DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Design and Implementation of a Secure and Scalable Verification Scheme for Outsourced Computation in an Edge Computing Marketplace

Christopher Harth-Kitzerow

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Design and Implementation of a Secure and Scalable Verification Scheme for Outsourced Computation in an Edge Computing Marketplace

Entwurf und Implementierung eines sicheren und skalierbaren Verifizierungsschemas für ausgelagerte Berechnungen in einem Edge-Computing-Marktplatz

Author:	Christopher Harth-Kitzerow
Supervisor:	Prof. Dr. Jörg Ott
Advisor:	Gonzalo Munilla Garrido
Submission Date:	15.12.2020

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.12.2020   Christopher Harth-Kitzerow

Acknowledgments

Abstract

Latency-sensitive Internet of Things (IoT) applications need edge computing to overcome their limited computing capabilities. Idle resources at the edge of a network could be utilized to outsource the computation of an IoT device. An edge computing marketplace could enable IOT devices (Outsourcers) to outsource computation to any participating node (Contractors) in their proximity. In return, these nodes receive a reward for providing computation and storage services.

As IoT devices are often computationally weak, it might be difficult for them to verify if responses returned by a third party Contractor are valid. In fact, Contractors have an incentive to return a computationally less expensive probabilistic result, to save resources while still collecting the reward. Likewise, an Outsourcer has no incentive to rightfully pay an honest Contractor after it has received all computational results, and expensive microtransactions prohibit real-time payment.

In this thesis, we propose a verification scheme that enables verification of arbitrary deterministic functions by letting the Outsourcer send random samples to an additional third party Verifier and compare responses with the ones from the Contractor. Our designed verification scheme also provides publicly verifiable proofs for the Contractor and the Verifier that can be used to enforce payment if all sent responses have been correct.

We consider that all participants described participants that engage in the verification scheme might be dishonest or lazy. Thus, we compiled a list of potential protocol violations that might threaten the integrity or quality of outsourced computation. Our verification scheme provides techniques to prevent or detect each of the identified protocol violations with a high probability.

Object detection is a relevant use case for IoT devices and a computationally expensive task. We tested our verification scheme with state-of-the-art pre-trained Convolutional Neural Network models designed for object detection. On all devices, our verification scheme caused less than 1ms computational overhead, and a neglectable network bandwidth overhead of a maximum of 84 bytes per frame, independent of the frame's size. We also implemented a multi-threaded version of our scripts that performs the task of our verification scheme in parallel to the object detection to eliminate any latency overhead of our scheme. We also utilized a non-blocking message pattern to receive and preprocess new frames in parallel to all other threads to achieve high frame rates. We evaluated the performance of outsourced object detection with our verification scheme with a regular mid-range GPU, and a low-end Edge Accelerator and were able to achieve more than 60 frames per second with either hardware. Our tests showed that a cheap setup of a low-end CPU combined with a low-end Edge Accelerator can be used as a Contractor for outsourced object detection.

Kurzfassung

Contents

1. Introduction

Use with pdfLaTeX and Biber.

1.1. Motivation

In an Edge Computing marketplace, we assume that the Outsourcer is a computationally weak IoT device that outsources real-time data to a Contractor to process. The Contractor can be an edge server or any device in proximity to the Outsourcer that has enough computational resources available to execute the assigned function with sufficient QoS. Resources at the edge might be unavailable at times due to the following reasons:

1. High demand

2. No existing edge hardware near requesting device

3. Software/hardware limitations

Certain techniques such as resource reservation can be used to ensure a certain reliability for Outsourcers to being able to find an available contractor at all times. But even if resources in an edge computing marketplace are unreliable, they can be used to provide temporal QoS improvements compared to a service by a cloud server. Thus, IoT devices that are usually served by the cloud such as smart home assistants may be able to temporarily improve latency when outsourcing to an edge server in proximity. The following temporal quality of service improvements can still add value to an application:

1. An IoT device can temporarily improve latency when being served by an edge server in proximity

2. An IoT can transfer large amounts of data quickly and with low cost

3. An IoT device can offload storage to an Edge Server which might compress the data before sending it over the internet

4. An IoT device can turn off long-range communications technology to save energy

The following figure illustrates different temporal QoS improvements.

An example for an outsourcing task in Edge computing scenarios with high relevance for IoT devices is object detection for the following reasons:

1. Reliable object detection requires high computational complexity
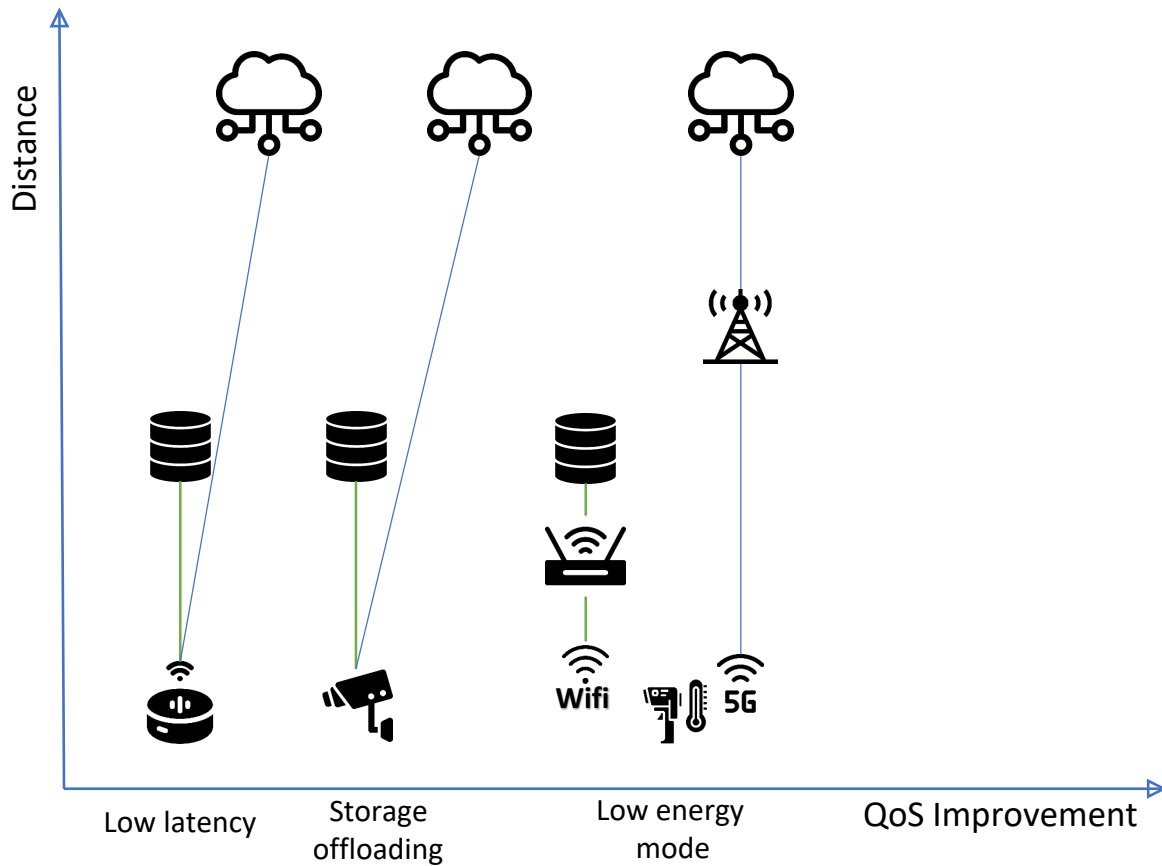
Figure 1.1.: Temporary QoS improvements

2. Object detection with real-time images requires low latency

3. Sending image streams in real-time requires high bandwidth

Also, there are multiple uses cases available for IoT devices that are required to perform object detection.

1. A CCTV tries to identify persons or other objects of interest

2. A connected vehicle tries to identify objects on the road

3. A passenger is displayed real-time information of surrounding objects via augmented reality

4. A robot automatically picks up objects for relocation in a warehouse

The state-of-the-art technique to perform object detection is with a Convolutional Neural Network (CNN). CNNs currently offer the best performance of object detection techniques. Pre-trained models are typically between 100MB and 2GB in size and can go as low as 5MB if

designed for specialized APUs. IoT devices might not be able to perform inference themselves due to the following reasons:

1. Lack of storage

2. Lack of computational power

3. Incompatible software

1.2. Assumptions

We assume the following points for outsourced object detection in an Edge Computing marketplace:

1. Contractors (edge servers) are stationary (reappearing actors)

2. Outsourcers (IoT devices) are mobile (reappearing and adhoc actors)

3. Limited set of outsourced task types (reappearing task types)

4. Both parties that agree to an outsourcing contract are rational, expected payoff maximizers

5. The payoff of the players are linear functions of their costs and payments/benefits (and risk-neutral payoff function)

6. Outsourcers pay Contractors for each processed image (this does not mean that micro-transactions or synchronous payment have to be used)

7. Cloud servers distributed pre-trained models to edge server

Additionally, we assume that the other required components of an edge computing marketplace such as resource matching, or a payment scheme is already implemented.

The following figure illustrated a scenario that matches out assumptions and expectations of a typical scenario in an edge computing marketplace:
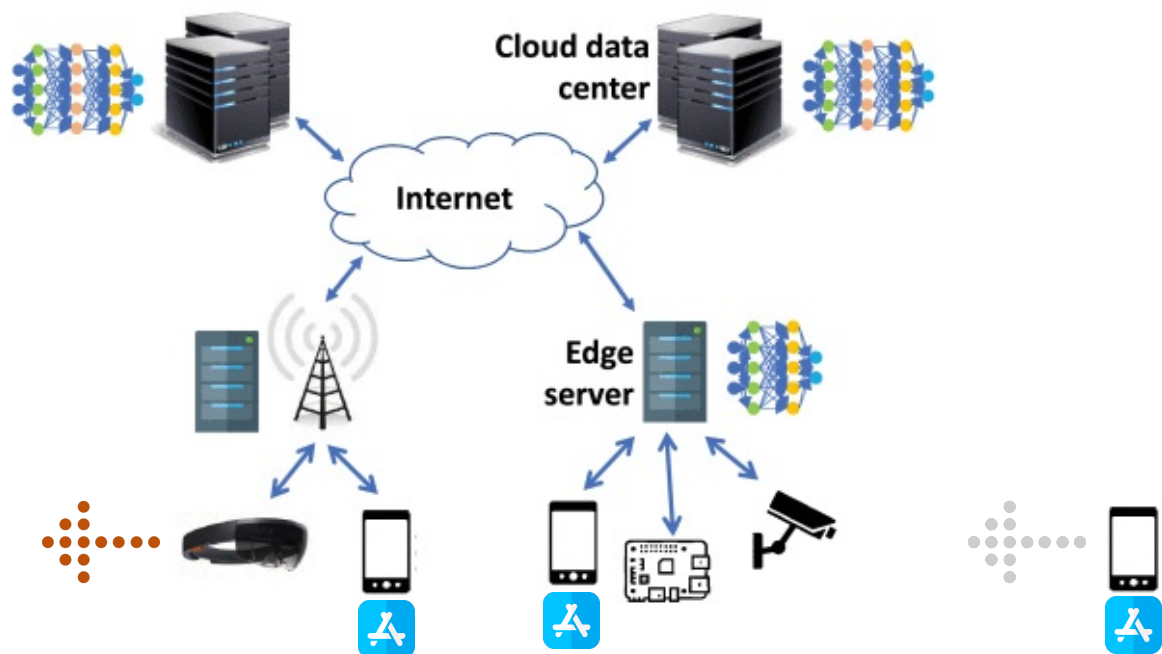
Figure 1.2.: Scenario of an edge computing marketplace with mobile IoT devices outsourcing reappearing object detection task types

1.3. Problem Statement

1.4. Research Objective

We identified the following issues when evaluating existing verification schemes proposed by current academic literature:

1. Some existing verification schemes use approaches such as Fully Homomorphic encryption that are not practical for complex functions in real-time yet.

2. We found no comprehensive list of possible protocol violations in computing marketplaces that threaten the security of a designed verification scheme.

3. Existing verification schemes often rely on trusted third parties (TTPs) that might be unavailable or become a bottleneck in real-world use cases. Other verification schemes utilize Blockchains instead of trusted third parties to record communication between participants. However, frequent Blockchain transactions cause high transaction costs and high computational overhead from a system perspective.

4. Existing verification schemes usually assume that the Outsourcer or a third-party Verifier is a trusted entity, or only design the security of their scheme from the perspective of one party rather than from an ecosystem perspective.

The objective of our research is to solve these issues by first identifying a verification approach that is practicable today and only causes low computational overhead for each participant. Afterward, we aim to compile a comprehensive list of potential protocol violations that this approach might be vulnerable to. We will then analyze which techniques proposed by current academic literature can be added to our preferred approach to prevent identified protocol violations. Using this knowledge, we intend to combine these techniques and develop new ones to design a verification scheme that is ideally resistant to all of our identified protocol violations. We expect our verification scheme to rely on a minimal amount of trusted third parties or Blockchains, and consider that the Outsourcer, Contractor, and potential Verifiers may act dishonestly or lazy.

1.5. Thesis Approach

2. Literature Review

2.1. Components of an Edge Computing Marketplace

This section gives an overview of components and their requirements that make an edge computing marketplace viable.

Table 2.1.: Components of an Edge Computing Marketplace

Component	Requirements
Verification of Outsourced Computation [1]	Secure
	Scalable
	Efficient
Payment [2]	Low fees
	Secure
	Scalable
Matching and price-finding [3]	Fair prices
	High Matching rate
Privacy Preservation [4] [5] [6]	Preserving privacy of sent data
	Preserving privacy of sender information
	Efficient
Network Architecture [7]	Low overhead
	Computation centric

From these identified components, we will focus on designing a secure verification scheme for outsourced computation that is resistant to cheating attempts, scales with many participants and causes low computational overhead for each participant.

2.2. Verification of Computation

This section gives an overview of different types of functions that can be verified by schemes that can be found in current academic literature.

There are a few verification schemes that can be used for arbitrary functions that rely for example re-execution, trusted hardware, or encryption. Other verification schemes instead only work on certain functions, allowing them to exploit mathematical properties that can be applied to certain function types. These tailor-made verification schemes may restrict the usability of a verification scheme to work on other functions than the ones that it was designed for but may improve efficiency compared to re-executing the function that needs to be verified.

Verification Schemes for Arbitrary Functions

Verification schemes for arbitrary functions generally work on either of the following three principles that the Outsourcer can utilize:

1. Restricting software or hardware access of the party conducting the computation thus preventing any modification.

2. Encrypting all values the function is executed on thus making it infeasible for the party conducting the computation to generate fake outputs[8] [9].

3. Re-executing the function on all inputs or random ones and comparing if the outputs returned by the party conducting the computation is identical.

The following paragraphs show a more detailed overview of the different techniques used for verification schemes for arbitrary functions.

Software Virtualization   Lightweight containers have become increasingly popular since the introduction of cloud computing. Containers can be integrated into the host operating system thus reducing the software overhead compared to virtual machines. When running containers on third party hardware, code integrity becomes critical. Currently, one of the most widespread types of containers are Docker containers that promise a lightweight and secure option for running code on different platforms and machines. However, Docker containers were found to be vulnerable to multiples attacks such as backdoors, phishing attacks, account hijacking, and unprotected API calls[10].
These security vulnerabilities leave multiple options for a dishonest third party to return fake results even when only having access to a secured docker container.

Trusted Execution Envoirements   Trusted execution environments (TEEs) like Intel's Software Guard Extensions (SGX) provide an isolated environment to execute code, protected from all other software running on the same machine. TEEs are architectural extensions integrated into processors that provide security guarantees against. It promises secure execution of code even when it runs on third party hardware. Apart from high computational overhead, there are multiple unsolved security challenges associated with SGX, Even with the latest update of SGX, attackers could get access to the attestation key which is used to attest computation as genuine results originated from the secure enclave[11]. Thus, when

a response is returned to the outsourcing machine, Intel SGX fails to provide secure proof about the validity of the result in practice.

Fully Homomorphic Encryption Fully Homomorphic Encryption (FHE) is a form of encryption that enables functions to be executed on encrypted data. The result of the computation is encrypted. By decrypting the result, the output is identical to the operations performed on the unencrypted data. [8] [9] The main problem with FHE is that encrypting and decrypting create high computational overhead which makes its application not practical for most use cases. Also, weak outsourcing machines will most likely not be able to encrypt all inputs in a reasonable time.

Re-execution Arbitrary deterministic functions can be verified by re-executing the outsourced function on all or certain inputs. Re-execution can be performed by the outsourcing machine itself or can be also outsourced to another machine. When comparing an output gained by performing re-execution and an output received, it can be verified whether both outputs are identical[**eisle**].

It will be shown in chapter 4 that even with a fraction of samples that are re-executed a dishonest party can be detected with high confidence. Re-execution is a practical approach of verifying computation of arbitrary function with the possible disadvantage that if the outsourcing machine is too weak to perform re-execution of the outsourced samples, a third party is needed for re-executing the function on its behalf.

Each of the presented techniques has certain advantages and disadvantages. The advantage of software virtualization is that the function and inputs itself do not have to be modified and there is almost no computational overhead introduced to the machine outsourcing the computation. The disadvantage is, however, that currently there is no entirely secure method available that prevents interference of any kind.

Likewise, the advantage of trusted execution environments is that inputs and functions do not have to be transformed, While in theory security guarantees of trusted execution environments can be higher than the one's of software virtualization, both currently fail to prevent certain attacks. Additionally trusted execution environments cause significant computational overhead to the executed function.

Encrypting all values with homomorphic encryption has the advantage of being secure against any type of interference with the function. Any modification can be easily detected as the machine conductin the computation is not able to generate valid encrypted outputs based on arbitrary values. Therefore, homomorphic encryption is usually the most computationally expensive verification scheme of all available methods.

Re-executing the outsourced function on all or certain values to verify if results gained from re-executing match received results from the party conducting the computation. The re-execution can be either computed by the machine that receives results from the outsourced computation or a third party.

The advantage of this method is that it is very cost-efficient and can lead to less than 1%

overall overhead of the verification scheme while still maintaining a high detection rate of false results. The disadvantages of this method are that it requires the outsourcing party or a third party to conduct computation.

Due to its cost-efficiency and high security, we conclude that re-executing computation is currently the most practical way of verifying computation for arbitrary function. A summary of verification schemes for arbitrary function can be found below in

Approach	Advantages	Disadvantages
Software Virtualization [10]	low computational overhead, inputs and function do not have to be transformed	not secure
Trusted Execution Environments [11]	inputs and function do not have to be transformed	not secure
Fully Homomorphic Encryption [8] [9]	highly secure	Extreme computational overhead, often unpractical
Re-execution [12]	low computational overhead when sampling is used, secure	need for a third party if outsourcing machine is not able to re-execute the function

Verification Schemes for Specific Functions

While the approaches listed in the last section can be used for arbitrary function, there are also verification schemes designed for specific functions that exploit mathematical properties specific to that function. For example, for certain function types there might be a function $v(x, y)$ that given an input x and an output $y = f(x)$ efficiently verifies the correctness of computing y. We evaluated the following verification schemes for specific function.

Pre-Image Computation   Pre-Image computation describes function types that given an input Domain D and a value y, the computations should return all values $x \in D$ for which $f(x) = y$ [13] [14]. While verifying $f(x) = y$ for a given y and input domain is trivial for most functions, the challenge when designing a verification scheme for Pre-Image computation lays in ensuring that all values x are processed and not only a subsection.

