D4.1: Report on the Desirable Properties for Analysis

Revision: v.3.0

Work package	WP4
Task	T4.1, T4.2, T4.3 and T4.4
Due date	30/Sep/2023
Submission date	30/Sep/2023
Deliverable lead	Nobuko Yoshida (OXF)
Version	2.0
Authors	Nobuko Yoshida (OXF); António Ravara (NOVA); Sebastian Mödersheim (DTU); Ivan Prokic (UNS); Silvia Ghilezan (UNS); Ping Hou (OXF); Simona Kasterovic (UNS), João Costa Seco (NOVA), Tamara Stefanovic (UNS), Carla Ferreira (NOVA)
Reviewers	Carlos Coutinho (CMS); Nuno Preguiça (NOVA)
Abstract	This document reports on the identified desirable properties for analysis. This deliverable will outline the list of properties for which analyses will be developed in the rest of the WP; the properties will be categorised according to the task to which they will be assigned.
Keywords	interaction behaviour, verification and validation, analysis, behavioural types, security, data conversion and integrity, deployment and orchestration integration.

www.project-tardis.eu

Grant Agreement No.: 101093006
Call: HORIZON-CL4-2022-DATA-01

Topic: HORIZON-CL4-2022-DATA-01-03
Type of action: HORIZON- RIA

DISCLAIMER

Funded by
the European Union

Funded by the European Union (TARDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**to specify R, DEM, DEC, DATA, DMP, ETHICS, SECURITY, OTHER***	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✓
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

Executive Summary

The TaRDIS project's objective is to build a distributed programming toolbox that simplifies the development of decentralized applications deployed in a diverse setting. Work Package 4 (WP4) focuses on pioneering formal analyses to assess the soundness, security, and reliability of heterogeneous swarms. These analyses will be specifically tailored to the TaRDIS models, ensuring that desirable *security*, *data integrity*, *AI coordination*, and *interaction properties* are satisfied, aligning with TaRDIS use cases and requirements. To this end, in this report, we categorise the properties regarding to TaRDIS use cases that necessitate analysis and verification. Additionally, we delve into both the existing and advanced verification techniques that will be employed to validate these properties.
Our key contributions for this report encompass various aspects:

Firstly, we identify the challenges arising from intelligent swarms as well as use cases related to verification and analysis.

Secondly, we categorise the properties that will undergo in-depth analysis in the upcoming WP4 deliverables, organising them based on the specific tasks to which they are assigned.

Thirdly, we classify the existing verification techniques and discuss how TaRDIS will go beyond the state-of-the-art.

Finally, we summarise the desirable models and properties that are specifically relevant to the TaRDIS use cases.

Here are some **key highlights** we aim to showcase as the outcomes of the M9 report, which effectively tackle the challenges related to formally analysing the soundness, security, and reliability of heterogeneous swarms:

- OXF has published two papers: one that addresses the challenge to account for unreliability and failures persists for session types, and another that supports protocols allowing dynamic participant joining.
    - David Castro-Perez and Nobuko Yoshida, "Dynamically Updatable Multiparty Session Protocols: Generating Concurrent Go Code from Unbounded Protocols", in ECOOP 2023. Click or tap here to enter text.
    - Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou, "Designing Asynchronous Multiparty Protocols with Crash-Stop Failures", in ECOOP 2023. (Barwell et al., 2023)
- NOVA has published five papers, two dealing with replicated data consistency levels, two dealing with concurrency control and safe data use, and one dealing with data confidentiality.
    - Kevin De Porre, Carla Ferreira, and Elisa Gonzalez Boix, "VeriFx: Correct Replicated Data Types for the Masses", in ECOOP 2023. (De Porre, 2023)
    - Marco Giunti, Hervé Paulino, and António Ravara, "Anticipation of Method Execution in Mixed Consistency Systems", in SAC 2023. (Giunti et al., 2023)
    - Hervé Paulino, Ana Almeida Matos, Jan Cederquist, Marco Giunti, João Matos, and António Ravara, "AtomiS: Data-Centric Synchronization Made Practical", OOPSLA 2023. (Paulino, 2023)
    - Pedro Rocha and Luís Caires, "Safe Session-Based Concurrency with Shared Linear State", ESOP 2023. (Rocha & Caires, 2023)
    - Eduardo Geraldo, João Costa Seco, and Thomas Hildebrandt. Data-Dependent Confidentiality in DCR Graphs. In PPDP 2023. (E. Geraldo et al., 2023)

- UNS has published two papers: one that develops in edge systems a Python Testbed for two generic Federated Learning Algorithms – a centralized and a decentralized, and the follow-up that formally specifies and verifies these two algorithms using the Communicating Sequential Processes calculus (CSP) and the Process Analysis Toolkit (PAT) model checker.
    - Ivan Prokić, Silvia Ghilezan, Simona Kašterović, Miroslav Popovic, Marko Popovic, Ivan Kaštelan, "Correct orchestration of Federated Learning generic algorithms: formalisation and verification in CSP". In ECBS 2023. CoRR abs/2306.14529 (2023). (Prokic et al., 2023)
    - Miroslav Popovic, Marko Popovic, Ivan Kastelan, Miodrag Djukic, Silvia Ghilezan, "A Simple Python Testbed for Federated Learning Algorithms" in ZINC 2023. (Popovic et al., 2023)

TABLE OF CONTENTS

List of Figures

Abbreviations

ML	Machine Learning
DL	Deep Learning
FL	Federated Learning
FLA(s)	Federated Learning Algorithm(s)
PTB-FLA	Python Testbed for Federated Learning Algorithms
ST	Session Types
MPST	Multiparty Session Types
CFSM	Communicating Finite State Machines
API	Application Programming Interface
AGV	Automated Guided Vehicle
ERP	Enterprise Resource Planning
MES	Manufacturing Execution System
PNT	Position, Navigation and Timing
DER	Distributed Energy Resources
SGAM	Smart-Grid Architectural Model
IoT	Internet of Things
DP	Differential Privacy
CSP	Communicating Sequential Processes calculus
CSP	Communicating Sequential Processes calculus
PAT	Process Analysis Toolkit

1. INTRODUCTION

The Work Package 4 (WP4) develops novel formal analyses for determining whether a heterogeneous swarm is *sound*, *secure,* and *reliable*. Specifically, analyses will apply to the TaRDIS models to ensure that desirable *security*, *data integrity*, *AI coordination,* and *interaction properties* are satisfied, with properties chosen according to the TaRDIS use cases and requirements. WP4 will facilitate safe usage of the AI and data primitives from WP5 and WP6. The developed tooling will be incorporated into the TaRDIS APIs and IDE and AI optimisation framework.

1.1 Formal Specifications

This document reports the M6 delivery (D4.1) which focuses on formal specifications for requirements delivered in D2.1 “Report on the initial requirements analysis from co-design". In particular, it will focus on formal definitions of properties based on behavioural types, replicated data convergence and integrity requirements, security and privacy requirements, and federated learning orchestration for heterogeneous swam. The key points with respect to Tasks 4.1, 4.2, 4.3 and 4.4 are given below.

Task 4.1

Task 4.1 [Leader OXF] develops novel formal analyses for interaction behaviour. For D4.1, this task defines novel properties and techniques based on behavioural types. In particular, to determine whether systems expressed using the TaRDIS models (T3.1) satisfy desirable behavioural properties, e.g., protocol compliance, *communication safety, deadlock freedom* and *liveness*, we explain how to extend current state-of-the art techniques, e.g., syntax-based analyses and model-checking-based analyses, to support an event-based setting where system entities are heterogeneous, may dynamically join, leave, fail, and not have complete views of the system. We examine the requirement and use case analyses (D2.1) to determine the behavioural properties of communication that are critical in decentralised systems in the cloud-edge continuum. We also discuss how to *compositionally* analyse these defined formal properties.

Specifically, Task 4.1 uses the theories of:

1. *Typestates* (Strom & Yemini, 1986a), which considers that resources have internal state that determines which operations are “safe” and define the behaviour of such resources in terms of state machines, akin to protocol descriptions.
2. *Session Types* (Honda, K., Vasconcelos, V. T., & Kubo, M., 1998), namely adhering to the propositions-as-types foundation, based on Linear Logic, to describe communication patterns of systems’ components over channels.
3. *Multiparty Session Types* (Honda et al., 2008, 2016), which is a type discipline for concurrent and distributed systems, a global description of the system’s intended behaviour considering all participants’ actions.

Task 4.2

Task 4.2 [Leader NOVA] concerns the development of compositional static analysis techniques for checking data convergence and integrity in the presence of data replication. The developments in these first six months were threefold.

Firstly, assuming the nodes that replicated the data have the same consistency model (weak consistency), VeriFx – a specialised programming language for Replicated Data Types (RDTs) – is under development: programmers implement RDTs atop functional collections and express correctness properties that are verified automatically.

Secondly, assuming nodes can have different consistency models, to deal efficiently with sequences of operations on different replicas, it is useful to know which operations commute with others and thus, when can an operation not requiring synchronisation be anticipated wrt others requiring it, thus avoiding unnecessary waits.

Finally, to ensure data integrity in concurrent applications, it is crucial to guarantee access to shared resources in mutual exclusion. The standard approach, which is difficult as reasoning is cumbersome, is control centric. We develop an alternative methodology that is instead data-centric, only requiring to identify the resources to protect.

Task 4.3

Task 4.3 [Leader DTU] develops novel and connect existing verification techniques for security properties. Amongst other things, we consider privacy-type properties aimed at protecting the relationship between data, actions, and entities. Task 4.3 employs both symbolic methods, to detect vulnerabilities early in the design, and abstract interpretation techniques that can capture key properties of devices to enable security proofs. We plan to formalise the security proofs for key building blocks (that are made available in the TaRDIS API) in Isabelle/HOL to obtain a very high security guarantee through machine-checked proofs. We will employ compositional reasoning by defining assumptions and guarantees of devices in their interaction such that the composition of an entire system is secure even when some devices have vulnerabilities or cannot be trusted. The composition will define minimal guarantees that devices must satisfy (e.g., not leaking certain keys) so that recovery is still possible. We are currently investigating how to incorporate information flow analysis to analyse code under the assumption that the communication primitives ensure given confidentiality and integrity properties. The analyses developed in Task 4.3 will complement those from the other WP4 tasks, enabling reliable communication that also respects privacy protocols, that data is replicated consistently and securely, and that distributed AI primitives use data which they are entitled to access. Task 4.3 will first identify desirable security analyses, leveraging requirements and use case analyses (WP2); and second, iteratively develop compositional analyses for the identified properties, incorporating feedback from TaRDIS evaluation tasks, and ensuring integration with the unified interface and tooling developed by WP3.

Task 4.4

Task 4.4 [Leader UNS] develops an integration framework for the safe orchestration of the decentralised machine learning model (Leader UNS). Overall, the framework is facilitating the coordination of the machine learning primitives developed in WP5 over a communication system using techniques based on process algebras and behavioural types. To the best of our

knowledge, there is no previous work on the formal verification of distributed machine learning algorithms.
Thus, as a starting point Task 4.4 has made an effort to build a common language for the two research communities, specifically, Federated Learning and Behavioural Types. Towards this goal and for D4.1, in Task 4.4

- We have used process algebra of Communication Sequential Processes (CSP) (Hoare, 1985) to provide simple formal specifications of two generic federated learning algorithms (Popovic et al., 2023).
- Based on these CSP specifications we performed model-checking-based analyses using Process Analysis Toolkit (PAT) (Sun et al., 2009).
- We have proved the correctness of the two generic FL algorithms by showing their deadlock freedom and termination.

Further, for D4.1, Task 4.4 inspects the federated learning requirements of the use cases (D2.1) to determine the behavioural properties that are to be defined and verified.

1.2 Results Summary

Our key contributions are:

- We **identify the challenges of intelligent swarms and use cases** related to verification and analysis.
- We **categorise the properties** that will undergo further analysis in the subsequent deliverables of the WP4, organising them according to their respective assigned tasks.
- We **classify the existing verification techniques** and discuss h**ow the TaRDIS will advance beyond the state-of-art.**
- We s**ummarise the desirable models and properties** which are specifically related to the TaRDIS use cases.

Overall, we consolidate various requirements, properties, and advanced techniques for analysing interactions in distributed systems. These are applied to TaRDIS models to ensure the satisfaction of desirable security, data integrity, AI coordination, and interaction properties, tailored to specific TaRDIS use cases and requirements.

1.3 Deliverable Structure

We begin the report by providing a comprehensive introduction to the complexities of intelligent swarms, highlighting the associate challenges tied to each use case (Section 2). We then categorise the properties that will be analysed throughout the rest of the WP based on their respective tasks (Section 3). Subsequently, in Section 4, we classify the existing verification techniques related to these properties, delving into both the state-of-the-art and upcoming advancements in verification methodologies. Furthermore, we delineate potential avenues for future research. Finally, we present a concise list of the desirable properties unique to each use case, which necessitate verification (Section 5).

2. THE CHALLENGE OF INTELLIGENT SWARMS

2.1 General Introduction

Developing and managing distributed systems is a highly complex task requiring expertise across different domains. This complexity becomes more pronounced when considering swarm systems which are highly dynamic and require fully or partial decentralised solutions to cope with the scale and heterogeneity of devices and execution environments. Developing correct, reliable, and secure systems in such contexts requires developers to reason about aspects across multiple layers of the system and across heterogenous devices and communication mediums.

TaRDIS development environment aims to assist developers in building correct systems by taking advantage of sophisticated techniques, as described in this document, to automatically analyse the interactions between the different components of the distributed system. This aims to ensure correctness-by-design of applications considering specifications of both the application invariants and the considered execution environment.

2.2 Identified Challenges by Use Case

A comprehensive discussion of the specific challenges pertaining to different use cases can be found in Section 3 of Deliverable 2.1. In this subsection, we provide a concise overview of the challenges we have identified concerning the analysis of interactions within distributed systems for each use case.

2.2.1 Actyx

Context: Next-generation factories are built from intelligent components that collaborate autonomously to perform mission-critical tasks without central infrastructure. Implementing this dynamic machine-to-machine cooperation correctly and resiliently is not yet possible. Rigid classical automation approaches lack the flexibility and agility to efficiently express such high-level orchestration.

Challenges: Actyx has identified challenges related to the specification and verification of the general structure of communication and participant behaviour in their swarm system, the integration of these behaviours within their system using specific programming languages, and ensuring the resilience of the system.

2.2.2 EDP

Context: The multi-level smart charging concept is built to overcome the foreseen electric vehicle charging problem. The concept is divided into three core energy and network levels. These levels aim to provide a local energy balance by matching energy generation with local flexible loads of the consumption – energy consumers with controlled usage of energy.

Challenges: The challenges identified by EDP primarily pertain to the decentralization of their energy grid control system and the corresponding adjustments required for components, communication, and data configurations.

2.2.3 GMV

Context: Next-generation of swarm satellite constellations in Low Earth Orbit will be characterized by an increasing number of satellites for which Orbit Determination and Time will be more and more challenging. They will strongly take advantage of the Inter-Satellite-Link technology for both communication and navigation measurements, with the goal of achieving more autonomy respect to the ground.

