VeriEdge: Verifying and Enforcing Service Level Agreements for Pervasive Edge Computing

Xiaojian Wang[1], **Ruozhou Yu**[1], Dejun Yang[2], Huayue Gu[1], Zhouyu Li[1]

[1] North Carolina State University

[2] Colorado School of Mines

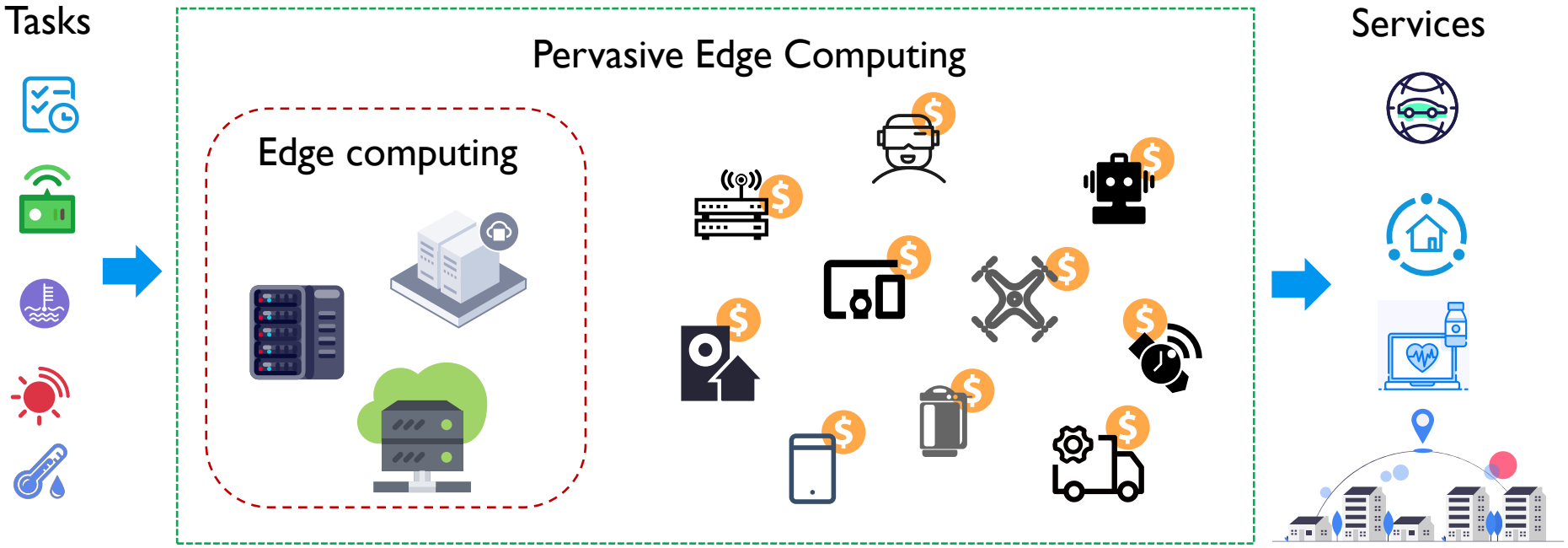
NC STATE UNIVERSITY

Outlines

- Background and Motivation
- Models and Problem Statement
- Solution Design
- Security Analysis
- Performance Evaluation
- Conclusions

Pervasive Edge Computing (PEC)

- Edge computing and **PEC**
  - Low latency
  - Network traffic reduction
  - **Edge devices can freely enter or exit the market**
  - **Device heterogeneity and dynamicity**