Ringers   An efficient solution to this problem is the use of true ringers and bogus ringers. The main idea of this approach is to pre-compute the function for a set of values $x \in D$ (true ringers) and $x \notin D$ (bogus ringers). Only if returned values x by the party conducting the whole computation match results of these pre-computed values, the result is accepted[13] [14].

Magic numbers   Another approach is the use of 'magic numbers' which rewards the party conducting the computation when finding a value y that is contained in the set of magic numbers. The more pre-images x a party computes, the more likely it is to find an x satisfying

$f(x) = y$ and gain a reward. Thus, magic numbers can serve as milestones to ensure that only a party that runs the computation on all inputs is guaranteed to be rewarded fully[15].

While both presented approaches to verifying pre-image computations, it is questionable if there are many use cases for pre-image computations in edge marketplaces.

Matrix Multiplication Matrix multiplication is a common operation for image processing as well as a computationally complex component of convolutional neural network training and inference. Currently, the most efficient matrix multiplication algorithm runs in $O(n^{(}2.3729))$ time.

Freivald's algorithm In general, when calculating matrix multiplication $AXB = C$, Freivald's algorithm can be used to verify $AXB = C$ given A, B, C in $O(n^2)$ with high probability [16].

While this property might be useful, using a verification algorithm that needs the result as an input instead of performing re-execution always bears the problem of lazy behavior where a machine might return that $AXB = C$ without ever running the verification algorithm. For this reason, most verification schemes available for matrix multiplication do not make use of Freivald's algorithm but instead use techniques that are not relying on a verification algorithm that needs the result matrix C as an input.

Permutation There are verification schemes for matrix multiplication based on Homomorphic encryption, secret sharing [17], and permutation [18] [19]. Verification schemes based on permutation promise to feature input and output privacy like Homomorphic encryption but with higher efficiency. As the matrix multiplication problem has to be encrypted to a secure matrix multiplication problem, permutation is still not practical with current hardware to perform real-time computations. Also, there is criticism that a solution using permutation can be even less practical than Homomorphic encryption due to high communication costs.

Matrix transformation Another approach is to instead of encrypting the input matrix A and B to generate a random matrix A' and B' and then sending $A' - A$ and $B' - B$ to on of two remote servers while the other remote server receives different transformed matrices. [20]. As only the outsourcing machine knows the original matrices and the two remote servers receive a different transformed matrix, collusion should be not possible. The advantage of this scheme is that it additionally provides input privacy and creates less than $O(n^2)$ computational complexity, We can conclude that this scheme uses complete re-execution but provides input privacy through matrix transformation. Also, we analyzed that this scheme also works with sampling-based re-execution.

While presented schemes feature input privacy in addition to secure verification, these features come with high complexity with the exception to [20]. Thus, we can conclude that general-use sampling-based re-execution is the best technique for verifying matrix multiplication. If the Outsourcer is powerful enough to conduct verification itself, Freivald's algorithm can improve the performance of this verification. Transforming matrices before

sending them to remote machines according to [20] might be a useful extension for providing input privacy also for verification schemes based on re-execution.

Matrix Inversion   Matrix inversion problems can be found in scientific and engineering problems. Matrix inversion comes with the same computational complexity as matrix multiplication. The correctness of a result can also be checked similarly: Given an input matrix A resulting matrix B it simply has to be checked whether AXB is equal to the identity matrix. Freivald's algorithm can be used to do this verification more efficient [16].

   Secret sharing and Monte-Carlo verification   One proposed verification scheme for verifying matrix inversion uses secret sharing and two remote servers that perform matrix inversion [21]. This has the advantage that input privacy is preserved. This scheme utilizes a Monte-Carlo verification algorithm to only check if random samples are correct and not all outputs.

   In conclusion, verifying matrix inversion is a similar problem as verifying matrix multiplication and therefore comes with the same complexity implication. Overall, matrix multiplication offers more use cases than matrix inversion.

Linear Equations   Linear equations are a popular type of computational tools used in analyzing and optimizing real-world systems. Large scale linear equations are computationally complex to solve.

   One proposed verification scheme for verifying large scale linear equations is to use Homomorphic encryption and outsource only certain complex operations to a remote server [22]. This scheme features the general advantages and disadvantages of using Homomorphic encryption.

Polynomials   Polynomial functions are fundamental in engineering and scientific problems.
   One proposed verification scheme utilizes arithmetic circuits to calculate the polynomial and ensure input and output privacy. The Contractor generates a proof that is checked by a third party Verifier whether computation was executed correctly, without revealing inputs [23]. Signatures are used for recording messages. In their test setup, the verification scheme takes 22ms per delegated polynomial function. Security concerns with the initial scheme have been addressed in a later publication [24]

Regression Analysis   Linear Regression Analysis is a data analysis technique applied across multiple domains. Performing linear regression over a large dataset is a computationally expensive task.

   Input transformation   One proposed verification scheme for verifying linear regression analysis features input and output privacy and an efficient verification algorithm that works on encrypted results. Given an input problem $\varphi(X, Y)$ the Outsourcer uses a secret key k to transform the problem into $\varphi_k(X', Y')$. The advantage of using a transformed problem

for linear regression analysis is that outputs can be verified on the transformed problem without re-transforming it first [25]. To verify if computation has been carried out correctly the Outsourcer has to check whether the regression parameter β' can be used to satisfy $Y' = X' * \beta'$. Thus verifying linear regression analysis is computationally inexpensive due to its trivial verification algorithm.

Convolutional Neural Networks   In deep learning, a Convolutional Neural Network(CNN) is a class of deep neural networks that is usually applied to image analyzing digital images. CNNs achieve state of the art performance in classifications tasks such as object recognition, optical character recognition, face recognition, and natural language processing. Compared to other types of neural networks, CNNs are easily trained with fewer parameters and connections while providing a better recognition rate. However, both training a CNN and using a CNN for predictions on new data (inference) is still extremely computationally expensive and often requires the use of powerful GPUs or APUs. For weak machines, outsourcing the inference of a CNN is particularly interesting to utilize recognition results for its application.

   Partial verification   One proposed verification scheme utilizes embedded proofs to generate a proof over a part of the used matrix multiplication during inference instead of the whole process [26]. Essentially, this scheme further optimizes re-execution by only checking parts of one iteration instead of re-executing the whole computation. It should be noted, however, that only verifying the matrix multiplication part is not sufficient to verify the integrity of the whole inference process.

   Zero-knowledge succinct non-interactive argument   Another approach is to use Zero-knowledge succinct non-interactive arguments (zk-SNARK). zk-SNARK have the advantage that they provide input and output privacy and privacy of the used model. However, zk-Snarks require a proof time between 8 hours [27] to 10 years [28] and 80GB to 1400TB storage overhead. This means that while offering valuable privacy-preserving properties on top of verification and outsourcing, zk-SNARK are far from practical.

   Matrix and weights transformation   SecureNets introduces a verification scheme where both data X and weight matrix W get transformed to X' and W' before sent to the Contractor to preserve input and mode privacy. When receiving a result Y' based on X' and Y' the Contractor simply randomly checks if $y'_{i,j}$ equals $w'_{i,:} \cdot x'_{:,j}$ for a random row i and a random column j [29]. Thus, this scheme uses an optimized re-execution protocol and preserves input and model privacy.

   Arithmetic circuits and random challenges   SafetyNets introduces a verification scheme for deep neural networks that can be presented as arithmetic circuits. This has the advantage that model privacy is preserved. The Outsourcer sends random challenges to the Contractor and aborts when a random challenge is not successfully met [30]. This is a similar concept to

using sampling-based re-execution. The disadvantage of this scheme is that it only works with some types of neural networks.

Homomorphic encryption   AlexNet claims to be the first Homomorphic CNN on encrypted data with GPUs. The authors use GPUs to accelerate CNN performance over encrypted data. While their Homomorphic CNN can classify images with state-of-the-art accuracy, inference takes between 5-300 seconds per image [31]. This shows that Homomorphic encryption is not practical yet for verifying the inference of a CNN.

In summary, the practical approaches for verifying CNN inference rely on re-execution or similar concepts. With application-specific adjustments used by schemes such as SecureNets re-execution can be further simplified and input/output, and model privacy can be added at additional computational cost.

Other functions

Input transformation   Input transformation can also be applied to other functions. The general approach is to transform an input x with a random value r to create x' that can be outsourced to a remote machine. This approach works whenever the resulting y' of applying outsourced function f on x' can be re-transformed to y by the Outsourcer. If this approach can be combined with re-execution to enable secure re-execution that is resistant to collusion and preserves input and output privacy. One proposed scheme utilizes input transformation for computing the characteristic polynomial and eigenvalues of a matrix [32].

Arithmetic circuits   Arithmetic circuits can be used for all problems that can be represented as an arithmetic circuit. Using arithmetic circuits has the same benefit of fully homomorphic encryption but is less computationally expensive. When a problem can be presented as an arithmetic circuit [33] proposes a verification scheme with a third-party Verifier. The disadvantage of this scheme is that computational cost is still high and the Outsourcer is assumed to be honest.

In summary, we conclude that approaches for verification schemes for specific functions still show several similarities. Most of these schemes use either a modified form of re-execution or encryption with their approaches to enable secure verification. Encryption can sometimes replace the need for re-execution as the Contractor is not able to generate a false encrypted response that meets simple integrity checks when decrypted. Therefore, these schemes are also not practical for specific functions. specially schemes that rely on Homomorphic encryption are also not practical in real-time when optimized for specific functions.

Schemes that use re-execution are mostly practical and use either a more efficient verification algorithm than re-executing the whole functions such as only re-executing parts of a function, random challenges, or efficient algorithms such as Freivald's algorithm. Additionally, input transformation seems a promising technique that works for a range of different functions and enables input and output privacy at a much lower computational cost than Homomorphic encryption, secret sharing, or schemes that rely on arithmetic circuits.

Schemes that try to optimize re-execution by only re-performing parts of the function over one input are problematic as incorrect results that were caused by other parts of the function can be overlooked. We conclude that it is more secure to decrease the input size of values that are re-executed rather than focusing on parts of the function.

After assessing multiple schemes with different approaches we conclude that sampling-based re-execution is practical for arbitrary functions and features low computational overhead. If the use case also requires privacy-preservation then input transformation is a technique that works for a range of specific functions and preserves input and output privacy. Also, Input transformation in combination with re-execution can easily prevent lazy Verifier behavior and collusion attempts. If the Outsourcer uses two different random numbers to transform the inputs sent to Verifier and Contractor, collusion between Verifier and Contractor is infeasible as they cannot produce two transformed outputs that are equal after re-transforming without knowing the random numbers used by the Outsourcer.

It should be noted that nearly all evaluated verification schemes in this section are not secure if the threat model includes a dishonest Outsourcer or colluding parties. Most verification schemes are designed from an outsourcer perspective and not from an ecosystem perspective where all parties are protected against dishonest or lazy behavior. The following sections will focus on how different verification schemes target threats that can result in a scenario with re-execution and Verifiers.

The tables 2.2 and 2.3 below summarizes verification scheme approaches for specific functions, their use cases, and their trade-offs.
'

Table 2.2.: Verification scheme approaches for different function types

Function type	Use Case	Approach	Advantage	Disadvantage
Pre-Image Computation	Scientific function evaluations	Ringers [13] [14]	Efficient verification	Computational overhead for Outsourcer to calculate bogus ringers
		Magic Numbers [15]	Efficient verification	Appraoch assumes honest Outsourcer
Matrix Multiplication	Fundamental operation for image processing and Neural Networks	Freivald's Algorithm [16]	More efficient than re-execution	Requires result matrix as input - promotes lazy behavior
		Secret sharing [17]	More efficient than Homomorphic Encryption	High computational overhead
		Permutation [18] [19]	More efficient than Homomorphic Encryption	Not practical
		Matrix Transformation [20]	Input and output privacy, low computational overhead	-
Matrix Inversion	Scientific and Engineering problems	Freivald's Algorithm [16]	More efficient than re-execution	Requires result matrix as input - promotes lazy behavior
		Secret sharing and monte-carlo verificaiton [21]	More efficient than Homomorphic Encryption	High computational overhead
Linear Equations	Analyzing and Optimizing real-world systems	Homomorphic Encryption [22]	Input and output privacy	Not practical

Table 2.3.: Verification scheme approaches for different function types

Function type	Use Case	Approach	Advantage	Disadvantage
Polynomials	Fundamental functions in engineering and scientific problems	Arithmetic Circuits [23] [24]	More efficient than Homomorphic Encryption	Moderate computational overhead
Regression Analysis	Data analysis	Input transformation [25]	More efficient than Homomorphic Encryption	-
Convolutional Neural Networks	Classification tasks, Natural language processing	Partial Verification [26]	More efficient than re-execution	Part of the computation remains unverified
		zk-SNARK [27] [28]	Input, output, and model privacy	Not practical
		Matrix and weights transformation [29]	More efficient than zk-SNARKs	Transforming complex models can be difficult
		Arithmetic Circuits and random challenges [30]	Efficient verification	Only works with certain CNNs
		Homomorphic Encryption [31]	Input, output, and model privacy	Not practical in real-time

2.3. Verification of Convolutional Neural Network Inference

As we test our designed verification scheme with object detection using Convolutional Neural Networks, we explain in this section why sampling-based re-execution is also currently the most practical technique to verify neural network inference.

As discussed in the previous sections we evaluated 5 verification approaches for verifying CNN inference and 4 approaches for verifying arbitrary functions. As software virtualization and trusted execution environments have still too many security problems, we do not consider them as a secure choice for verifying neural network inference. Fully Homomorphic encryption, zk-SNARKs, and also AlexNet which is based on Homomorphic encryption are still far off from allowing inference on real-time video/image streams. Arithmetic circuits are more efficient than Homomorphic encryption but transforming an existing CNN to an arithmetic circuit is not only difficult from a methodological point of view but also comes with additional computational overhead. Transforming input matrix and weights before sending them to a remote server is a promising approach for preserving input, output, and model privacy but also has the problem that transforming CNN models with a complex architecture is difficult.

This leaves us with re-execution as an approach that can verify arbitrary functions and does not require any transformation of the CNN model or the input values. This also makes this technique particularly efficient for the Outsourcer. Since CNN inference is a complex task and computationally weak machines may not even be able to perform sampling-based re-execution in real-time, a third-party Verifier may be included in verification. The possible optimization of only verifying parts of the matrix multiplications of the CNN inference is not advised due to the risk that other parts of the functions are not executed properly.

In summary, sampling-based re-execution is out preferred technique to verify CNN inference due to its following properties:

1. Low computational overhead for Outsourcer

2. No methodologically difficult modification of the CNN model required

3. Provides high security if all potential threats (collusion, lazy, and dishonest behavior) are addressed

4. Little to no computational overhead for Contractor and Verifier

2.4. Verification of outsourced Computation using Re-execution

Lots of verification schemes use re-execution performed by the Outsourcer or a third party entity to verify the integrity of outsourced computation results. As discussed in previous chapters, re-execution is a practical and secure way to verify arbitrary functions. However, a verification scheme that is resistant to different threats has to feature additional measurements on top of re-execution to ensure that no party can cheat in the process. Using re-execution without additional measurements does not protect for example from collusion where Verifier

and Contractor agree on a false response that when generated saves computational resources in comparison with performing the actual computation. Likewise, there have to be measurements so that the Outsourcer is not able to reject payment or falsely accuse Verifier or Contractor of cheating. In this section, we analyze different verification schemes that use re-execution or similar approaches and give an overview of assumptions and measurements these schemes introduce to deal with dishonest behavior or unintended protocol violations such as timeouts.

Outsourcing computation is a variation of the principal-agent problem. The Outsourcer can be viewed as the principle that outsources a job to a Contractor that can be viewed as the agent. Multiple attributes make this problem sensitive to dishonest behavior.

1. The Outsourcer is not able or does not intend to verify the correctness of the whole outsourced computation.

2. The Contractor wants to utilize minimal resources for outsourced computation.

3. Contractor might be "lazy but honest", lazy and dishonest or malicious to save resources or harm the Outsourcer.

4. The Outsourcer might be dishonest to avoid payment after correct computation.

Verification schemes

Uncheatable grid computing [34]   This verification scheme introduces commitment-based sampling. The Contractor commits to a Merkle tree hash every few intervals and sends it to the Outsourcer. The Outsourcer then randomly selects a few samples to recompute them and send a proof of membership challenge to the Contractor. Only if the verified output matches the initial response and only if the proof of membership challenges is successful, the contract is continued otherwise, the Contractor is detected cheating.

The major advantage of this scheme is that an Outsourcer can detect a cheating Contractor with high probability and low computational overhead. Merkle trees are used to require only one signature per specified interval instead of signing all responses. Sampling-based re-execution is used to verify results with low computational overhead and high probability.

The disadvantage of this scheme is that it does not feature an ecosystem perspective and therefore does not deal with dishonest Outsourcers. This additionally leads to the problem that the only action the Outsourcer can take is to abort a contract. It cannot enforce a fine towards the Contractor as there are no digital signatures used in this scheme to record communication securely. Also, it does not consider that a third party Verifier might be needed to perform the re-execution. Other possible contract violations than dishonest behavior such as timeouts or low response rates are not considered. Assuming no third party Verifier is needed to perform sampling-based re-execution, the following security problems remain.

1. The Contractor can claim to not have sent a false response when detected due to the lack of signatures.

2. The Outsourcer can reject payments due to the lack of signatures.

These disadvantages lead to the conclusion that this scheme can only be used from the Outsourcer's perspective to detect dishonest behavior of a Contractor and abort the contract if dishonest behavior is detected. This makes cheating a risk-free strategy for the Contractor apart from losing an active contract. This scheme has to be modified if the Outsourcer is not powerful enough to perform re-execution and relies on a third-party Verifier.

Security and privacy for storage and computation in cloud computing [35]   This verification scheme uses sampling-based re-execution to verify results of outsourced computations from an Outsourcer to a cloud server. Within specified intervals, the cloud server commits to signed Merkle roots over its results when returning them to the Outsourcer. The cloud user can instruct a Verifier to verify these results by checking if the signature of the signed Merkle root is correct and if random samples were computed and returned correctly. The Verifier sends back the result to the cloud user which can then decide on further actions.

The main advantages of this scheme are that no third party is required during the verification process except the Verifier, Outsourcer, and the Contractor. Also, digital signatures are used to provide a record of all outputs that the Contractor sent. Merkle trees are used to require only one signature per specified interval instead of signing all responses. Sampling-based re-execution is used to verify results with low computational overhead and high probability.

The major disadvantage of this scheme is that Verifier and Outsourcer are assumed to be honest. This can lead to the following security problems:

1. The Outsourcer and the Verifier collude to falsely accuse the Contractor of cheating

2. The Contractor and the Verifier collude to send back false results by investing minimal computational resources

3. A lazy Verifier can always respond that responses to be verified are correct to not invest any computational resources

4. It is not specified how a Contractor can enforce payment from a dishonest Outsourcer that tries to reject payment

These security problems lead to the conclusion that this verification scheme only works if the Verifier and the Outsourcer can be assumed as honest. In this case, it successfully detects all cheating attempts by a Contractor with high probability, low computational overhead, and no requirement of a third party.