Challenges: GMV's identified challenges involve decentralising a single satellite into a large constellation of satellites, specifying, and verifying the required interactions between these satellites, and achieving autonomy to reduce dependency on ground station support.

2.2.4 Telefónica

Context: Smart homes include a wide range of devices designed to work together as a swarm to make our lives more convenient and comfortable. Many devices are built to incorporate artificial intelligence algorithms to improve their functionality. However, the heterogeneity of these devices makes it difficult to share the intelligence without sharing data with each other or to a central location, raising concerns about privacy. Further, with so many devices collecting data about us, there is a risk that our personal information may be compromised. The use-case aims to develop a privacy-preserving federated learning framework that can work in a hierarchical fashion.

Challenges: The identified challenges by Telefónica address investigating a Hierarchical Federated Learning framework for mobile environments that enables cross-device and cross-app (i.e., on-device cross silo) Federated Learning, and offer an as-a-service solution to provide app developers with user-friendly tools and APIs.

3. CATEGORIES OF PROPERTIES

In this section, we categorise the properties that will undergo further analysis in the subsequent deliverables of the WP, organising them according to their respective assigned tasks.

3.1 Behavioural Properties to Discipline Interactions

The primary objective of analyses for interactions behaviour is to develop innovative techniques based on behavioural types, specifically, Typestates and (Multiparty) Session Types. These techniques are particularly aimed at determining whether systems expressed using the TaRDIS models satisfy desirable interaction behavioural properties. These properties include:

Communication Safety

The exchanged data in a well-defined sequence of interactions (referred as a communication protocol) adheres to the expected type, ensuring the absence of any type errors. For instance, consider the interactions between browser clients and a server in a web-based distributed system, communication safety ensures that all endpoints progress without type errors, conforming to a specified protocol.

Deadlock-freedom

Communications will eventually terminate. More specifically, a communication task can be completed through permitted interactive actions. For instance, consider a communication protocol, deadlock freedom ensures that the protocol will end successfully, allowing all participants to avoid getting stuck.

Termination

Communications will terminate finitely. More specifically, a communication task can be completed through a finite number of permitted interactive actions. For instance, consider a message-passing process, termination ensures that the process will reach the final state in a finite number of reduction steps.

Never-termination

Communications will persist indefinitely. More specifically, a communication task can be executed infinitely through permitted interactive actions. For instance, consider a message-passing process, never-termination indicates that the process can always infinitely reduce.

Liveness

Every event in a communication task can be completed through permitted interactive actions. For instance, consider a message-passing process, liveness guarantees that each input or output action of the process can be performed eventually.

Protocol conformance and completion

Valid sequences of calls of an API's methods can be defined with a typestate specification (henceforth called *protocol*). The code can be statically type-checked to ensure that clients following the protocol never generate run-time errors like null-pointer exceptions (a safety property known as *protocol conformance*). Moreover, in the absence of divergent computation, it is also possible to ensure that such client code complete the API's protocol (a weak liveness property known as *protocol completion*). In the context of a communication protocol, local conformance of all participants ensures the network's overall conformance to the initial protocol. Likewise, when considering a message-passing process, protocol conformance guarantees that the process behaves conforming to its declared types.

3.2 Properties for Data Management and Replication

Heterogenous swarms require the development of decentralized data management techniques that rely on data replication to provide high availability and low latency. These techniques resort to weak consistency solutions enhanced with support for strong consistency when required by applications. When adopting weak consistency models, a replica executes locally an operation requested by a client without any coordination with other replicas and immediately returns to the client. The operation is later propagated in the background, leading to different execution orders at different replicas. To guarantee that replicas' state converge it is necessary to address the conflicting concurrent updates made by clients at different replicas. Replicated Data Types (RDTs) that resemble sequential data types (e.g., counters, sets, maps), guarantee convergence-by-design by providing efficient and deterministic data reconciliation solutions. These data types will serve as basic building blocks for developing heterogenous swarm applications. However, although individual RDTs guaranteed convergence, when combining RDTs that might not the case.

Even though RDTs guarantee state convergence, most do not guarantee well-formedness properties of application data, commonly called integrity invariants (e.g., the account balance must be nonnegative). In general, these data properties are global properties that cannot be ensured under weak consistency and require coordination between replicas. To address this tension between consistency and availability, TaRDIS will provide support for mixed consistency models that may execute some operations under weak consistency and others under strong consistency. For instance, a replicated counter that can be both incremented and decremented ensures convergence as both operations commute. However, if we consider the integrity invariant that the counter must be nonnegative, then this invariant cannot be guaranteed without some coordination, but strong consistency can be avoided. The insight is that increments can be executed under eventual consistency, while decrements require strong consistency.

Lastly, dynamic partial replication will also be needed. There are multiple reasons for having partial replication. For instance, to keep private data in the clients' devices, storage capacity limitations of devices, among others.

The properties to be studied here are concerned with the following two properties.

State convergence

This property expresses that replicas eventually converge to the same state. As discussed above, in weak consistency models each replica might execute operations in different orders. Therefore, it is crucial to ensure that, independently of the execution order, two replicas that have received the same operations do indeed converge to the same state.

Data integrity preservation

This property is concerned with guaranteeing data integrity invariants of the application data. These include referential integrity, uniqueness constraints, numeric constraints, among others.

Beyond verifying these properties for heterogenous swarm systems at large, the goal is to provide static analysis tools that help developers determine the necessary mechanisms to ensure these two properties. For instance, given the application code assuming a sequential setting, determine where RDTs should be used, and where stronger consistency is needed.

3.3 Properties for Security

For verifying the security and privacy properties of the system, we are going to employ two basic approaches: Firstly, the verification of communication protocols that use cryptographic means to protect communication from leaking information, tampering with information and unauthorised access. Secondly, we will use information flow control techniques applied to event-based languages to analyse systems for illegal flows that are introduced by programming mistakes. This aims to prevent classified information from being “leaked” into public places and to prevent untrusted information from “leaking” into a trusted information base.

Transmission Security Properties

Here are the classical properties of security protocols: authentication and confidentiality for the data transmitted. This includes that the parties agree on each other’s name and the content of the transmission, that the transmission is not a replay, and that the data is kept confidential. Such properties can be specified by marking which data must be kept confidential between whom and who is expected to be authenticated on which data. Special cases are injective vs. non-injective agreement, unicast vs. multicast, and unauthenticated endpoints.

Information Flow Properties

In the analysis for information flow control of the code, we have the counterpart of the transmission properties, namely lattices of labels for confidentiality and integrity levels, so that information of a certain confidentiality/integrity level cannot be used in places of lower confidentiality level nor of higher integrity level, respectively. The specification is in the form of lattices of security levels, which can be linked to the roles of individuals in the system, owners, readers, or writers of data. We follow and will push the state-of-the-art on the use of dependent security levels to make the analysis more flexible and usable in real cases.

Compositional Specification

More generally, an API interface between protocols (with shared databases) can be specified, or an interface between an application and transmission protocols, specifying which interaction the components have on their interface, which modifications they can make to the shared databases, and when they can declassify shared secrets. This then requires each component to perform only actions allowed by their interface (Hess et al., 2023), under the assumption that all other components do as well.

Privacy-type Properties

These are more advanced properties for communication protocols, such as unlinkability, i.e., a third person cannot see if two actions were performed by the same device/participant or a new one. Such a property can be specified via privacy variables (e.g., agent names) and releasing formulas about them (e.g., the owner of a device may know which actions are done by their device, and which are not).

Cryptographic Compliance

When a security protocol is specified as a multi-party session type, we want to infer that for a given cryptographic knowledge of each role, each role can perform all their steps. Moreover, we can automatically derive all cryptographic checks that the role can (and in fact must) perform at each step of the protocol; this in fact ensures that the role is executable and unambiguous.

Accountability

Whenever there are no cryptographic means to enforce that users behave according to protocol, we want to ensure that they are accountable for their actions, i.e., when violating the rules (that can include general laws, but also contractual obligations and protocol descriptions) they run the risk of being detected and reprehended. Accountability involves a specification of three things: the rules that limit what actions participants may and may not do; what can possibly be detected; and a "judicial" process, detailing how to derive in case of a detection of misbehaviour. To verify this property is that the judicial process will never "convict" an innocent participant, it will convict at least somebody, and for certain misbehaving actions, there is a positive risk to get convicted. This allows then a verification of the system under the assumption that that said those actions will not occur.

Resilience and Recoverability

These are more advanced properties of security protocols that we describe in more detail in Section 4.2 (advanced beyond the state-of-the-art) and where it is at this point not entirely clear how to specify them within TaRDIS. We will therefore leave this for future deliverables.

3.4 Properties for Decentralised Machine Learning Models

Federated Learning Models (FL) are the best suited **Machine Learning Models** for heterogenous swarms (McMahan, 2019). The communication in FL can be organised as a centralised organisational structure, decentralised organisational structure, or their combination. Each of these communicational structures can have different topologies which have to ensure that the necessary information is propagated.

One of the main goals is to identify the communication topologies of the use cases, distinguishing e.g., between static hierarchical scenarios, dynamic peer-to-peer scenarios, and combinations of the two. The objective is to ensure that the TaRDIS models support the most dynamic scenarios, and address the more static scenarios as special cases, without introducing excessive burden or difficulties for the programmers.

Swarm topology sufficiency: The topology of the system must be dense enough to ensure that the necessary information is propagated. The communication can be organised in two ways: centralised organisational structure and decentralised organisational structure. In both organisational structures different topologies can be implemented such as: star topology, tree topology, full mesh topology, partial mesh topology and ring topology. In **star topology**, all nodes (devices) are connected to a central node (device). A variant of star topology is **tree topology.** This topology has a hierarchical flow of data. In a **full mesh topology**, each node is connected to every other node in the network. This topology ensures the highest level of redundancy, since there is always an alternative path for data to reach its destination if one path fails. However, this topology is expensive and difficult to manage in large networks. In order to reduce the cost and complexity of the network a **partial mesh topology** can be used. A partial mesh topology is a compromise between the full mesh topology and other topologies such as star topology. In this topology, some nodes are connected to all other nodes, while others are connected to few nodes. Another type of network topology is **ring topology**. In this topology, nodes are connected in a circular manner, forming a closed loop. Data travels around the ring in one direction, passing through each device until it reaches its destination.

These techniques are particularly aimed at determining whether systems expressed using the TaRDIS models satisfy desirable FL properties, which include:

FL Roles of agents

Some of the agents involved in the FL algorithm can have limited capabilities. The FL Roles of agents property ensure clients receive only the data they can process. Assigning roles to agents can ensure that “small” clients cannot receive “large” data. Moreover, for “low power” clients it ensures that the data and frequency can be limited, whereas the “higher power” clients can function in full capacity. This contributes to the efficiency of the model and green energy.

FL Data privacy

Ensure statically that only the model parameters can be sent by the clients and servers, and not the actual data that always should remain private to the clients and servers. This contributes to privacy protection in the model.

FL Message delivery

In case of asynchronous communications, agents acting like servers should have large enough buffers to support receiving messages from all clients preventing system congestion with received messages. This contributes to the liveness of the model.

FL Clients equality

Clients participating in one round of FL algorithm should equally contribute to the algorithm - behaviours in which a single client sends multiple messages within a single round should be avoided. This would prevent cases in which a single client is making unwanted large influence on the FL algorithm. This contributes to the fairness of the model and uniform participation of the clients.

4. VERIFICATION TECHNIQUES

In this section, we categorise the existing verification techniques relevant to the identified desirable properties, exploring both the current state-of-the-art and forthcoming advancements in verification approaches. Additionally, we outline potential directions for future work.

4.1 CATEGORIES OF VERIFICATION TECHNIQUES

4.1.1 Behavioural Types for Interaction Analysis

Significant research effort has been devoted to ensuring the safety and reliability of communicating systems. A key approach is that of behavioural types (Hüttel et al., 2016) (BTs), which specify the intended interaction patterns of systems, such that well-typed systems adhere to the prescribed interactions. BTs can be incorporated into existing languages (Ancona et al., 2016) and describe both internal and external system behaviour. The main approaches for the former are typestates (Strom & Yemini, 1986b); for the latter, contracts, and session types (Honda et al., 1998).

Session types provide a lightweight, type system–based approach to message-passing concurrency. This type discipline is further advanced by Multiparty Session Types (MPST), which enable the specification and verification of communication protocols among multiple message-passing processes in concurrent and distributed systems. MPST ensure that protocols are designed to guarantee desirable behavioural properties, i.e., communication safety, deadlock-freedom, and liveness (Scalas & Yoshida, 2019). By adhering to a specified MPST protocol, participants can communicate reliably and efficiently. From a practical perspective, MPST have been implemented in various programming languages, e.g. Go (Castro-Perez et al., 2019), Java (Hu & Yoshida, 2016), Rust (Cutner et al., 2022), Python (Demangeon et al., 2015), Scala (Cledou et al., 2022), Typescript (Miu et al., 2021), and OCaml (Yoshida et al., 2021), enabling their applications and providing safety guarantees in real-world programs.

Based on behavioural type system methodologies, the type-level behavioural properties that align with the scope of the TaRDIS APIs are verified for correctness utilising exhaustive static reasoning methods, such as *static type checking* and *model checking*, whenever feasible. These approaches enable us to provide comprehensive and concise feedback to programmers regarding any identified problems.

- Type Checking API's Code against Protocol Specifications:
    - Incorporating the protocol as its behavioural specification within the API code allows to check/enforce that client code interacting with the API correctly follows the protocol.
    - Moreover, the protocol can also be used to monitor requests arriving at the API, only allowing, for each client, the sequences of requests prescribed by the protocol.
- Model Checking Type-Level Behavioural Properties:
    - Express the type-level behavioural properties, such as communication safety, deadlock-freedom, termination, never-termination, and liveness, as modal μ-calculus formulas (Groote & Mousavi, 2014); and

  - Use model checkers, such as mCRL2 (Bunte Olavand Groote, 2019) to check the correctness of these behavioural properties.