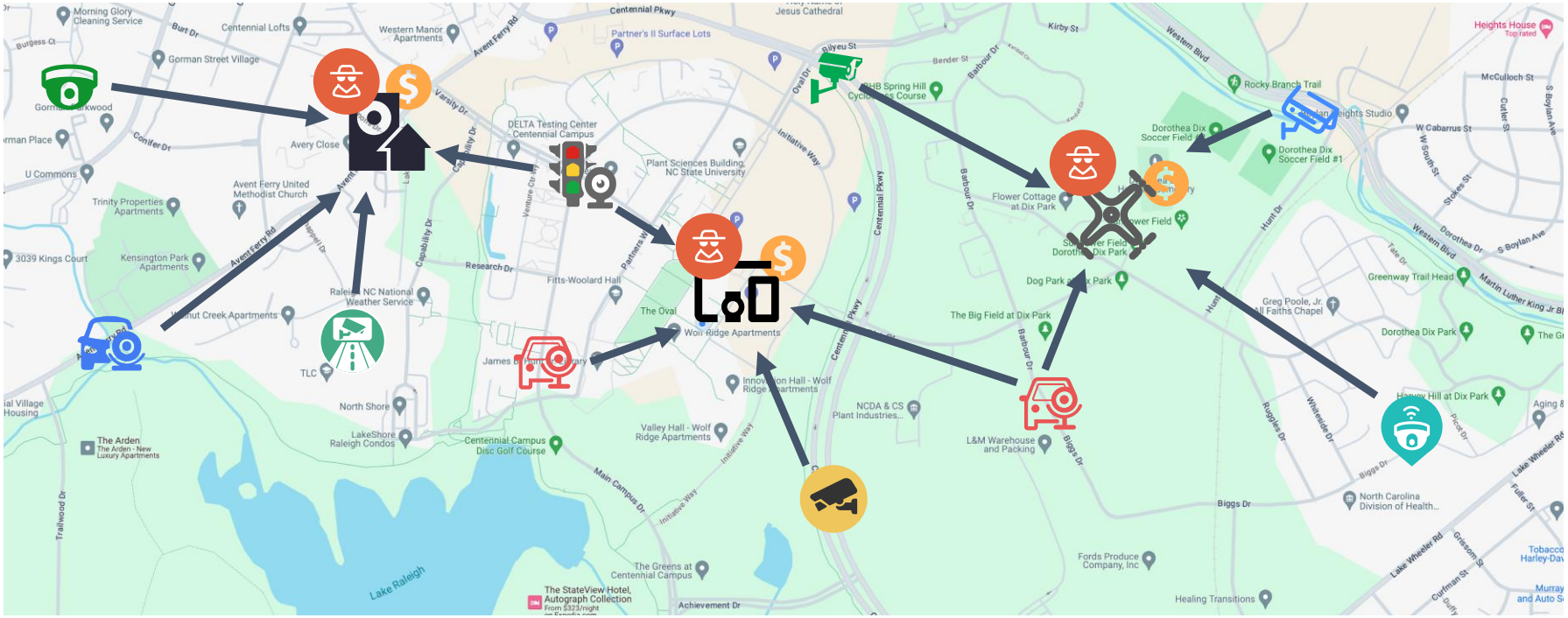


Service Level Agreements (SLAs)

- Mutual untrust between users and edge devices

SLA Compliance

A PEC device may not faithfully process the offloading video data while still trying to claim the service reward.



Monitoring SLA is imperative

Challenges in SLA Verification

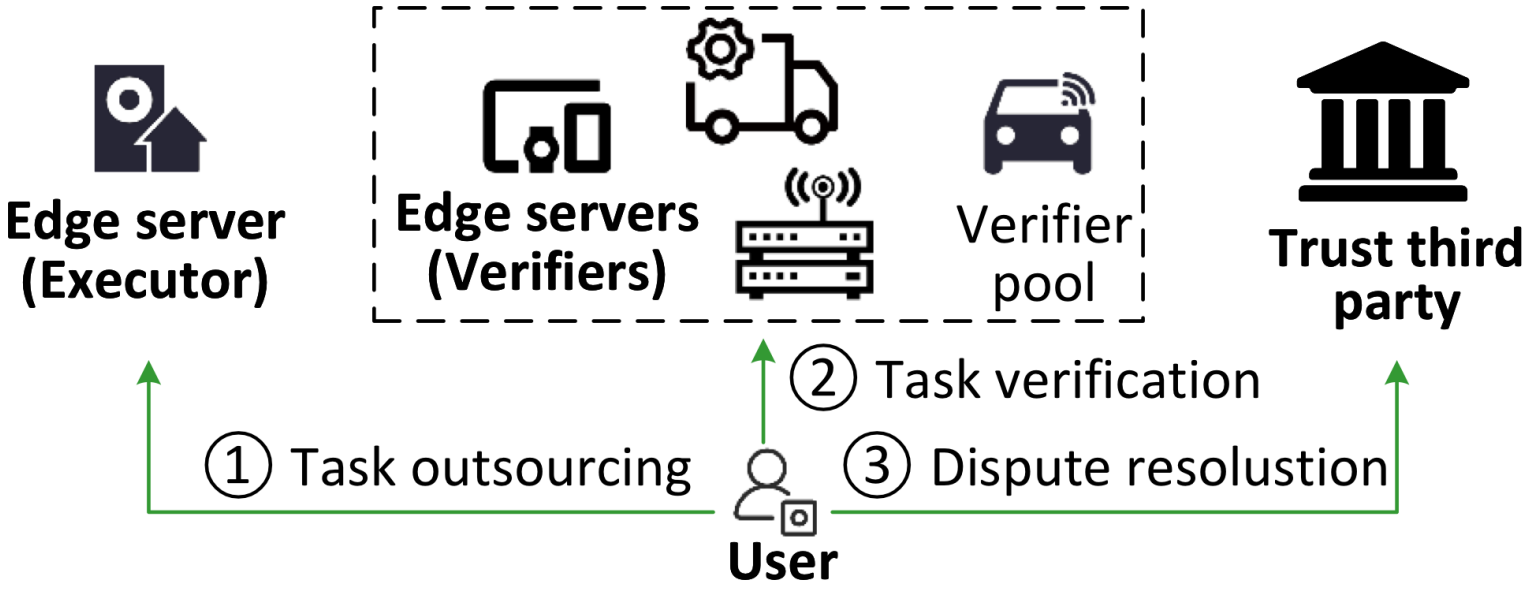
- Limited user resources hinder independent verification
  - Rely on external verifiers to assist.
- Verifiers can be untrusted as well
  - Verifiers are driven by financial interests.
- Fairly opportunities for verifier participation
  - For market stability, long-term viability, and preventing dominance or collusion by resource-intensive devices.

