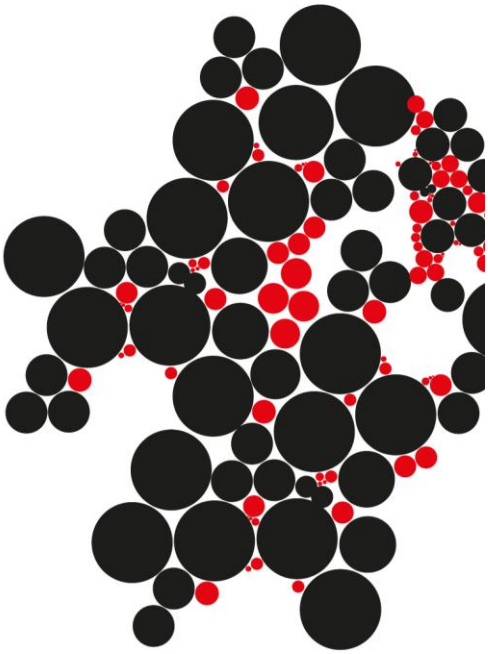
MASTER THESIS



CLAIMING SECURITY PROPERTIES IN A SERVICE ORIENTED ARCHITECTURE

S.L.C. Verberkt

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE
PROF. DR. IR. A.J. MOUTHAAN

EXAMINATION COMMITTEE
prof. dr. D. Pavlovic
prof. dr. S. Etalle
prof. dr. J.L. van den Berg
dr. W.A. Bruls
drs. ing. G.E.A. Smit

UNIVERSITY OF TWENTE.

23-11-2012

IBM Nederland
P.O. Box 9999
1006 CE Amsterdam

University of Twente
Faculty of EEMCS
P.O. Box 217
7500 AE Enschede

Thesis to obtain the degree of Master of Science

Claiming Security Properties in a Service Oriented Architecture

S.L.C. Verberkt[I]

23th November 2012

Graduation committee:
prof. dr. D. Pavlovic
prof. dr. S. Etalle
prof. dr. J.L. van den Berg (TNO)
dr. W.A. Bruls (IBM Nederland)
drs. ing. G.E.A. Smit (IBM Nederland)

[I] s.l.c.verberkt@student.utwente.nl

Thesis to obtain the degree of *Master of Science* on *23th November 2012* at the *Faculty of Electrical Engineering, Mathematics and Computer Science* of the *University of Twente* by *S.L.C. Verberkt* born on *10th December 1988* in *Woerden* under the supervision of the graduation committee consisting of *prof. dr. D. Pavlovic* (first supervisor), *prof. dr. S. Etalle*, *prof. dr. J.L. van den Berg*, *dr. W.A. Bruls* and *drs. ing. G.E.A. Smit*.

Acknowledgements

With the submission of this thesis, five years of education at the University of Twente come to an end. Being able to study at possibly the most wonderful university of the Netherlands has been a great opportunity and an amazing environment. It will be great honour to become an ambassador of this institution.

Writing this thesis and graduating today would not have been possible without the graduation committee. The author is very grateful for the supervision, advise, and knowledge given by prof. dr. Dusko Pavlovic. Furthermore, this gratefulness extends to the other members of the graduation committee, namely: prof. dr. Sandro Etalle, prof. dr. Hans van den Berg, dr. Wiel Bruls and drs. ing. Gerard Smit.

For the helpful comments, reviews and pointers, I would like to thank drs. ir. Lydia Duijvestijn, prof. dr. Rob van der Mei and ing. Jeroen Slobbe. Their reviews helped immensely with increasing the quality of the work.

I would like to thank friends and family for their support over the years. A special word of thanks is for my parents, who helped in opening a lot of doors that helped enabling my graduation today. To this, I want to add a small moment of silence for my grandfather, prof. dr. Arie Duijvestijn, who introduced me to the wonderful world of computer science at an early age. Finally, I will be forever thankful to Silke Kücking for keeping the promise she made in her thesis by being loving and supportive each day.

Abstract

The rise of service oriented computing as a popular software development paradigm brings the inability to compose security properties back into the spotlights. Since the modular nature of services is one of the key concepts within service oriented architectures (SOAs), a lot of security efforts in this field are rendered useless. This is amplified by the fact that services are built upon an uncoupled architecture.

Additionally, service oriented systems need to be able to bootstrap trust in real time, as another main feature of service oriented systems is the opaqueness of the services. As no prior communication or knowledge further than a standardised application programming interface (API) can be assumed due to the uncoupled nature of services, making an informed decision about the honesty of a specific service is very hard. Therefore, a trust infrastructure or other means of coping with dishonest services is mandatory.

The present research will tackle both the composition problem and the issue of trust within a service oriented context. It is aimed at enabling the techniques for proving security properties within standard SOAs and finding security properties for use within this architecture while posing only realistic assumptions on the service oriented environment. Furthermore, a concise and effective model for bootstrapping trust will be introduced.

A precise model for services and service compositions is built using partially ordered multisets (pomsets). This model conforms to the characteristics of service oriented computing, with an emphasis on the uncoupled nature and opaqueness of services.

The secure protocol composition framework of Datta et al. [18] is adapted to a service oriented context. This is done by overcoming the differences between service oriented computing and protocols, and combining the framework with our model of service oriented computing. The resulting framework solves the problem of secure service composition.

Finally, certification and reputation-based trust infrastructures are discussed and proposed to solve the issue of trust raised by opaque services. By using trust infrastructures to bootstrap trust in the honesty of individual services, it becomes possible to make informed decisions on which services to trust. As means of mitigating the issue of trust against a trustworthy majority, the problem of the Byzantine generals of Lamport, Shostak and Pease [42] is discussed and adapted to our model of service compositions. This approach minimises the effect of dishonest services by normalising anomalies caused by service poisoning.

In conclusion, the present research proposes a framework for the secure composition of services. Additionally, trust infrastructures are proposed to decide upon the honesty of services, and an adaptation of the Byzantine generals problem is used to mitigate service poisoning.