4.1.2 Verification Techniques for Distributed Data Management

- Verification and Synthesis of RDTs:
  - The verification and synthesis of RDTs rely on static analyses done at compile time. In the operation-based consistency approach, the analyses evaluate a high-level specification of the application, more specifically, each operation pre and post-conditions, plus the data invariants to be guaranteed. Several works (Gotsman et al., 2016; Houshmand, 2019; Li et al., 2020) rely on this high-level specification to determine the safety of an RDT, however these works differ on the assumptions made. While some assume causal consistency (Gotsman et al., 2016), others rely only on eventual consistency (Houshmand, 2019; Li et al., 2020). According to the CISE (Gotsman et al., 2016), a logic for reasoning about the correctness of a distributed application operating on top of a causally-consistent database, an RDT is safe if: (1) it is safe in sequential execution; (2) converges; and (3) the precondition of each operation is stable under the effect of any other concurrent operation. Formally, a database computation is defined by a partial order on operations, representing causality, and a conflict relation (between operations) that further constrains the partial order. A crucial aspect of this logic is that instead of reasoning about all possible interactions between operations, the logic reasons over each operation individually under a set of assumptions on the behaviour of other operations. Recently, the other works (Houshmand, 2019; Li et al., 2020) extend the above safety conditions with a dependency analysis that is used to determine which operations require causal delivery or whether eventual consistency is enough. These (four) safety conditions can be leveraged to define a deterministic ordering relation between concurrent operations, thus enabling the construction of RDTs from sequential data types. The ordering relation is used at runtime to avoid conflicts by locally (re-)ordering conflicting operations when possible and coordinating operations only if correctness cannot be guaranteed otherwise.
- Conflict and Dependency Analyses:
  - Language-based static analysis can extract information at compile-time on which operations can commute with which other operations and thus get information that can be used by the run-time support to decide on call anticipations of operations in replicas without compromising consistency. Data-centric concurrency control (DCCC) shifts the reasoning about concurrency restrictions from control structures to data declaration. It is a high-level declarative approach that abstracts away from the actual concurrency control mechanism(s) in use.

4.1.3 Verification Techniques for Security Properties

For the verification of security protocols, several methods and tools exist that we plan to employ and improve upon, namely:

- Abstract Interpretation, e.g., like *ProVerif* and *PSPSP* (Blanchet, 2016; Hess et al., 2021)

- Model-checking tools like *OFMC* (Mödersheim & Viganò, 2009).
- Privacy-verification tools like *Noname* (Fournet et al., 2023).

A major aspect of integrating different methods is compositional reasoning, i.e., modelling components with their interfaces, ensuring every component adheres to their interface, and an attacker cannot interfere with the components in a way that arises only from the composition. As far as feasible, we plan to use the PSPSP/StateParComp framework of the proof assistant Isabelle/HOL, which ensures that the composition is machine-checked. This is especially helpful in composition since subtle requirements may otherwise be overlooked. Moreover, it allows for the use of the automated verification module PSPSP on many components.

Information flow techniques are applied in this context in complement to the verification of security protocols to ensure the proper cryptographic properties. Information flow checks for confidentiality and integrity of data assuming that all communication and interaction work within the expected parameters. The expected techniques to be used here vary from static type checking, when possible, to dynamic verification with runtime monitors when necessary (E. Geraldo, Santos, & Costa Seco, 2021). Additionally to using a hybrid approach to information flow adapted to declarative event-based languages (E. Geraldo et al., 2023), we are also applying a technique called value-dependent security labels (E. Geraldo et al., 2023; E. Geraldo, Santos, & Costa Seco, 2021; Lourenço & Caires, 2015) that allows for a more flexible and fine-grained definition of security compartments that dynamically adapt to the data being processed and the context of the computation, e.g. the user accessing the data.

4.1.4 Verification Techniques for Federated Learning

Like other communication protocols, we can use the MPST for the specification of federated learning protocols and rely on type-checking (possibly combined with model-checking) to verify their desirable properties. However, some of the federated learning protocols utilize communication patterns that cannot be directly modelled using the existing MPST models. Another direction toward this goal is to

- Use Communicating Sequential Processes Calculus (CSP) (Hoare, 1985) for modelling federated learning protocols. The CSP provides modelling of the concurrency primitives as follows
    - the system components are CSP processes;
    - communication between the system components is performed through the communication channels;
    - the system of parallel processes communicating asynchronously (i.e., without barrier synchronization) is assembled via interleaving of the CSP processes.
- Express a behavioural property of considered protocols (e.g., safety, liveness, deadlock freedom) as a linear temporal logic formula and use model-checkers, such as Process Analysis Toolkit (PAT) (Sun et al., 2009), to verify the correctness of the property.

4.2 Looking Beyond the State-Of-The-Art

4.2.1 Type Systems for Behavioural Analysis of Interactions

State-of-the-art: Extensions to core theories of session types and multiparty session types include features to support heterogeneous swarms: Failure handling extensions include affine sessions, permitting processes to fail, and coordinator-based failure handling techniques

(Viering et al., 2021); other extensions enable dynamically evolving connections between protocol participants (Hu & Yoshida, 2017) and dynamically sized participant pools (Demangeon & Honda, 2012). Recent works preliminarily address compositional verification (Barbanera et al., 2021) of open systems (Horne, 2020) and a session-typed, higher-order, core language (CLASS) that supports concurrent computation with shared linear state (Rocha & Caires, 2021).

Beyond state-of-the-art: In WP4, Task 4.1, we intend to build upon the above approaches, enabling the application of behavioural types to heterogeneous swarms.

CLASS, the first proposal for a foundational language able to flexibly express realistic concurrent programming idioms, features a type system ensuring all the following three key properties: CLASS programs never misuse or leak stateful resources or memory, they never deadlock, and they always terminate. CLASS expressiveness is illustrated with several examples involving memory-efficient linked data structures, sharing of resources with linear usage protocols, and sophisticated thread synchronisation.

In order to address the challenge to account for unreliability and failures persists for session types, (Barwell et al., 2023) has introduced a toolchain that utilises asynchronous MPST with *crash-stop* semantics to support failure handling protocols. Additionally, to support protocols in which participants can join an already existing session (dynamic participants), (Castro-Perez & Yoshida, 2023) has proposed multiparty session types extended with the ability to add unbounded participants dynamically during a protocol execution.

Multiparty protocols with crash-stop failures

We introduce a top-down methodology for designing asynchronous multiparty protocols with crash-stop failures: (1) We use an extended asynchronous MPST theory, which models crash-stop failures (Cachin et al., 2011), and show that the usual session type guarantees remain valid, i.e. communication safety, deadlock-freedom, and liveness; (2) We present a toolchain, **Teatrino**, for implementing asynchronous multiparty protocols, under our new asynchronous MPST theory, in Scala, using the Effpi concurrency library (Scalas et al., 2019)

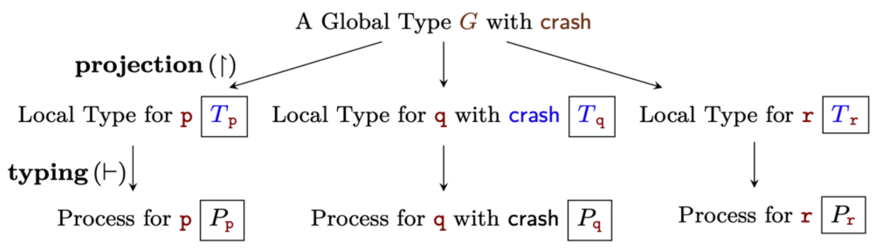
A Global Type G with crash
projection ($\upharpoonright$)
Local Type for p $\boxed{T_p}$
Local Type for q with crash $\boxed{T_q}$
Local Type for r $\boxed{T_r}$
typing ($\vdash$)
Process for p $\boxed{P_p}$
Process for q with crash $\boxed{P_q}$
Process for r $\boxed{P_r}$

Figure 1: Top-down view of MPST with crash.

As depicted in Figure 1, the top-down design of multiparty protocols with crash-stop failures begins with a given *global* type (top), and implementations rely on *local* types (bottom) for participants, obtained from the global type. Well-typed implementations (processes) that conform to a global type are guaranteed to be *correct by construction*, enjoying full guarantees (safety, deadlock-freedom, liveness) from the theory.

We model crash-stop failures, i.e., a process may fail arbitrarily and cease to interact with others. This model is simple and expressive and has been adopted by other approaches (Barwell et al., 2022; Brun & Dardha, 2023).Using global types in our design for handling failures in multiparty protocols presents two distinct advantages: (1) global types provide a simple, high-level means to both specify a protocol abstractly and automatically derive local types; and (2) desirable behavioural properties such as communication safety, deadlock-freedom, and liveness are guaranteed by construction. We focus on *asynchronous* systems, where messages are buffered whilst in transit, since most communication in the real distributed world is asynchronous.

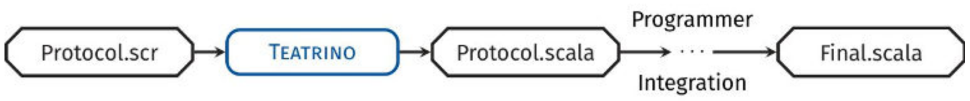

Figure 2: Workflow of Teatrino.

On the practical side, we present a code generator toolchain, **Teatrino**, to implement our MPST theory. As depicted in Figure 2, our toolchain takes an asynchronous multiparty protocol as input, using the protocol description language Scribble (Yoshida et al., 2013), and generates Scala code using the Effpi concurrency library as output. Our code generation technique, as well as the Effpi library itself, utilises the type system features introduced in Scala 3, including match types and dependent function types, to encode local types in Effpi. This approach enables us to specify and verify program behaviour at the type level, resulting in a more powerful and flexible method for handling concurrency. By extending Scribble and Effpi to support *crash detection and handling*, our toolchain **Teatrino** provides a lightweight way for developers to take advantage of our theory, bridging the gap on the practical side. We demonstrate the feasibility of our methodology and evaluate **Teatrino** with examples incorporating crash handling behaviour.

Dynamically updatable multiparty session protocols

Existing MPST frameworks do not support protocols with dynamic unbounded participants and cannot express many common programming patterns that require the introduction of new participants into a protocol. This poses a barrier for the adoption of MPST in languages that favour the creation of new participants (processes, lightweight threads, etc) that communicate via message passing, such as Go or Erlang. To tickle this challenge, we introduce *Dynamically Updatable Multiparty Session Protocols* (**DMst**), a new MPST theory that supports protocols with an *unbounded* number of fresh participants, whose communication topologies are *dynamically updatable*. We prove that **DMst** guarantees deadlock-freedom and liveness. We implement a toolchain, **GoScr** (Go-Scribble), which generates Go implementations from **DMst**, ensuring by construction, that the different participants will only perform I/O actions that comply with a given protocol specification. We evaluate our toolchain by (1) implementing representative parallel and concurrent algorithms from existing benchmarks, textbooks and literature; (2) showing that **GoScr** does not introduce significant overheads compared to a naive implementation, for computationally expensive benchmarks; and (3) building three realistic protocols (dynamic task delegation, recursive Domain Name System, and a parallel Min-Max strategy) in **GoScr** that could not be represented with previous theories of session types.

Next Steps: We will expand upon the research developed in (Barwell et al., 2023) to study different crash models (e.g., crash-recover), as well as failures of other components (e.g., link failures). In addition, we aim to generalise failure handling to accommodate the new crash models and failures, while also relaxing network reliability assumptions. This will enable systems to effectively address network and hardware failures.

We plan further to enhance MPST to incorporate time specifications, affinity, and exception-handling mechanisms. This will tackle the challenges of handling failures, particularly timeouts, that may occur during the execution of communication protocols.

We will also extend MPST to facilitate the dynamic joining and leaving of protocol participants, leveraging (Castro-Perez & Yoshida, 2023) and work on Conversation Types, and explore the integration of verification techniques such as model checking to augment BT and MPST applicability. Moreover, we will study the composition techniques of MPST to enable more comprehensive forms of open system reasoning, fostering both compositional and broader understandings of system behaviour.

All these next-step efforts will provide methodologies to facilitate the reliability of communications across all heterogeneous swarms, while aiding in the verification of properties within the initial TaRDIS toolset.

4.2.2 Verification of RDTs

State-of-the-art: RDTs verified with VeriFx can be transpiled to mainstream languages (currently Scala and JavaScript). VeriFx provides libraries for implementing and verifying Conflict-free Replicated Data Types (CRDTs) and operational transformation functions. These libraries implement the general execution model of those approaches and define their correctness properties.

Beyond the state-of-the-art: Currently there are two alternative approaches for the verification of RDTs, operation-based or data-centric. We aim to combine these two separate verification techniques. Moreover, we aim to analyze the correctness of RDTs that are built by composing existing ones.

Commutation of operations with different consistency requirements

State-of-the-art: To achieve an automatic approach to determine operations commutation, we develop a language-based static analysis to extract information at compile-time and thus get information that can be used by the run-time support to decide on call anticipations of operations in replicas without compromising consistency. We illustrate the formal analysis on several paradigmatic examples and briefly present a proof-of-concept implementation in Java.

Beyond the state-of-the-art: We plan to mechanically prove sound the approach described. The algorithms are already implemented in Coq and the envisaged results are formally stated. We will build on this work to develop the proofs.

Data integrity in concurrent applications

State-of-the-art: We developed AtomiS, a new DCCC approach that requires only qualifying types of parameters and return values in interface definitions, and of fields in class definitions.

The latter may also be abstracted away in type parameters, rendering class implementations virtually annotation-free. From this high-level specification, a static analysis infers the atomicity constraints that are local to each method, considering valid only the method variants that are consistent with the specification, and performs code generation for all valid variants of each method. The generated code is then the target for automatic injection of concurrency control primitives that are responsible for ensuring the absence of data-races, atomicity-violations, and deadlocks.

Beyond the state-of-the-art: We will show that AtomiS formally guarantees thread-safety properties like absence of atomicity violations, data-races, and deadlocks.

4.2.3 Verification of Security Properties

Protocol Verification

State-of-the-art: There is a rich body of research on the verification, where we limit ourselves to black-box models of cryptography (aka Dolev-Yao-style models (Dolev & Yao, 1983)) where we assume that the intruder only uses the normal cryptographic operators (i.e., composing and decomposing messages with known keys) but does not attempt crypto-analysis. One line of works is in bounded model-checking with symbolic techniques (Mödersheim & Viganò, 2009) that allow for analysing protocols with rather complex concepts, but need to impose a bound on the number of steps that honest agents can perform, or otherwise would not terminate on secure protocols. In fact, this bound usually has to be usually extremely low (two or three protocol sessions) and thus these approaches work very well for quickly finding security flaws, but have limited value for positive statements.

To overcome the infinity and state-explosion problem, some tools follow an abstract interpretation approach, most notably the tool ProVerif (Blanchet, 2016). The abstract interpretation here maps fresh messages that have been created in the same context to the same constant, and also ignores the temporal structure, lumping together all messages that are ever available to the intruder, or rather an over-approximation thereof. One can then often efficiently prove that this over-approximation contains no secrets without the exploration of a state-space. While this works very well on simple protocols, the underlying monotonous framework does not allow for protocols that have a mutable long-term state such as a database of requests that have not been processed yet. For this reason, several tools have considered a modified version of the abstraction approach, to allow for a small amount of state while maintaining the advantages of abandoning the state space exploration, namely AIF/AIF-omega/set-pi (Mödersheim & Bruni, 2016) to abstract data by membership in sets, StatVerif (Arapinis et al., 2011) to abstract by the state of some memory cells, and GS-Verif (Cheval, Cortier, et al., 2018) to handle mutable maps.

Another is the inductive method of Paulson in the proof assistant Isabelle/HOL (Bella, 2007; Paulson, 1998). The advantage of this approach is that proofs are machine-checked, so do not rely on the correctness of a verification tool. Moreover, the nature of such a proof assistant does allow a wide range of proof techniques (basically what is accepted mathematics) where a particular abstraction or model-checking approach may not work. On the other hand, most of the proof work has to be done manually with only smaller proof steps being done automatically by the engine. The work PSPSP (Hess et al., 2021) integrates into Isabelle/HOL the abstraction approach of AIF, allowing for automated proofs for protocols with long-term mutable state that have the high security of machine checked proofs.