Our Solution: VeriEdge

- Commit-then-sample
  - Perform lightweight sampling and verification of intermediate computation results with non-repudiability.

- Crypto-based verifier selection and computation verification
  - Ensure verifiable fairness and a high probability for misbehavior detection.

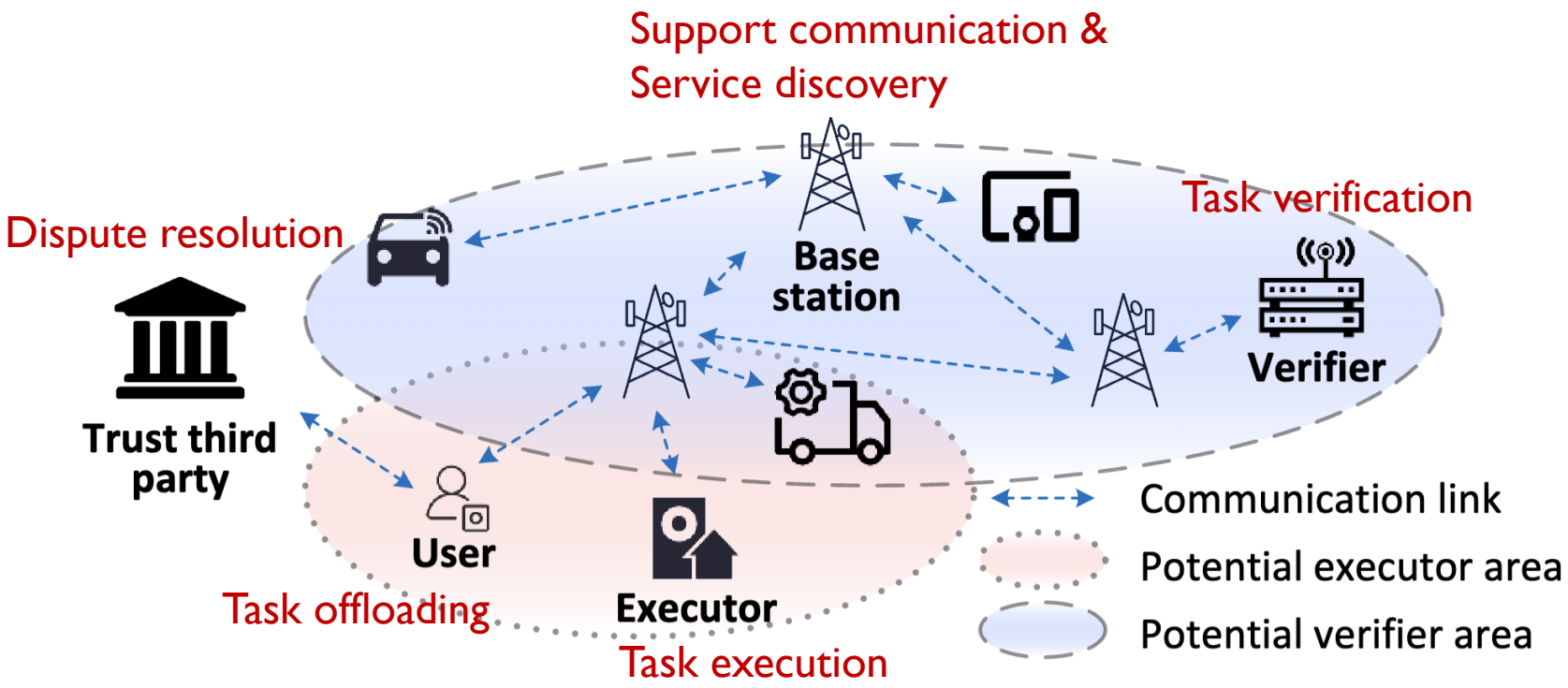
VeriEdge Overview



Outlines

- Background and Motivation
- Models and Problem Statement
- Solution Design
- Security Analysis
- Performance Evaluation
- Conclusions

System Model



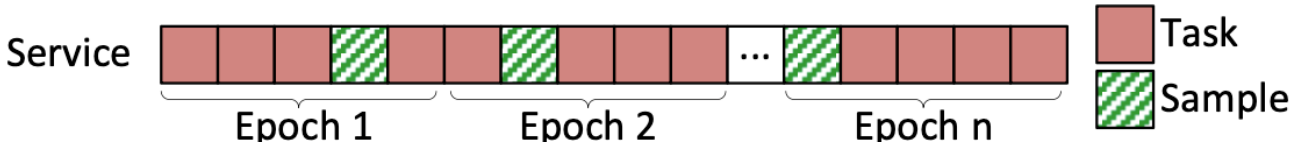
Threat Model and Security Goals

- Malicious executor
- Colluding executor and verifier
- Malicious users
- Trust Base station and TTP
- All parties communicate via authenticated secure channels

- ✓ SLA compliance
- ✓ Non-manipulable verification
- ✓ Dispute resolution

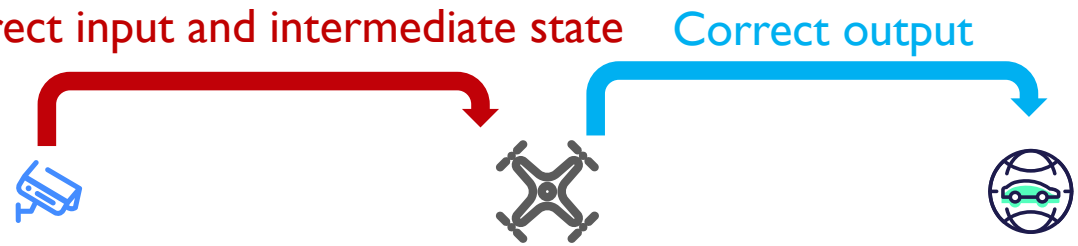
Problem Formulation

- Service Model



n epochs with epoch length $l_{\varrho_i} = 5$ tasks. Sampling rate $\eta_s = 0.2$.

- User's goal
  - To verify that each task is faithfully executed



Problem Formulation

- Goal: Define an edge outsourcing and verification framework ensures (Probabilistic) SLA compliance.

q-Algorithm

An executor/verifier executes the correct algorithm Φ_s and returns the correct result with probability q, and executes an arbitrary algorithm and returns potentially fake or malicious result with probability 1 - q.
An honest executor/verifier always executes a 1-algorithm.