Keywords

service oriented computing, security properties, composition, uncoupled services, opaque services, trust infrastructures, Byzantine generals problem, service poisoning

Contents

Abstract III

List of Figures VII

List of Tables IX

List of Theorems XI

1 Introduction 1
1.1 Organisation . . . 2

2 Services and Service Oriented Computing 5
2.1 Service Composition . . . 7
2.2 Partially Ordered Multisets . . . 12
2.3 Modelling Service Oriented Systems . . . 19
2.4 Differences from Protocols . . . 21

3 Secure Composition 25
3.1 Secure Composition of Protocols . . . 27
3.1.1 Cord Calculus . . . 27
3.1.2 Protocol Logic . . . 33
3.1.3 Proof System . . . 39
3.2 Secure Composition of Services . . . 46
3.3 Security Properties and Predicates . . . 48
3.4 Example Cases . . . 54
3.4.1 Sending Secrets . . . 55
3.4.2 Authenticating Services . . . 57
3.5 On Adversaries . . . 60

4 The Need for Trust 63
4.1 Implementing Trust . . . 63
4.1.1 Certification . . . 64
4.1.2 Reputation . . . 66
4.2 High Frequency Trading . . . 68
4.3 The Byzantine Generals . . . 70
4.3.1 Oral Messages . . . 71
4.3.2 Signed Messages . . . 76
4.3.3 Application to Service Oriented Computing . . . 78

5 Related Work **81**
5.1 Security in Service Oriented Computing 81
5.2 Secure Composition 82
5.3 Trust Infrastructures 82

6 Conclusion and Discussion **83**
6.1 Future Research 85

Bibliography **1-1**

Acronyms **2-1**

List of Figures

2.1 Services for a personal scientific papers repository 6
2.2 Example of a service composition for a news site 8
2.3 Example of a dancing steps diagram 9
2.4 Example of a service choreography for a news site 10
2.5 Example of a service orchestration for a news site 11
2.6 Example of a labelled partial order 13
2.7 A labelled partial order that is isomorphic with Figure 2.6 13
2.8 A labelled partial order that is homomorphic with Figure 2.6 14
2.9 Example of a pomset 15
2.10 The pomset for $g(c)$ 16
2.11 The mapping function g applied to the pomset of Figure 2.9 16
2.12 The sequential composition of the pomset of Figure 2.9 with itself . . . 17
2.13 The parallel composition of the pomset of Figure 2.9 with itself 18
2.14 Pomset for Figure 2.2 21
2.15 Example of a protocol run 22

3.1 Example of a service composition to illustrate non-interference 50
3.2 Example of a service composition for a travel agency to illustrate non-interference 51
3.3 Example of a service composition to illustrate invocation 54
3.4 A secure echo service 55
3.5 The secure echo service of Figure 3.4 in a composition 56
3.6 The secure echo service of Figure 3.4 in an insecure composition 56
3.7 A service composition for authentication 58

4.1 A hierarchical public key infrastructure 66
4.2 A reputation infrastructure 67
4.3 Example of a service layout for a high frequency trading system 69
4.4 Problem of the Byzantine commander with 3 participants and a dishonest lieutenant [42] 72
4.5 Problem of the Byzantine commander with 3 participants and a dishonest general [42] 72
4.6 The result of $OralMessage(1)$ from the viewpoint of the second lieutenant with a dishonest lieutenant [42] 74
4.7 The result of $OralMessage(1)$ with a dishonest general [42] 74
4.8 The result of $SignedMessage(1)$ with a dishonest general [42] 78
4.9 The example of Figure 4.3 with a rogue service 79

List of Tables

2.1 The values for μ of the labelled partial order of Figure 2.6 13
2.2 The map by which the labelled partial order of Figure 2.6 is isomorphic to Figure 2.7 14
2.3 The map by which the labelled partial order of Figure 2.6 is homomorphic to Figure 2.8 14

List of Theorems

2.1 Definition (Labelled partial order [68, 27, 65]) . . . 12
2.2 Definition (Homomorphism of labelled partial orders [27]) . . . 12
2.3 Definition (Partially ordered multiset [27, 65]) . . . 14
2.4 Definition (Substitution of partially ordered multisets [27]) . . . 14
2.5 Definition (Sequential composition of partially ordered multisets [27]) 15
2.6 Definition (Parallel composition of partially ordered multisets [27]) . . 17
2.7 Definition (Process [64]) . . . 19
2.8 Definition (Sequential composition of processes [27]) . . . 19
2.9 Definition (Parallel composition of processes [27]) . . . 19
2.10 Definition (Addition of processes [27]) . . . 19

3.1 Definition (Processes [18]) . . . 26
3.2 Definition (Process Triple [18, 17]) . . . 26
3.3 Definition (Process Soundness [18, 17]) . . . 26
3.4 Definition (Free variables [23]) . . . 31
3.5 Definition (Bound variables [23]) . . . 32
3.1 Theorem (Crosstalk [23]) . . . 32
3.2 Theorem (Basic reaction steps for cord spaces [18]) . . . 33
3.6 Definition (After [18]) . . . 35
3.3 Theorem (General composition methodology [18]) . . . 45
3.4 Theorem (Parallel Composition [17]) . . . 46
3.5 Theorem (Sequential Composition [17]) . . . 47
3.6 Theorem (Service Composition) . . . 47
3.7 Definition (Has [18]) . . . 49
3.8 Definition (Fresh [18]) . . . 49
3.9 Definition (Honesty [18]) . . . 49
3.10 Definition (Secret terms) . . . 51
3.11 Definition (Public terms) . . . 52

4.1 Definition (Trust and distrust [71]) . . . 63
4.2 Definition (Trust predicate) . . . 64
4.1 Theorem (Oral message algorithm [42]) . . . 73
4.2 Theorem ($OralMessage(m)$ with loyal general [42]) . . . 75
4.3 Theorem ($OralMessage(m)$ [42]) . . . 76
4.4 Theorem (Signed message algorithm [42]) . . . 76
4.5 Theorem ($SignedMessage(m)$ [42]) . . . 78

1 Introduction

Since its introduction, service oriented computing has become a major technique within the field of computer science [76]. In a service oriented system, entities, either automated or human agents, expose themselves as services that can be invoked by other entities, including other services [54]. Of these types of services, this research mainly concerns fully automated services, although the research could be extended to include human agents. In a networked world like the Internet, this results in multitudes of uncoupled services. In Chapter 2 we will discuss the concept of services and their characteristics in depth.

Although service oriented computing is popular, the trustworthiness and security of it are still subject to a lot of questions and require thorough research [55]. The security aspects of service oriented architectures (SOAs) require full validation to be able to guarantee entry-point to end-point security.

Services are characterised by several properties, of which their uncoupled nature and composability stand out [71, 54, 24]. Uncoupled, when referring to services, means that no prior contact between or prior knowledge about other services can be assumed. Therefore, for a responsible usage of services, trust needs to be bootstrapped or security needs to be proven.

Additionally, the uncoupled nature means that services can be opaque, which makes it virtually impossible to prove security. This is due to the fact that obfuscation is considered impossible [8], which makes it impossible to check any proof without giving away the code, and the inability to prove that the code that has been validated is indeed the code that is running. This is why a SOA commonly requires a trust architecture to cope with opaque services.