Beyond the state-of-the-art: We plan to deploy these tools for security verification in TaRDIS, and this is likely to work "out-of-the-box" for many problems already. However, we envision several extensions. First, some cryptographic primitives require modelling of algebraic properties (e.g., applying a verification step to a zero-knowledge proof, as in the EDP case study) and the support for algebraic reasoning is a challenge for all these tools. Moreover, several modern protocols use ratchet mechanisms, which are not supported by the tools currently either, and that are in some sense at odds with the abstract interpretation approaches denoted above. We therefore plan to extend and adapt the abstraction approach for ratchet mechanisms. Moreover, we will investigate alternative ways to prove such protocols manually in Isabelle/HOL and try to obtain a general paradigm for conducting such proofs. The long-term goal is to obtain methods for handling larger classes of protocols automatically.

Protocol Composition

State-of-the-art: When running several protocols together on the same communication medium, where the protocol may share for instance the same public-key infrastructure, there can arise new attacks that the protocols in isolation would not have had, e.g., the intruder can abuse a message from one protocol and play it in the context of another protocol where it is interpreted in a different way by participants. The motivation for protocol composition is that the verification of the variety of protocols for instance on the Internet is far too big to verify them together as one system. Moreover, we cannot expect all protocol developers in the world to coordinate their efforts with all other protocol developers. Finally, a simple update to one protocol may require the entire system to be verified again. The first works on protocol composition concern just parallel composition, i.e., protocols that are basically unrelated except that they share a key infrastructure and the communication medium. Here the main proof argument is that this composition is sound as long as messages of the two protocols are sufficiently different such that they cannot be abused by the intruder in another protocol. A next step is sequential composition, where one the result from one protocol (e.g., a negotiated session key) is the input to the next protocol.

A major generalisation has been achieved with the works of Hess et al (Hess et al., 2023) to allow for the composition of stateful protocols. This in particular allows for sets to be shared between protocols. A typical example is a web server that is "speaking" several protocols, and maintains a database of all orders/tasks that are currently open. In this way, a complex service can be decomposed into smaller units that can be verified independently. This paradigm is so general that it subsumes parallel composition and sequential composition. The fact that this is also implemented in Isabelle/HOL and connected to the PSPSP tool allows for a complete machine-verified security proof of complex systems where small components have been either verified automatically or manually. This is particularly interesting as the compositionality theorems often have subtle requirements and the entire proof is only accepted when all these requirements are satisfied.

Another relevant development is the vertical composition result of Gondron et al. (Gondron & Mödersheim, 2021). Here the idea is that one protocol provides a kind of communication channel that can be used by an application protocol to transmit payload messages. This result builds on the aforementioned stateful composition result: the payload protocol that wants to send a payload message from A to B puts the payload message into a shared set outbox(A,B) where the channel protocol picks it up, applies encryption operations to it (or, if needed, an entire hand-shake protocol between A and B) and sends out the encrypted message on an insecure communication medium. On B’s end, the channel protocol unpacks the received message, performs necessary checks (e.g., message authentication codes) and then delivers the message into the shared set inbox(B,A), where the application can pick it up. For the application protocol this is no more involved than sending and receiving operations and being able to rely on the guarantee given by the channel (e.g., authentication/integrity and confidentiality), but its verification is independent of how the channel protocol works. Dually, the channel protocol can be oblivious to the messages the application protocol is transmitting. It is ensured that this works both in the case that the payload message is known or unknown to the intruder, as well as both when the payload message is fresh or repeated.

Beyond the State-of-the-art: We plan to offer as part of the TaRDIS API a variety of such channels or, more generally, message transmission interfaces. This will include the many standard channels like those provided by TLS where we have typically an unauthenticated client and an authenticated server, and this can be composed with an authentication mechanism (e.g., password-based login, SSO, OAuth) to authenticate the client and obtain a full secure channel.

A major development we plan is to combine the verification of security protocols with the verification of information flow of the TaRDIS applications via the compositionality reasoning. The idea is that we define a security lattice for confidentiality and integrity that is combined with access control, so that data in the program is labelled within this lattice, and similar are the API calls for sending and receiving messages that connect to the security protocols. On the protocol side, we have then to verify that the transmission mechanism is strong enough to ensure the requires integrity, confidentiality, and authorization requirements; on the application side we have to verify that no information flows in a way that contradicts the security policy induced by the lattice, i.e., that unauthorised entities cannot manipulate data and data is not leaked to them.

While the initial goal is to (manually) verify a set of transport mechanisms used in the case studies, we aim for a general automated verification of such vertical composition, and that embedded into the Isabelle proof assistant if feasible.

Protocol Privacy

State-of-the-art: Privacy-type properties for security protocols are more challenging than standard secrecy goals, because they are not about the secrecy of randomly chosen cryptographic secrets like a secret key, but about guessable data such as the names of participants, requested orders, etc. An example relevant to the project is *unlinkability*, i.e., that an observer cannot tell whether two actions have been performed by different entities or the same entity. The classical approach here are observational equivalence approaches (Blanchet et al., 2008; Cortier et al., 2007; Delaune et al., 2008) in some form: a notion of indistinguishability between processes that is applied to two possible scenarios. In the example we may have a process where any number of cards can perform one single session with a card reader and a process where each card can run multiple sessions. There have been several approaches towards this verification problem: DEEPSEC (Cheval, Kremer, et al., 2018) is currently probably the most advanced tool in that it supports the most privacy properties, but it is limited to a bounded number of sessions. The infinite session tools like ProVerif have been equipped with a notion of bi-processes, i.e., processes where each message has a left and right variant. This allows for integration into the unbounded verification approaches; however, it requires, roughly speaking, that all conditions in the program either yield true for both variants or false for both, i.e., the two variants can be distinguished based on conditions.

With alpha-beta privacy (Mödersheim & Viganò, 2019) another approach was proposed that departs from distinguishability-based notions and rather represents a state space where every state contains as a formula beta the information that the intruder can infer from their observations and their knowledge of the protocol - this is basically a symbolic execution of the protocol by the intruder who does in general not know all the concrete values and thus not whether a condition evaluates positively or negatively. Besides that, every state has a formula alpha that describes what information has been publicly released and the intruder is thus cleared to know. It is then defined as a violation of privacy if in any reachable state the intruder can infer from beta any (relevant) information about the payload data that was not released in alpha. There is a first prototype implementation of an automated verification tool for alpha-beta privacy for a bounded number of sessions (Fournet et al., 2023). This is similar to DEEPSEC and other bounded session approaches in that similar challenges have to be overcome, and the state explosion hits already for rather small examples.

Beyond the state-of-the-art: For TaRDIS, privacy-type properties can be highly relevant, since we want to prevent, e.g., in a collaborative environment where not all participants trust each other, that one cannot directly observe a competitors customer base or working patterns. Since the specification of desired privacy properties as observational equivalence can be a challenge for programmers, we see an advantage in being able to specify simply where information is released; however, this also need an extension over existing alpha-beta privacy, as we usually will deal with releases not to a general public, but rather to selected communication partners.

Moreover, in connection with verifying compositionality security protocols with information flow analysis of protocols, we believe it can be sufficient that programmers just need to specify the security labels for the communication end-points and data, and the privacy verification can infer a specification of alpha-beta privacy from there. This will minimise the burden on programmers and designers in specifying the properties that must be analysed. We thus plan to extend compositionality results also for alpha-beta privacy properties and develop infinite-state verification techniques tailored to channel protocols, namely exploiting that the channels in a vertical composition are oblivious to the content of the payload messages. As explained above, this restricts the problem to fall into a much simpler class of privacy properties that conditions of the channel protocol do not depend on.

Behavioural types for security

State-of-the-art: Multiparty session types have already been used to describe security protocols and their properties, even though the concept is geared towards concurrency properties like deadlock-freedom. For instance, (Bruni et al., 2021) describes a privacy-enhancing protocol between non-trusting parties using the general API of a trusted platform module. Such specifications can allow for a more comprehensive and intuitive view of the communication structure and is thus beneficial for programmers, especially given the envisioned use of MPSTs throughout the project.

Beyond the state-of-the-art: It turns out that in such descriptions we cannot directly obtain the local behaviour of each participant by an endpoint projection: it is necessary to understand what the recipient of a message can check (e.g., by decryption, comparing cryptographic hashes and MACs) and how they can compose messages from their current knowledge. This can help avoid subtle specification mistakes: that programmers forget to specify checks that can and should be made, or that a formal model has messages that one end sends but the other can never receive. In both cases an actually vulnerable protocol may falsely be classified as secure. We are currently developing a formal definition of end-point projection for MPSTs that can handle cryptographic operators in an appropriate way.

Accountability, Resilience and Recoverability

State-of-the-art: These three topic complexes are not on a critical path for TaRDIS in the sense that they are nice-to-have, but not essential. Accountability is a set of mechanisms for giving incentives for participants to behave honestly where this cannot be directly enforced (Alhadeff et al., 2012; Kunnemann et al., 2019; Küsters et al., 2010; Mödersheim & Cuellar, 2021), e.g., a certificate server could maliciously issue certificates for people who are not eligible in exchange for a bribe. However, accountability can in this case ensure that the server runs a risk of being detected and penalised for such illegal behaviour. Resilience is about mechanisms to ensure that even after an attacker has compromised some components of a system, other components continue to function and ensure at least their most basic security goals (Jacomme & Kremer, 2018). Recoverability is about the ability of a system to recover to a secure state after a compromise (Cohn-Gordon et al., 2016).

Beyond state-of-the-art: All three topics are with a bit of manual specification work in the realm of what protocol verification tools can already analyse. This can however be a bit of tedious work, e.g., generating many scenarios with scripts and feeding them into verification tools. Time permitting, we plan to investigate whether we can integrate easier ways to specify the three properties for given systems, and improve verification methods for these specifications, avoiding that a large set of scenarios has to be enumerated and checked but can rather be handled symbolically.

Information Flow Control

State-of-the-art: Information flow control a technique that relies on the good behaviour of the system regarding external agents and effectively maintains confidentiality and information integrity withing a system, preventing programming errors from introducing information leaks and consumption of untrusted data. Information flow control goes back to the seminal work of Denning (Denning, 1976), where she proposes a fixed lattice model, base for type systems (Volpano et al., 1996), monitors (Austin & Flanagan, 2009), and hybrid approaches (E. Geraldo, Santos, & Costa~Seco, 2021; Toro et al., 2018) aiming to ensure data confidentiality. Some approaches trade the fixed-lattice model for more flexible mechanisms such as the decentralized label model (Myers & Liskov, 2000) or value-dependent security lattices (E.

Geraldo, 2022; E. M. P. C. R. Geraldo, 2018; Lourenço & Caires, 2015). Independently of the means of enforcement, information flow control is used in many programming languages like Java (E. Geraldo & Costa Seco, 2019; Myers & Liskov, 2000; Snelting et al., 2014), JavaScript (Santos et al., 2018), and OCaml (Simonet, 2003), or tools like JOANA (Snelting et al., 2014), FlowDroid (Arzt et al., 2014), TaintDroid (Enck et al., 2014), and Snitch (E. Geraldo, Santos, & Costa~Seco, 2021).

However, the digitalisation of even-based languages calls for high-level treatments of information flow control that acknowledge the workflows employed (van der Aalst et al., 2017). Some formalisms support access control, but few allow for information flow control. A notable exception is the analysis of non-interference in Petri Nets (Busi & Gorrieri, 2009), applied to business processes in (Accorsi Rafael and Lehmann, 2012; Lehmann & Fahland, 2012). There is also work applying the declarative specification and verification of information flow control in process calculi; some consider only the control flow and the sequence of messages (Abadi et al., 1999; Honda et al., 2000; Kobayashi, 2005), while others integrate imperative languages in the calculus, expanding the verification scope (Honda et al., 2000).

The formal connection between processes and data is a well-known problem addressed by different approaches in the literature (Costa Seco et al., 2018; Galrinho et al., 2021; van der Aalst et al., 2017). The process-data connection may help overcome one of the disadvantages of traditional information flow control approaches for imperative languages. Existing tools can be too strict, making it too difficult to adapt to real-world scenarios. In this context, we highlight the contribution of data-dependent information flow control (Lourenço & Caires, 2015), which is a promising approach to defining security compartments that capture the essence of realistic software.

Beyond state-of-the-art: We will push the state-of-the-art in applying information flow control in event-based structures by studying static analysis techniques that prevent erroneous situations of confidentiality and integrity of data. We will further extend that into developing hybrid approaches in this context, preventing all errors with the most confidence and precision possible. The use cases of the project are rich test beds for such techniques mixing personal information with sensitive computations.

4.2.4 Verification of Federated Learning Orchestration

Formal specification of FL frameworks

State of the art: Originally, federated learning (FL) was introduced by McMahan et al. (McMahan et al., 2017) as a decentralised approach to model learning that leaves the training data distributed on the mobile devices and learns a shared model by aggregating locally computed updates. Besides preserving local data privacy, FL is robust to the unbalanced and non-independent and identically distributed (non-IID) data distributions, and it reduces required communication rounds by 10–100x as compared to the synchronized stochastic gradient descent algorithm. Inspired by (McMahan et al., 2017), Bonawitz et al. (Bonawitz et al., 2017) introduced an efficient secure aggregation protocol for federated learning, and Konecny et al. (Konečný et al., 2017) presented algorithms for further decreasing communication costs. More recently, Bonawitz et al. (Bonawitz et al., 2022) and Perino et al. (Perino et al., 2022) focused on data privacy.

Nowadays, there are many FL frameworks. The most prominent such as TensorFlow Federated (TFF) (McMahan, n.d.), BlueFog (Ying, Yuan, Chen, et al., 2021; Ying, Yuan, Hu, et al., 2021) and Flower (Beutel et al., 2020) are well supported and accepted and they work well in the cloud-edge continuum. However, they are not deployable to edge only, they are not supported on OS Windows, and they have numerous dependencies that make their installation far from trivial.

Recently, in 2021, Kholod et al. (Kholod et al., 2021) made a comparative review and analysis of open-source FL frameworks for IoT, covering TensorFlow Federated (TFF) from Google Inc (*TensorFlow Federated: Machine Learning on Decentralized Data*, n.d.), Federated AI Technology Enabler (FATE) from Webank's AI department (*An Industrial Grade Federated Learning Framework*, n.d.), Paddle Federated Learning (PFL) from Baidu (*An Open-Source Deep Learning Platform Originated from Industrial Practice*, n.d.), PySyft from the open community OpenMined (*A World Where Every Good Question Is Answered*, n.d.), and Federated Learning and Differential Privacy (FL&DP) framework from Sherpa.AI (*Privacy-Preserving Artificial Intelligence to Advance Humanity*, n.d.). They found out that application of these frameworks in the IoTs environment is almost impossible.