(Probabilistic) SLA compliance

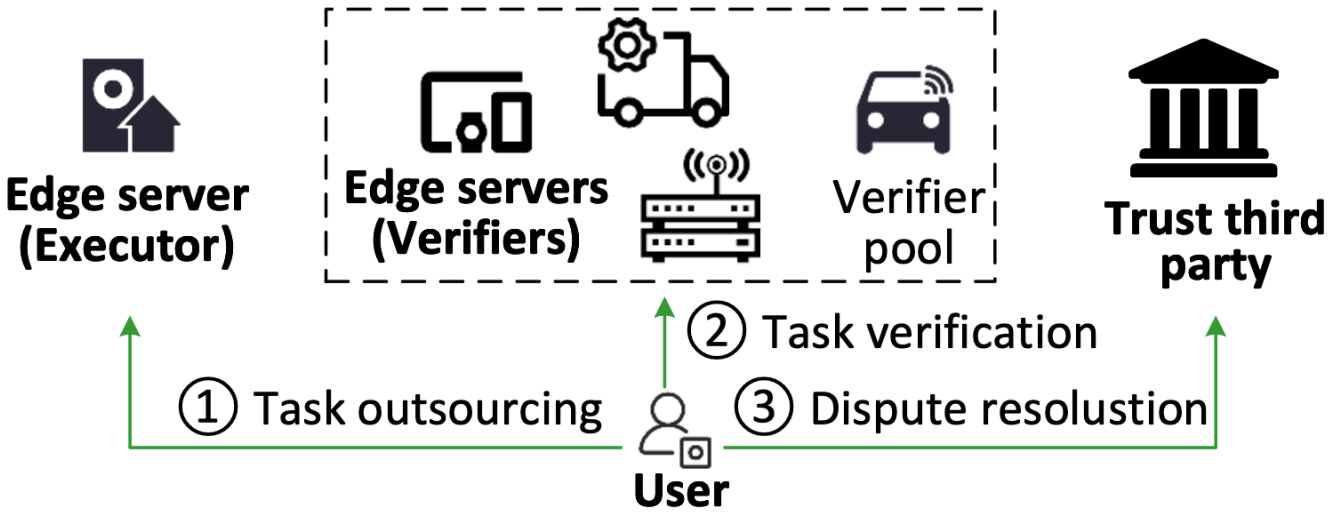
Given an executor using q-algorithm with q<1, for any pre-collusion ratio δ_s and sampling rate η_s, unterminated outsourcing indicates that user's SLA requirements are met with probability

$\Pr[\text{ executor is faithful }] > 1 - \epsilon(n)$,
where $\epsilon(n)$ is a negligible function in the number of epochs n, i.e., for all constants c, there exists an integer N such that for all $n > N$, $|\epsilon(n)| < \frac{1}{n^c}$.

Outlines

- Background and Motivation
- Models and Problem Statement
- Solution Design
- Security Analysis
- Performance Evaluation
- Conclusions

VeriEdge



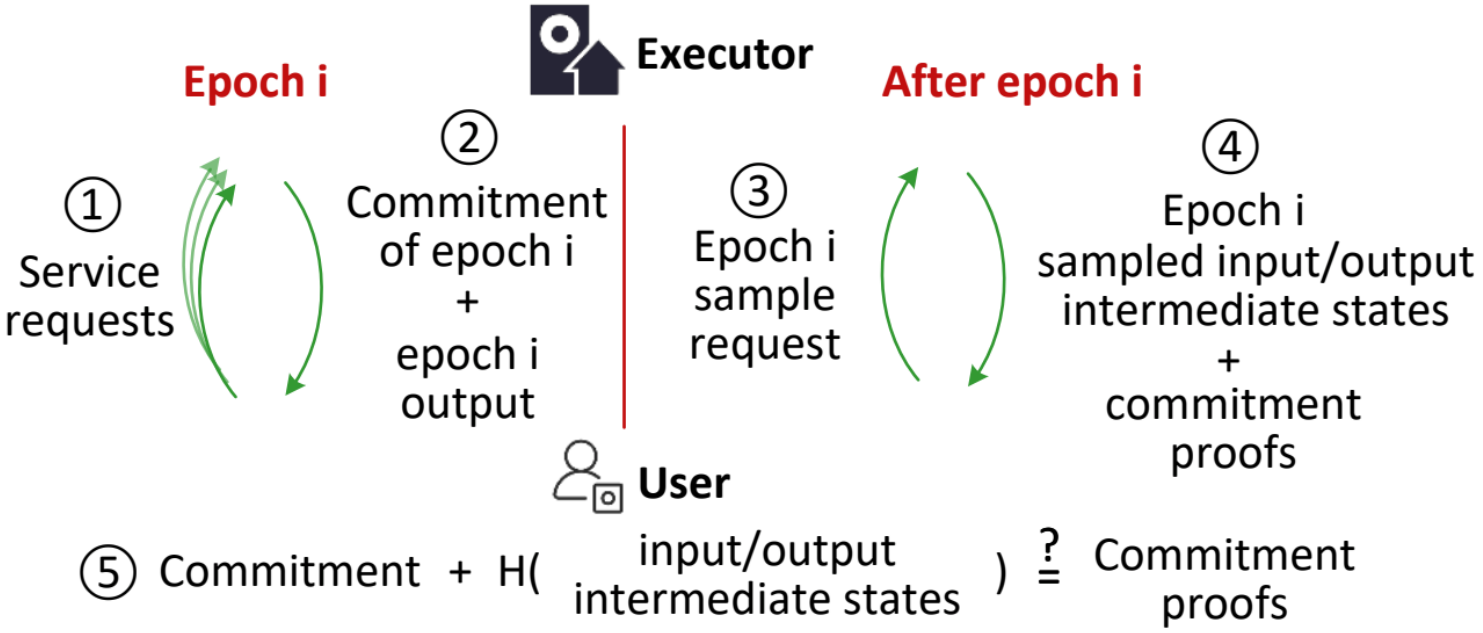
- Task Execution and Verifiable Sampling 1/3
- VRF-based Verifier Selection 2/3
- Dispute Resolution 3/3