A major problem when trying to prove security of web services is the fact that security properties generally do not hold under composition or refinement [48, 52, 35, 73], as web services are meant for composition [54]. Composition of services means that atomic services are used to build larger systems. These small services are composed to larger services by using several standard technologies for the composition of and the interfacing between services. However, their uncoupled nature makes proving security properties a difficult or impossible task, as those properties commonly do not hold under composition or refinement.

Furthermore, common proofs of security properties depend on the model in use and the possibly implicit assumptions made [57]. In order to advertise provable trustworthy web services, it is important to overcome these problems, as otherwise any security claim would be useless as soon as a web service is chosen to be part of a system.

The problem of composition has already been subject to research [18, 17]. This resulted in a solution based on annotating protocols using the triples of Hoare [32], which

enables secure composition of protocols. In our research, this approach by Datta et al. [18] will be used as a starting point for solving the stated problem. In Chapter 3, we will discuss this method for secure composition in depth.

It is important to see why services differ from protocols, as this shows why the present paper aims at a new goal and does not merely recap the previous work of Datta et al. [18], which concerns the secure composition of protocols. Most importantly, where protocols are built for a certain task, services are publicly exposed agents – either human or software – that may interact. Thus, they have more capabilities than the participants in a single protocol run. Actually, they go literally beyond protocols, as the communication between services makes use of standardised protocols.

Given the stated findings, it can be concluded that we are in need of a way to prove security properties for web services that hold under composition given realistic assumptions. In order to do this, we will combine current standardised service oriented computing techniques with the knowledge of proving security properties in modular systems or protocols. Therefore, the problem statement of the present research is:

Problem Statement. How can we guarantee security properties in a service oriented architecture, thereby taking into account the characteristics of service oriented computing in general and the uncoupled and opaque characteristics specifically?

1.1 Organisation

To find an answer to this problem statement, we will split it up in a number of research questions, which we will discuss and answer in this paper. We will start with a discussion of service oriented computing in Chapter 2. After discussing services and their characteristics, we will touch upon the mechanisms and concepts of service composition in Section 2.1. Furthermore we will discuss modelling the services within our context by introducing Research Question 1, which is considered in Section 2.3. This model makes use of so-called partially ordered multisets (pomsets), which are extensively explained in Section 2.2. Finally, we ask ourselves why past solutions built around protocols are not directly applicable with Research Question 2 in Section 2.4.

Research Question 1. How can we model services such that their key characteristics, namely uncoupled, composable, and opaqueness, and their distinction from protocols and other modular constructs are honoured?

Research Question 2. On what characteristics differ services from protocols such that the work on secure composition of protocols by Datta et al. [18] is not directly applicable to services?

After pursuing the thread of service orientation, we shift our focus to secure composition in Chapter 3. Here, we will discuss the groundwork for secure composition based on the work of Datta et al. [18], which is discussed in Section 3.1. This framework for secure composition leads to two problems when we try to apply it to service oriented computing. These problems are formulated in the form of Research Question 3 and Research Question 4. Our adaptation of the model of Datta et al. [18] is discussed

in Section 3.2 and equipped with additional predicates in Section 3.3. Additionally, we will discuss example cases of the secure composition of services in Section 3.4. After discussing this formal model, we lay out an attacker model based on Research Question 5 in Section 3.5. To security research, the existence of an attacker model is mandatory, as anything is secure if the attacker is non-existent.

Research Question 3. How can limitations raised by opaque services when applying the secure composition technique of Datta et al. [17] be solved?

Research Question 4. How can the state explosion when applying the secure composition technique of Datta et al. [18] be prevented?

Research Question 5. What attacker model do we need to reflect all the challenges specific to the secure composition of services?

Finally, we will discuss trust and the need for it in Chapter 4. The issue of trust is discussed as the security claims concerning opaque services cannot always be validated first hand, which constitutes a problem, as insecure services cannot be composed securely. Trust can be used to evaluate whether those security claims should be accepted or not. The central problems in this chapter are formulated by Research Question 6 and Research Question 7. The former question, which is concerned in Section 4.1, discusses how trust could be incorporated in the proposed model of service oriented computing. The latter question ask how the issue of trust could be mitigated by relying on the trustworthiness of the majority. This concept is introduced in Section 4.2 using an example based on high frequency trading and brought to existence in Section 4.3 based on the problem of the Byzantine generals – which is also explained in that section.

Research Question 6. How can we implement trust in our model, thereby keeping in mind the paradox of trust of Pavlovic [56]?

Research Question 7. How can we apply solutions for the problem of the Byzantine generals of Lamport, Shostak and Pease [42] to compensate for dishonest participants?

The related work is mentioned in Chapter 5. In Chapter 6, all research questions will briefly be discussed, thereby giving an easy overview of all the given answers to these questions including the locations where these answers have been provided. This results in the main contribution, i.e. the answer to the problem statement. Finally, the open threads are discussed as pointers for future research in Section 6.1.

2 Services and Service Oriented Computing

The concept of services originates from the field of marketing, but it became later on a popular concept in the world of computer science [71]. MacKenzie et al. [45] define a service as "a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description". In other words, services provide a uniform interface to capabilities exposed by human or automated agents. It should be noted that a web service is the same thing with the difference that it is accessible over the Internet, commonly using an uniform resource locator (URL) that points to the service [54].

To illustrate this, suppose that you have a website that you use to distribute your scientific papers. Traditionally, someone goes to that website, clicks around and reads what he finds. Now, if this website were to be converted into a service oriented system, one would build services for all atomic tasks, such as: searching a paper, retrieving a paper, and listing all papers. In this case, we consider any task that cannot be cut into smaller pieces without resembling the original task as an atomic task. Please note that this differs from atomic operations: listing all papers is an atomic task, although this may consist of multiple atomic operations, e.g. merely getting a list of all papers from the database is not a task that can be delivered to an end user.

The services that are built to deliver the atomic tasks will make use of standardised interfaces such that they are accessible by any person or computer using this interface. This way, someone could, for example, build another service which uses your services to create an aggregated list of all scientific papers by all experts on a certain subject.

The services for the discussed paper repository have been illustrated in Figure 2.1. Services will henceforth be illustrated using the visualizations in this figure, i.e. services are drawn as circles and the names of the services are written under these circles. Additionally, inside the circle, a variable for usage in formulas may be written, e.g. $S_1 = search\_papers$. Please note that these services are not ordered or coupled in any way.

Characteristics Services are commonly classified as being uncoupled, abstracted, persistent, autonomous, granular, stateless, discoverable and composable [71]. We will discuss these characteristics to gain a more detailed view on what services are. This way, a step further towards a general understanding of service orientation is made.

