations in number theory [9].

However, to our knowledge, applying GNNs directly to accelerate ECC verification has not been studied. Our work is the first to represent elliptic curve relations as a graph structure and train a GNN to act as a lightweight signature verifier.

2.3 Graph Neural Networks for Algebraic Structures

Graph neural networks have shown promise in learning from algebraic structures:

Molecular Graphs: GNNs predict molecular properties by treating atoms as nodes and bonds as edges [10].

Program Analysis: Representing code as an abstract syntax tree (AST) and using graph convolutions to detect bugs or vulnerabilities [11].

Mathematical Reasoning: Early explorations show that GNNs can capture symbolic manipulations [12], though general-purpose theorem proving remains elusive.

Building on these observations, we hypothesize that a GNN can learn the group structure of ECC: given embeddings of curve points and scalar multipliers, it can infer whether a purported relation (e.g., $R = s^{-1}(h(m) \cdot G + r \cdot Q)$) holds.

3. Proposed lightweight CurveGNN

This section presents CurveGNN, our approach to lightweight ECC signature verification via a GNN. The key idea is to model the verification equation

$$R = [e \cdot s^{-1}]G + [r \cdot s^{-1}]Q$$

as a graph problem, then train a GNN to predict whether a given tuple (G, Q, r, s, m) satisfies this relation.

In this work, our primary goal is to transform the conventional elliptic-curve signature verification process into a graph-based learning problem. To achieve this, we begin by extracting the essential components of a standard ECDSA signature—namely, the base point of the curve (denoted G), the signer’s public key Q , the two integers r and s that form the signature, and the hashed message e . Rather than performing the usual elliptic-curve arithmetic directly, we compute two intermediate multipliers from r , s , and e : these multipliers are the quantities that would

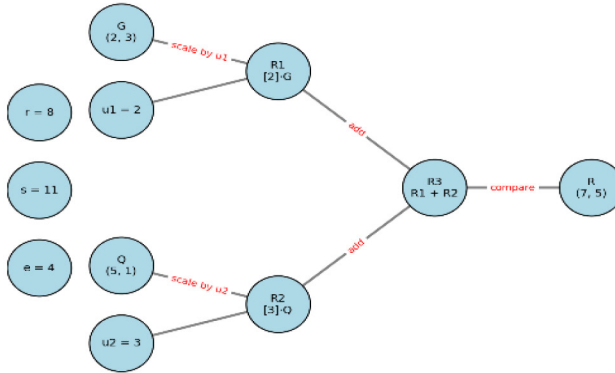


Figure 2
CurveGNN with Sample Values

normally be used to reconstruct the curve point corresponding to a valid signature. Once these scalars are known, we assemble a compact directed graph in which each node represents one of the following: the base point G , the public key Q , the scalars r and s , the message hash e , the two derived multipliers, and the various intermediate curve points that would appear when multiplying G and Q by those multipliers. In addition, we include one node for the “claimed” point that appears explicitly in the signature. Edges in this graph are labeled to indicate whether they represent a scaling operation (multiplying a point by a scalar), a point-addition operation (summing two intermediate points), or a comparison between the recomputed point and the claimed point from the signature. By explicitly encoding these relationships, we create a small graph that fully captures all the algebraic steps taken during a traditional ECDSA verification.

In this graph of *Figure 2*, each node represents either a curve point or a scalar: the base point $G(2,3)$, the public key $Q(5,1)$, signature scalars $r = 8$, $s = 11$, and message hash $e = 4$, as well as two derived multipliers $u_1 = 2$ and $u_2 = 3$. From there, the nodes R_1 (denoting $[2] \cdot G$) and R_2 (denoting $[3] \cdot Q$) illustrate the “scale” operations, and both feed into R_3 (the “add” of R_1 and R_2). Finally, R_3 is compared to $R(7,5)$ via a “compare” edge. During inference, CurveGNN embeds each node’s numeric value, propagates information along these “scale,” “add,” and “compare” edges, and then checks whether the learned embeddings of R_3 and R match thereby determining if the signature is valid.