Preliminaries

- Commitment
  - Com(x,r) → c: It takes a statement x and a random value r as input, and outputs a commitment c.
  - Verify(c,x,r) → {0,1}: It takes a commitment c, a statement x and a random value r as input, and outputs 1 if c = Com(x, r), and 0 otherwise.
  - ✓ Binding and Hiding

Task Execution and Verifiable Sampling 1/3

- Commitment and verification
  - Executor cannot know which tasks will be sampled before executing tasks.
  - Executor cannot return wrong intermediate states of a sampled task to mislead and evade verification.

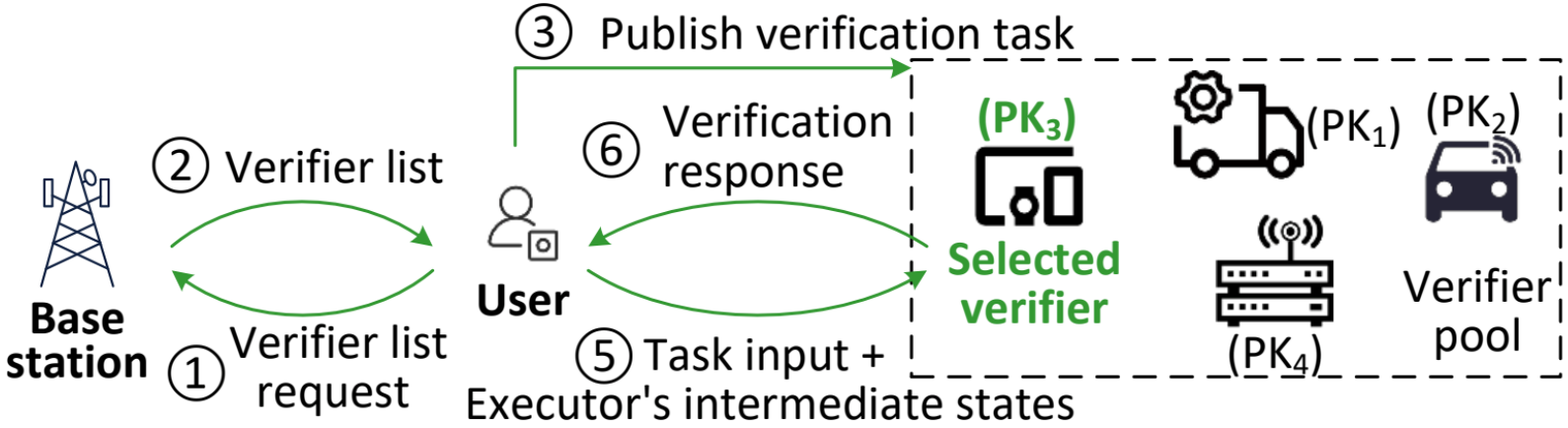


Preliminaries

- Verifiable Random Function (VRF)
  - VRFKeyGen(1^k)→(SK,PK)
    - It takes security param- eter k as input, and outputs a (SK,PK) key pair.
  - RFProve(SK,x)→(y,π)
    - It takes secret key SK and an input x, and generates an output y and its proof π.
  - VRFVerify(PK,x,y,π)→{0,1}
    - It takes the public key PK, input x, output y and proof π as input, and outputs 1 if (y,π)=VRFProve(SK,x) and 0 otherwise.
  - ✓ **Uniqueness, Provability, Pseudorandomness**