Therefore, developing a FL framework targeting smart IoTs in edge systems is still an open challenge. More recently, in 2023, Popovic et al. proposed their solution to that challenge called Python Testbed for Federated Learning Algorithms (PTB-FLA) (Popovic et al., 2023). The work has been carried out within the TaRDIS project task 5.1.

PTB-FLA was developed with the primary intention to be used as a FL framework for developing federated learning algorithms (FLAs), or more precisely as a runtime environment for FLAs. The word "testbed" in the name PTB-FLA that might be misleading was selected by ML & AI developers in TaRDIS project because they see PTB-FLA as an "algorithmic" testbed where they can plugin and test their FLAs. Note that PTB-FLA is neither a system testbed, such as the one that was used for testing the system based on PySyft in (Cheng Shen and Wanli Xue, 2021), nor a complete system such as CoLearn (Feraudo et al., 2020) and FedIoT (Zhang et al., 2021) (for more elaborated comparison with CoLearn and FedIoT see Section I.A in (Popovic et al., 2023)).

PTB-FLA is written in pure Python to keep the application footprint small so to fit to IoTs, and to keep installation as simple as possible (with no external dependencies). PTB-FLA supports both centralised and decentralised FLAs. The former is as defined in (McMahan et al., 2017), whereas the latter are generalised such that each process (or node) alternatively takes server and client roles from (McMahan et al., 2017) or more precisely, it switches roles from server to client and back to server. We describe the one-shot FLA execution of both algorithms.

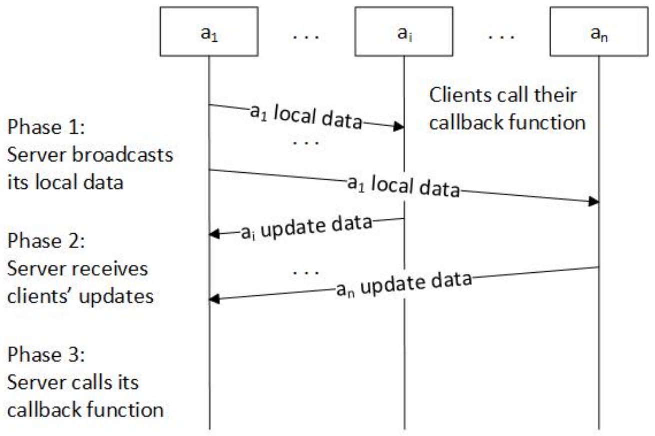
a1
a_i
a_n
Clients call their callback function
Phase 1: Server broadcasts its local data
a1 local data
a1 local data
Phase 2: Server receives clients' updates
a_i update data
a_n update data
Phase 3: Server calls its callback function

Figure 3: The generic centralised one-shot FLA execution.

The generic centralised one-shot FLA has three phases, see Figure 3 (here a_1 is the server and $a_i, i = 2, \ldots, n$, are the clients). In the first phase, the server broadcasts its local data to the clients, which in turn call their callback function to get the update data and store the update data locally. In the second phase, the server receives the update data from all the clients (in any order, caused by arbitrary delays), and in the third phase, the server calls its callback function to get its update data (i.e., aggregated data) and stores it locally. Finally, all the instances return their new local data as their results.

Unlike the generic centralised FLA that uses the single field messages carrying data, the generic decentralised FLA uses the three field messages carrying: the messages sequence number (i.e., the phase number), the message source address (i.e., the source instance network address), and the data (local or update).

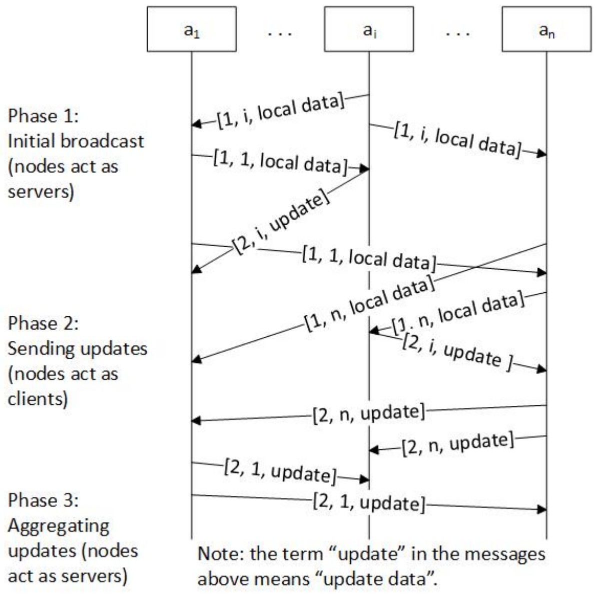
a1
a_i
a_n
Phase 1: Initial broadcast (nodes act as servers)
[1, i, local data]
[1, i, local data]
[1, 1, local data]
[2, i, update]
[1, 1, local data]
[1, n, local data]
[1, n, local data]
[2, i, update]
Phase 2: Sending updates (nodes act as clients)
[2, n, update]
[2, n, update]
[2, 1, update]
[2, 1, update]
Phase 3: Aggregating updates (nodes act as servers)
Note: the term "update" in the messages above means "update data".

Figure 4: The generic decentralised one-shot FLA execution.

The generic decentralised one-shot FLA has three phases, see Figure 4. In the first phase, each instance acts as a server, and it sends its local data to all its neighbours. These messages have the sequence number 1, each instance sends $(n-1)$ such messages and is also the destination for $(n-1)$ such messages.

In the second phase, each instance acts as a client, and it may receive either a message with the sequence numbers 1 or 2. In the latter case, it just stores it in a buffer for later processing in the third phase, whereas in the former case, it calls the client callback and sends the update data in the reply to the message source. Note that during the second phase, the instance does not update its local data, it just passes the update data it got from the client callback function. Since messages are sent asynchronously, they may be received in any order. Figure 4 shows a scenario where the instance a_1 receives the messages in the messages sequence $1-2-1-2$, which is out of the phase order, whereas the instances $a_i$and a_n receive the messages in the sequence $1-1-2-2$, which is in the phase order. However, by using the abovementioned buffering, the instance a_1 postpones processing of the phase 2 messages until the third phase. The second phase is completed after the instance received and processed all $2(n-1)$ messages. In the third phase, each instance again acts as a server, and it calls the server callback function to get its update data (e.g., aggregated data) and stores it locally. Finally, all the instances return their new local data as their results.

PTB-FLA enforces a restricted programming model, where a developer writes a single application program, which is later instantiated and launched by the PTB-FLA launcher as a set of independent processes, and within their application program, a developer only writes callback functions for the client and the server roles, which are then called by the generic federated learning algorithms hidden inside PTB-FLA.

Correct orchestration is one of the main challenges of FL frameworks nonetheless it has not yet received proper attention. There is an urging demand for formal specification and verification of FL frameworks. So far, PTB-FLA usage has been illustrated and validated by three simple examples in (Popovic et al., 2023). PTB-FLA has not been formally verified.

Beyond the state of the art: Within WP4, Task 4.4, we have formally verified the correctness of two generic FL algorithms, a centralised and a decentralised one using the Communicating Sequential Processes calculus (CSP) and the Process Analysis Toolkit (PAT) model checker. All details are presented in (Prokic et al., 2023).

The work has been done in two phases:

1) In the first phase, presented below, we construct CSP models of the generic centralised and decentralised FLAs as faithful representations of the real Python code. We construct these models in a bottom-up fashion in two steps. In the first step, we construct processes corresponding to generic FL algorithm instances, and in the second step, we construct the system model as an asynchronous interleaving of n FL algorithm instances.
2) In the second phase, presented further in this document in Section 4.1.8, we formally verify the CSP models constructed in the first phase in two steps. In the first step, we formulate desired system properties, namely deadlock freeness (safety property) and successful FLA termination (liveness property). We formulate the latter property in two equivalent forms (reachability statement and always-eventually LTL formula). In the

second step, we use PAT to automatically prove formulated verification statements (Sun et al., 2009).

The CSP models are constructed bottom-up as a faithful representation of the real Python code and they are automatically checked top-down by PAT.

To the best of our knowledge, this is the first result that formally verifies decentralised FLAs.

We present two models: one for the centralised algorithm and another one for the decentralised algorithm.

Modeling centralised algorithm

Figure 5 shows a CSP model for our centralised algorithm. Lines 2-3 define number of nodes (NoNodes) (indexed with 0, 1, 2, . . .) with the server (FlSrvId) having the largest index, and other nodes being clients. We remark we could set here the index of the server node with the smallest index, but this would in fact make our model less intuitive because of the channel manipulation (as explained below). Lines 4-5 define arrays of local data ldata and private data pdata - one per each node. The communication channels are defined in lines 8-9. The array of channels server2client - one per each client (hence, NoNodes−1 channels) are used for the server broadcast of their local data to the clients (one channel per client). Notice that the indexes of array elements are generated starting with 0, hence the channel index indicates the index of the client node. Since we consider one-shot algorithm the server sends their local data only once, hence the channels are specified to have FIFO buffers of size 1.

```
 1  // PTB-FLA
 2  enum {False, True};
 3  #define NoNodes 3;
 4  #define FlSrvId 2;
 5  var ldataArr[NoNodes];
 6  var pdataArr[NoNodes];
 7  var terminated;
 8  channel server2client[NoNodes-1] 1;
 9  channel clients2server NoNodes-1;
10
11  FlCentralised(noNodes, nodeId, flSrvId, ldata, pdata) =
12    if(nodeId == FlSrvId) {
13      CeServer(noNodes, nodeId, flSrvId, ldata, pdata)
14    } else {
15      CeClient(noNodes, nodeId, flSrvId, ldata, pdata)
16    };
17
18  CeServer(noNodes, nodeId, flSrvId, ldata, pdata) =
19    {terminated = False} ->
20    CeBroadcastMsg(0, noNodes, nodeId, ldata);
21    CeRcvMsgs(0, noNodes-1);
22    {terminated = True} -> Skip;
23
24  CeBroadcastMsg(id, noNodes, nodeId, ldata) =
25    if(id != nodeId) {
26      server2client[id]!ldata -> Skip
27    };
28    if(id < noNodes-1) {
29      CeBroadcastMsg(id+1, noNodes, nodeId, ldata)
30    };
31
32  CeRcvMsgs(i, noMsgs) =
33    if(i < noMsgs) {
34      clients2server?update -> CeRcvMsgs(i+1, noMsgs)
35    };
36
37  CeClient(noNodes, nodeId, flSrvId, ldata, pdata) =
38    server2client[nodeId]?srvLdata ->
39    clients2server!ldata+srvLdata ->
40    Skip;
41
42  SysCentralised() =
43    |||nodeId:{0..NoNodes-1}
44    @FlCentralised(NoNodes,
45                   nodeId,
46                   FlSrvId,
47                   ldataArr[nodeId],
48                   pdataArr[nodeId]);
```

Figure 5: CSP model for centralised algorithm.

Channel **clients2server** is used in the second phase of our algorithm, i.e. for clients replying to the server with the update data. The FIFO size of this channel is **NoNodes**−1, since all clients reply with a single update.

Lines 11-16 define a generic node as a CSP process with parameters of the number of nodes, identification of the node, index of the server, their local and private data. We remark that parameters *sfun*, *cfun*, and *noIters*, also present in fl_centralised, were considered out of the scope for this model. Based on the node index the process proceeds as the server node **CeServer** or as one of the client nodes **CeClient**.

The server node is modeled in lines 18-22. The process first checks if it is terminated: if not it performs the broadcasting of the local data via **CeBroadcastMsg**, then proceeds to phase 2 by receiving updates via **CeRcvMsgs**. The successful termination is modeled with **Skip**. The broadcasting of server's local data **CeBroadcastMsg** is defined in lines 24-30. The server sends **ldata** on channels **server2clients[id]** (if **id** is not their own index), and then recursively calls itself with index increased by 1 - if the index is less than **noNodes**−1. Since **CeServer** passes **id** to **CeBroadcastMsg** to be 0, the server will send the local data to all the clients exactly once. Once the broadcast is done, the server starts receiving clients' updates on channel **clients2server** as defined with **CeRcvMsgs** in lines 32-35.

The client process is defined with **CeClient** in lines 37-40. The client with index **nodeId** first receives server's local data on channel **server2client[nodeId]**, and then replies updated server's local data with its own local data (here for simplicity modeled with addition) on channel **clients2server**, after which client process successfully terminates.

The system consisting of **NoNodes**−1 clients and a single server is then modeled as the interleaving of the **FlCentralised** processes (lines 42-48), since all processes but one indexed **FlSrvId** are instantiated as clients (and the one indexed **FlSrvId** is instantiated as a server).

Modelling decentralised algorithm

The CSP model for our decentralised algorithm is given in Figure 6. Albeit more complex than the centralised one, the decentralised algorithm yields a slightly simpler CSP model. The reason is that all nodes in the system have the same behaviour. In phase 1 all nodes behave as servers broadcasting their local data to all other nodes, which in turn update the data and return an answer in phase 2. All the nodes receive messages from all other nodes as they arrive, but first process the messages from phase 1 and only then deals with the messages from the phase 2. We model this behaviour with assigning two channels to each process (i.e. node). One channel is for receiving messages from other processes, called **tonode**, with buffer of size 2*(**NoNodes**-1) (line 7), since the node will receive messages from all other nodes from both phases. The other channel assigned to node, called **buffer** (line 8), serves only for storing messages from the second phase while all messages from the first phase are processed - later in phase 3 the same node will read those messages. Hence, the buffer size of these channels are **NoNodes**-1.

The node processes are defined with **FlDecentralised** in lines 10-15. Process first broadcasts their local data with **DeBroadcastMsg** (defined in lines 17-23) - which behaves in the same way as **CeBroadcastMsg** in the centralised algorithm, except that the sent messages now contain not only field for local data of the node, but also fields marking the phase (here 1) and the node's index (that the receiving node uses for the reply in phase 2).

```
 1  // PTB-FLA
 2  enum {False, True};
 3  #define NoNodes 3;
 4  var ldataArr[NoNodes];
 5  var pdataArr[NoNodes];
 6  var terminated;
 7  channel tonode[NoNodes] 2*(NoNodes-1);
 8  channel buffer[NoNodes] NoNodes-1;
 9
10  FlDecentralised(noNodes, nodeId, ldata, pdata) =
11    {terminated = False} ->
12    DeBroadcastMsg(0, noNodes, nodeId, ldata);
13    DeRcvMsgs(0, noNodes, nodeId, ldata);
14    DeRcvMsgs2(0, noNodes, nodeId);
15    {terminated = True} -> Skip;
16
17  DeBroadcastMsg(id, noNodes, nodeId, ldata) =
18    if(id != nodeId) {
19      tonode[id]!1.nodeId.ldata -> Skip
20    };
21    if(id < noNodes-1) {
22      DeBroadcastMsg(id+1, noNodes, nodeId, ldata)
23    };
24
25  DeRcvMsgs(i, noNodes, nodeId, ldata) =
26    if(i < 2*noNodes-2) {
27      tonode[nodeId]?phase.from.nodeldata ->
28      if(phase == 1){
29      tonode[from]!2.nodeId.ldata+nodeldata ->
30      DeRcvMsgs(i+1, noNodes, nodeId, ldata)
31      } else {
32      buffer[nodeId]!phase.from.nodeldata ->
33      DeRcvMsgs(i+1, noNodes, nodeId, ldata)
34      }
35    };
36
37  DeRcvMsgs2(i, noNodes, nodeId) =
38    if(i < noNodes-1) {
39      buffer[nodeId]?phase.from.update ->
40      DeRcvMsgs2(i+1, noNodes-1, nodeId)
41    };
42
43  SysDecentralised() =
44    |||nodeId:{0..NoNodes-1}
45    @FlDecentralised(NoNodes,
46                     nodeId,
47                     ldataArr[nodeId],
48                     pdataArr[nodeId]);
```

Figure 6: CSP model for decentralised algorithm.