In related work, the term loosely coupled is often used instead of the term un-

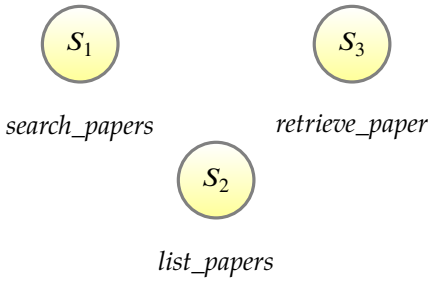


Figure 2.1: Services for a personal scientific papers repository

coupled to describe the coupling properties of services [e.g. 24, 54]. Uncoupled means that no prior contact between or prior knowledge about other services can be assumed. Loose coupling hints at the fact that, in practice, there are sometimes dependencies between services, although these should be minimised [71]. However, as the notion of loose coupling also allows uncoupled services, it is considered too ambiguous for the context of this research. This is due to the fact that loose coupling may allow a certain amount of prior knowledge when this is considered inevitable in the context of the specific information system at hand. However, given the general level of reasoning that we use, it is not possible to formalise this notion without gravely complicating the model, which would result in an unusable model of service orientation. Therefore, we will keep strictly to uncoupled services instead of loosely coupled services.

To illustrate the concept of service orientation, we could have a service that gives information on the weather or one that allows you to search a database of books. These services would be deployed somewhere, in order to be accessible by other services or potential users. Services can be regarded as software that can be invoked over a network. For example, we can define a program that will calculate the sum, e.g. *calculate_sum*. One can imagine that if this service is invoked with 2 and 4 as parameters, the result would be 6.

To further discuss the common characteristics of services, we will quickly explain them. To start, abstraction means that services only advertise essential information and abstract themselves from any implementation-related or comparable details [71]. For example, a traditional system would mention that it runs on and requires a certain operating system, e.g. Microsoft Windows, whereas a service oriented system simply does not care about those details and is abstracted from them. Please note that a service provider has to be aware of the details of the implementation for obvious reasons, but that he does not share those details in the context of the service oriented architecture (SOA). Additionally, the granularity ensures that services are not only abstracted from the details, but are also placed on the highest level possible, thereby even more simplifying the development process.

The persistence of services means that they are reusable and will exist for a longer period of time [71]. Simply put, this means that a service is built with a more long-term view in mind. In combination with the characteristic of abstraction, this ensures that

a service is not built for use in a single system but for use by multiple parties. This also explains autonomy, as a service should be able to stand on its own feet during their lifetime.

The previously explained uncoupled nature directly explains why services should be stateless: no prior contact can be assumed [71]. If no prior knowledge or contact can be assumed, maintaining state simply does not make sense. This also means that if a certain service provider would offer multiple services, these still would not share state. In other words, if a service provider has two services, they should not share a certain parameter – please note that, although this may occur in practice, those examples cannot fully be deemed service oriented computing.

Additionally, it should be noted that if a service provider uses techniques for gaining high availability such as load balancing or mirroring, those techniques can be regarded as part of the internals of the service. For example, when you go to your local grocery store, you could regard that grocery store as a service for buying certain articles. However, within the shop, you may be able to choose between several payment counters. In other words, within the service a certain amount of load balancing is implemented, which requires coordination between the several counters, but not with other services.

Comparably, the order of invocation of services should not influence the outcome, except if the output of the first service is indirectly or directly used as input for the second service. Once again, this has to do with the statelessness of services: if there is no state, time- or order-based effects cannot be remembered.

Finally, being able to discover and compose services is what makes the system work. This means that one should be able to find existing services and use them to build complete systems using these services [71]. In our example of Figure 2.1, one could already see how this enables others to use those services to create an aggregation of scientific papers or other innovative applications with minimal effort.

The key characteristics of services that we will focus on in this research are the fact that they are uncoupled, that they are meant for composition and that services are generally opaque to other services. The uncoupled nature means that no prior communication between or knowledge about services can be assumed [54]. Services make use of (open) standards in their communication, which are assumed general knowledge and are not of interest for our research. In addition, opaqueness refers to the inability to look inside a service to see how it works or how it is built. This results in Research Question 1.

Research Question 1. How can we model services such that their key characteristics, namely uncoupled, composable, and opaqueness, and their distinction from protocols and other modular constructs are honoured?

2.1 Service Composition

As noted before, services rely heavily on the concept of composition. The general idea is that atomic services are aimed at delivering a certain capability. By composing

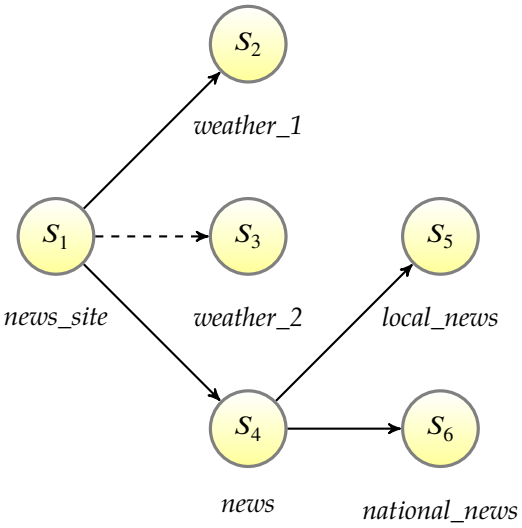


Figure 2.2: Example of a service composition for a news site

all these services a larger system can be built, which performs a complete process. Furthermore, if requirements change or certain services prove unreliable, the atomic services can easily be replaced by other services or the complete composition can be changed.

To illustrate the idea of service compositions, Figure 2.2 displays an example composition in which a news site is central. This site makes use of two weather services, such that either one can be used when the other is unavailable. Please note that, in the actual run shown in this figure, *weather_1* is used. Furthermore, the example shows that a chain could be built in which a used service also makes use of other services.

In practice, two types of service composition exist [62]. Namely, orchestration, which describes the way services interact on a message level, and choreography, which keeps track of the sequence of messages, which may involve multiple parties and sources [61]. Thus, where orchestration is a centralised approach in which the service chain is controlled from the viewpoint of one party, choreography is collaborative and lets all parties describe the part they play in the service composition [71].

Choreography To give a good distinction between the two composition mechanisms, imagine a dancing steps diagram. Those diagrams can be bought printed on a rug, such that one or more persons are able to follow the directions on the diagram. This means that the dancers place their feet on the printed feet on the rug and move them as the arrows point. An example of such a diagram for an imagined dance is shown in Figure 2.3.