Bitcoin-based Fair Payments for Outsourcing Computations of Fog Devices [1]   This scheme uses commitment based sampling and assumes that outsourcing jobs are exclusively pre-image computations. Commitment-based sampling refers to the use of Merkle trees that are first use to commit to a range of output values via a Merkle root hash. The Outsourcer

then sends random challenges picked by sampling techniques to verify results with high probability. At the start of the scheme, the Contractor and the Outsourcer create a Bitcoin contract to create a tamperproof, publicly verifiable agreement. The contract includes the task and a deposit transaction by the Outsourcer that can be redeemed by the Contractor if its condition is met. Once the Contractor sends results to the Outsourcer that satisfies the challenge, the Contractor gets automatically paid by the Bitcoin transaction. A trusted third party is used to resolve conflicts.

The proposed verification scheme has a few advantages. Using Merkle trees for committing to outputs is an efficient way to verify a large number of outputs with only a few or even one sample and with low communicational overhead. Using a Blockchain is a very secure method of creating records of key communication and automatically enforce payments.

There are however also major disadvantages, why we do not recommend this verification scheme for most use cases. One major disadvantage of this scheme is that it exploits trivial verification properties of pre-image computation which makes it unusable for other more frequent use cases. Also, the scheme assumes to have a trusted third party available at all times for conflict settling. Besides, there is a security problem as responses from the Contractor are not digitally signed. Therefore, no proof is available for the Outsourcer to reject payment properly. For this reason, the scheme does not prevent a dishonest Contractor from sending false or no responses and does not prevent a dishonest Outsourcer to claim that no valid computation result was received.

Integrity verification of cloud-hosted data analytics computations [36]   This verification scheme uses artificial data values to enable verification even for computationally weak Outsourcer. This scheme is mainly targeted at use cases with a large amount of data such as machine learning or deep learning tasks. The key idea of this verification scheme is that the Outsourcer adds artificial data values to its inputs and checks whether the Contractor reacts accordingly. For example, the Outsourcer could insert an object at a certain position in an image and verify if an object detection response sent by the contractor detects the artificially added object successfully.

The main advantage of this method is that it can be utilized by a weak Outsourcer without the need for a third-party Verifier. Also, the computational overhead of modifying input data is low.

There are some disadvantages however to the technique itself. If we stick to the example with an image, then artificially added objects can block other relevant parts of the images. Thus, we advise to only use this technique rarely for data that is not used for further analysis but rather discarded. Also, a Contractor might detect an unregular change of two consecutive images when an object is added artificially and react with honest computation accordingly. This issue is addressed by the paper but comes with additional computational overhead of transforming input data. It should finally be noted that even if an object is added artificially, there might be the chance that the CNN performing object detection is not able to detect it even if it behaves honestly. Thus, this technique has to be used probabilistically where missing detections are tolerated to a certain extent. The alternative of not placing the artificial data

randomly but falling back on a set precomputed to ensure detection with 100% probability is not advised as the Contractor can learn this limited set of precomputed images over time.

If these concerns are addressed, adding artificial data values can be a promising technique to replace re-execution in certain use cases due to higher efficiency. It should be noted, however, that this technique still has to be supported by other measures that prevent dishonest behavior by Outsourcer and Contractor. As this verification scheme was not designed from an ecosystem perspective, the usual drawbacks that result from not using signatures or other forms of records come along with this scheme if no measures are added.

Game-Theoretic Analysis of an Incentivized Verifiable Computation System [37] This paper modifies an existing verification scheme with re-execution and Verifiers by adjusting game-theoretic incentives and prevention of Sybil attacks. These modifications are also relevant from previously discussed verification schemes in this chapter that suffer from possible collusion of Contractor and Verifier. At first, this scheme proposes to randomly select a Contractor and a Verifier and to assign fines and bounties when a party is detected cheating or successfully accuses another party of cheating. Sybil attacks refer to the problem that a Contractor or Verifier can generate multiple identities to be selected both as Contractor and as Verifier. This scheme proposes two third parties - an arbiter and a judge to prevent this behavior. The verification scheme uses Ethereum as a Blockchain solution to securely enforce payment, fines, and bounties. Merkle Trees are used to commit to multiple outputs at once.

The advantage of this scheme is that it successfully prevents collusion of Verifier and Contractor if both parties are rational, expected payoff maximizing entities. Also, it is the only scheme that we analyzed that addresses Sybil attacks.

The disadvantage of this scheme is that preventing Sybil attacks comes with the high overhead of a third party and Ethereum transactions. Also the Outsourcer, judge, and arbiter are assumed to be honest which may not be a valid assumption.

Anticheetah: Trustworthy computing in an outsourced (cheating) environment [38] The authors of this paper propose a system called Anticheetah which outsources a computation to multiple nodes in a multi-round approach. Each round different inputs are sent to each node and compared by a trusted master node in case the same input got sent to more than one node. Each round all nodes are evaluated by the master node for the following criteria in order of importance: Reliability, cost-efficiency, and timeliness. This leads to an accurate reputation balance over time for each node. In each round, nods are selected first that have the best reputation balance. If a node gets its reputation decreased in reliability even though it was honest, due to a multi-round approach this scenario does not have a big impact on overall node balance. The scheme assumes guaranteed message delivery, no lost messages, and zero communication overhead.

The advantage of this scheme is that due to its multi-round reputation system, collusion between multiple nodes is disincentivized. Also, through a reputation system that also includes cost-efficiency and timeliness, it can be ensured that over time only nodes that can provide a high quality of service remain as Contractors.

The disadvantage of this scheme is that multiple nodes are needed for serving as a Contractor. In scenarios where not many nodes are available for computation, this can be a problem. Also, the master node causes an additional need for trusted third party resources. The assumptions of this scheme such as no lost messages and zero communication overhead are not realistic for scenarios with weak connection.

Efficient fair conditional payments for outsourcing computations [39]   This verification scheme uses a semi-trusted third party to solve disputes between participants. The verification scheme uses true and bogus ringers that were discussed earlier. These lead to efficient verification for certain functions. The verification scheme is designed to prevent dishonest and lazy Contractors from succeeding with their strategy. Only in case of conflict or delays the semi-trusted party is required to participate. Signatures over key communication are used to let the third-party come to a rightful conclusion in case of dispute.

The advantage of this scheme is that it works for dishonest Contractors and to a certain extent also to dishonest Outsourcers.

The disadvantage of this scheme is that it requires a semi-trusted third party that can become the bottleneck of the system and might not be available at all times.

Mechanisms for Outsourcing Computation via a Decentralized Market [12]   The authors of this verification scheme propose a smart contract-based protocol running on a Blockchain to set incentives for Outsourcers and Contractors to participate in an outsourcing market place. For verification, their scheme uses semi-trusted third parties. The incentives in the system a set in a way that dishonest behavior is discouraged. For verification, their scheme uses sampling-based re-execution.

The advantage of this scheme is that incentives are set in a way that collusion is made unlikely when dealing with rational pay-off maximizing participants.

The main disadvantage of this scheme is that current transaction fees in Ethereum are too costly for short contracts. Also, Verifiers have to be partially-trusted.

Incentivized outsourced computation resistant to malicious Contractors [40]   This verification scheme proposes that an Outsourcer sends it inputs to at least two Contractors performing the identical task. A trusted timestamping server is used to keep records of inputs, results, and timeliness, and a trusted central bank is used to conduct payment. The general procedure of this scheme consists of outsourcing the same job to $n = 2$ Contractors and accepting the result if responses match. If responses do not match, the job gets outsourced again to $n_{new} = n^2$ Contractors until no conflicts arise. The timestamping server records all communication between Contractors and Outsourcer to ensure verifiability of the outcomes. The incentives in this system are sent in a way that being honest is the dominant strategy, and rationally expected pay-off maximizers will not collude.

In their simulations, the authors detected that with a fine-reward ratio of 20% and 25% malicious Contractors, on average 2.04 Contractors are needed until agreement is reached.

The minimum total computation complexity of outsourcing is 200% of the original job plus the overhead of the system which is 2% on average according to the authors.

The advantage of this scheme is that the timestamping server records communication from both the Outsourcer and Contractors. Therefore, it not only prevents dishonest behavior from Contractors but also from the Outsourcer which might try to refuse payment. Also, the quality of service violations can be punished by comparing the times of Outsourcer and Contractor communication on the timeserver.

The main disadvantage of this scheme is that a trusted timestamping server is needed that records nearly all communication of all ongoing contracts of different Outsourcers and Contractors at all times. Thus, the timestamping server can easily become the network or performance bottleneck of the system. Also, the Contractors perform complete re-execution which leads to a minimum computational overhead of 100% per job alone accounted for by that decision.

Approaches to prevent protocol violations   In summary, the evaluated verification schemes are mainly designed to prevent dishonest behavior of a Contractor and assume the Outsourcer to behave honest. Only a few examples such as [40] also protect against dishonest behavior from the Outsourcer. The following approaches are used by different schemes to ensure secure verification of outsourced computation and accountability for dishonest behavior:

Complete/Sampling-based re-execution   This technique refers to re-executing the outsourced function overall inputs or random sample inputs. If the Outsourcer is not powerful enough to perform re-execution, it can outsource the re-execution to a Verifier (third-party) such as in [40].

Signatures   Digital Signatures are a secure way to keep records of communication that can not be tampered with. If an input and related job information is signed by the Outsourcer before sent to the Contractor, the Contractor can prove having received this data from the Outsourcer to any third party, since it is unable to generate a valid signature of the Outsourcer over data by itself. A few schemes such as [1] use Outsourcer signatures to commit to a contract with a specified reward.

Likewise, if the Contractor signs results and related job/input data before sending them back to the Outsourcer, the Outsourcer can prove to any third party which values the Contractor responded with. Schemes such as [35] use Contractor signatures to commit to responses when sending them. This way, if a false response from a dishonest Contractor is detected, the signature can act as proof that the Contractor indeed sent the signed value.

Distributed computing with multi-round inputs   This is a special technique introduced by [38]. A job gets outsourced to multiple Contractors with certain overlap and compared by a trusted master node. Inputs are sent in multiple rounds. In each round, Contractors are evaluated based on reliability and performance criteria. This scheme ensures that after a

few rounds a near-optimal matching of honest Contractors and a trusted Outsourcer can be achieved. However, it requires a large ability of Contractors to work properly.

Commitment to messages using Merkle Trees Merkle Trees can be used as a data structure to efficiently verify if data is contained in a large collection. The root hash of a Merkle tree can be used to verify if data is contained in the whole tree with $log_2(n)$ steps. Verification schemes such as [37] use this attribute of Merkle trees in combination with signatures or trusted third parties to commit to large amounts of inputs or outputs by only committing to the Merkle tree root hash.

Committing to inputs via a Merkle Tree has the advantage that the Outsourcer can first commit to the inputs by signing their Merkle root hash and sending the actual inputs without adding a digital signature or involving a third party. The disadvantage of committing to inputs is that all inputs must be known in advance as the commitment should be made before sending data. This makes input commitment only feasible for use cases without real-time data.

Committing to responses via a Merkle Tree has the advantage that the Outsourcer can commit to a collection of responses before receiving a proof of membership challenge by the Outsourcer. When receiving a challenge from the Outsourcer, the Contractor is not able to quickly re-compute a response and send it instead since it would not meet the proof of membership challenge. This way, instead of signing all data, it is sufficient for the Contractor to only sign Merkle root hashes and challenges in a specified interval, thus improving efficiency. The disadvantage of commit over responses via Merkle trees is that there is always a certain delay after receiving a response, before it is committed. This can lead to a loss of records in case of a dispute if the verification does not consider this risk.

Use of trusted third parties and Blockchains for records and conflict resolving Some Verification schemes such as [1] use trusted third parties (TTPs) or Blockchains to record the communication of participants. In case of dispute, the record can simply be reviewed by any participant on the Blockchain or by contacting the TTP. Also, the Blockchain or TTP can execute code to settle conflicts without bias. The disadvantage of recording all or key communication on a Blockchain or with the help of the TTP is that the amount of communication from all different contracts of a system can easily lead to them being the performance or bandwidth bottleneck of the system. Also, not in all scenarios, it can be assumed that a TTP is available. On the other hand, a Blockchain may add a large amount of computational complexity to the system, especially when "proof of work" is used.

Reputation system A reputation system is used by verification systems such as [38] to evaluate multiple Contractors based on their reliability and performance. A reputation system can exist both locally at node level or globally at a TTP or Blockchain. The reputation system can give incentives for almost no computational overhead to let participants behave honestly and provide a good quality of service. While in public systems, a negative reputation system

Approach	Advantage	Disadvantage
Complete/Sampling-based re-execution [1] [35] [34] [36] [37] [40]	Efficient verification approach to prevent dishonest behavior, works for arbitrary functions	More efficient verification algorithms are available for certain function types
Signatures [1] [35] [39] [37] [40]	Can be used to record payment promises and responses in a tamperproof manner	Account for computational overhead for each signed value
Distributed computing with multi round inputs [38]	Provides a near optimal performance and reliability matching over time	Requires a large amount of available Contractors
Commitment to messages via Merkle Trees [1] [35] [34] [37]	Can be used to reduce the computational overhead of signatures by signing only Merkle root hash and challenges	Leads to small delays in commitment/secure recording when a value is received
TTPs or Blockchains used for records or conflict resolving [1] [39] [37] [12] [40]	Can be used to securely store records and handle conflicts without bias	Requires an available TTP, or accounts for high comutational complexity wen using a PoW Blockchain
Reputation system [38]	Promotes honest behavior and good quality of service	Works best globally with the need of a TTP or Blockchain to store reputation for each node
Blacklisting [38]	Can be used at node/entity level to punish any form of protocol violation	Works best with a reputation system or node register that prevent or disincentivize genearting a new identity

can be avoided by simply creating a new identity, a positive reputation that cannot be faked may still lead to economical benefits for participants that adhere to the desired protocol.

Blacklisting   Blacklisting is a simple technique used by verification systems such as [38] to cancel all future relationships with a node at the entity or node level. Blacklisting is most useful when combined with measurements such as a reputation system or a registry system at a TTP that respectively disincentivize or prevent participants that got blacklisted to resume relationships by generating a new identity. Blacklisting can be used to punish the quality of service violations or dishonest behavior.

The following table illustrates which evaluated verification schemes utilize which techniques.

2.5. Protocol violations

Protocol violations can be categorized into the following two types:

1. Dishonest behavior such as sending back false responses, rejecting legitimate payment, collusion.

2. Quality of Service (QoS) violations such as timeouts, low response rate, or frame loss.

The key difference between both scenarios is that dishonest behavior is always intentional, while QoS violations are usually not intentional. Dishonest behavior can be further categorized into dishonest behavior by a single participant, and dishonest behavior by multiple colluding participants. This section will give an overview of identified possible protocol violations and which of the approaches explained in the last section are used by related work to prevent these scenarios. We assume the scenario, that a contract consists of an Outsourcer, a Contractor, and a Verifier. The Outsourcer outsources computation to the Contractor and random samples to the Verifier.
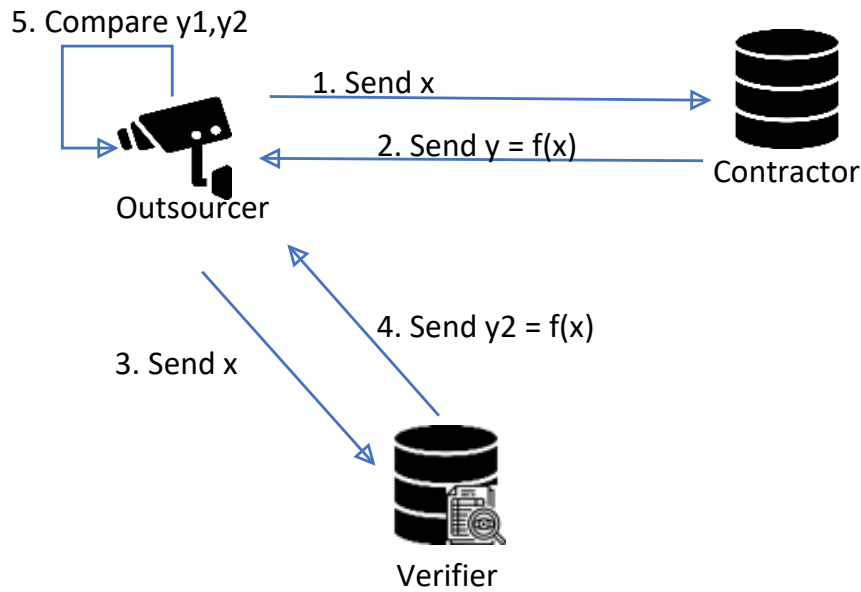
Figure 2.1.: Assumed scenario

Dishonest behavior by one participant   Each participant may behave dishonestly on its own if no measurements are set to protect against it. Generally, the incentive of the Outsourcer to act dishonestly is to reject payments to Verifier, or Contractor. The incentives of the Verifier, or Contractor to act dishonestly are to send back false responses that can be generated with less computational effort than performing the function assigned by the Outsourcer. The following scenarios can happen if one party acts dishonestly.

Contractor sends back false responses to save resources   In this scenario, the Contractor sends back false responses to the Outsourcer. The false response can be a q-algorithm that returns the correct result with probability q and is computationally less expensive to perform than the function assigned by the Outsourcer. This behavior can be detected by existing verification schemes that use re-execution, often by introducing a third party Verifier such as in [40]. The following figure illustrates the dishonest behavior.

1. Send x

2. Send y ≠ f(x)

Outsourcer
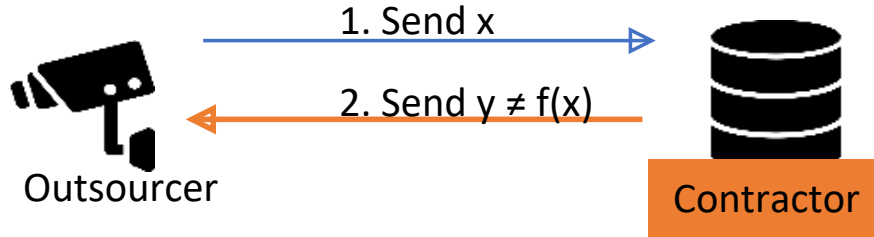
Contractor

Figure 2.2.: Contractor sends back false responses to save resources

Verifier sends back false response to save resources   Likewise, the Verifier might also use a q-algorithm that returns the correct result with probability q and is computationally less expensive to perform than the function assigned by the Outsourcer. As multiple existing schemes assume that Verifiers are honest entities, dishonest behavior can go unnoticed. However, schemes such as [40] outsource the function and inputs of two unequal responses of Contractor and Verifier to additional Verifiers to detect the party that has cheated. The following figure illustrates the dishonest behavior.

Outsourcer sends different input to Contractor and Verifier to refuse payment   An Outsourcer has multiple ways to reject payments. One example is, if it sends two different inputs to Verifier and Contractor which leads to unequal responses even if the other participants are behaving honestly. If communication of the Outsourcer is not recorded, this method can be exploited by the Outsourcer to reject payment while claiming that identical inputs were sent to the Contractor and Verifier. While schemes such as [35] use digital signatures we found no evidence that these schemes also propose to digitally sign all inputs and related information sent by the Outsourcer. Also, [40], who proposes a trusted timestamping server to record communication did not specifically mention this scenario. Thus, we consider this potential dishonest behavior as unaddressed by current academic literature. The following figure illustrates the dishonest behavior.