The node then proceeds with receiving messages from all other nodes with **DeRcvMsgs**, and finally (phase 3) process the messages from the second phase with **DeRcvMsgs2**.

DeRcvMsgs is given in lines 25-35. Here we deviate from the centralised algorithm: node receives all messages from both phases from the other nodes and then performs an analysis on the phase of the received message. If the phase is 1, the node replies updated data to **from**

they received message in the first place, marking the phase of the message 2. If, on the other hand, the phase is 2, the node stores the message to their own channel **buffer[nodeId]**. Once the node process all messages from phase 1 (and buffers all messages from phase 2), **DeRcvMsgs2** (lines 37-41) is used to read from the **buffer[nodeId]**, which behaves in the same way as **CeRvcMsgs** from the centralised algorithm.

The system of **NoNodes** nodes is finally modeled as the interleaving of the **FlDecentralised** processes in lines 43-48.

Next steps: Multiparty Asynchronous Session Types (MPST) were tailored to describe distributed protocols relying on asynchronous communications. Hu and Yoshida extended MPST in (Hu & Yoshida, 2017) with explicit connection actions to support protocols with optional and dynamic participants. These extended MPST enabled modelling and verification of some protocols in cloud-edge continuum in (Simic et al., 2021). However, we could not use these extended MPST to model the generic centralised and decentralised FLAs, because we could not express arbitrary order of message arrivals that take place at an FLA instance.

The design of robust protocols for coordination of peer-to-peer systems is difficult because it is hard to specify and reason about their global behaviour. Recently, (Kuhn et al., 2023) presented an approach where a so-called swarm protocol is a global system specification, whereas swarm protocol projections to machines are local specifications of peers. They claim that swarms are dead- lock free, but liveness is not guaranteed in their theory. We find this approach interesting and in our future work we plan to investigate whether it would be feasible for our generic FLAs.

At present, we identify some of the differentiating points between (Kuhn et al., 2023) and our work: (i) in their approach communication of peers is conducted through a shared log instead of point-to-point message passing; (ii) they model peers using finite state automata, while we use (CSP) processes; (iii) they model protocols in the style of MPST via top-down approach (projecting global type onto peers to obtain local type specification) while we only write local processes specifications, that we ensemble together to obtain global protocol behaviour; (iv) they use TypeScript language and develop tools to check protocol conformance at runtime through equivalence testing, whereas our protocols are written in Python language, modelled in CSP, and we use PAT to prove deadlock freeness and liveness.

We plan further to combine the two approaches in order to work towards the verification of properties of the TaRDIS initial toolset of WP6 along with requirements of WP2 and developments of WP3 and WP5.

Formal Verification of Distributed AI

State-of-the-art: The most prominent FL frameworks include Flower (Beutel et al., 2020) (for centralised federated learning) and BlueFog (Ying, Yuan, Hu, et al., 2021) (for fully distributed learning), promising scalable decentralised ML workloads on heterogeneous edge devices. More recently, as part of the TaRDIS project, (Popovic et al., 2023) proposed a framework called Python Testbed for Federated Learning Algorithms (PTB-FLA). However, none of the above frameworks have been utilizing methods for formal verification, lacking trustworthy methodologies that can provide safe and reliable systems.

Beyond state-of-the-art: As a first step towards formal verification of the FL algorithms we conducted an investigation on the formal verification of the two generic FL algorithms introduced in (Popovic et al., 2023) (and presented above in this document). To achieve the formal verification, we first modeled the protocols of the two algorithms using the CSP process algebra (Hoare, 1985) (the CSP models are also given above in this document). The correctness of the CSP models is automatically checked by PAT, which supports the system analysis in two ways: simulation and model checker. We have used the latter one.

The correctness of the centralised and decetralised algorithms is verified by proving the deadlock freeness (safety property) and successful termination (liveness property). The properties of algorithms are stated in the form of queries, called assertions, which are checked by PAT (Sun et al., 2009).

```
1  // ...
2  // CSP model for centralised algorithm
3  // ...
4
5  #assert SysCentralised() deadlockfree;
6  #define Terminated (terminated == True);
7  #assert SysCentralised() reaches Terminated;
8  #assert SysCentralised() |= []<> Terminated;
```

Figure 7: Verifying centralised algorithm.

The assertions that formally verify the correctness of the centralised algorithm are shown in Figure 7. The assertion given in line 5 of Figure 7 claims that the centralised algorithm is **deadlock-free**. PAT model checker performs Depth-First-Search or Breath-First-Search algorithm to check if the assertion is true. It explores unvisited states until a non-terminated state with no further move called a deadlock state is found or all states have been visited.

The assertion given in line 7 of Figure 7 claims that the centralised algorithm reaches a terminated state. This assertion is checked by performing Depth-First-Search algorithm. PAT model checker repeatedly explores all unvisited states until it finds a state at which the condition Terminated is satisfied or it visits all the states. The condition **Terminated** is a proposition defined as a global definition (line 6 in Figure 7).

PAT supports the full syntax of the linear temporal logic (LCL), which is used in the last assertion of Figure 7 that claims our centralised algorithm satisfies formula **[]<> Terminated**. The modal operator [] reads as *always* and the operator <> reads as *eventually*, so the statement asserts our centralised algorithm *always eventually reaches the terminated state*.

```
// ...
// CSP model for decentralised algorithm
// ...

#assert SysDecentralised() deadlockfree;
#define Terminated (terminated == True);
#assert SysDecentralised() reaches Terminated;
#assert SysDecentralised() |= []<> Terminated;
```

Figure 8: Verifying decentralised algorithm.

The proof of the correctness of our decentralised algorithm is given in Figure 8 and follows the same explanations given for the centralised one.

Next steps: We will build upon the research developed in (Prokic et al., 2023) to design a framework for the safe orchestration of decentralised swarm ML. Further developments of TaRDIS will enable the coordination of the ML primitives ensuring that each primitive has access to the data that it requires. The framework will ensure the safe execution of machine learning actions at collaborative smart edge-nodes. Finally, we will integrate the analyses developed across WP4 with the AI-based optimisation developed in WP5. Resource orchestration will be made transparent and hence more trustworthy by exploiting transparent and secure data management from WP6 that include swarm ML/DL models and models for reinforcement learning(Södergård et al., 2020). We will investigate local model explainability based on LIME (Ribeiro et al., 2016) and local surrogate decision trees to be linked with allocated resource schemes to increase explainability of decisions on resource allocations to humans.

5. EXPECTED RESULTS FOR USE CASES

In this section, we demonstrate the desirable models and properties associated with use cases that necessitate specification and verification.

5.1 FAILURE MODELS

- **Actyx, GMV**

1. Devices can be inaccessible for arbitrary but bounded periods of time; during this time local computation on the device may or may not be possible (i.e., device can lose network connectivity or battery, application can be stopped and restarted, …).
2. Devices can be destroyed—fail-stop mode.

Byzantine faults are not a concern at this stage because all devices are centrally managed by a single entity, and potential middleware bugs are not being considered for now.

- **EDP**
    1. Grid: A failure in the connectivity of prosumers to the community infrastructures can disrupt the system.
        - Assumption regarding overall system architecture: there is a central entity (MainProvider, e.g., EDP) that owns a trusted, known, and centralised infrastructure. This infrastructure is assumed to have high availability (99.999%).
    2. Processes: Failures to provide energy can result from not meeting consumption requests and energy offer timeframes. Another failure scenario can occur if there is a delayed response to a system failure in providing a failback power source to all requests.
    3. Optimization: The system should optimize the matching between producers and consumers within the energy community, resorting to other communities or the grid only when there are no local alternatives available for production, consumption, or storage.
- **Telefónica**

    A core part of Federated-Learning-as-a-Service (FLaaS) is privacy. Failure to protect participant data can lead to privacy breaches and other consequences. Proper privacy-preserving techniques should be implemented to prevent data exposure. Hence, Telefónica needs to account for these failure models:

    1. *Network:* FLaaS relies on communication between the server and participating devices or clients. Communication failures, including network outages, delays, and packet loss, can disrupt the learning process.
    2. *Data:* In FLaaS, data distribution across participants may be imbalanced, leading to biased models. Addressing data imbalance challenges and ensuring that the model remains fair, and representative is essential.
    3. *Model:* The aggregation process of model updates from participants can be inefficient or error-prone, affecting the quality of the global model. Implementing efficient and robust aggregation algorithms is crucial.

4. *Users and nodes:* Participants in FLaaS may drop out due to various reasons, such as device unavailability, network issues, or user preferences. Managing participant dropouts without compromising the model's performance is a challenge. Further, nodes (in the hierarchy) might exhaust their computational, memory, or bandwidth resources, affecting their ability to participate effectively.

5.2 Desirable Liveness Properties

- **Actyx**
  1. A workflow always terminates, either with an error condition or by reaching a terminal state; this assumes that non-modelled state value computations terminate, and it assumes that we limit workflow recursion using established techniques such as gas/fuel consumption or timeouts.
  2. Failures (incl. running out of gas/fuel) are handled by other workflows, e.g., using an escalation or supervision scheme, also allowing compensating actions or recovery to restore a valid overall system state.
  3. All non-failing participants in a workflow reach eventual consensus on the sequence of states traversed by the execution of this workflow; some participants may lump together workflow states if they don’t care about their distinction.
- **EDP**
  1. There is always a way of satisfying a consumption or production request.
- **GMV**
  1. Navigation filter shall eventually converge.
- **Telefónica**
  1. Participants are allowed to join or leave the system dynamically.
  2. Feedback mechanisms and incentives should be implemented in FLaaS to motivate participants for continued engagement.
  3. Ensuring the absence (or at least minimising) data imbalance can be tackled with quality (quantitative) models for data integrity (as a balanced aggregation of parts).

5.3 Desirable Safety Properties

- **Actyx**
  1. Deadlock-freedom: no participant can get stuck; if the local event log puts the local view of the workflow into a state where the participant may act (i.e., publish events) then it can do so, which will advance the local workflow view. Together with termination (liveness property), this implies lock- and starvation-freedom.
  2. Causality captured in the workflow is maintained in each participant’s local view (e.g., “this event can only be seen after that other event has been seen”); this allows real-world safety properties like “the robot can only take the workpiece after it has been placed on the shelf”.
  3. The programmer can formulate safety constraints like “event A never applies before event B”; since we aim for dynamic systems, we start scoping these properties to each workflow instance—we regard them as isolated even though they can in principle interfere through overlapping event subscriptions

4. Conflict-detection: deadlock-freedom in this setting implies the possibility of divergent event histories which will be reconciled as events are disseminated (i.e., undoing invalidated state computations).
5. Each participant can detect when reconciliation has invalidated any of its prior actions (i.e., external effects incl. event emission); this allows compensating actions to be taken.
6. Compensating event emissions shall not break the other properties listed above.

- **EDP**
  1. Consumption and production requests are always matched in a compatible way.
  2. The data provided by the devices to the community/grid is used in accordance with specified guidelines.
- **GMV**
  1. Deadlock-freedom: entities become available for synchronous communication in predetermined time slots as pairs. However, if only a single entity becomes available, it may result in blocking, potentially leading to a deadlock. In simpler terms, the satellite control code must be designed to prevent the occurrence of deadlocks.
  2. Navigation accuracy shall be at least X meters.
- **Telefónica**
  1. Concurrent updates from multiple participants can be handled.
  2. Fault tolerance: resilient to failures, including server crashes, communication interruptions, or participant dropouts.
  3. Presence of malicious: compromised participants can send incorrect or malicious updates to the centre server, affecting the integrity of the global model. Mechanisms are employed to detect and mitigate the impact of Byzantine participants.
  4. Efficient and robust aggregation algorithms require good sound concurrency control (approaches to control data races are key for consistency/integrity)

5.4 Desirable Security and Privacy Properties

- **Actyx**
  1. All communication between participants (i.e., event generation, and receiving an event in a callback function) must be encrypted such that both confidentiality and injective agreement (integrity and authentication, protected from replay) on the transmitted events is ensured. This will be verified for the transmission protocols underlying the TaRDIS API that is handling the events exchange in a heterogenous swarm.
  2. The correct deployment of the channels by the application must be verified: the communication may need to be separated w.r.t. a setup of subscriptions for certain event types. Moreover, no communication should be performed outside the TaRDIS event-based model, or, if it is deemed necessary, verified that this cannot leak secrets or import untrusted information.
  3. To an external observed (non-participant) it must not be observable which events are taking place, including what type of event. We may assume here that an attacker cannot perform a precise timing/traffic analysis (which to protect against

would require dummy events and batching of messages that may not be practically feasible).

4. Any protocols that are deployed for updating cryptographic material (e.g., creating a new key when a participant leaves a group) should ensure security against outsiders and malicious insiders (e.g., participants who shall leave a group but try to manipulate the update key exchange such that they continue to have access to the group communication).
5. We will also investigate practically feasible ways to allow the above security and privacy in presence of malicious participants, by either using further cryptographic means (e.g., restricting access to certain parts of the communication, or using zero-knowledge proofs) as well as non-cryptographic means such as accountability.
6. We intend to leverage ML for anomaly detection in event logs, with federated learning being a potential approach.

- **EDP**
  1. Only the model parameters of distributed ML in a heterogenous swarm can be sent by the clients and servers. The actual data should always remain private to the clients and servers. This will be verified statically. These properties contribute to privacy protection in the TaRDIS model.
  2. All communication between participants must ensure confidentiality and injective agreement on the transmitted events is ensured or any other communication channels that are employed. This will be verified for the transmission protocols underlying the TaRDIS API that is handling the events exchange in a heterogenous swarm.
  3. The correct deployment of the channels by the application must be verified: each generation of an event (or sending of a message) must only be visible to the participants that are allowed to see it, and vice-versa events (or received messages) can only be reacted upon when it comes from a participant who is authorized to influence the respective data.
  4. To an external observer (non-participant) it must not be observable which events are taking place, including what type of event. We may assume here that an attacker cannot perform a precise timing/traffic analysis (which to protect against would require dummy events and batching of messages that may not be practically feasible).
  5. The observability requirement explicitly includes unlinkability goals (as far as feasible): it should be impossible (or very hard) to obtain profiles of participants linking their actions together. This may involve the use of special zero-knowledge primitives in the implementation of communication channels and protocols.
  6. Any protocols that are deployed for updating cryptographic material (e.g., creating a new key when a participant leaves a group) should ensure security against outsiders and malicious insiders.
  7. Contracts and agreements between participants should be accountable, i.e., in case of a breach of an agreement, enough evidence is available to prove the breach to a third party (e.g., in court or to an arbitration entity).
- **GMV**
  1. All communication between participants must ensure confidentiality and injective agreement on the transmitted events is ensured or any other communication channels that are employed. This will be verified for the transmission protocols

   underlying the TaRDIS API that is handling the events exchange in a heterogenous swarm.
2. The correct deployment of the channels by the application must be verified: each generation of an event (or sending of a message) must only be visible to the participants that are allowed to see it, and vice-versa events (or received messages) can only be reacted upon when it comes from a participant who is authorized to influence the respective data.
3. Any protocols that are deployed for updating cryptographic material (e.g., creating a new key when a participant leaves a group) should ensure security against outsiders and malicious insiders.
4. The satellites share their models' coefficients, which are used for orbit propagation, and enhance their individual estimates through federated learning.