Being a term from the world of dancing, one can understand that the diagram of dancing steps is an excellent example of a choreography. This choreography describes

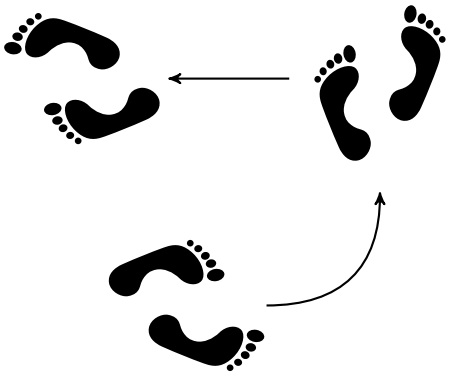


Figure 2.3: Example of a dancing steps diagram

how the dancers should move their feet in order to perform the dance together. In this diagram, the dance is not viewed through the eyes of one of the participants, but described in a global manner. Additionally, which specific services are used is not described either, only the roles that are required to perform the dance – e.g. if this were a tango, the choreography would simply require a leading and a following participant and does not define which specific persons it has in mind.

The example of the dance shows that a choreography simply happens, without any orchestrating role. Basically, the dancers enter the dancing floor where they will – try to – find one or more dancing partners to perform the choreography they have in mind. The choreography itself is, in this sense, an abstracted overview on how that dance will go.

To return to services, Figure 2.4 illustrates a choreography for the composition of Figure 2.2. As can be seen in this figure, the choreography describes how the different roles in the composition of a news portal website can and should interact. Basically, for a certain news site, a weather source and a news aggregator for news sources is required.

A choreography leaves room for alternatives, e.g. a different weather source can be used when the first choice fails. As we already saw in the composition of Figure 2.2, it is also possible to have multiple news sources. In the abstraction of Figure 2.4, this is not reflected. However, any news source would interact using the same rules with the news aggregation service, i.e. there is only one role for a news source service to assume.

Orchestration As opposed to choreography, service orchestration looks at to composition from a certain viewpoint. Where the metaphor for choreographies can be found on the dancing floor, we can illustrate orchestration by visiting a concert. Orchestration refers to how an orchestra works with a focus on the role of the conductor. Please note that this conductor is allowed to play an instrument himself.

Every musician in an orchestra knows his instrument and the music he should be

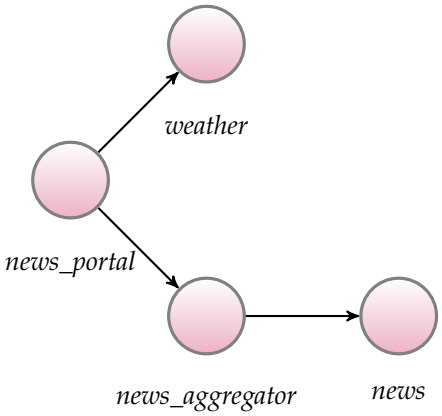


Figure 2.4: Example of a service choreography for a news site

playing. However, when and how he plays is controlled by the conductor who, from a central position, instructs all the musicians in the orchestra. Thus, orchestration has a central orchestrator – in this example the conductor.

To illustrate orchestration in a service oriented context, one could see the example of Figure 2.5 as a composition orchestrated from a central viewpoint [62] – namely, the viewpoint of *news_site*. Thus, in this orchestration, *news_site* takes the roll of central composer. Additionally, one could see the part of this figure containing the services *news*, *local_news* and *national_news* as a sub-composition orchestrated from the viewpoint of *news*.

Please note that, in practice, there are forms of orchestration that are implemented using a so-called orchestration engine [71]. This engine has the task of controlling the execution of the service composition. In other words, the service orchestration is composed and ran from the viewpoint of this engine. If we do not mention an orchestration engine explicitly, the role of the orchestration engine is also performed by the service from which viewpoint the orchestration is built.

Projection It should be noted that a choreography can easily be mapped to an orchestration by the process of projection [33]. This can be illustrated by the dancing example as follows: every dancer on the floor will take a certain role of the choreography and play his part in the dance from this viewpoint. Thus, an orchestration could be built by taking the leading roles and change their steps from the global view of the choreography to those he should perform from his own viewpoint.

To illustrate the act of projection, imagine the service choreography of Figure 2.4. When performing a run in accordance with this choreography, a number of services is able to fill in one of the roles in this dance. Those services are: *news_site*, *weather_1*, *weather_2*, *news*, *local_news* and *national_news*. Those can respectively fulfil the following roles in the choreography: *news_portal*, *weather* (twice), *news_aggregator* and *news*

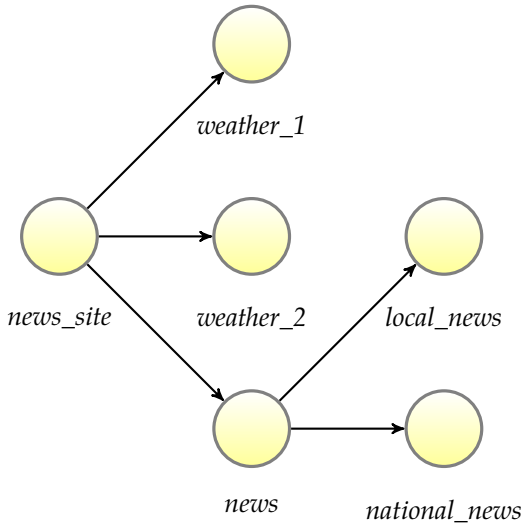


Figure 2.5: Example of a service orchestration for a news site

(twice).

Every service can lookup its role in the choreography and make this its viewpoint. For example, the service *news_site* sees that it takes the role of *news_portal*, which requires a *weather* service and a *news_aggregator*. With this information, the *news_site* service can build an orchestration in which it has both *weather_1* and *weather_2* as candidates for the role of *weather* and the service *news* for the position of *news_aggregator*. This way, an orchestration from the viewpoint of *news_site* has been built.

Additionally, the service *news* can take the central position in the orchestration from the viewpoint of *news_aggregator*. He will find that both *local_news* and *national_news* fit the role of *news*. Please note that this makes those alternatives and not both used as opposed to the orchestration of Figure 2.5 due to the generalised nature of the projection algorithm. If we wanted this effect, the original choreography should have reflected this distinction.

Runs When a service composition is executed, an actual run happens. As we already learned, compositions may offer alternatives, select based on the input or provide other control logic. A run reflects what is executed in one particular instance.

For example, Figure 2.2 shows, when one ignores the dashed lines, a run of the orchestration of Figure 2.4. In this run, the service *weather_1* has specifically been selected as an alternative over *weather_2*. However, in the composition both these services are still regarded as equal.