Contractor or Verifier tries to avoid global penalties when convicted of cheating or QoS violations   Even when a Verifier or Contractor is detected cheating by an Outsourcer it has multiple ways to avoid system-wide penalties if confronted. While it cannot prevent the Outsourcer from taking actions such as blacklisting in consequence, it might falsely communicate to a third party arbitrator trying to solve the conflict. If communication is not
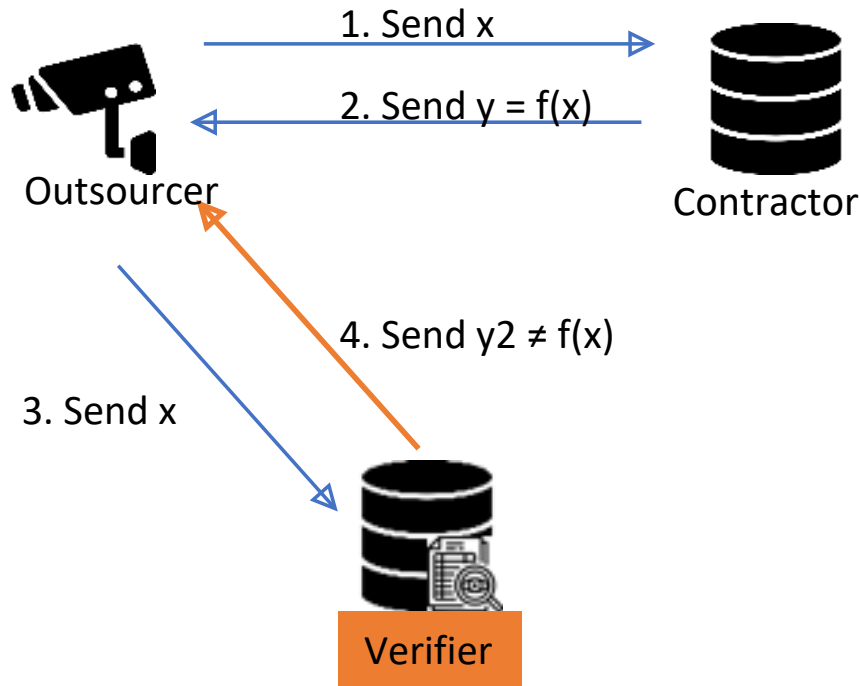
Figure 2.3.: Verifier sends back false responses to save resources

recorded in a tamperproof manner, it can claim to the Arbitrator that it has not received inputs or quickly re-compute a correct response before sending it to the Arbitrator. Verification schemes that utilize digital signatures for Contractors to commit to responses such as [37] can prevent this behavior. The following figure illustrates the dishonest behavior.

Participant refuses to pay even if obliged to by the protocol   Being obliged by the protocol to pay a reward or to pay a fine is not sufficient if that reward or fine cannot be enforced. This problem could be solved naively by waving fines and only using real-time payment. However, this strategy leads to a large amount of micro-transaction, large transaction fees, and latency bottlenecks of payment providers. Also, fines can be important disincentives to prevent dishonest behavior to take place. Verification schemes such as [1] solve this problem by utilizing a TTP or Blockchain that supports deposits and payment on another entity's behalf. The following figure illustrates the dishonest behavior.

Dishonest behavior via collusion   Collusion refers to the secret collaboration of at least two participants. As Contractor and Outsourcer have fundamentally different interests, collusion is only plausible for either Contractor and Verifier to trick the Outsourcer into believing a false result, or Outsourcer and Verifier to reject payment for the Contractor. The following scenarios can happen if two parties collude:
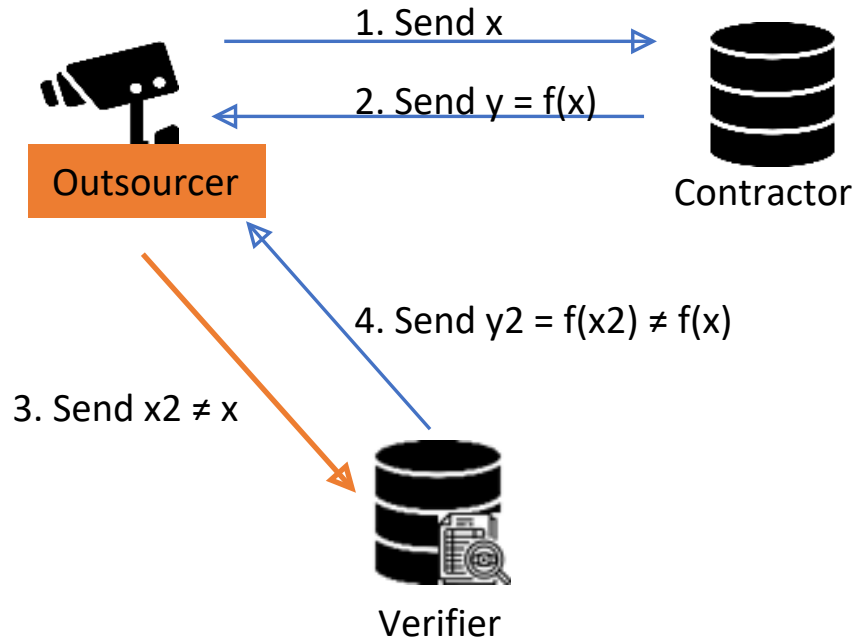
Figure 2.4.: Outsourcer sends different input to Contractor and Verifier to refuse payment

Outsourcer and Verifier collude to refuse payment and save resources   Even if inputs of the Outsourcer are recorded via digital signature it cannot prevent that an Outsourcer colludes with a Verifier to send identical inputs to Contractor and Verifier, but receiving a false response from the Verifier, claiming it was the real one. The incentive for the Outsourcer to act dishonestly with this behavior is to reject payment to an honest Contractor. The incentive for the Verifier is to save computational resources by returning a false response and not conducting the assigned function. Potentially, the Verifier could also be rewarded with a proportion of the saved Contractor reward by the Outsourcer. We found no evidence that this scenario is prevented by any of the evaluated verification scheme. Even in schemes such as [40] that outsource unmatching responses to additional Verifier, it seems that (1) this decision is not enforced by the protocol and (2) there seems to be no random assignment of Verifiers which leads to the problem that the Outsourcer can freely pick a colluding Verifier. Thus, we consider this potential dishonest behavior as unaddressed by current academic literature. The following figure illustrates the dishonest behavior.

Contractor and Verifier collude to save resources   The biggest risk for the Outsourcer is if the Contractor and the Verifier collude and send false responses to the Outsourcer. Unknowingly, the Outsourcer will continue outsourcing and pay for false responses. The Contractor and Verifier both save resources by being able to use a computationally inexpensive algorithm rather than the function assigned by the Outsourcer. Schemes such as [40] lower the possibility of collusion by designing their incentive structure in a way that being honest is
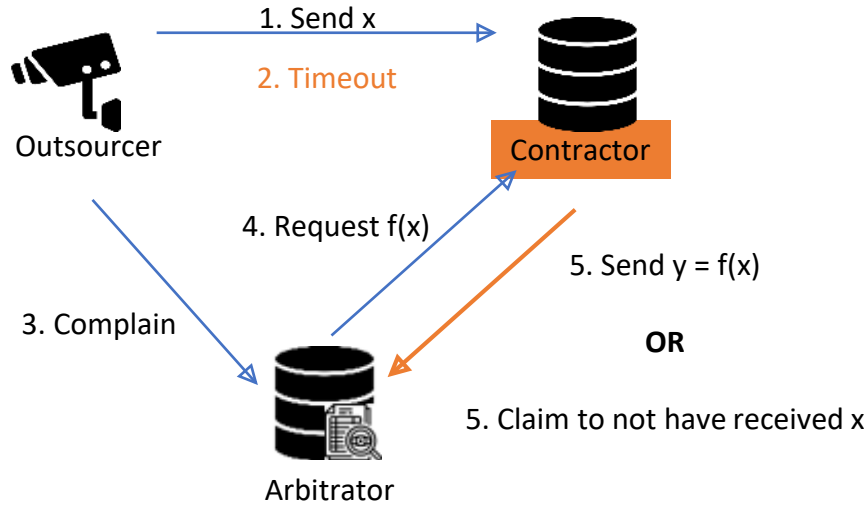
Figure 2.5.: Contractor or Verifier tries to avoid global penalties when convicted of cheating or QoS violations

a dominant strategy for rational expected pay-off maximizing participants. This method only ensures probabilistic protection and may fail if the Verifier and the Contractor are aware of each other's identities. The following figure illustrates the dishonest behavior. The following figure illustrates the dishonest behavior.

Quality of Service violations Quality of Service (QoS) violations refers to the measurement of overall service performance. An Outsourcer is likely to expect a certain QoS from a Contractor and a Verifier such as maximum response time, or minimum response rate. QoS violations can be (1) unintended in case of weak connections, or unexpected timeouts or (2) reckless in case of overutilizing resources to ensure 100% resource utilization at all time or (3) intended to harm an entity. The problem is that it is hard to tell which of the three cases is true if QoS violations take place. Thus, dishonest behavior may not be assumed, and penalties might be lower or non-existent compared to proven dishonest behavior as described above. The following QoS violations might occur.

Timeouts At any point during a contract, a participant may time out due to network problems, over-utilization, or other problems. While some Verification schemes ignore that problem since an Outsourcer could simply cancel the contract, others such as [38] keep local reputation systems that frequently compare all Contractors in their performance. This implicitly leads to the blacklisting of weak performers. Another approach is that the Outsourcer commits to its inputs via Merkle Trees or sends them to a trusted time-stamping server such as in [40]. These techniques come with computational or communicational overhead but can be used to create a globally usable record of when inputs are sent and responses are received. The following figure illustrates the QoS violation.
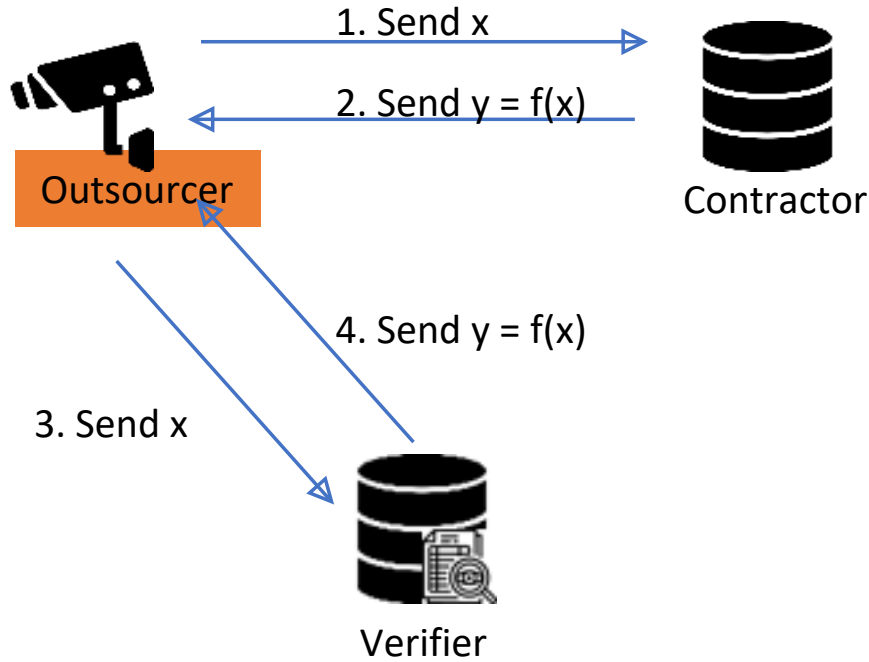
Figure 2.6.: Outsourcer refuses to pay even if obliged to by the protocol

Low Response Rates   Low response rates can happen due to weak connection, or if a participant is not able to computationally process a function in the expected time. The same techniques as for timeouts can be used to prevent this behavior.

High Response Time   High response times can happen due to network latency, or if a participant is not able to computationally process a function in the expected time. The same techniques as for timeouts can be used to prevent this behavior.

External Threat: Message Tampering   While previously explained protocol violations are all caused internally by the participants, there might also be external threats such as an attacker that tries to tamper with messages to harm a participant. We assume that all schemes that utilize signatures such as [39] are resistant to these types of attacks, as an external attacker cannot generate a valid digital signature of a participant over a message it did not send. It should be noted, however, that using Merkle trees along with signatures leads to a brief time window for message tampering between sending an unsigned message and sending the signed Merkle root hash. In this short time window, messages can be tampered with, but the tampering will be detected once the Merkle root does not match received inputs. The following figure illustrates this external threat.

In summary, we identified multiple protocol violations and summed them up by 11 description and 4 types. It should be noted that while 9 out of the identified 11 protocol
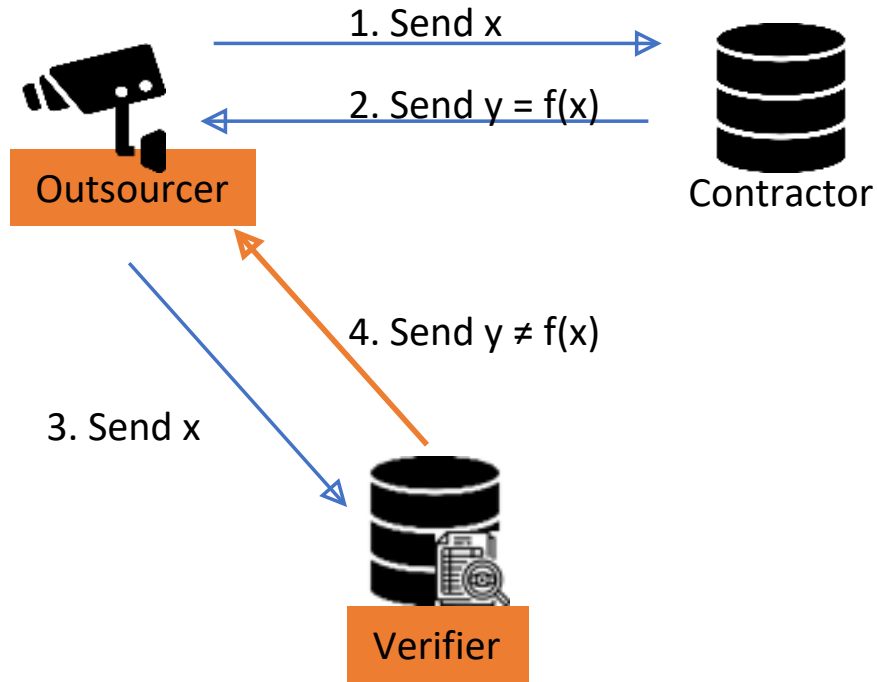
Figure 2.7.: Outsourcer and Verifier collude to refuse payment and save resources

violations seemed to be addressed by techniques used by current academic literature, we found no verification scheme that utilizes at least one technique per identified violation. This leaves the following research practice gaps to design a verification scheme that adheres provides the following attributes:

1. Prevents all of the 9 protocol violations that already have been addressed by existing solutions

2. Proposes solutions for the 2 unaddressed protocol violations

3. Minimizes the use of TTPs, Blockchains, and computationally expensive techniques to remain practicable and scalable

The following table provides an overview of the described potential protocol violations and by which techniques they are addressed by the evaluated academic literature. The next section will introduce our designed verification scheme that aims to prevent all 11 identified protocol violations while remaining efficient, scalable.

Table 2.4.: Protocol Violations and techniques to prevent them proposed by evaluated literature

Type of Violation	Referred Number	Description	Techniques
Dishonest Behavior by Individual	1	Contractor sends back false responses to save resources	Re-execution, utilization of third party Verifiers if required
	2	Verifier sends back false response to save resources	Re-outsourcing input to additional Verifiers if responses do not match
	3	Outsourcer sends different input to Contractor and Verifier to refuse payment	Not addressed
	4	Contractor or Verifier tries to avoid global penalties	Digital Signatures
	5	Participant refuses to pay even if obliged to by the protocol	TTP or Blockchain that is authorized ot conduct payment on behalf on another entity
Dishonest Behavior via Collusion	6	Outsourcer and Verifier collude to refuse payment and save resources	Not addressed
	7	Contractor and Verifier collude to save resources	Incentives designed in a way that being honest is a dominant strategy
QoS Violation	8	Timeout	Blacklisting, Review System,Recording timestamps of Merkle root hashes of inputs and responses to track QoS violations globally
	9	Low Response Rate	Same as Nr 8
	10	High Response Time	Same as Nr 8
External Threat	11	Message Tampering	Digital Signatures
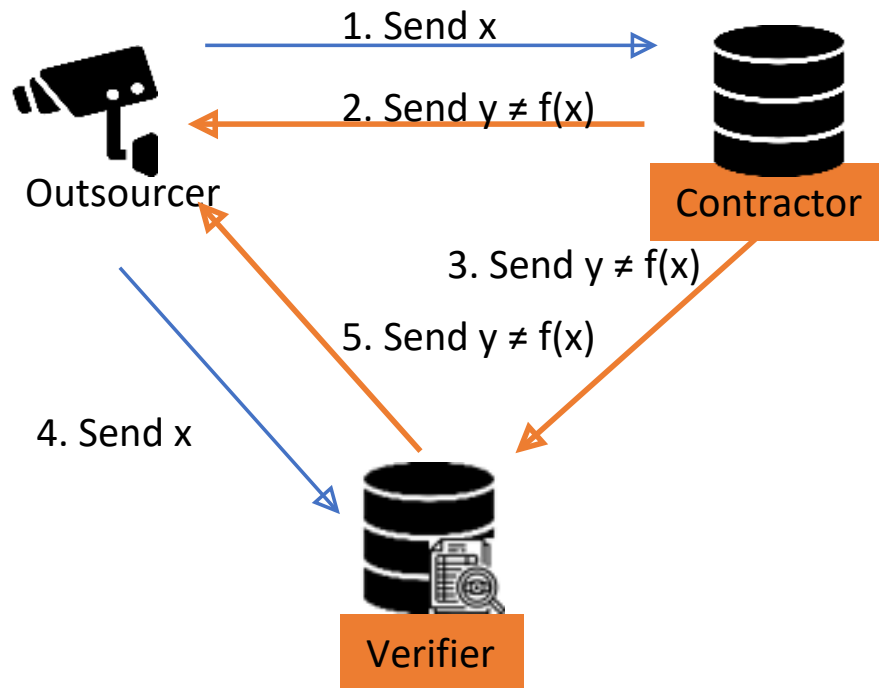
Figure 2.8.: Contractor and Verifier collude to save resources
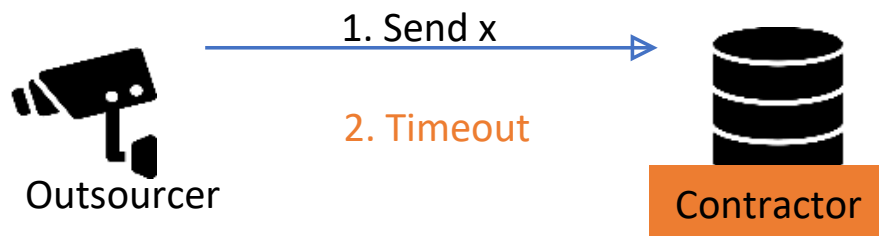
Figure 2.9.: Contractor times out
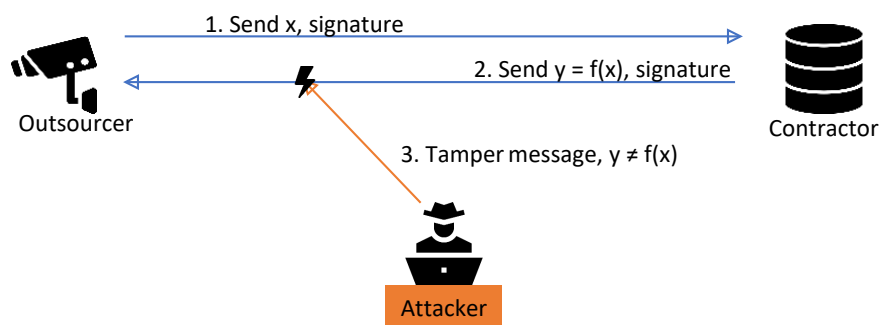
Figure 2.10.: Message Tampering by an external attacker

3. Designed Verification Scheme

This section will introduce our designed verification scheme. According to the research gaps compiled in the last section the following points were of our priority when we designed the scheme:

1. Prevents all of the 9 identified protocol violations that already have been addressed by the evaluated solutions

2. Proposes solutions for the 2 unaddressed identified protocol violations

3. Minimizes the use of TTPs, Blockchains, and computationally expensive techniques to remain practicable and scalable

As our scheme uses re-execution as a verification technique, it can be applied to any deterministic function. Also, the designed verification scheme is suitable for both latency-sensitive Edge Computing marketplaces with scarce resources as well as cloud computing marketplaces with unverified cloud service providers. Nevertheless, we focus on the Edge Computing scenario and outsourced object detection as an application when we introduce assumptions and implementation to provide specific results.