- **Telefónica**
  1. Only the model parameters of FL can be sent by the clients and servers. The actual data should always remain private to the clients and servers. This will be verified statically. These properties contribute to privacy protection in the TaRDIS model.
  2. Besides the privacy properties on the application/machine learning level, all communication between participants must ensure both confidentiality and injective agreement on any communication channel (including the communication of events). This shall be a verified property of the transmission protocols underlying the TaRDIS API (or whatever communication channels shall be used). These properties shall also hold amongst honest participants when some participants are dishonest (e.g., compromised or malicious users).
  3. The correct deployment of the channels by the application must be verified, including that guessable, known or repeated messages communicated by the application can impact the goals.
  4. Any protocols that are deployed for updating cryptographic material (e.g., creating a new key when a participant leaves a group) must ensure that the updated key is secure, possibly also removing access to an attacker who had successfully hacked into the system.

5.5 Desirable Data Convergence and Integrity Properties

Actyx's liveness property 3 also doubles as a data convergence property, as “eventual consensus on the sequence of states traversed by the execution of this workflow” implies convergence on the system overall state (thus on data). It also implies data integrity as no two participants that can change the state won't eventually reach consensus on the sequence of states traversed by the execution of the workflow. Safety properties 2 and 4 of Actyx are critical to ensure this liveness property.

6. CONCLUSIONS

TaRDIS development environment's primary goal is to assist developers in building correct systems through automatic analysis of interactions between various components within a distributed system. This approach ensures that applications are inherently designed for correctness, taking into account both application invariants and the specifics of the execution environment.

As a first step to address these challenges, we categorise the properties regarding to TaRDIS use cases that require analysis and verification. Additionally, we delve into both established and sophisticated verification techniques that will be utilised to validate these properties.

The key contributions of this report include identifying the challenges arising from intelligent swarms as well as use cases related to verification and analysis, categorising the properties that will undergo in-depth analysis in the upcoming WP4 deliverables, organising them based on the specific tasks to which they are assigned, classifying the existing verification techniques and discussing how TaRDIS will go beyond the state-of-the-art, and summarizing the desirable models and properties that are specifically relevant to the TaRDIS use cases. We consolidate these insights and apply them to TaRDIS models to ensure that they satisfy the desirable security, data integrity, AI coordination, and interaction properties, as aligned with the specific TaRDIS use cases and requirements.

Funded by the European Union

7. Bibliography

A world where every good question is answered. (n.d.). https://www.openmined.org

Abadi, M., Banerjee, A., Heintze, N., & Riecke, J. G. (1999). A Core Calculus of Dependency. In A. W. Appel & A. Aiken (Eds.), *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999* (pp. 147–160). ACM. https://doi.org/10.1145/292540.292555

Accorsi Rafael and Lehmann, A. (2012). Automatic Information Flow Analysis of Business Process Models. In A. and K. E. Barros Alistair and Gal (Ed.), *Business Process Management* (pp. 172–187). Springer Berlin Heidelberg.

Alhadeff, J., Van Alsenoy, B., & Dumortier, J. (2012). The accountability principle in data protection regulation: Origin, development and future directions. In *Managing Privacy Through Accountability*. https://doi.org/10.1057/9781137032225_4

An Industrial Grade Federated Learning Framework. (n.d.). https://fate.fedai.org/

An Open-Source Deep Learning Platform Originated from Industrial Practice. (n.d.). https://www.paddlepaddle.org.cn/en

Ancona, D., Bono, V., Bravetti, M., Campos, J., Castagna, G., Deniélou, P.-M., Gay, S. J., Gesbert, N., Giachino, E., Hu, R., Johnsen, E. B., Martins, F., Mascardi, V., Montesi, F., Neykova, R., Ng, N., Padovani, L., Vasconcelos, V. T., & Yoshida, N. (2016). Behavioral Types in Programming Languages. *Found. Trends Program. Lang.*, *3*(2–3), 95–230. https://doi.org/10.1561/2500000031

Arapinis, M., Ritter, E., & Ryan, M. D. (2011). StatVerif: Verification of stateful processes. *Proceedings - IEEE Computer Security Foundations Symposium*. https://doi.org/10.1109/CSF.2011.10

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., & McDaniel, P. (2014). FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 259–269. https://doi.org/10.1145/2594291.2594299

Austin, T. H., & Flanagan, C. (2009). Efficient Purely-Dynamic Information Flow Analysis. *Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, 113–124. https://doi.org/10.1145/1554339.1554353

Barbanera, F., Dezani-Ciancaglini, M., Lanese, I., & Tuosto, E. (2021). Composition and decomposition of multiparty sessions. *J. Log. Algebraic Methods Program.*, *119*, 100620. https://doi.org/10.1016/j.jlamp.2020.100620

Barwell, A. D., Hou, P., Yoshida, N., & Zhou, F. (2023). Designing Asynchronous Multiparty Protocols with Crash-Stop Failures. In K. Ali & G. Salvaneschi (Eds.), *37th European Conference on Object-Oriented Programming (ECOOP 2023)* (Vol. 263, pp. 1:1–1:30).

Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ECOOP.2023.1

Barwell, A. D., Scalas, A., Yoshida, N., & Zhou, F. (2022). Generalised Multiparty Session Types with Crash-Stop Failures. In B. Klin, S. Lasota, & A. Muscholl (Eds.), *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland* (Vol. 243, pp. 35:1–35:25). Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.CONCUR.2022.35

Bella, G. (2007). *Formal Correctness of Security Protocols*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68136-6

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., & Lane, N. D. (2020). Flower: A Friendly Federated Learning Research Framework. *CoRR*, *abs/2007.14390*. https://arxiv.org/abs/2007.14390

Blanchet, B. (2016). Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends® in Privacy and Security*, *1*(1–2). https://doi.org/10.1561/3300000004

Blanchet, B., Abadi, M., & Fournet, C. (2008). Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, *75*(1). https://doi.org/10.1016/j.jlap.2007.06.002

Bonawitz, K. A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., & Seth, K. (2017). Practical Secure Aggregation for Privacy-Preserving Machine Learning. In B. Thuraisingham, D. Evans, T. Malkin, & D. Xu (Eds.), *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017* (pp. 1175–1191). ACM. https://doi.org/10.1145/3133956.3133982

Bonawitz, K. A., Kairouz, P., McMahan, B., & Ramage, D. (2022). Federated learning and privacy. *Commun. ACM*, *65*(4), 90–97. https://doi.org/10.1145/3500240

Brun, M. A. Le, & Dardha, O. (2023). MAGπ: Types for Failure-Prone Communication. In T. Wies (Ed.), *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings* (Vol. 13990, pp. 363–391). Springer. https://doi.org/10.1007/978-3-031-30044-8_14

Bruni, A., Carbone, M., Giustolisi, R., Mödersheim, S., & Schürmann, C. (2021). Security Protocols as Choreographies. In D. Dougherty, J. Meseguer, S. A. Mödersheim, & P. D. Rowe (Eds.), *Protocols, Strands, and Logic - Essays Dedicated to Joshua Guttman on the Occasion of his 66.66th Birthday* (Vol. 13066, pp. 98–111). Springer. https://doi.org/10.1007/978-3-030-91631-2_5

Bunte Olav and Groote, J. F. and K. J. J. A. and L. M. and N. T. and de V. E. P. and W. W. and W. A. and W. T. A. C. (2019). The mCRL2 Toolset for Analysing Concurrent Systems. In L. Vojnar Tomáš and Zhang (Ed.), *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 21–39). Springer International Publishing.

Busi, N., & Gorrieri, R. (2009). Structural Non-Interference in Elementary and Trace Nets. *Mathematical. Structures in Comp. Sci.*, *19*(6), 1065–1090. https://doi.org/10.1017/S0960129509990120

Cachin, C., Guerraoui, R., & Rodrigues, L. E. T. (2011). *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer. https://doi.org/10.1007/978-3-642-15260-3

Castro-Perez, D., Hu, R., Jongmans, S.-S., Ng, N., & Yoshida, N. (2019). Distributed programming using role-parametric session types in go: statically-typed endpoint APIs for dynamically-instantiated communication structures. *Proc. ACM Program. Lang.*, *3*(POPL), 29:1–29:30. https://doi.org/10.1145/3290342

Castro-Perez, D., & Yoshida, N. (2023). Dynamically Updatable Multiparty Session Protocols: Generating Concurrent Go Code from Unbounded Protocols. In K. Ali & G. Salvaneschi (Eds.), *37th European Conference on Object-Oriented Programming (ECOOP 2023)* (Vol. 263, pp. 6:1–6:30). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ECOOP.2023.6

Cheng Shen and Wanli Xue. (2021). An Experiment Study on Federated Learning Testbed. In T. and S.-I. C. and J. A. Zhang Yu-Dong and Senjyu (Ed.), *Smart Trends in Computing and Communications* (pp. 209–217). Springer Singapore.

Cheval, V., Cortier, V., & Turuani, M. (2018). A little more conversation, a little less action, a lot more satisfaction: Global states in ProVerif. *Proceedings - IEEE Computer Security Foundations Symposium*, *2018-July*. https://doi.org/10.1109/CSF.2018.00032

Cheval, V., Kremer, S., & Rakotonirina, I. (2018). DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice. *Proceedings - IEEE Symposium on Security and Privacy*, *2018-May*. https://doi.org/10.1109/SP.2018.00033

Cledou, G., Edixhoven, L., Jongmans, S.-S., & Proença, J. (2022). API Generation for Multiparty Session Types, Revisited and Revised Using Scala 3. In K. Ali & J. Vitek (Eds.), *36th European Conference on Object-Oriented Programming (ECOOP 2022)* (Vol. 222, pp. 27:1–27:28). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ECOOP.2022.27

Cohn-Gordon, K., Cremers, C., & Garratt, L. (2016). On post-compromise security. *Proceedings - IEEE Computer Security Foundations Symposium*, *2016-August*. https://doi.org/10.1109/CSF.2016.19

Cortier, V., Rusinowitch, M., & Zălinescu, E. U. (2007). Relating two standard notions of secrecy. *Logical Methods in Computer Science*, *3*(3). https://doi.org/10.2168/LMCS-3(3:2)2007

Costa Seco, J., Debois, S., Hildebrandt, T., & Slaats, T. (2018). RESEDA: Declaring live event-driven computations as reactive semi-structured data. *Proceedings - 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference, EDOC 2018*. https://doi.org/10.1109/EDOC.2018.00020

Cutner, Z., Yoshida, N., & Vassor, M. (2022). Deadlock-free asynchronous message reordering in rust with multiparty session types. In J. Lee, K. Agrawal, & M. F. Spear (Eds.), *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022* (pp. 246–261). ACM. https://doi.org/10.1145/3503221.3508404

De Porre, K. ; F. C. ; G. B. E. (2023). VeriFx: Correct Replicated Data Types for the Masses. *37th European Conference on Object-Oriented Programming (ECOOP 2023)*.

Delaune, S., Ryan, M., & Smyth, B. (2008). Automatic verification of privacy properties in the applied pi calculus. *IFIP International Federation for Information Processing*, *263*. https://doi.org/10.1007/978-0-387-09428-1_17

Demangeon, R., & Honda, K. (2012). Nested Protocols in Session Types. In M. Koutny & I. Ulidowski (Eds.), *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings* (Vol. 7454, pp. 272–286). Springer. https://doi.org/10.1007/978-3-642-32940-1_20

Demangeon, R., Honda, K., Hu, R., Neykova, R., & Yoshida, N. (2015). Practical interruptible conversations: distributed dynamic verification with multiparty session types and Python. *Formal Methods Syst. Des.*, *46*(3), 197–225. https://doi.org/10.1007/s10703-014-0218-8

Denning, D. E. (1976). A Lattice Model of Secure Information Flow. *Communications of the ACM*, *19*(5), 236–243. https://doi.org/10.1145/360051.360056

Dolev, D., & Yao, A. C. (1983). On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, *29*(2). https://doi.org/10.1109/TIT.1983.1056650

Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., & Sheth, A. N. (2014). TaintDroid: An Information Flow Tracking System for Real-Time Privacy Monitoring on Smartphones. *Communications of the ACM*, *57*(3), 99–106. https://doi.org/10.1145/2494522

Feraudo, A., Yadav, P., Safronov, V., Popescu, D. A., Mortier, R., Wang, S., Bellavista, P., & Crowcroft, J. (2020). CoLearn: Enabling Federated Learning in MUD-Compliant IoT Edge Networks. *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 25–30. https://doi.org/10.1145/3378679.3394528

Fournet, L., Mödersheim, S., & Viganò, L. (2023). A Decision Procedure for Alpha-Beta Privacy for a Bounded Number of Transitions. In *2024 IEEE Computer Security Foundations Symposium*. IEEE Computer Society Press.

Galrinho, L., Seco, J. C., Debois, S., Hildebrandt, T., Norman, H., & Slaats, T. (2021). *ReGraDa: Reactive Graph Data* (pp. 188–205). https://doi.org/10.1007/978-3-030-78142-2_12

Geraldo, E. (2022). SNITCH: A Platform for Information Flow Control. *Integrated Formal Methods: 17th International Conference, IFM 2022, Lugano, Switzerland, June 7–10, 2022, Proceedings*, 365–368. https://doi.org/10.1007/978-3-031-07727-2_24

Geraldo, E., & Costa Seco, J. (2019). Snitch: Dynamic dependent information flow analysis for independent Java bytecode. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, *302*. https://doi.org/10.4204/EPTCS.302.2

Geraldo, E., Costa Seco, J., & Hildebrandt Thomas. (2023). Data-Dependent Confidentiality in DCR Graphs. *PPDP 2023: 25th International Symposium on Principles and Practice of Declarative Programming*.

Geraldo, E. M. P. C. R. (2018). *SNITCH: Dependent Dynamic Information Flow Analysis on Intermediate Java Code*. NOVA School of Science & Technology.