VRF-based Verifier Selection 2/3

- Dynamic verifier pool
  - Verifiable verifier selection
    - Selection process is fair and not manipulated

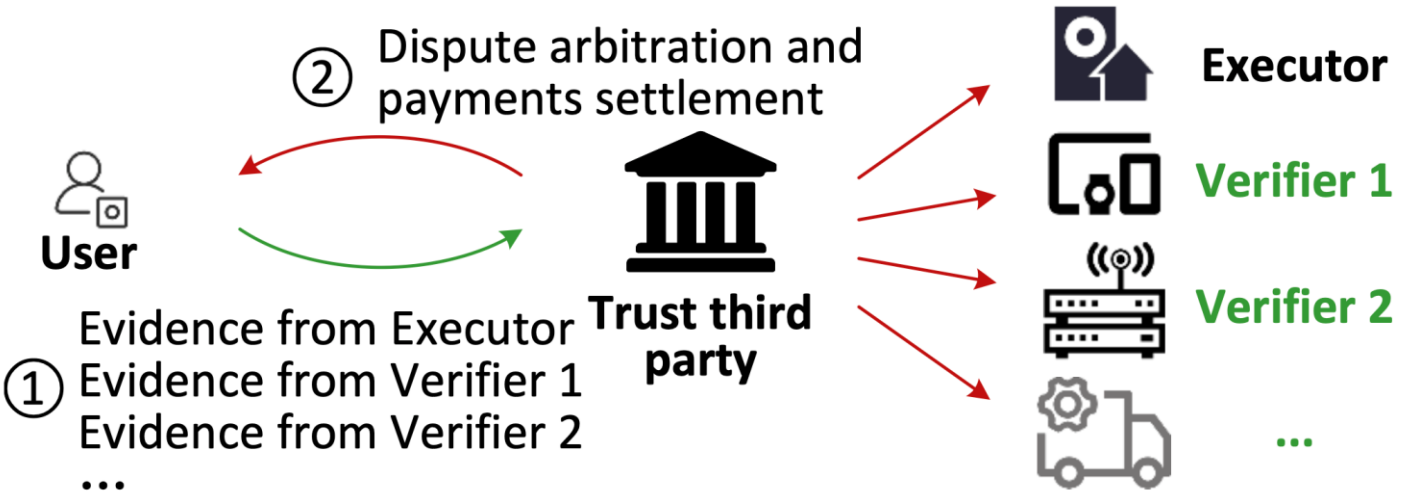


4 Verifiers generate verifiable random numbers

Public Key	PK_1	PK_2	**PK_3**	PK_4
Output	y_1	y_2	**y_3**	y_4
Proof	π_1	π_2	**π_3**	π_4

Dispute Resolution 3/3

- User can initiate a dispute to the TTP for arbitration
  - When results from the executor and (some of) the verifiers do not match



- epoch number for the dispute
- verifier list signed by base station
- proof list containing outputs
- proofs from all verifiers
- hash values of all samples' inputs
- random number

Outlines

- Background and Motivation
- Models and Problem Statement
- Solution Design
- Security Analysis
- Performance Evaluation
- Conclusions

Security Analysis

- Sound and correct execution

Theorem 1. *The probability of a malicious executor remaining undetected throughout n epochs, referred to as the* ***escape probability****, is* $\prod_{i=1}^{n} \left(q^{l_{\varrho_i} \cdot \eta_s} + (1 - q^{l_{\varrho_i} \cdot \eta_s}) \cdot \delta_s\right)$. □

- Non-manipulable dispute

Theorem 2. *Assume the SLA contract requires V verifiers for a majority vote. The probability that a malicious user can win the dispute for an arbitrary epoch is at most* $\delta_u^{\lfloor \frac{V}{2} \rfloor + 1}$. □

Outlines

- Background and Motivation
- Models and Problem Statement
- Solution Design
- Security Analysis
- Performance Evaluation
- Conclusions

Evaluation Settings

- Settings
  - Object tracking service
    - Real-time multi-object, segmentation and pose tracking with Yolov8
    - KITTI dataset
  - Platform
    - Phone, Raspberry Pi, Laptop, Desktop
  - Parameters
    - 100 epochs, each with100 tasks
    - Verifier pool 30 verifiers, pre-defined verifier number 2
    - Sampling rate 0.01
  - Baseline
    - Full replay without sampling