3.1. Overview

Our designed verification scheme begins at the point where an Outsourcer has successfully hired an available contractor for a certain reward per input. Both participants begin with an identical contract object that contains the reward per image, the type of function or model to be used, and other relevant data.

They first engage in a preparation phase. In the preparation phase, both participants generate a random number in a combined manner without the ability for any participant to predict the number. The random number r is only revealed to the Contractor but not to the Outsourcer. We called this process randomization. Randomization ensures that if there is a sorted list of n available Verifiers the random Verifier at position $r mod n$ is picked. To prevent planned collusion, neither Outsourcer nor Contractor can decide on their own which Verifier will be picked. Also, the Contractor is not informed about the choice of the Verifier so it is harder to contact it for an ad-hoc collusion attempt. Both Verifier and Contractor generate an identical contract. After that process, all contracts are checked if their specified fines, rewards, and bounties lead to a dominant strategy of being honest from a game-theoretic incentive. When designing the incentives in such a way, ad-hoc collusion of Verifier and Contractor can

be further prohibited. However, adhering to game-theoretic incentives is not enforced by the protocol.

After the preparation phase is done, the execution phase begins. In the execution phase, the Outsourcer digitally signs each input before sending it to the Contractor. The Contractor verifies the signature of the new input, performs the assigned function, and either digitally signs each response before sending it or signs a Merkle root hash over multiple inputs for higher efficiency. The Outsourcer finally verifies the signature of the response. The execution phase is performed as long as one party aborts the contract according to custom or due to a protocol violation. Once in a while, the Outsourcer sends a digitally signed sample input to the Verifier. Like the contractor, the Verifier verifies the signature of the new input, performs the assigned function, and digitally signs each response before sending it. When the Outsourcer receives a response from both Verifier and Contractor, it compares if they are equal and aborts the contract if they are not. Digital signatures are always generated over each input, respective index, and the contract hash to ensure a distinct record of each input. Additionally, the Outsourcer digitally signs with each new message to the Contractor, or the Verifier how many responses it has already received from that participant. Thus, it acknowledges a certain amount of outputs and commits to a payment to that party. If acknowledged output does not match the expected acknowledge rate set by Verifier or Contractor, they might cancel the contract due to protocol violations themselves. In all cases, at the end of the contract, every participant has a clear digitally signed record of honesty and payment promises from each participant it interacts with.

There are three potential scenarios for the closing phase: (1) A contract was aborted according to customs, (2) A contract is aborted due to a QoE violation, (3) a contract is aborted due to dishonest behavior. In the first, and usual case Verifier and Contractor submit their last received input which contains the latest number of acknowledged responses along with a copy of their contract to the payment settlement entity. If signatures match, the Payment settlement entity pays the specified reward per image specified in the contracts on behalf of the Outsourcer to the Verifier and the Contractor. Also, participants can decide to leave a positive review, assuming that the payment settlement entity might as well also feature a global review system. In the second case, a participant may blacklist the party that violated its QoS parameters and leave a negative review at the Payment settlement entity. The third case of dishonest behavior can only happen if a sample response of the Verifier was found unequal to the one of the Contractor. In this case, the Contractor is accused of cheating and receives a fine after a deadline. Within the deadline, the Contractor may perform one of our developed protocols called Contestation. If the Contractor re-outsources the input to additional random Verifiers, and responses match with the response of the Contractor, the Verifier is instead accused of cheating. Verifier and Contractor can perform Contestation as many times as they like at their own cost. The participant that receives the minority of Verifier support for their response is finally accused of cheating and has to pay a fine for cheating and needs to pay all additional verifiers consulted during Contestation. This process ensures that when a participant is accused of dishonest behavior, there is a guarantee to detect the cheating participant if >50% of available Verifiers respond with the correct response.

3.2. Initial Situation

In the initial situation, we assume there are an Outsourcer and a Contractor that have agreed to a contract. The contract contains a unique ID which makes it distinguishable from previous contacts with the same participant, the public keys of the participants that are also registered at the Payment settlement entity. Also, a reward per processed input and the function/model to be used is specified. Additionally, Contractor and Outsourcer may agree on fines, deposits, and bounties if a participant is caught cheating to increase the robustness of the protocol. Also, we assume multiple available verifiers are available and willing to agree on a contract with the Outsourcer to process random sample inputs. The following figures illustrate the initial situation.
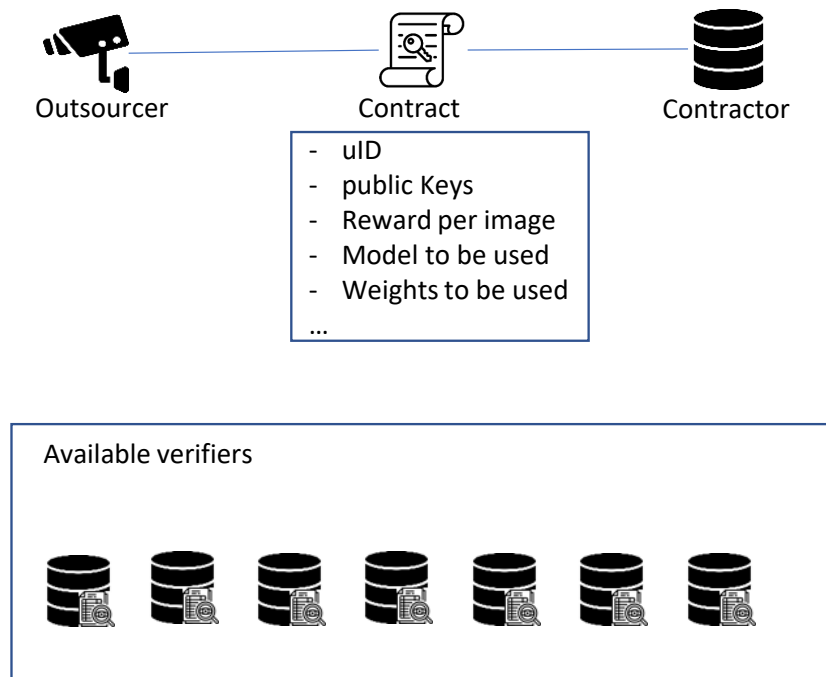
Figure 3.1.: Initial Situation

3.3. Preparation Phase

After the initial situation the objectives for the Preparation Phase are the following:

1. The Outsourcer and the Contractor should commit to a random Verifier

2. The Contractor should not know which Verifier is selected

3. The Verifier should not know which contractor is assigned to the job

   To explain these objectives we first have two distinguish between two types of collusion: Planned collusion and ad-hoc collusion.

We describe planned collusion as the scenario where two participants, either a Contractor and a Verifier, or a Contractor and an Outsourcer know each other initially and try to trick the other participant by colluding. To reduce the risk of planned collusion, Contractor and Outsourcer engage in a protocol we designed and called Randomization. The result of the protocol is the selection of a random Verifier that can not be set initially by the Outsourcer, nor the Contractor.

We describe ad-hoc collusion as the scenario where a Contractor and an Outsourcer do not know each other initially but still try to communicate and collude with each other because they expect individual benefits such as spending less computational resources by submitting a false response. This explains our second and third objectives. Thus, we designed Randomization in a way that Verifier and Contractor do not get to know each other's identity and no communication is present between them.

3.3.1. Randomization

At the start of Randomization, the Outsourcer and Contractor internally generate a large random number. The following steps are executed afterward:

1. The Outsourcer digitally signs the hash of its chosen number x and the contract hash without revealing the chosen number x before sending it to the Contractor.

2. The Contractor digitally signs the received hash by the Outsourcer along with its chosen value y, the contract hash, and a list of available Verifiers sorted by their public keys that meet the conditions specified by the Outsourcer in the contract. Along with this signed hash, the Contractor sends the value y and the list of available Verifiers for the Outsourcer to review. By signing the initially sent hash of the Outsourcer, the Contractor commits to have accepted a random number by the Outsourcer without knowing it.

3. If the list of available Verifier matches the current local list of the Outsourcer with high confidence, and the signature matches with all revealed messages, the Outsourcer contacts the Verifier at $(x * y) \mod n$ in the list of available Verifiers. The Outsourcer sends the signature of unsigned x, y, and the unsigned list to the Verifier. The Verifier might reject the contract if it's not located at the specified position in the list. If the Verifier does not respond, the whole protocol is repeated until an available Verifier agrees to the contract.

The figure below illustrates the protocol.

The result of this process is that a Verifier is selected that adheres to the defined objectives. As both the Outsourcer and the Contractor have signatures over each other's signed numbers, these can later be verified by the payment settlement entity if there is a dispute. Thus, a record is available if the Contractor and Outsourcer adhered to the protocol. This record is necessary to prevent the Outsourcer from ignoring the number of the Contractor to pick a Verifier by its choice.
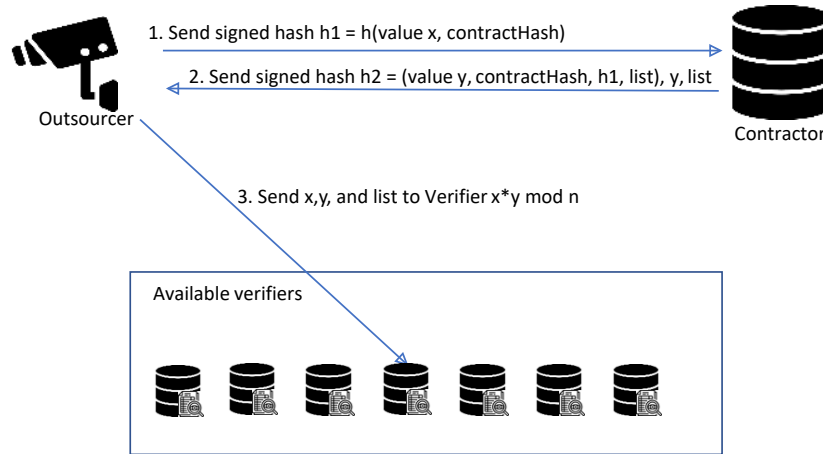
Figure 3.2.: Randomization Overview

The protocol does not add any noticeable overhead to the system as the preparation phase is only performed once per contract and is expected to take less than 5ms processing time on most machines. The results of the protocol are the following:

1. The Outsourcer and the Contractor should commit to a random Verifier

2. The Contractor does not know which Verifier was selected

3. The Verifier does not know which contractor was assigned to the job

4. Record of signed random numbers associated with the specific contract is available

3.3.2. Game-theoretic Incentives

After Randomization, both the Verifier and the Contractor have received a contract identical to a contract of the Outsourcer. Even when Verifier and Contractor do not know each other there is a theoretical risk for ad-hoc collusion. This is the case, for example, if there exists a q-algorithm that is computationally inexpensive but provides a correct result with a certain probability q. For object detection, this might be the naive response that no object was found and therefore no bounding boxes at specific co-ordinates have to be estimated and returned. If we assume there exists a most efficient q-algorithm, there is a possibility that a lazy Contractor and a lazy Verifier may act rationally by both deciding to use this most efficient, known q-algorithm without the need of ever planning to collude. As described in the literature review chapter 2, there is a solution to lazy behavior and collusion by designing incentives in a certain way. Verification schemes such as [40] have identified a relationship of incentives with adding fees cheating participants and bounties for honest participants such that being honest is a dominant strategy from a game-theoretic point of view. When designing incentives correctly, the use of a q-algorithm is not the decision with the highest expected payoffs, as the fine when detected cheating might be higher than $q * r$ (where r is

the reward when not detected cheating). Also, by adding a bounty to the participant that catches the other participant cheating, the expected payoff for honest behavior increases. The following figure illustrates both contract pairs with a bounty, reward, and fine.
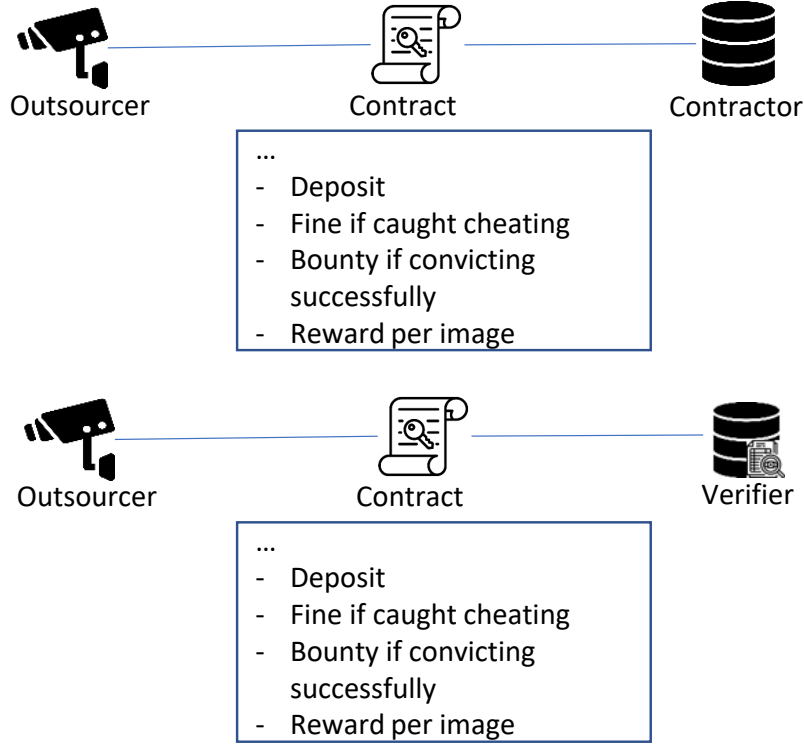
Figure 3.3.: Contracts with incentive related agreements

The payoff matrix with the use of a bounty can be seen below. If the bounty $b > cost1 + cost2$ where b is the bounty $cost1$ is the cost of performing the function and $cost2$ is the cost of performing the q-algorithm, and $(f + b) * (1 - q) - cost2 > r * q$ where f is the fine when caught cheating, and q is probability of the q-algorithm returning a correct result, ad-hoc collusion is not rational. In this case, the expected payoff when both participants act honestly is the highest. Even if not all values in the formulas are known before such as the success rate q or the exact costs, a moderate fine and bounty compared to the reward such as a 20:1 ratio are already sufficient with high probability. In the implementation, we will show how we found an upper bound for cost1 and cost2 to utilize for the calculation.

Finally, a deposit ensures in a real-world system that the fine can be paid as part of the balance is locked for spending in the meantime.

Due to the unlikeliness of ad-hoc collusion with a random Verifier, we chose to not enforce contracts adhering to described game-theoretic incentives. Nevertheless, if a payment scheme supports deposits, fines, and bounties game-theoretic incentives can be used to increase security at a tiny, non-measure-able computational overhead.

	Verifier	Diligent	Lazy
Contractor			
Diligent		r − cost1	r − cost1 + b
Lazy		r*q − (f+b)*(1-q) − cost2	r − cost2

Figure 3.4.: Payoff Matrix

3.4. Execution Phase

3.4.1. Sampling

Sampling refers to picking one random input out of a collection of inputs. In our verification scheme, the Outsourcer can send samples to the Verifier to check whether its response matches the response of the Contractor on the same input. We call this process sampling-based re-execution. Sampling-based re-execution has the advantage over complete re-execution that just with a few number of samples, a dishonest Contractor with a cheating rate of c can be detected with nearly 100% confidence. Thus, we can significantly improve the efficiency of the verification process at a neglectable security draw-down.

The following example should explain this statement. We construct a scenario where an Outsourcer sends 10000 inputs to a Contractor. If the 10000 inputs are a video stream at 60 frames per second, this contract is less than 3 minutes long.

Let's suppose an Outsourcer performs complete re-execution, meaning that it sends every input to both the Contractor and the Verifier. Assuming the Verifier is acting honestly in this scenario, a Contractor with any cheating rate c can be found with 100% confidence, but the computational overhead of the verification is at least 100% compared to not performing any verification. Thus, to achieve 100% confidence we need to check all

Let's suppose an alternative strategy, where the Outsourcer splits up the inputs in i intervals and sends only one random sample per interval to the Contractor. In this case the chance p of detecting a cheating attempt is $p = 1 - (1 - c)^i$. If we want to catch a cheating Contractor with at least 99% confidence we simply have to assume a cheating rate and solve the equation for $0.99 >= 1 - (1 - c)^i$. Let's assume a difficult case where the Contractor cheats only in 10% ($c = 0.1$) of all cases. In this case by solving the equation our interval size becomes $i >= 43.8$. This means, independent of the total number of inputs, the Outsourcer will catch a cheating Contractor even in a difficult and unlikely case with more than 99% confidence with only 44 samples. In our short contract with 10000 inputs, this results in a verification overhead of 0.44%. In case we send 100,000 inputs (less than 28 minutes of video at 60fps) this overhead already decreases to 0.044%. Thus, we can conclude that even for short contracts a sampling rate of less than 1% is sufficient to detect a rational, dishonest Contractor with nearly 100% confidence.

We expect that in real-world scenarios rational, dishonest Contractors try to cheat in more than 10% of the cases as with a cheating rate of 0.1 they would still need to perform at least 90% of required computational effort but risk being detected, losing their reward and perhaps even receiving a fine that exceeds their reward expectation. Furthermore, we expect average

contracts for most use cases to last more than 3 minutes as aborting a contract according to custom is only useful if (1) a Contractor with a better QoS, or cheaper rates becomes available, (2) the application is stopped, or (3) the Outsourcer moves out of range of the Contractor. Therefore, we consider our example as a stress test and suppose even less computational overhead in real-world scenarios.

Nevertheless, we recommend instead of setting a fixed number of samples, to instead set a sampling rate that checks samples in even intervals. After all, it is hard to predict how long a contract will last, as all parties can cancel the contract according to custom at any time without specifying a reason. The situation should be avoided that (1) either too few samples were sent because the contract was shorter than expected, or (2) that samples were sent too early because the contract was longer than expected. Case 2 either leads to an uneven distribution of samples or demands the Outsourcer to send additional samples whenever the contracts exceeds the expected length. For most use cases, we recommend setting a sampling rate between 0.01% and 1% and from the beginning on always storing 50 inputs locally that can be still verified after a contract ends after a surprisingly short time. This strategy ensures an even distribution of samples, as well as a high-security guarantee even with short contracts. While the resulting computational overhead is already very low further optimization could include lowering the sampling rate, the longer the contract goes on, or adjusting the sampling rate if a global reputation system is present.