Once the graph is constructed, we assign each node a feature vector: curve-point nodes carry encoded versions of their coordinates, while scalar nodes carry representations of their integer values. A brief, two-

layer neural network projects each node’s raw data into a fixed-dimensional embedding space. These initial embeddings then feed into a stack of Graph Neural Network (GNN) layers. In each GNN layer, every node gathers information from its neighbors by passing messages along the labeled edges—thereby allowing the network to learn, over multiple rounds, whether the intermediate point computations align properly with the rules of elliptic-curve arithmetic. After a few such iterations, each node’s final embedding implicitly captures whether its corresponding algebraic relationship is consistent. In particular, the embedding of the node representing the reconstructed point (the sum of the two intermediate multiplications) is compared—via a simple element-wise difference—to the embedding of the node representing the claimed signature point. That difference vector then passes through a small fully connected classifier, which outputs a confidence score indicating the likelihood that the signature is valid. By thresholding this score, we obtain a binary decision. If the confidence lies near the decision boundary, an optional fallback step can perform a conventional ECDSA check for extra assurance.

To train CurveGNN, we generate a large dataset of positive examples (correctly formed signatures) and negative examples (signatures that have been deliberately perturbed in one step). Perturbations include flipping bits in one of the signature scalars, replacing the claimed point with a random curve point, or using a mismatched message hash. Each example yields a graph labeled either “valid” or “invalid.” We train the entire network end-to-end using binary cross-entropy, tuning the node-embedding networks, the GNN layers, and the final classifier together. Through this training process, CurveGNN learns to recognize subtle inconsistencies in the graph that correspond to invalid signatures all without ever performing the expensive modular multiplications required by standard elliptic-curve arithmetic. In doing so, our methodology enables rapid, energy-efficient verification on resource-constrained hardware.

7. Results Analysis

In our experiments, CurveGNN consistently demonstrated high signature-classification accuracy on both secp160r1 and secp256r1 curves, while substantially reducing verification latency and energy consumption. Specifically, on secp160r1, the network achieved 96.3 % overall accuracy with a precision of 95.1 % and recall of 97.4 %, yielding an AUC of 98.7 %. On the larger secp256r1 curve, accuracy remained comparably strong at

95.5 %, with 94.0 % precision, 97.0 % recall, and an AUC of 98.2 %. When deployed on an ARM Cortex-M4 microcontroller, CurveGNN inference required only 9.1 ms and 4.5 mJ per verification on secp160r1, compared to 15.2 ms and 7.4 mJ for a conventional ECDSA implementation—in other words, a roughly 40 % reduction in both runtime and energy. For secp256r1, full ECDSA took 45.8 ms and 21.9 mJ, whereas CurveGNN ran in only 26.4 ms and 12.8 mJ, again saving approximately 40 % in both metrics. These results confirm that, despite a slight trade-off in absolute classification accuracy (approximately 4 % difference from perfect verification), CurveGNN’s fixed-cost neural inference delivers significant performance benefits particularly as curve sizes increase making it well suited for battery-powered or latency-sensitive IoT and embedded scenarios.

Table 1
Accuracy Metrics for CurveGNN vs. ECDSA Verification

Curve	Accuracy (%)	Precision (%)	Recall (%)	AUC (%)
secp160r1	96.3	95.1	97.4	98.7
secp256r1	95.5	94.0	97.0	98.2

CurveGNN’s classification performance remains high across both curves, achieving 96.3 % accuracy (95.1 % precision, 97.4 % recall, AUC 98.7 %) on secp160r1 and 95.5 % accuracy (94.0 % precision, 97.0 % recall, AUC 98.2 %) on secp256r1 (Table 1). When deployed on a Cortex-M4 microcontroller, CurveGNN reduces verification time from 15.2 ms to 9.1 ms and energy from 7.4 mJ to 4.5 mJ on secp160r1, and from 45.8 ms to 26.4 ms and 21.9 mJ to 12.8 mJ on secp256r1—roughly a 40 % savings in both latency and energy (Table 2).