Geraldo, E., Santos, J. F., & Costa Seco, J. (2021). Hybrid Information Flow Control for Low-Level Code. In R. Calinescu & C. S. Pasareanu (Eds.), *Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings* (Vol. 13085, pp. 141–159). Springer. https://doi.org/10.1007/978-3-030-92124-8_9

Geraldo, E., Santos, J. F., & Costa~Seco, J. (2021). Hybrid Information Flow Control for Low-Level Code. In R. Calinescu & C. S. Pasareanu (Eds.), *Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings* (Vol. 13085, pp. 141–159). Springer. https://doi.org/10.1007/978-3-030-92124-8_9

Giunti, M., Paulino, H., & Ravara, A. (2023). Anticipation of Method Execution in Mixed Consistency Systems. *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 1394–1401. https://doi.org/10.1145/3555776.3577725

Gondron, S., & Mödersheim, S. (2021). Vertical Composition and Sound Payload Abstraction for Stateful Protocols. *Proceedings - IEEE Computer Security Foundations Symposium*, *2021-June*. https://doi.org/10.1109/CSF51468.2021.00038

Gotsman, A., Yang, H., Ferreira, C., Najafzadeh, M., & Shapiro, M. (2016). *'Cause I'm Strong Enough: Reasoning about Consistency Choices in Distributed Systems*. https://doi.org/10.1145/2837614.2837625

Groote, J. F., & Mousavi, M. R. (2014). *Modeling and Analysis of Communicating Systems*. MIT Press. https://mitpress.mit.edu/books/modeling-and-analysis-communicating-systems

Hess, A. V., MÖdersheim, S. A., & Brucker, A. D. (2023). Stateful Protocol Composition in Isabelle/HOL. *ACM Transactions on Privacy and Security*, *26*(3). https://doi.org/10.1145/3577020

Hess, A. V., Mödersheim, S., Brucker, A. D., & Schlichtkrull, A. (2021). Performing Security Proofs of Stateful Protocols. *Proceedings - IEEE Computer Security Foundations Symposium*, *2021-June*. https://doi.org/10.1109/CSF51468.2021.00006

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Honda, K., Vasconcelos, V. T., & Kubo, M. (1998). Language Primitives and Type Discipline for Structured Communication-Based Programming. In C. Hankin (Ed.), *Programming*

Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings (Vol. 1381, pp. 122–138). Springer. https://doi.org/10.1007/BFb0053567

Honda, K., Vasconcelos, V. T., & Yoshida, N. (2000). Secure Information Flow as Typed Process Behaviour. In G. Smolka (Ed.), *Programming Languages and Systems, 9th European Symposium on Programming, ESOP 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings* (Vol. 1782, pp. 180–199). Springer. https://doi.org/10.1007/3-540-46425-5_12

Honda, K., Yoshida, N., & Carbone, M. (2008). Multiparty Asynchronous Session Types. *35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 273–284. https://doi.org/10.1145/1328897.1328472

Honda, K., Yoshida, N., & Carbone, M. (2016). Multiparty Asynchronous Session Types. *Journal of the ACM*, *63*(1–9), 1–67. https://doi.org/10.1145/2827695

Horne, R. (2020). Session Subtyping and Multiparty Compatibility Using Circular Sequents. In I. Konnov & L. Kovács (Eds.), *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)* (Vol. 171, pp. 12:1–12:22). Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.CONCUR.2020.12

Houshmand, F. (2019). Hamsaz: Replication Coordination Analysis and Synthesis *. *ACM Program. Lang. 3, POPL, Article*, *74*, 32. https://doi.org/10.1145/3290387

Hu, R., & Yoshida, N. (2016). Hybrid Session Verification Through Endpoint API Generation. In P. Stevens & A. Wasowski (Eds.), *Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings* (Vol. 9633, pp. 401–418). Springer. https://doi.org/10.1007/978-3-662-49665-7_24

Hu, R., & Yoshida, N. (2017). Explicit Connection Actions in Multiparty Session Types. In M. Huisman & J. Rubin (Eds.), *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings* (Vol. 10202, pp. 116–133). Springer. https://doi.org/10.1007/978-3-662-54494-5_7

Hüttel, H., Lanese, I., Vasconcelos, V. T., Caires, L., Carbone, M., Deniélou, P.-M., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H. T., & Zavattaro, G. (2016). Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.*, *49*(1), 3:1–3:36. https://doi.org/10.1145/2873052

Jacomme, C., & Kremer, S. (2018). An extensive formal analysis of multi-factor authentication protocols. *Proceedings - IEEE Computer Security Foundations Symposium*, *2018-July*. https://doi.org/10.1109/CSF.2018.00008

Kholod, I., Yanaki, E., Fomichev, D., Shalugin, E., Novikova, E., Filippov, E., & Nordlund, M. (2021). Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis. *Sensors*, *21*(1), 167. https://doi.org/10.3390/s21010167

Kobayashi, N. (2005). Type-based information flow analysis for the pi-calculus. *Acta Informatica*, *42*(4–5), 291–347. https://doi.org/10.1007/s00236-005-0179-x

Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2017). *Federated Learning: Strategies for Improving Communication Efficiency*. http://arxiv.org/abs/1610.05492

Kuhn, R., Melgratti, H. C., & Tuosto, E. (2023). Behavioural Types for Local-First Software. In K. Ali & G. Salvaneschi (Eds.), *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States* (Vol. 263, pp. 15:1–15:28). Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ECOOP.2023.15

Kunnemann, R., Esiyok, I., & Backes, M. (2019). Automated verification of accountability in security protocols. *Proceedings - IEEE Computer Security Foundations Symposium*, *2019-June*. https://doi.org/10.1109/CSF.2019.00034

Küsters, R., Truderung, T., & Vogt, A. (2010). Accountability: Definition and relationship to verifiability. *Proceedings of the ACM Conference on Computer and Communications Security*. https://doi.org/10.1145/1866307.1866366

Lehmann, A., & Fahland, D. (2012). Information Flow Security for Business Process Models - just one click away. *Proceedings of the 10th International Conference on Business Process Management - Demo Track (BPM 2012)*.

Li, X., Houshmand, F., & Lesani, M. (2020). Hampa: Solver-Aided Recency-Aware Replication. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12224 LNCS*. https://doi.org/10.1007/978-3-030-53288-8_16

Lourenço, L., & Caires, L. (2015). Dependent Information Flow Types. *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 317–328. https://doi.org/10.1145/2676726.2676994

McMahan, B. (n.d.). *"'Federated Learning from Research to Practice'", a presentation hosted by Carnegie Mellon University seminar series*. https://www.pdl.cmu.edu/SDI/2019/slides/2019-09-05Federated%20Learning.pdf

McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. In A. Singh & X. (Jerry) Zhu (Eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA* (Vol. 54, pp. 1273–1282). PMLR. http://proceedings.mlr.press/v54/mcmahan17a.html

Miu, A., Ferreira, F., Yoshida, N., & Zhou, F. (2021). Communication-safe web programming in TypeScript with routed multiparty session types. In A. Smith, D. Demange, & R. Gupta

(Eds.), *CC '21: 30th ACM SIGPLAN International Conference on Compiler Construction, Virtual Event, Republic of Korea, March 2-3, 2021* (pp. 94–106). ACM. https://doi.org/10.1145/3446804.3446854

Mödersheim, S., & Bruni, A. (2016). AIF-ω: Set-based protocol abstraction with countable families. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9635*. https://doi.org/10.1007/978-3-662-49635-0_12

Mödersheim, S., & Cuellar, J. (2021). Three Branches of Accountability. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 13066 LNCS*. https://doi.org/10.1007/978-3-030-91631-2_16

Mödersheim, S., & Viganò, L. (2009). The open-source fixed-point model checker for symbolic analysis of security protocols. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5705 LNCS*. https://doi.org/10.1007/978-3-642-03829-7_6

Mödersheim, S., & Viganò, L. (2019). Alpha-Beta Privacy. *ACM Trans. Priv. Secur.*, *22*(1). https://doi.org/10.1145/3289255

Myers, A. C., & Liskov, B. (2000). Protecting Privacy Using the Decentralized Label Model. *ACM Trans. Softw. Eng. Methodol.*, *9*(4), 410–442. https://doi.org/10.1145/363516.363526

Paulino, H. and A. M. A. and C. J. and G. M. and M. J. and R. A. (2023). AtomiS: Data-Centric Synchronization Made Practical. *ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)* .

Paulson, L. C. (1998). Inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, *6*(1–2). https://doi.org/10.3233/JCS-1998-61-205

Perino, D., Katevas, K., Lutu, A., Marin, E., & Kourtellis, N. (2022). Privacy-preserving AI for future networks. *Commun. ACM*, *65*(4), 52–53. https://doi.org/10.1145/3512343

Popovic, M., Popovic, M., Kastelan, I., Djukic, M., & Ghilezan, S. (2023). A Simple Python Testbed for Federated Learning Algorithms. *2023 Zooming Innovation in Consumer Technologies Conference (ZINC)*, 148–153. https://doi.org/10.1109/ZINC58345.2023.10173859

Privacy-Preserving Artificial Intelligence to advance humanity. (n.d.). https://sherpa.ai

Prokic, I., Ghilezan, S., Kasterovic, S., Popovic, M., Popovic, M., & Kastelan, I. (2023). Correct orchestration of Federated Learning generic algorithms: formalisation and verification in CSP. *CoRR*, *abs/2306.14529*. https://doi.org/10.48550/arXiv.2306.14529

Ribeiro, M., Singh, S., & Guestrin, C. (2016). ``Why Should I Trust You?'': Explaining the Predictions of Any Classifier. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 97–101. https://doi.org/10.18653/v1/N16-3020

Rocha, P., & Caires, L. (2021). Propositions-as-types and shared state. *Proceedings of the ACM on Programming Languages*, *5*(ICFP), 1–30. https://doi.org/10.1145/3473584

Rocha, P., & Caires, L. (2023). *Safe Session-Based Concurrency with Shared Linear State* (pp. 421–450). https://doi.org/10.1007/978-3-031-30044-8_16

Santos, J. F., Maksimovic, P., Naudziuniene, D., Wood, T., & Gardner, P. (2018). JaVerT: JavaScript verification toolchain. *Proceedings of the ACM Programming Languages*, *2*(POPL), 50:1–50:33. https://doi.org/10.1145/3158138

Scalas, A., & Yoshida, N. (2019). Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, *3*(POPL), 30:1–30:29. https://doi.org/10.1145/3290343

Scalas, A., Yoshida, N., & Benussi, E. (2019). Verifying message-passing programs with dependent behavioural types. In K. S. McKinley & K. Fisher (Eds.), *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019* (pp. 502–516). ACM. https://doi.org/10.1145/3314221.3322484

Simic, M., Prokic, I., Dedeic, J., Sladic, G., & Milosavljevic, B. (2021). Towards Edge Computing as a Service: Dynamic Formation of the Micro Data-Centers. *IEEE Access*, *9*, 114468–114484. https://doi.org/10.1109/ACCESS.2021.3104475

Simonet, V. (2003). Flow Caml in a Nutshell. In G. Hutton (Ed.), *Proceedings of the first APPSEM-II workshop* (pp. 152–165).

Snelting, G., Giffhorn, D., Graf, J., Hammer, C., Hecker, M., Mohr, M., & Wasserrab, D. (2014). Checking probabilistic noninterference using JOANA. *It Inf. Technol.*, *56*(6), 280–287. http://www.degruyter.com/view/j/itit.2014.56.issue-6/itit-2014-1051/itit-2014-1051.xml

Södergård, C., Tuikka, T., & et al. (2020). *Strategic Research, Innovation and Deployment Agenda: AI, Data and Robotics Partnership* (S. Zillner, D. Bisset, M. Milano, & E. Curry, Eds.; 3rd ed.).

Strom, R. E., & Yemini, S. (1986a). Typestate: A Programming Language Concept for Enhancing Software Reliability. *IEEE Trans. Software Eng.*, *12*(1), 157–171. https://doi.org/10.1109/TSE.1986.6312929

Strom, R. E., & Yemini, S. (1986b). Typestate: A Programming Language Concept for Enhancing Software Reliability. *IEEE Trans. Software Eng.*, *12*(1), 157–171. https://doi.org/10.1109/TSE.1986.6312929

Sun, J., Liu, Y., Dong, J. S., & Pang, J. (2009). PAT: Towards Flexible Verification under Fairness. In A. Bouajjani & O. Maler (Eds.), *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings* (Vol. 5643, pp. 709–714). Springer. https://doi.org/10.1007/978-3-642-02658-4_59

TensorFlow Federated: Machine Learning on Decentralized Data. (n.d.). https://www.tensorflow.org/federated

Toro, M., Garcia, R., & Tanter, É. (2018). Type-Driven Gradual Security with References. *Type-Driven Gradual Security with References. ACM Transactions on Programming Languages and Systems*, *40*(4), 1–55. https://doi.org/10.1145/3229061ï

van der Aalst, W. M. P., Artale, A., Montali, M., & Tritini, S. (2017). Object-Centric Behavioral Constraints: Integrating Data and Declarative Process Modelling. In A. Artale, B. Glimm, & R. Kontchakov (Eds.), *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017* (Vol. 1879). CEUR-WS.org. http://ceur-ws.org/Vol-1879/paper51.pdf

Viering, M., Hu, R., Eugster, P., & Ziarek, L. (2021). A multiparty session typing discipline for fault-tolerant event-driven distributed programming. *Proc. ACM Program. Lang.*, *5*(OOPSLA), 1–30. https://doi.org/10.1145/3485501

Volpano, D. M., Irvine, C. E., & Smith, G. (1996). A Sound Type System for Secure Flow Analysis. *J. Comput. Secur.*, *4*(2/3), 167–188. https://doi.org/10.3233/JCS-1996-42-304

Ying, B., Yuan, K., Chen, Y., Hu, H., Pan, P., & Yin, W. (2021). *Exponential Graph is Provably Efficient for Decentralized Deep Training*.

Ying, B., Yuan, K., Hu, H., Chen, Y., & Yin, W. (2021). BlueFog: Make Decentralized Algorithms Practical for Optimization and Deep Learning. *CoRR*, *abs/2111.04287*. https://arxiv.org/abs/2111.04287

Yoshida, N., Hu, R., Neykova, R., & Ng, N. (2013). The Scribble Protocol Language. In M. Abadi & A. Lluch-Lafuente (Eds.), *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers* (Vol. 8358, pp. 22–41). Springer. https://doi.org/10.1007/978-3-319-05119-2_3

Yoshida, N., Zhou, F., & Ferreira, F. (2021). Communicating Finite State Machines and an Extensible Toolchain for Multiparty Session Types. In E. Bampis & A. Pagourtzis (Eds.), *Fundamentals of Computation Theory - 23rd International Symposium, FCT 2021, Athens, Greece, September 12-15, 2021, Proceedings* (Vol. 12867, pp. 18–35). Springer. https://doi.org/10.1007/978-3-030-86593-1_2

Zhang, T., He, C., Ma, T., Gao, L., Ma, M., & Avestimehr, S. (2021). Federated Learning for Internet of Things. *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 413–419. https://doi.org/10.1145/3485730.3493444