3.4.2. Signatures

Since in our verification scheme participants communicate with non-monitored peer-to-peer communication we need a way to securely record payment promises and dishonest behavior. Otherwise, an Outsourcer could claim to never have received any responses from a Contractor or a Verifier, and could accuse another party of faking records that tell otherwise. Likewise Contractor and Verifier could deny that a dishonest response originated from them and could claim to have processed more responses than they did. For a payment settlement entity, it is impossible to solve a dispute and hold entities accountable without tamper-proof records.

Digital signatures can solve this problem as each participant can have a public key registered at the payment settlement entity, and only knows its private key that can be used to sign messages. As forging a digital signature of another entity is infeasible without knowing the associated private key, any message that is signed with the public key of a participant can be used as a verifiable record of communication. Thus, if the Contractor or the Verifier commit to a response by generating a digital signature over it and sending it along with the response, the Outsourcer can prove to any third party that the response indeed originated from that participant. Likewise, if the Outsourcer signs how many responses it has received from a Contractor, or a Verifier along with other contract-related information, this record can be used to redeem payment from a payment settlement entity on behalf of the Outsourcer's account.

Figure 3.5 shows a high-level overview of sampling in combination with digital signatures. $x = r$ refers to the property of our scheme that an input x is only sent to the Verifier if its index is chosen as a random number within the current interval.

When an Outsourcer sends an input with our scheme it is required to always send a
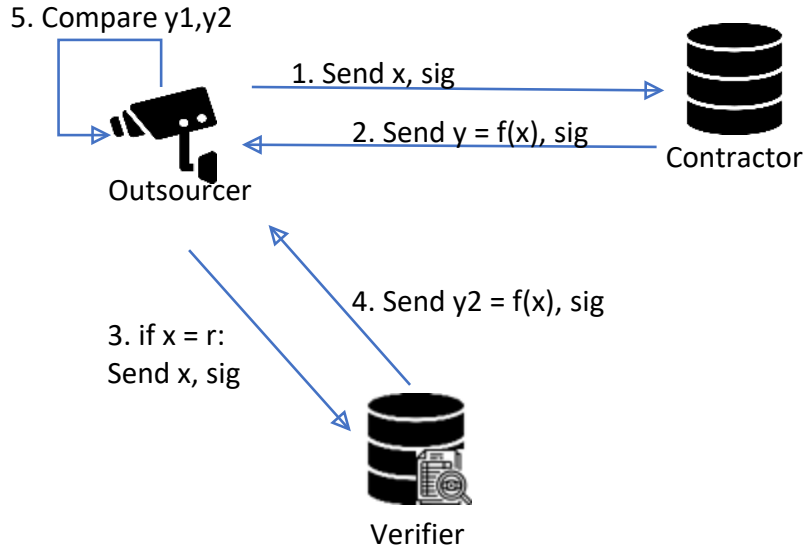
Figure 3.5.: Execution phase with signatures

signature with it, which is signed over the current input index, the contract hash, and the input itself. This ensures that each signature can be traced back to exactly one unique input in a contract context. Even if an identical input is sent later in the contract or in another contract, signatures do not match as the index number or the contract hash would be different. Whenever receiving an input from the Outsourcer, they simply have to check whether the input and input index send along the signature, and the contract hash only know to two participants (which they own a copy of) can be used to verify the signature. If the signature is not valid, they abort the contract according to a QoS violations, leave a bad review to the Outsourcer and blacklist it. The contract cannot be aborted due to dishonest behavior as it cannot be proven where a fake signature originated from.

When the Contractor or the Verifier sends a response to the Outsourcer, they also send the input index that the response is associated with and the input, along with a digital signature forged over contract hash, input index, and the input itself. The Outsourcer verifies each signature and aborts the contract according to a QoS violation if the signature does not match the designated values and may also leave a bad review to the participant and blacklist it. If a signed response returned by the Verifier and the Contractor over the same input index is found to be unequal, the Outsourcer now has a record of both participants that committed to their responses. The Outsourcer always assumes that the Contractor is dishonest in this case and can abort the contract due to dishonest behavior, sending both unequal responses and their signatures as proof to the payment settlement entity. The Contractor has the chance to prove that it was instead the Verifier that sent the false response, which is explained in detail in the section on Contestation.

There also needs to be a way for the Outsourcer, and the Verifier to redeem payment. As we consider scenarios with frame loss, weak participants, and weak network connections,

we do not assume that just because a message was sent by a participant it was also received and processed by the other participant. Thus, the Outsourcer acknowledges each received by sending a counter of acknowledged responses from that participant along with each input and integrates the counter into its signature. Verifier and Contractor specify in their internal parameters which loss rate they tolerate before aborting the contract due to Qos violations. All parameters that specify a maximum loss rate or number of consecutively unacknowledged outputs mustn't be revealed to the Contractor. Otherwise, the Contractor may exploit this parameter to always acknowledge only enough inputs so that the other participant continues the Contract. The sent counter of acknowledged outputs is a 32 bit (4 bytes) integer that causes neglectable network overhead and comes with another advantage: When a contract is aborted no matter if according to customs or other reasons, the last input received from the Outsourcer contains a signed number of all acknowledged responses throughout the contract. Thus, Contractor and Verifier only need to store the latest input in their memory and submit this single input after the contract ended to the payment settlement entity to redeem their payment. With this approach, we also do not need any TTP that monitors or records communication between participants. Sending counters alongside inputs and signatures is illustrated in figure 3.6.
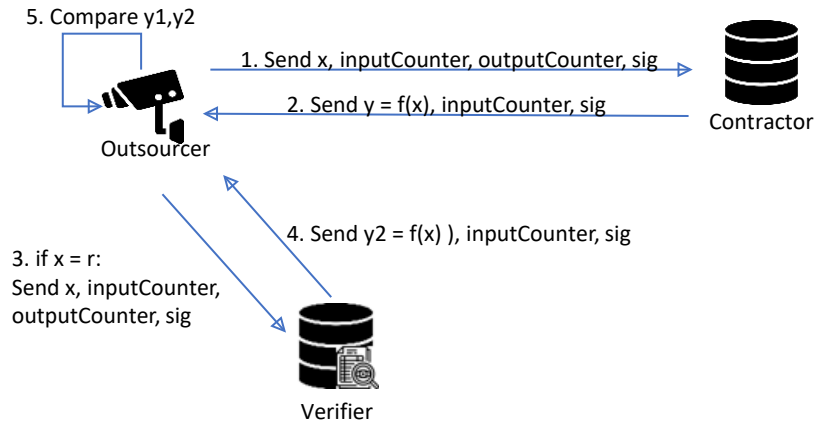
Figure 3.6.: Execution phase with signatures and acknowledged responses

The Outsourcer also sets internal parameters such as a minimum receive/process rate from Contractor and Verifier, a maximum number of consecutive inputs that remain without a response. Unlike the Contractor and the Verifier, it may decide to publish these values as a lower amount of responses received also results in fewer rewards for those participants. Nevertheless, we decided to not share any internal parameters between participants that lead to QoS violations. These, internal, disclosed parameters that can be set by each participant according to their expectation in combination with the consequences of aborting a contract due to bad QoS, make our scheme compatible with lossy networks and weak participants while giving incentives for participants to provide reliable service. These techniques are used to prevent the identified QoS violations (number 8-10 in the list of possible protocol violations2.4), timeouts, low response rate, and high response time.

By utilizing digital signatures, our verification scheme is also resistant to message tampering (number 11 in the list of possible protocol violations 2.4), and Contractor or Verifier trying to change their record or making false claims when accused of cheating (number 4 in the list of possible protocol violations 2.4). A message that is tampered with by an attacker immediately leads to the receiving participant aborting the contract as the attacker can not forge a valid signature of the tampered message.

3.4.3. Improving Efficiency of the Execution Phase

Signing a digital stream via one-time signatures

1. Sign each chunk at index i while sending the newly generated public key with chunk i-1

2. One-time signature schemes are faster than multi-use digital signature schemes in theory

3. Real-world implementations of Lamport and Winternitz one-time digital signatures schemes were found to be slower than ED25519 in the benchmarks

Inputs			outputs
W	X	Y	Q = A⊕B⊕C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Figure 3.7.: Truth table of 3 input XOR

Signing a digital stream via Merkle Trees

1. Instead of signing each response, the outsourcer only signs the root of the Merkle root in a specified interval and the associated challenge sent afterward

2. Implementation reduces total time spent on signing and verifying

3. Merkle trees are also useful when not all responses are significant for the Outsourcer but it still wants to make sure that all inputs are processed
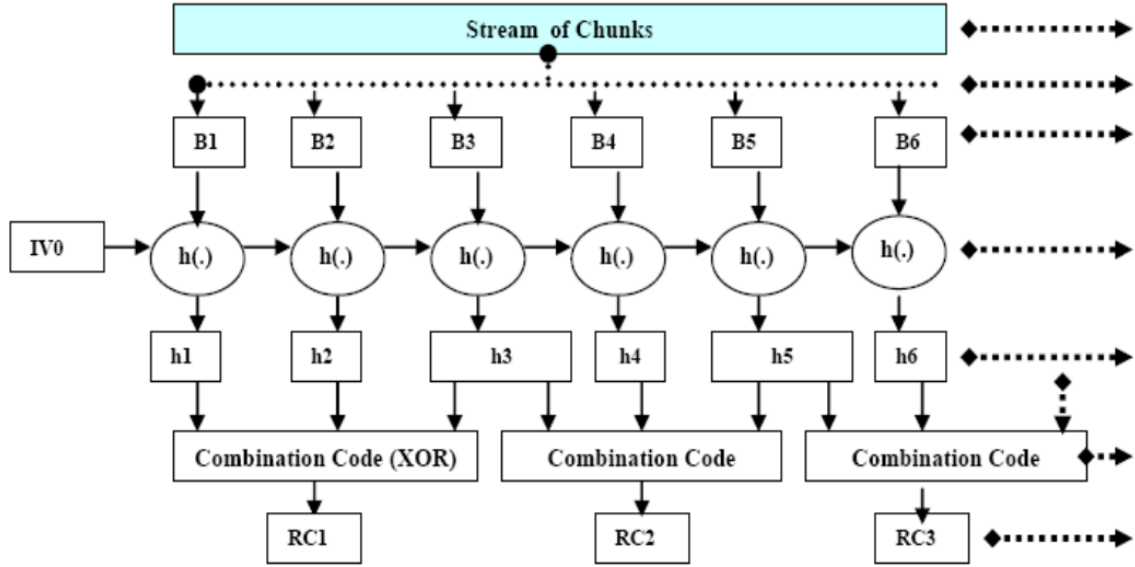
Figure 3.8.: Hash Chain of a digital stream with combination codes using XOR
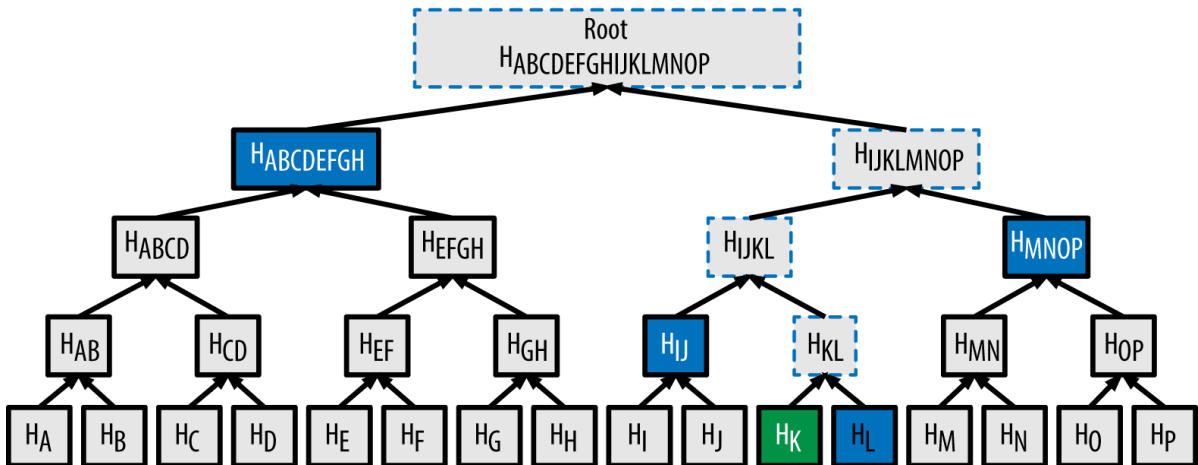
Figure 3.9.: Visualization of a Merkle Tree and a Proof of Membership challenge

3.5. Closing Phase

Aborting a contract Each party can abort a contract at any time If the aborting party sends an abort message to each counterparty the contract is aborted according to custom

Redeeming Payment At any point in time, the latest message of the Outsourcer contains a signed output counter of the counterparty This latest message can be submitted to a payment settlement entity for proof of service to redeem payment

Comment The Payment Settlement Entity does not have to be located at the edge and can handle payments with an arbitrary delay

3.5.1. Review System

3.5.2. Blacklisting

3.5.3. According to Custom

Verifier

1. Send x, inputCounter, outputCounter, contract, sig of Outsourcer

Contractor

Payment Settlement Entity

2. Transfer reward per image * outputCounter from Outsourcer account to Verifier account and Contractor account after a deadline
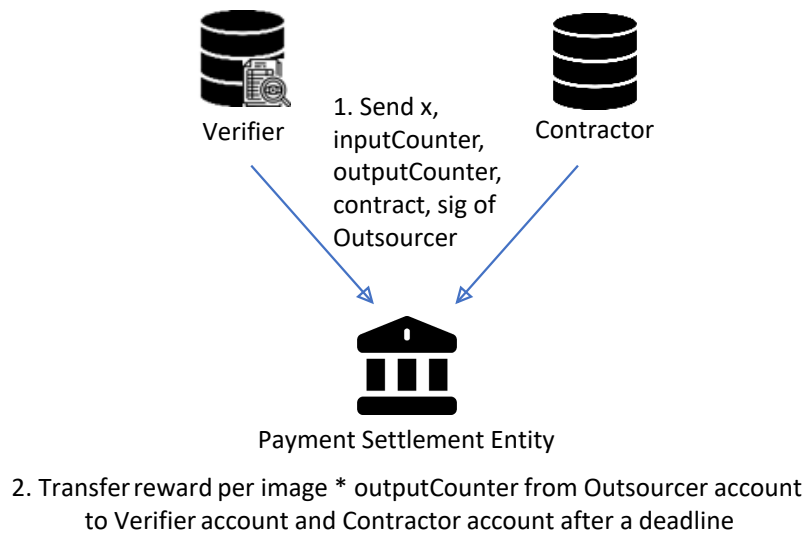
Figure 3.10.: Redeeming payment after closing a contract

3.5.4. Quality of Service Violations

Aborting a contract

1. Each party can set parameters internally that set a maximum delay or minimum response rate to specify when a contract is aborted due to QoE violations

2. A party can decide to give bad reviews and blacklist another party if QoE violations occur

Review system

1. A positive review balance of a contractor indicates honest and reliable behavior and may lead to increased reward per image and lower sampling rate

2. A positive review balance of an outsourcer may lead to a discount

3. A negative review balance can be reset to neutral by generating a new key pair but will not lead to cost savings or increased rewards

4. Payment settlement entity might also be able to maintain the review systems
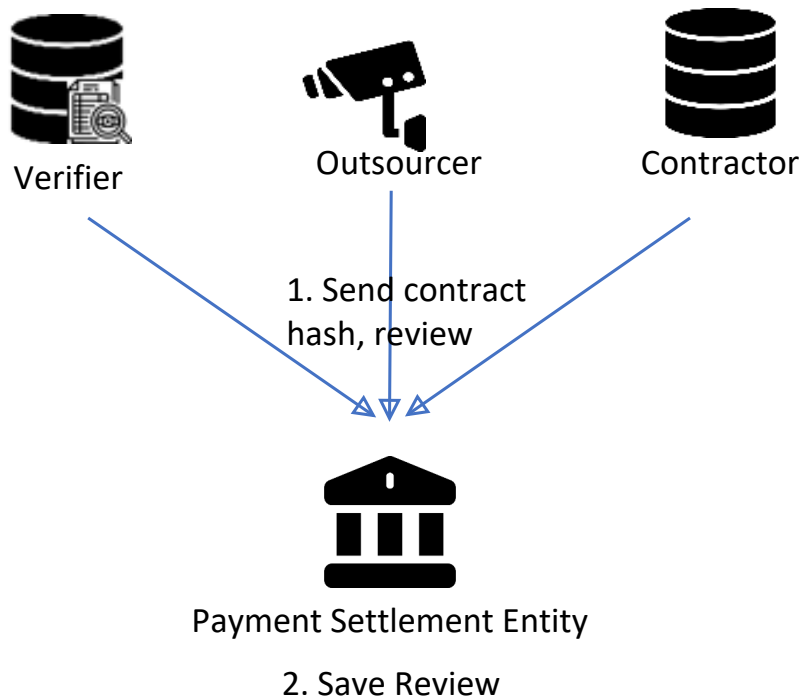
Figure 3.11.: Reviewing a participant after QoE violation

3.5.5. Dishonest Behavior

Aborting a contract

1. The Outsourcer can abort a contract due to dishonest behavior if a response sent by the Verifier and a response sent by the Contractor belonging to the same input are unequal

Reporting unequal sample

1. After contract abortion, within a deadline, the Outsourcer can report the two unequal responses to the Payment settlement entity

2. If set, the payment settlement entity issues a fine to the Outsourcer and transfers a bounty to the Verifier

3. Contractor has the right to contest within that deadline

Outsourcer

1. Send x, verifier response, outsourcer response, contract, signatures

Payment Settlement Entity

2. Reject any incoming payment redeem request by outsourcer
Issue fine and transfer bounty if specified after a deadline
Allow contestation within a deadline
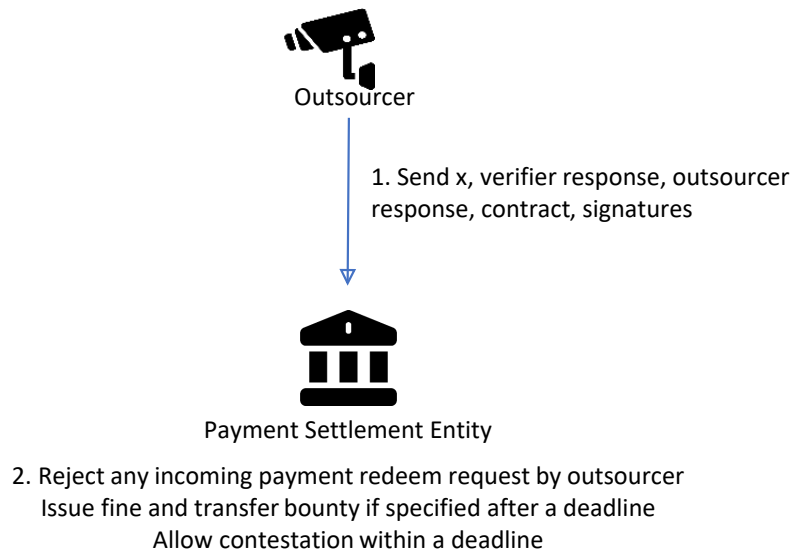
Figure 3.12.: Reporting dishonset behavior