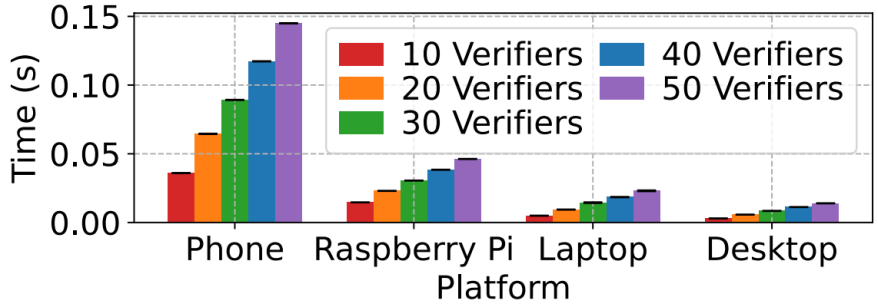
Communication and Computation Overhead

Description	Communication Cost (bytes)*			Execution Time (ms)		
	Message	VeriEdge	Baseline	Step	VeriEdge	Baseline
Obtaining inputs for verification (verifiable sampling in VeriEdge)	User → executor task request	86759200	86759200	Executor Merkle tree construction	0.33	0.34
	Executor → user Merkle tree root	113	113	User got results from executor	2778.76	3719.26
	Executor → user results	72	72	Executor generated proof	**28.45**	30.81
	User → executor sample request	515	-	User got proof response from executor	**30.29**	34.48
	User got proof from executor	**1789**	70124	User validated proofs from executor	**1.07**	79.56
VRF-based verifier selection	User → BS verifier list request	330	330	User got BS verifier list	61.82	90.27
	BS → user verifier list response	832	832			
	User → first verifier task request	**2612023**	86703281	Verifier key generation	124.27	96.21
				User found first verifier	30.56	30.56
	First verifier → user response	**552**	14160	User got results from first verifier	**643.78**	19762.93
				User checked correctness	**0.07**	221.44
Dispute resolution	User → second verifier request	**2612023**	86703281	User got results from second verifier	**642.77**	22205.78
	Second verifier → user response	**552**	14160			
	User → TTP dispute request	**56700**	104865	User got the dispute result from TTP	20.04	108.86
	TTP → user dispute response	16	17			

- Compared to raw application without verification
  - ↑ communication cost by 0.0028%
  - ↑ execution time by 1.14%
- Compared to Baseline
  - ↓ communication cost
  - ↓ execution time

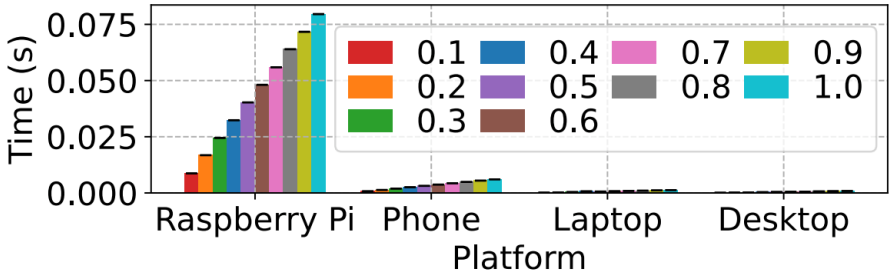
User End Computation Overhead

VeriEdge is lightweight, practical, and efficient.



VRF-based verifier selection time.

Verifier Selection <150ms



Merkle tree commitment checking time with different sampling rates.

Commitment checking time <80ms

Outlines

- Background and Motivation
- Models and Problem Statement
- Solution Design
- Security Analysis
- Performance Evaluation
- Conclusions

Conclusions

- VeriEdge
  - A framework for SLA verification and enforcement in dynamic PEC environments with untrusted edge devices
- Commit-then-sample
  - Perform lightweight sampling and verification of intermediate computation results with non-repudiability.
- VRF based verifier selection and computation verification
  - Ensures verifiable fairness and a high probability for misbehavior detection
- Stateful object tracking application evaluation
  - Efficiency and scalability

This research was supported in part by NSF grants 2045539, 2007391, 2008056 and 2008935.
The information reported here does not reflect the position or the policy of the federal government.

Thank you very much!

Q&A?