Specific runs are not of high interest to this research, as they represent unique instances of compositions that can be regarded as a whole. Compositions, on the other hand, can result in various runs, which makes it impossible to evaluate the security for

the complete system. Therefore, the security problems raised by composing services should be regarded from the level of compositions.

2.2 Partially Ordered Multisets

Following Research Question 1, we will build a model for service oriented computing, that enables us to pursue secure composition of services. In such a model, the uncoupled nature and the ability to compose services are very important, as they present the challenges for secure composition. Additionally, we will also note opaqueness as an important and challenging feature.

Labelled Partial Orders In this research, we will model services using partially ordered multisets (pomsets) [27, 65]. A pomset, as stated in definition 2.3, is the isomorphism class of a labelled partial order, which is defined in definition 2.1. Such a labelled partial order provides us with a set of vertices V that is partially ordered by $\leq$. Additionally, the labelling function μ assigns symbols from the alphabet Σ to the vertices $v \in V$. This makes it possible to have multiple occurrences of the same symbol, which can, for example, be used for multiple parallel executions of the same task.

Definition 2.1 (Labelled partial order [68, 27, 65])**.** A labelled partial order is a 4-tuple $(V, \Sigma, \leq, \mu)$ consisting of: a vertex set V; an alphabet Σ; a partial order $\leq$ on V which is represented as a binary relation in $V \times V$; and a labelling function $\mu : V \to \Sigma$ assigning symbols to vertices.

A labelled partial order is said to be homomorphic with another labelled partial order if and only if there is a mapping from the set of vertices to the set of vertices of the other labelled partial order that preserves labelling and order. This is formalised in definition 2.2. Additionally, following the definition of isomorphism, two labelled partial orders are said to be isomorphic if and only if there is a mapping between the two sets of vertices that preserves labelling and order [27]. In other words, two labelled partial orders are isomorph when either one is homomorphic with the other one.

Definition 2.2 (Homomorphism of labelled partial orders [27])**.** If we have two labelled partial orders $P = (V_P, \Sigma_P, \leq_P, \mu_P)$ and $Q = (V_Q, \Sigma_Q, \leq_Q, \mu_Q)$ and there is a mapping τ from P to Q such that Equation 2.1 and Equation 2.2 hold, τ is a homomorphism.

$$\forall v \in V_P : (\mu_P(v) = \mu_Q(\tau(v))) \tag{2.1}$$

$$\forall v, w \in V_P : (v \leq_P w \Rightarrow \tau(v) \leq_Q \tau(w)) \tag{2.2}$$

To give an example, if we have a labelled partial order with $V = \{A, B, C, D\}$, $\Sigma = \{a, b, c\}$, $A < B < C$ and μ as shown in Table 2.1, we could draw this as shown in Figure 2.6. In this figure, the vertexes are names A, B, C, and D, which is shown with

v	A	B	C	D
$\mu(v)$	a	c	b	c

Table 2.1: The values for μ of the labelled partial order of Figure 2.6

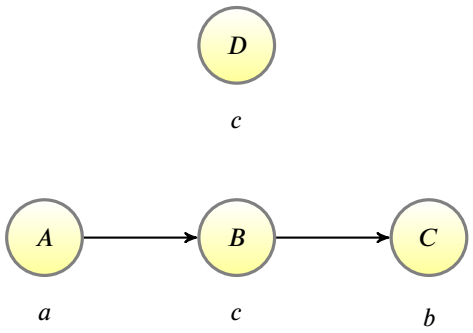


Figure 2.6: Example of a labelled partial order

the central captions on the nodes. Additionally, the nodes represent certain actions – namely, a, b, and c – that are added as caption beneath them and can occur multiple times, as shown by the double occurrence of c. Finally, a partial order is introduced where A comes before B, which comes before C, and D is executed somewhere in parallel with the sequence of A, B, and C.

One can easily see that the labelled partial order of Figure 2.6 is isomorphic with the one displayed in Figure 2.7 with the map displayed in Table 2.2. Comparably, Figure 2.8 is a homomorphism of this figure with the map displayed in Table 2.3.

Partially Ordered Multisets As stated before, a pomset is the isomorphism class of some labelled partial order. In other words, a pomset abstracts from the specific

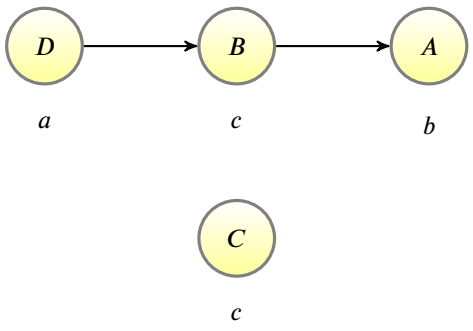


Figure 2.7: A labelled partial order that is isomorphic with Figure 2.6

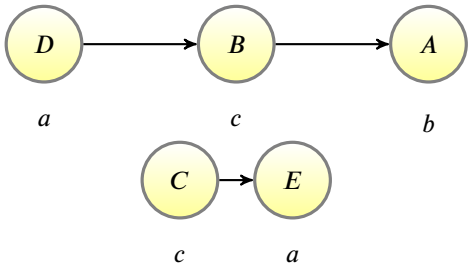


Figure 2.8: A labelled partial order that is homomorphic with Figure 2.6

v	A	B	C	D
$\tau(v)$	D	B	A	C

Table 2.2: The map by which the labelled partial order of Figure 2.6 is isomorphic to Figure 2.7

contents of the set of vertices and focusses on the structure. This has been formalised in Definition 2.3. Thus, if we were to abstract the pomset out of this labelled partial order, we would end up with Figure 2.9.

Definition 2.3 (Partially ordered multiset [27, 65])**.** A partially ordered multiset – or pomset – P is the isomorphism class of a labelled partial order and is denoted as $[V, \Sigma, \leq, \mu]$.

Pomsets do not require that the tasks are atomic, i.e. it is possible to look inside a task is to find another pomset [27]. Formally, this is captured by introducing substitution to the model. For this, we can have a function g that maps every a in the alphabet Σ to some pomset, which is captured by Definition 2.4.