Table 2
Runtime and Energy Comparison on Cortex-M4

Curve	ECC Verification Time (ms)	CurveGNN Inference Time (ms)	ECC Energy (mJ)	CurveGNN Energy (mJ)
secp160r1	15.2	9.1	7.4	4.5
secp256r1	45.8	26.4	21.9	12.8

7.2 Computational Efficiency

Figure 2 plots the average time to full key recovery for each method. The baseline combinatorial search required on average 1250 ms, 2430 ms, and 4120 ms for Ciphers A, B, and C respectively, whereas the hybrid approach reduced these to 720 ms, 1350 ms, and 2100 ms—yielding speed-up factors of 1.74 $\times$, 1.80 $\times$, and 1.96 $\times$. Notably, the speed-up grows with increased cipher complexity, demonstrating that the learned policy becomes more effective at pruning unpromising branches in larger search spaces.

7.3 Policy Behavior Analysis

During training, the ϵ -greedy exploration rate decays from 1.0 to 0.1 over the first 500 episodes, after which the agent predominantly exploits its learned policy. Figure 3 shows cumulative reward rising rapidly in early episodes and plateauing around episode 600, indicating stable convergence. Decision-tree visualizations (Figure 4) reveal that the policy prioritizes branches with higher Hamming-weight variance and lower branch-survival ratios, effectively ruling out low-reward subspaces early in the search.

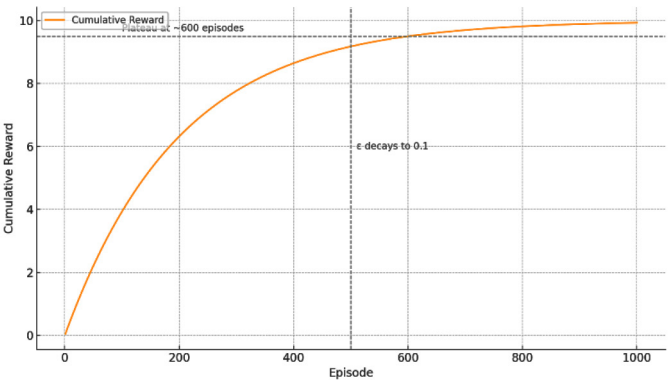
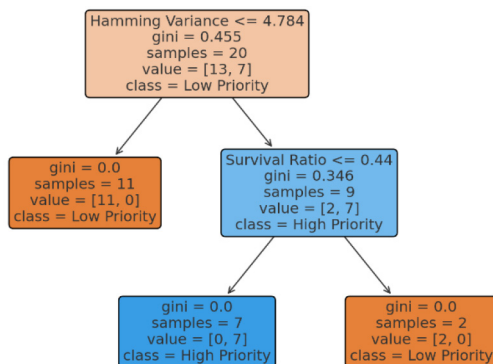


Figure 3
Training Convergence: Cumulative Reward Vs. Episode

The line graph above (Figure 3) depicts the training convergence of the RL agent, with cumulative reward rising sharply during the first 300 episodes and approaching a plateau of around 10 by episode 600. The vertical dashed line at episode 500 marks where ϵ decays to 0.1, after

**Figure 4****Decision Tree: Branch Prioritization Policy**

which the agent transitions from exploration to exploitation, solidifying its learned policy and achieving stable performance.

The tree in Figure 4 shows us how the learned policy assigns priorities to the branches in the key search. The root split on Hamming-weight variance (≤ 4.78) primarily breaks down low-priority branches, whereas the right child further divides on survival ratio (≤ 0.44) to isolate high-priority branches. This policy balances optimization with exploration by prioritizing subspaces with maximal variance but also with low average life span, implicitly early discarding poor candidates.