3.5.6. Contestation

Contestation

1. Contractor can decide to send the reported sample to 2 random verifiers to prove innocence

   After successful Contestation:

2. Initial Verifier was proven to have been the dishonest party

3. Issue fine to initial verifier and bounty to contractor

   After contestation result:

4. A party can always respond to a contestation result by presenting a majority of random verifiers supporting their output

Comment

1. If > 50% of available Verifiers are honest, Contestation serves as a guarantee that the party which responded with a false response is found guilty

2. In combination with a nearly 100% detection rate of cheating using sampling, any cheating parting will be successfully found guilty

3. The combination of Contestation and Sampling make any cheating attempt irrational

Verifier R1

Verifier R2

1. Send x

2. Send y1 = f(x), sig1

3. Send y2 = f(x), sig2

Contractor

4. Send y1, y2, sig1, sig2

Payment Settlement Entity

2. If y1 and y2 match previous sent response:
Reject any incoming payment redeem request by verifier
Re-Issue fine and transfer bounty if specified after a deadline
Allow contestation within a deadline
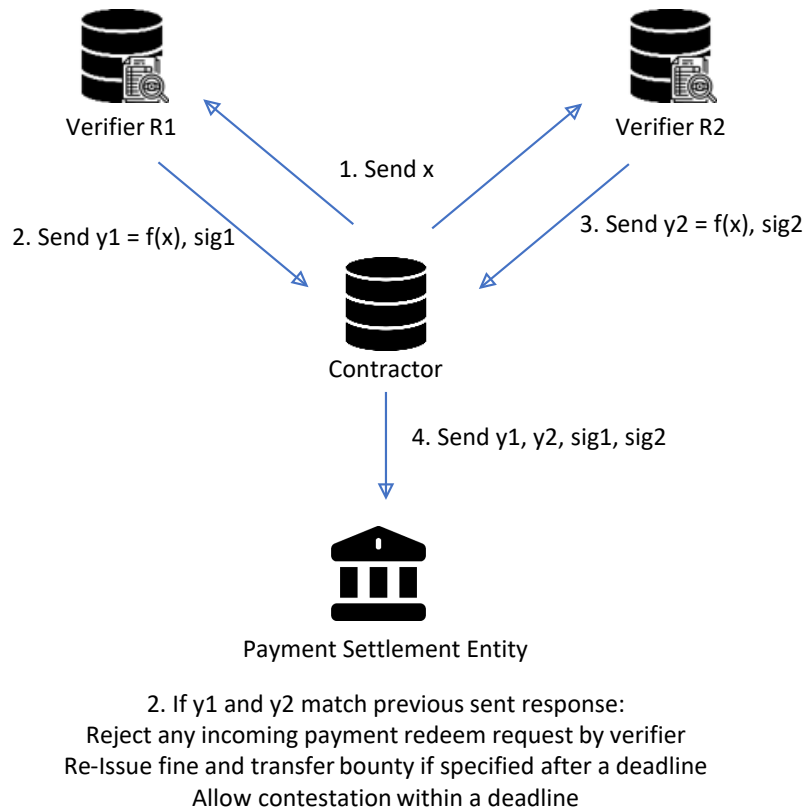
Figure 3.13.: Contestation
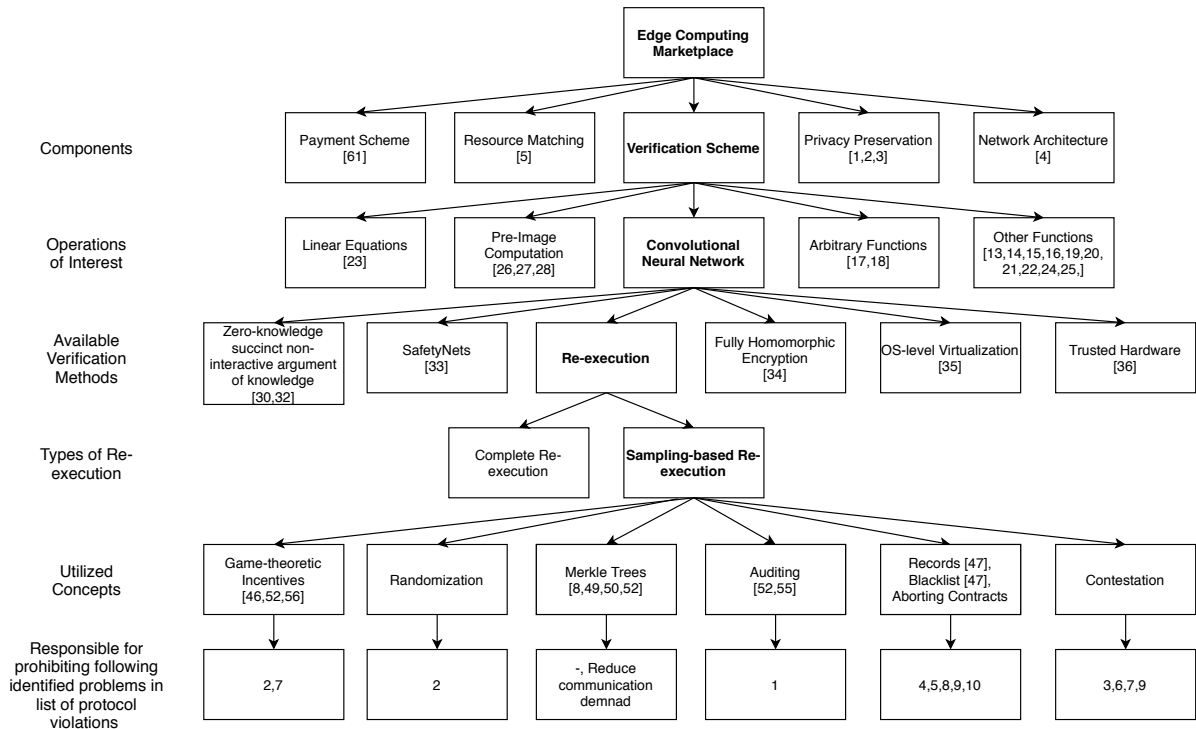
Figure 3.14.: Overview of literature review and designed scheme (outdated reference numbers)

4. Implementation

4.1. Software Architecture
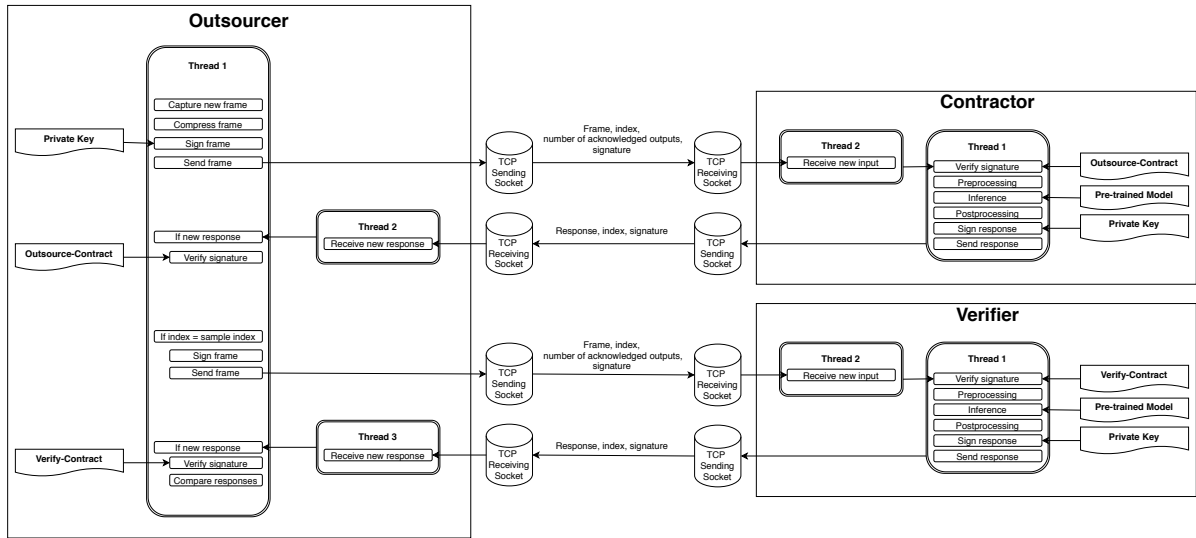
Figure 4.1.: Software Architecture without multithreading of key tasks using a regular GPU
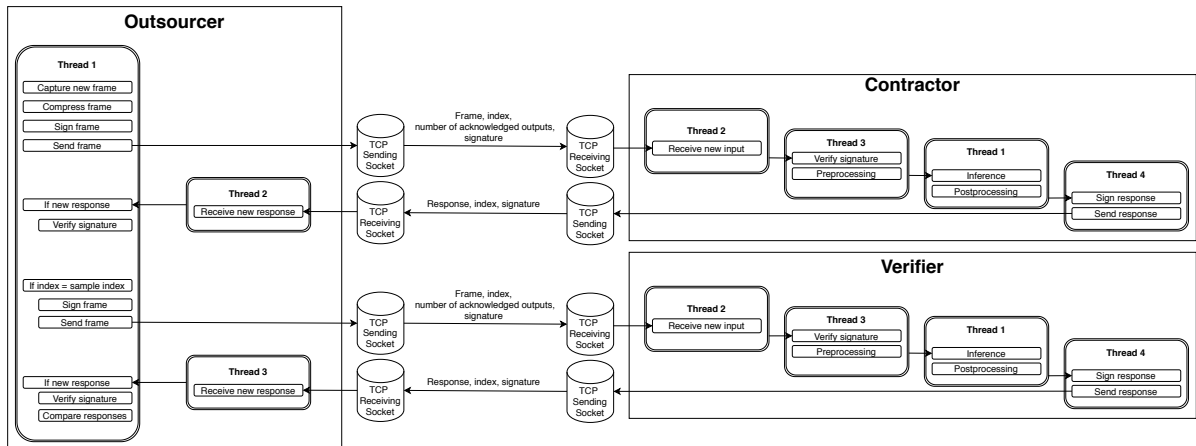
Figure 4.2.: Software Architecture with multithreading of key tasks using a regular GPU
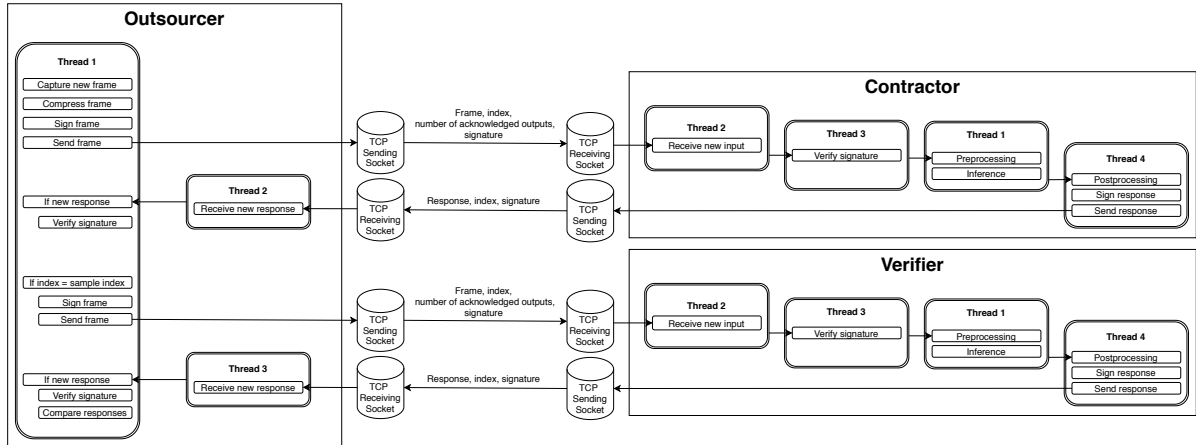
Figure 4.3.: Software Architecture with multithreading of key tasks using a Coral USB Accelerator

4.2. Improving Run-time of Implementation

4.2.1. Benchmark of Digital Signature Algorithms in Python

4.2.2. Non-blocking Message Pattern
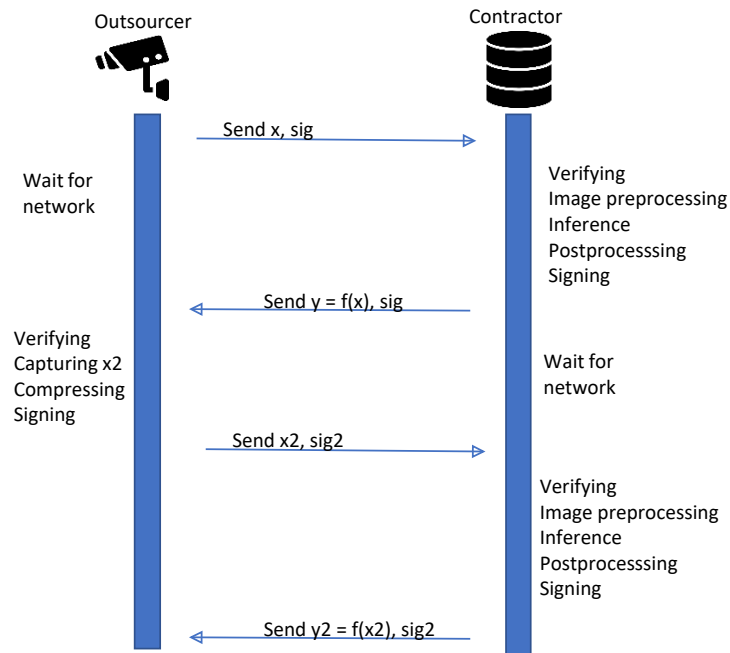
Figure 4.4.: Blocking Message pattern
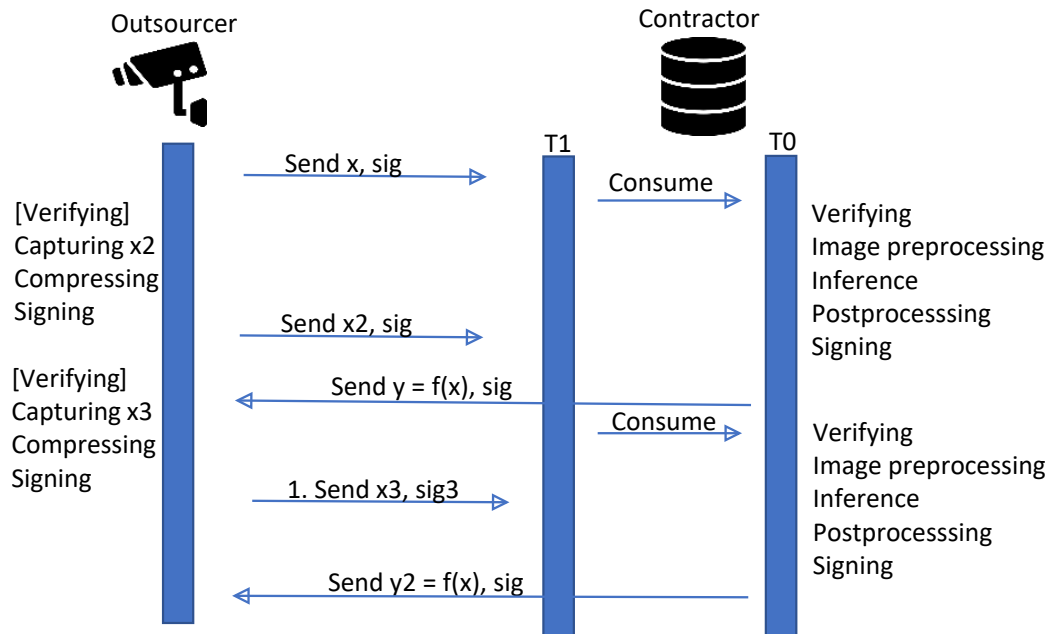
Figure 4.5.: Non-Blocking Message pattern

4.2.3. Parallel Execution

Thread 1 (ms per frame)
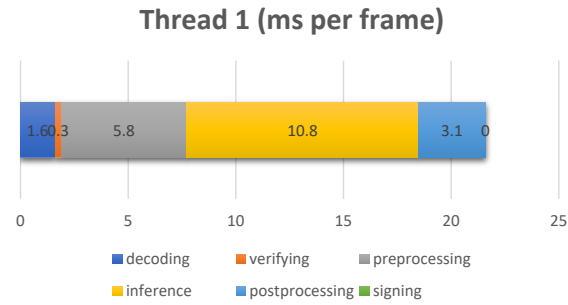
Figure 4.6.: Main thread in regular implementation

Thread 2 (ms per frame)
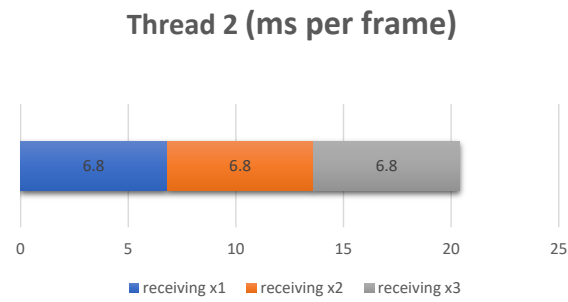
Figure 4.7.: Thread 2 in regular implementation
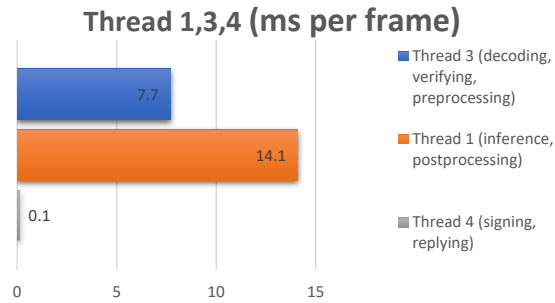
Figure 4.8.: Main threads in mutlithreading implementation
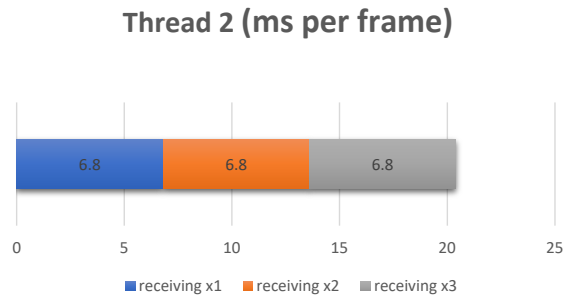
Figure 4.9.: Thread 2 in mutlithreading implementation
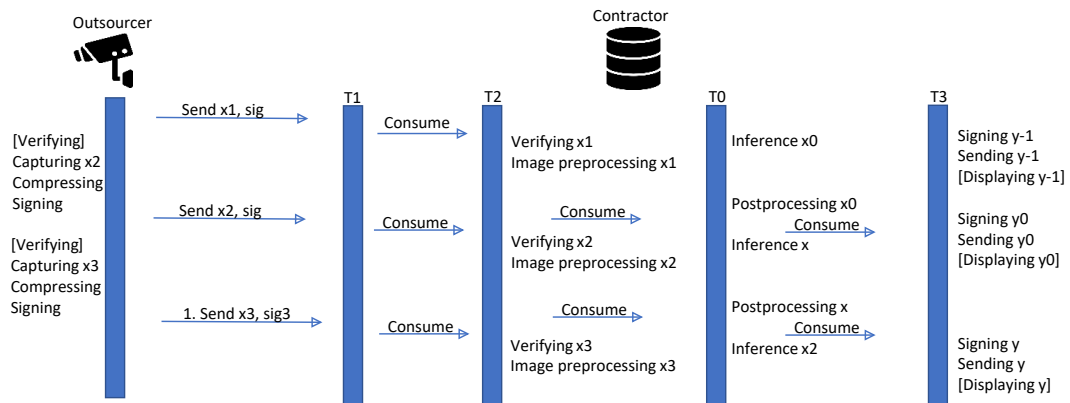
Figure 4.10.: Four threads visualized