Definition 2.4 (Substitution of partially ordered multisets [27])**.** We have a partially ordered multiset $P = [V_P, \Sigma_P, \leq_P, \mu_P]$ and a mapping function g that maps each a in Σ_P to some pomset denoted by $g(a) = [V_{g(a)}, \Sigma_{g(a)}, \leq_{g(a)}, \mu_{g(a)}]$. $P[g]$ is the substitution of P using mapping function g and is a new pomset $P[g] = [V_{P[g]}, \Sigma_{P[g]}, \leq_{P[g]}, \mu_{P[g]}]$, in which every instance of a is replaced with an instance of its corresponding pomset $g(a)$. By combining μ_P with mapping function g, we get a function f which maps each

v	A	B	C	D
$\tau(v)$	D	B	A	C

Table 2.3: The map by which the labelled partial order of Figure 2.6 is homomorphic to Figure 2.8

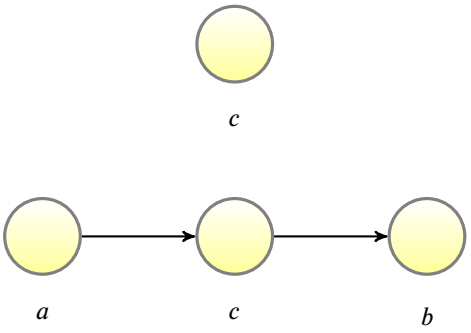


Figure 2.9: Example of a pomset

vertex of P to a pomset. For the resulting pomset $P[g]$, the following formulas hold:

$$V_{P[g]} = \uplus_{v \in V_P} V_{f(v)} \tag{2.3}$$

$$\Sigma_{P[g]} = \cup_{v \in V_P} \Sigma_{f(v)} \tag{2.4}$$

$$\leq_{P[g]} = \left(\cup_{v \leq_P u} V_{f(v)} \times V_{f(u)} \right) \cup \left(\cup_{v \in V_P} \leq_{f(v)} \right) \tag{2.5}$$

$$\mu_{P[g]} = \uplus_{v \in V_P} \mu_{f(v)} \tag{2.6}$$

When creating the substitution $P[g]$ by applying mapping function g on pomset P, one needs to combine the mapping function μ_P that maps each vertex $v \in V_P$ to a symbol $\sigma \in \Sigma_P$ with g [27]. This results in a function f that maps each vertex $v \in V_P$ to a pomset. As shown in Definition 2.4, this results in $V_{P[g]}$ being the disjoint union of all $V_{f(v)}$ and $\mu_{P[g]}$ being the disjoint union of all $\mu_{f(v)}$ for all $v \in V_P$. Comparable, $\Sigma_{P[g]}$ is the union of all $\Sigma_{f(v)}$ for all $v \in V_P$. Finally, $v \leq_{P[g]} u$ holds only when $v \leq_{f(w)} u$ for some $w \in V_P$ with $v, u \in V_{f(w)}$ or when $v \in V_{f(w)}$, $u \in V_{f(z)}$, and $w \leq_P z$.

Suppose that we have a mapping function g, which maps the symbol c to the pomset shown in Figure 2.10. Furthermore, the mappings of other symbols result in the input, e.g. $g(a)$ is equal to a pomset consisting of just a node with the symbol a. If we apply this map to the pomset of Figure 2.9, we get the pomset shown in Figure 2.11.

Two pomsets can be composed using either sequential composition – or concatenation – and parallel composition [27]. Sequential composition is formalised in Definition 2.5, which states that the composition consists of the sum of both pomsets where all vertices of the first pomset precede those of the second pomset in the partial ordering. To illustrate concatenation of two pomsets, Figure 2.12 shows the concatenation of the pomset of Figure 2.9 with itself.

Definition 2.5 (Sequential composition of partially ordered multisets [27])**.** The sequential composition of two pomsets P and Q is denoted by $P.Q = [V_{P.Q}, \Sigma_{P.Q}, \leq_{P.Q}$

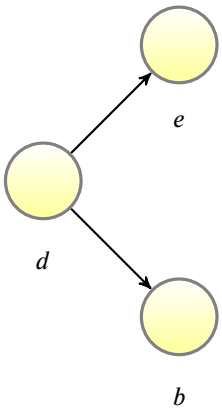


Figure 2.10: The pomset for $g(c)$

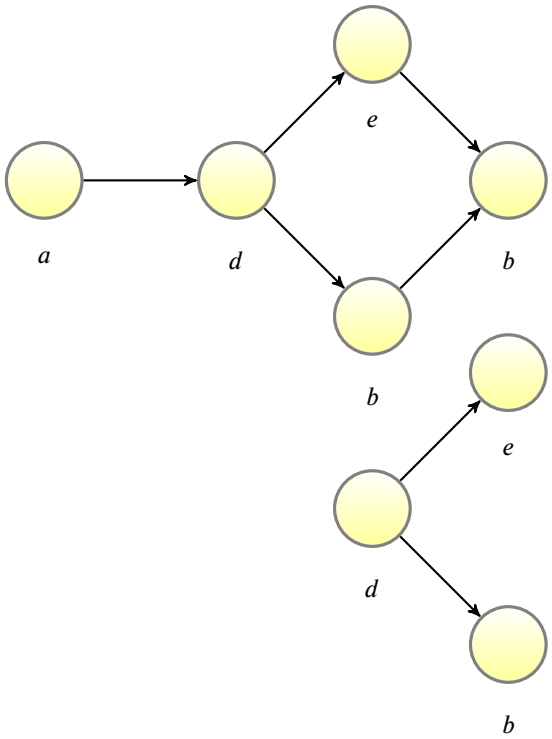


Figure 2.11: The mapping function g applied to the pomset of Figure 2.9

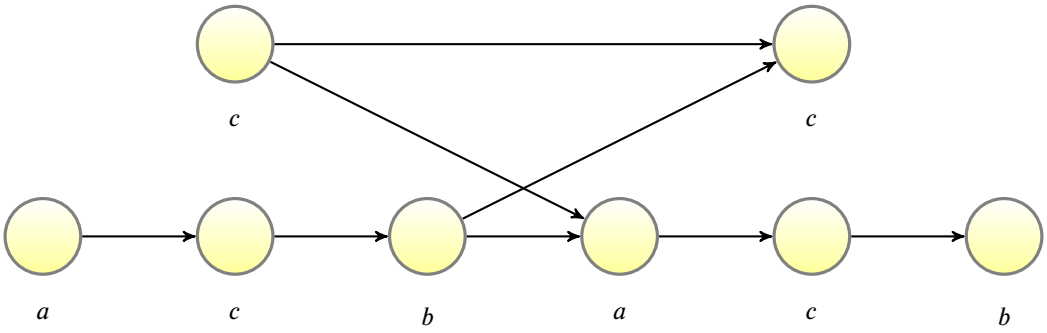


Figure 2.12: The sequential composition of the pomset of Figure 2.9 with itself

$, \mu_{P.Q}]$. For sequential composition, the following formulas hold:

$$V_{P.Q} = V_P \uplus V_Q \tag{2.7}$$
$$\Sigma_{P.Q} = \Sigma_P \cup \Sigma_Q \tag{2.8}$$
$$\leq_{P.Q} = \left(\cup_{v \in V_P \times u \in V_Q} v \leq u\right) \cup \leq_P \cup \leq_Q \tag{2.9}$$
$$\mu_{P.Q} = \mu_P \uplus \mu_Q \tag{2.10}$$

Parallel composition is explained in Definition 2.6, which simply states that the composition consists of the sum of both pomsets. The parallel composition of two pomsets is shown Figure 2.13 in which the pomset of Figure 2.9 is composed with itself.