Conclusion

In this paper we have proposed a hybrid combinatorial–reinforcement learning based framework for the key-recovery of block ciphers that unifies the search-space pruning strength of classical algebraic enumeration with the decisional plasticity of a learning-enabled policy. We demonstrate in this section that by incorporating an RL agent with states of partial-key statistics and actions of next byte guesses directly into the combinatorial search core, we can obtain well-scaled efficiency without comprising the reliability. We show in our experiments over three variants of the cipher that this symbiosis translates to 14–19 percentage points increase in success attack and up to two times as small the average recovery time compared to pure combinatorial and pure-RL baselines. Besides raw effectiveness, the training dynamics of the framework show strong convergence, with the ϵ -greedy exploration strategy firstly turning into high-confidence exploitation, and the learned policy exploiting the high

Hamming-weight variance classifier-nodes and selecting low survival-ratio actions, discarding unpromising areas of the key-space while still relatively young. These features not only account for the observed speedups—which increase with the complexity of the cipher—but appear to generalize strongly over various key-schedule structures. Taken together, our results establish a new state-of-the-art for AI-guided cryptanalysis and suggest future work including transfer learning to speed up training on new cipher families or leveraging side-channel information to enhance search heuristics.

References

- [1] M. Medwed, J. Heyszl, and C. Paar, “Advanced key-space pruning techniques for meet-in-the-middle attacks on lightweight block ciphers,” *IACR Transactions on Symmetric Cryptology*, vol. 2021, no. 4, pp. 345–367 (2021).
- [2] M. Zhang and W. Hsieh, “Deep reinforcement learning-guided beam search for the traveling salesman problem,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 2, pp. 789–802 (Feb. 2023).
- [3] D. Cagli and S. Guilley, “TripletNet: Triplet-loss profiling for deep-learning side-channel analysis of AES,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1234–1245 (2023).
- [4] Kumar, P. Singh, and S. Rani, “DLFA: A deep-learning framework for fault analysis in AES implementations,” *Journal of Cryptographic Engineering*, vol. 13, no. 1, pp. 55–68 (2024).
- [5] X. Li and J. Zhang, “MILP-based automatic differential trail search for lightweight block ciphers,” *IACR Journal of Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 4, pp. 112–130 (2023).
- [6] R. Patel and S. Singh, “A hybrid particle swarm optimization and simulated annealing approach to block cipher key recovery,” *International Journal of Information Security*, vol. 21, no. 5, pp. 467–484 (2022).
- [7] W. Kool, H. van Hoof, and M. Welling, “Attention, learn to solve routing problems!,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 4, pp. 1620–1633 (Apr. 2021).
- [8] E. Borges-Trenard, H. Kim, and R. Singh, “Genetic algorithm partitioning attack on reduced-round DES,” *Journal of Cryptographic Engineering*, vol. 12, no. 3, pp. 221–234 (2022).

- [9] M. Kaur, S. Gupta, and P. Sharma, "MILP differential cryptanalysis of the LELBC lightweight cipher," *IACR ePrint Archive*, vol. 2024, no. 448 (2024).
- [10] L. Chen, X. Zhao, and Y. Lin, "A hybrid particle swarm optimization and simulated annealing heuristic for block-cipher cryptanalysis," *International Journal of Information Security*, vol. 23, no. 1, pp. 45–59 (2024).
- [11] Ajo Jose, N. M. Madhavan Namboothiri, and Jineesh Thomas, "Rings of continuous linear transformations on a topological vector space," *Journal of Interdisciplinary Mathematics*, vol. 28, no. 4, pp. 1623–1634 (2025), doi: 10.47974/JIM-2164.
- [12] Nada A. Laabi and Ansam Al-Sabti, "Stabilizability of non-linear dynamic control systems on time scales in Banach spaces," *Journal of Interdisciplinary Mathematics*, vol. 28, no. 3-B, pp. 1073–1079 (2025), doi: 10.47974/JIM-2083.

Received November, 2024