4.3. Threat Model

4.4. Test Setup

Implementation and Testing 1
Contractor Hardware: i7-3770K + GTX 970
Verifier Hardware: i5-4300U
Model: Yolov3, Yolov4, tiny and non-tiny versions

  Implementation and Testing 2
Contractor Hardware: i7-3770K + Coral USB Accelerator
Verifier Hardware: i5-4300U + Coral USB Accelerator
Model: Mobilenet SSD V2
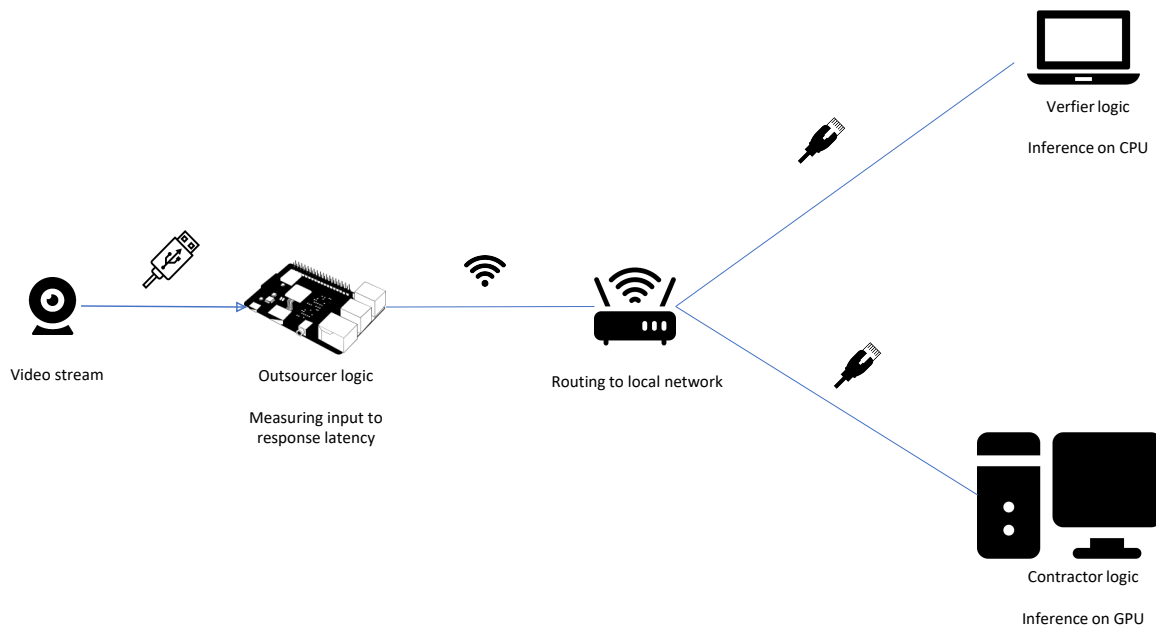
Figure 4.11.: Implementation and Testing with regular GPU and CPU
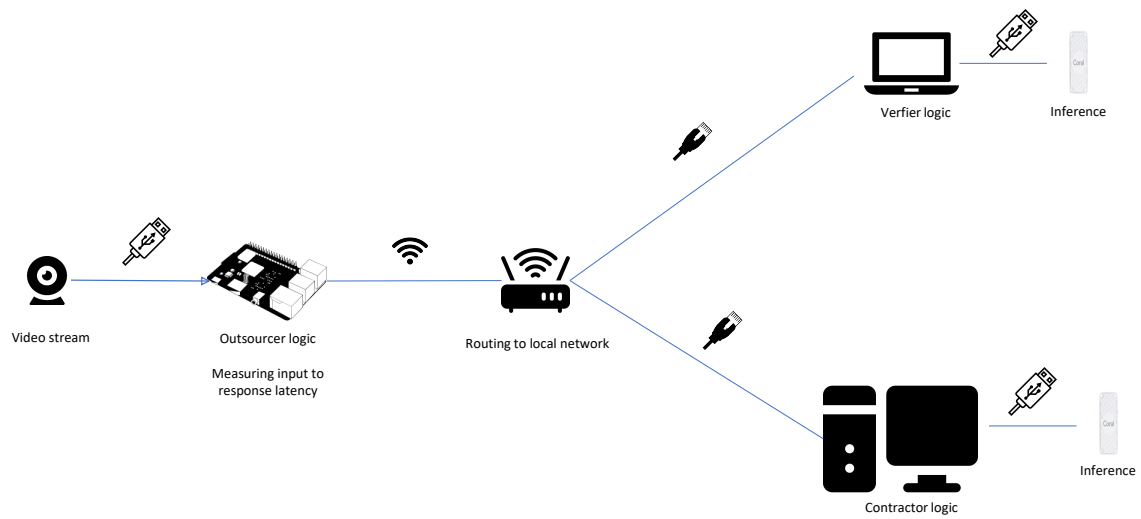
Figure 4.12.: Implementation and Testing with Edge Accelerator

5. Evaluation

5.1. Results

Execution time progress over time
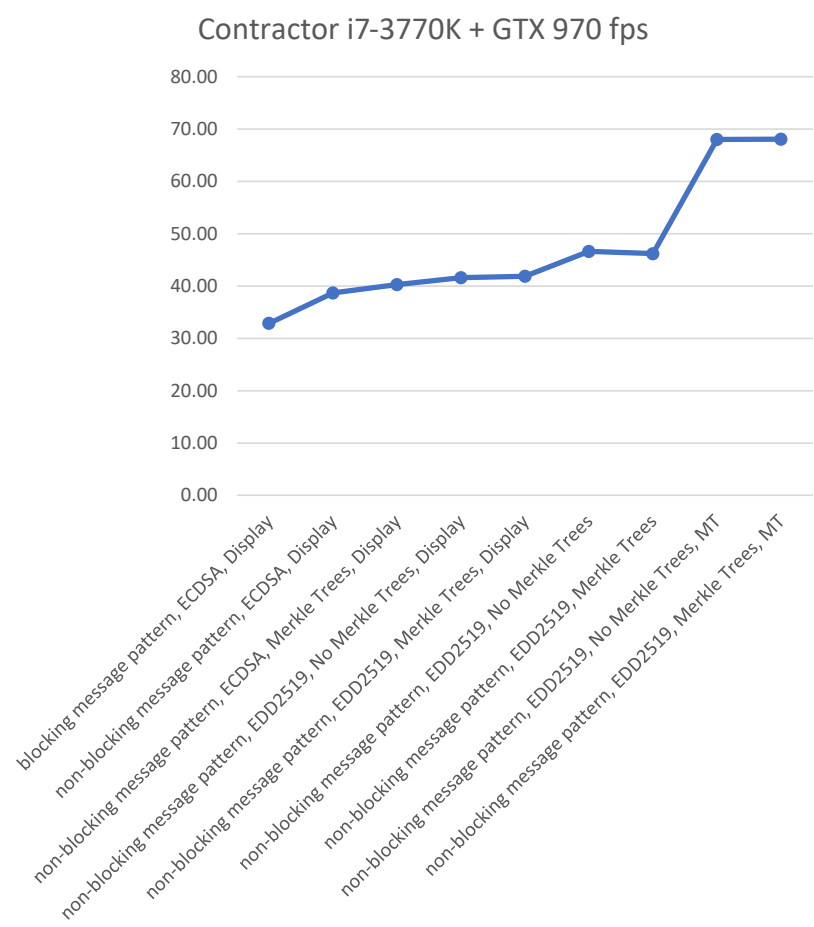
Figure 5.1.: Performance with different setups on regular GPU

Table 5.1.: Key Results

Participant	Device	CPU	GPU	Model	Frames per second	Milliseconds per frame	% spent on network wait	% spent on application processing	% spent on verification scheme	ms spent on verification scheme
Outsourcer	Raspberry Pi Model 4B			Mobilenet SSD V2 300*300	236.00	4.24	0.00	78.70	21.30	0.90
Outsourcer	Raspberry Pi Model 4B			Yolov4 tiny 416*416	146.90	6.81	0.00	85.10	14.90	1.01
Contractor	Desktop PC	Core i7 3770K	GTX 970	Yolov4 tiny 416*416	68.06	14.69	0.00	100.00	0.00	0.00
Contractor	Desktop PC	Core i7 3770K	Coral USB Accelerator	Mobilenet SSD V2 300*300	63.59	15.73	0.00	100.00	0.00	0.00
Contractor	Notebook	Core i5 4300U	Coral USB Accelerator	Mobilenet SSD V2 300*300	49.30	20.28	0.00	100.00	0.00	
Verifier	Notebook	Core i5 4300U	Coral USB Accelerator	Mobilenet SSD V2 300*300	28.75	34.78	?	0.64	?	?

Contractor i7-3770K + Edge TPU fps
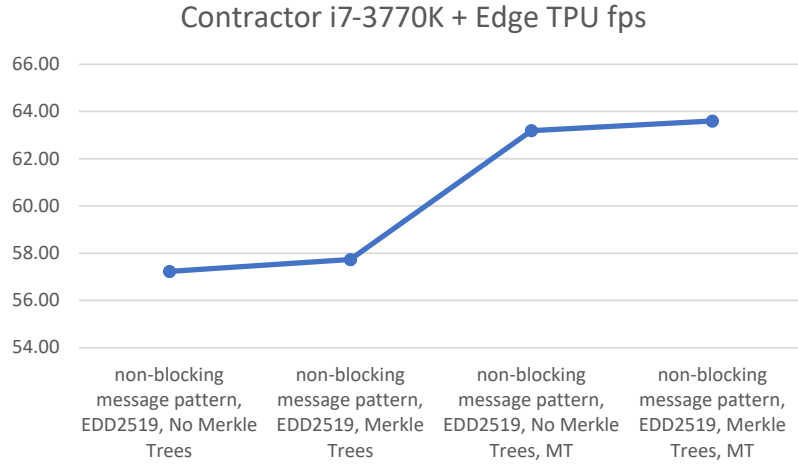
Figure 5.2.: Performance with different setups on Edge Accelerator

5.1.1. Performance

5.1.2. Network Bandwidth Overhead

The average 416x416 jpg frame was 120 KB in size in our tests. Network bandwidth overhead is neglectable at a maximum of 84 bytes per frame consisting of a 512 bit (64 bytes) large signature, and at most 5 32 bit (4 bytes each) integers such as image index, acknowledged responses, and other contract-related information.

5.2. Comparison with other Verification Schemes

5.3. Security of Designed Verification Scheme

5.4. Efficiency of Designed Verification Scheme

5.5. Scalability of Designed Verification Scheme

6. Conclusion

6.1. Future Work

6.2. Summary

A. General Addenda

A.1. Detailed Addition

B. Figures

B.1. Example 1

✓

B.2. Example 2

✗

List of Figures

List of Tables

Bibliography

[1] H. Huang, X. Chen, Q. Wu, X. Huang, and J. Shen. "Bitcoin-based fair payments for outsourcing computations of fog devices". In: *Future Generation Computer Systems* 78 (Dec. 2016). DOI: 10.1016/j.future.2016.12.016.

[2] R. Rahmani, Y. Li, and T. Kanter. "A Scalable Distriubuted Ledger for Internet of Things based on Edge Computing". In: *Seventh International Conference on Advances in Computing, Communication and Information Technology-CCIT 2018, Rome, Italy, 27-28 October, 2018*. Institute of Research Engineers and Doctors (IRED). 2018, pp. 41–45.

[3] A. Zavodovski, S. Bayhan, N. Mohan, P. Zhou, W. Wong, and J. Kangasharju. "DeCloud: Truthful Decentralized Double Auction for Edge Clouds". In: May 2019. DOI: 10.1109/ICDCS.2019.00212.

[4] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu. "Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues". In: *IEEE Access* 6 (2018), pp. 18209–18237.

[5] Y. Wang, Z. Tian, S. Su, Y. Sun, and C. Zhu. "Preserving Location Privacy in Mobile Edge Computing". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. May 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761370.

[6] M. Gheisari, Q.-V. Pham, M. Alazab, X. Zhang, C. Fernández-Campusano, and G. Srivastava. "ECA: An Edge Computing Architecture for Privacy-Preserving in IoT-based Smart City". In: *IEEE Access* PP (Aug. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2937177.

[7] I. Psaras. "Decentralised Edge-Computing and IoT through Distributed Trust". In: June 2018, pp. 505–507. DOI: 10.1145/3210240.3226062.

[8] Y. C. Chunming Tang. *Efficient Non-Interactive Verifiable Outsourced Computation for Arbitrary Functions*. Cryptology ePrint Archive, Report 2014/439. https://eprint.iacr.org/2014/439. 2014.

[9] C. Xiang and C. Tang. "New verifiable outsourced computation scheme for an arbitrary function". In: *International Journal of Grid and Utility Computing* 7 (Jan. 2016), p. 190. DOI: 10.1504/IJGUC.2016.080187.

[10] T. Combe, A. Martin, and R. Di Pietro. "To Docker or Not to Docker: A Security Perspective". In: *IEEE Cloud Computing* 3.5 (2016), pp. 54–62.

[11] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom. *SGAxe: How sgx fails in practice*. 2020.

[12]   S. Eisele, T. Eghtesad, N. Troutman, A. Laszka, and A. Dubey. "Mechanisms for Outsourcing Computation via a Decentralized Market". In: *arXiv preprint arXiv:2005.11429* (2020).

[13]   B. Carbunar and M. Tripunitara. "Fair Payments for Outsourced Computations". In: *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. 2010, pp. 1–9.

[14]   B. Carbunar and M. V. Tripunitara. "Payments for Outsourced Computations". In: *IEEE Transactions on Parallel and Distributed Systems* 23.2 (2012), pp. 313–320.

[15]   P. Golle and I. Mironov. "Uncheatable Distributed Computations". In: vol. 2020. Apr. 2001, pp. 425–440. DOI: `10.1007/3-540-45353-9_31`.

[16]   R. Freivalds. "Probabilistic Machines Can Use Less Running Time." In: *IFIP congress*. Vol. 839. 1977, p. 842.

[17]   M. J. Atallah and K. B. Frikken. "Securely Outsourcing Linear Algebra Computations". In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '10. Beijing, China: Association for Computing Machinery, 2010, pp. 48–59. ISBN: 9781605589367. DOI: `10.1145/1755688.1755695`. URL: `https://doi.org/10.1145/1755688.1755695`.

[18]   X. Lei, X. Liao, T. Huang, and F. H. Rabevohitra. "Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud". In: *Inf. Sci.* 280 (2014), pp. 205–217.

[19]   Z. Cao and L. Liu. "A note on "achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud"". In: (Mar. 2016).

[20]   D. Benjamin and M. J. Atallah. "Private and Cheating-Free Outsourcing of Algebraic Computations". In: *2008 Sixth Annual Conference on Privacy, Security and Trust*. 2008, pp. 240–245.

[21]   X. Lei, X. Liao, T. Huang, H. Li, and C. Hu. "Outsourcing Large Matrix Inversion Computation to A Public Cloud". In: *IEEE TRANSACTIONS ON CLOUD COMPUTING* 1 (July 2013), pp. 78–87. DOI: `10.1109/TCC.2013.7`.

[22]   C. Wang, K. Ren, J. Wang, and Q. Wang. "Harnessing the Cloud for Securely Outsourcing Large-Scale Systems of Linear Equations". In: *IEEE Transactions on Parallel and Distributed Systems* 24.6 (2013), pp. 1172–1181.

[23]   W. Song, B. Wang, Q. Wang, C. Shi, W. Lou, and Z. Peng. "Publicly Verifiable Computation of Polynomials Over Outsourced Data With Multiple Sources". In: *IEEE Transactions on Information Forensics and Security* 12.10 (2017), pp. 2334–2347.

[24]   X. Wang, K.-K. R. Choo, J. Weng, and J. Ma. "Comments on "Publicly Verifiable Computation of Polynomials Over Outsourced Data With Multiple Sources"". In: *IEEE Transactions on Information Forensics and Security* PP (Aug. 2019), pp. 1–1. DOI: `10.1109/TIFS.2019.2936971`.

[25] J. Meena, S. Tiwari, and M. Vardhan. "Privacy preserving, verifiable and efficient outsourcing algorithm for regression analysis to a malicious cloud". In: *Journal of Intelligent & Fuzzy Systems* 32 (Apr. 2017), pp. 3413–3427. DOI: 10.3233/JIFS-169281.

[26] H. Chabanne, J. Keuffer, and R. Molva. "Embedded Proofs for Verifiable Neural Networks". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1038.

[27] S. Lee, H. Ko, J. Kim, and H. Oh. "vCNN: Verifiable Convolutional Neural Network". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 584.

[28] J. Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: May 2016, pp. 305–326. ISBN: 978-3-662-49895-8. DOI: 10.1007/978-3-662-49896-5_11.

[29] X. Chen, J. Ji, L. Yu, C. Luo, and P. Li. "SecureNets: Secure Inference of Deep Neural Networks on an Untrusted Cloud". In: *ACML*. 2018.

[30] Z. Ghodsi, T. Gu, and S. Garg. "SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4675–4684. ISBN: 9781510860964.

[31] A. A. Badawi, J. Chao, J. Lin, C. F. Mun, J. J. Sim, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar. *Towards the AlexNet Moment for Homomorphic Encryption: HCNN, theFirst Homomorphic CNN on Encrypted Data with GPUs*. 2018. arXiv: 1811.00778 [cs.CR].

[32] X. Hu and C. Tang. "Secure outsourced computation of the characteristic polynomial and eigenvalues of matrix". In: *Journal of Cloud Computing* 4 (Dec. 2015). DOI: 10.1186/s13677-015-0033-9.

[33] G. Xu, G. T. Amariucai, and Y. Guan. "Delegation of Computation with Verification Outsourcing: Curious Verifiers". In: *IEEE Transactions on Parallel and Distributed Systems* 28.3 (2017), pp. 717–730.

[34] W. Du, J. Jia, M. Mangal, and M. Murugesan. "Uncheatable grid computing". In: *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE. 2004, pp. 4–11.

[35] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen, and A. V. Vasilakos. "Security and privacy for storage and computation in cloud computing". In: *Information sciences* 258 (2014), pp. 371–386.

[36] H. Wang. "Integrity verification of cloud-hosted data analytics computations". In: *Proceedings of the 1st International Workshop on Cloud Intelligence*. 2012, pp. 1–4.

[37] M. Nabi, S. Avizheh, M. V. Kumaramangalam, and R. Safavi-Naini. "Game-Theoretic Analysis of an Incentivized Verifiable Computation System". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 50–66.

[38] R. Di Pietro, F. Lombardi, F. Martinelli, and D. Sgandurra. "Anticheetah: Trustworthy computing in an outsourced (cheating) environment". In: *Future Generation Computer Systems* 48 (2015), pp. 28–38.

[39]   X. Chen, J. Li, and W. Susilo. "Efficient fair conditional payments for outsourcing computations". In: *IEEE Transactions on Information Forensics and Security* 7.6 (2012), pp. 1687–1694.

[40]   A. Küpçü. "Incentivized outsourced computation resistant to malicious contractors". In: *IEEE Transactions on Dependable and Secure Computing* 14.6 (2015), pp. 633–649.