Definition 2.6 (Parallel composition of partially ordered multisets [27])**.** The parallel composition of two pomsets P and Q is denoted by $P||Q = [V_{P||Q}, \Sigma_{P||Q}, \leq_{P||Q}, \mu_{P||Q}]$. For sequential composition, the following formulas hold:

$$V_{P||Q} = V_P \uplus V_Q \tag{2.11}$$
$$\Sigma_{P||Q} = \Sigma_P \cup \Sigma_Q \tag{2.12}$$
$$\leq_{P||Q} = \leq_P \cup \leq_Q \tag{2.13}$$
$$\mu_{P||Q} = \mu_P \uplus \mu_Q \tag{2.14}$$

Advantages of Pomsets Pomsets have one main advantage over graphs when modelling service oriented systems, which lies within the uncoupled property of service oriented computing. Uncoupledness of services refers to the emphasis on reducing dependencies between the service contract, its implementation, and its service consumers [71, 24, 54]. Where graphs provide much stronger relations between the nodes, pomsets adapt much better to this absence of coupling by only providing constraints on the order of invocation. Furthermore, even the ordering is only partial, which also allows for completely unordered systems.

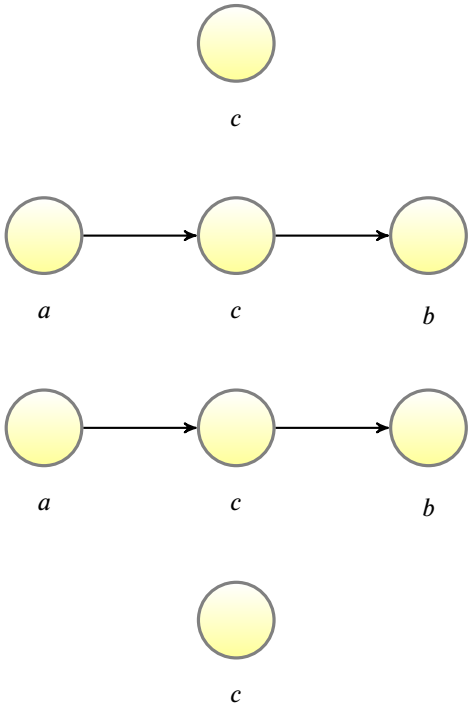


Figure 2.13: The parallel composition of the pomset of Figure 2.9 with itself

It should be noted that the interaction between services could also be modelled using techniques from the process calculus, but this suffers from comparable complications. Although there are implementations of models of service oriented computing using process calculus [e.g. 72, 66, 13], they are rather strict and specific. Bravetti and Zavattaro [13] show that implementing loose coupling is possible. However, the strict definition of uncoupled services in this research does not play well with the models implemented using process calculi. Additionally, due to their more precise nature, the process calculi tend to be too specific and detailed than necessary for our research, which makes it harder to keep an overview of the system. Furthermore, usage of process calculus would leave even less room for integrating the security properties of the atomic services into the bigger picture and making security preserving composition of services possible.

Processes Pratt [64] defines a process as a set of pomsets [63] – see also definition 2.7. Thus, as Gischer [27] puts it, one can see a process as being "identical to the set of its possible behaviours" – which are modelled using pomsets. In this notion, the process adds an element of choice to the pomset, which describes all events including their partial ordering and possible duplicate execution. Furthermore, the process existing of all possible pomsets over Σ is denoted as Σ^*.

Definition 2.7 (Process [64])**.** A process P is a set of pomsets p_i with $0 \leq i < n$.

As we saw with pomsets, it is also possible to compose two processes. The sequential composition of processes is shown in Definition 2.8 and the parallel composition in Definition 2.9. Those forms simply compose all the elements of the pomsets captured by the process. Furthermore, the addition of two processes results in one larger process that contains the pomsets of both processes, as stated in Definition 2.10.

Definition 2.8 (Sequential composition of processes [27])**.** The sequential composition of two processes P and Q is defined as:

$$P.Q = \{p.q | p \in P \wedge q \in Q\} \tag{2.15}$$

Definition 2.9 (Parallel composition of processes [27])**.** The parallel composition of two processes P and Q is defined as:

$$P||Q = \{p||q | p \in P \wedge q \in Q\} \tag{2.16}$$

Definition 2.10 (Addition of processes [27])**.** The addition of two processes P and Q is defined as the union of those two processes, i.e. $P \cup Q$.

2.3 Modelling Service Oriented Systems

In order to adapt pomsets to service oriented computing, we will see Σ as the set of services and V as the set of actions on these services, i.e. the possible service invocations.

The ordering serves to provide minimal orders of invocation of the services – for example, an authentication service will have to be invoked before a service that requires an authenticated user. Nevertheless, given the uncoupled nature of service oriented systems, in principle there is no ordering. Therefore, all ordering is introduced by the service compositions that are made.

As already stated in Section 2.1, we do not consider runs in the model, but service compositions. In other words, pomsets will be used to model compositions and not individual instances of this composition. As explained, modelling individual runs would bypass the main problem of the paper, as those are already composed to a single process and can, thus, be verified as a whole.

For the sake of simplicity, we will not consider composition logic other than sequential and parallel composition. This means that, for example, and-compositions and or-compositions will be considered both as parallel compositions, even though or-compositions can be mutually exclusive. Although this creates the unnecessary requirement of verifying parallel composability of mutually exclusive or-compositions, this strongly simplifies the model.

To give an example, we will return to the example service composition for a news website that was introduced in Figure 2.2. If we were to build a pomset out of this composition, we would get the following alphabet shown in Equation 2.17. Furthermore, we would get a partial ordering similar to the one shown in Equation 2.18 and Equation 2.19. Please note that, although the order is written as if it concerns the services, it actually orders the specific invocations. Nevertheless, due to the fact that no service occurs twice, this notation is used as simplification. This example is also shown in Figure 2.14, which is similar to the original figure, with omission of the names of the vertices.

$$\Sigma = \{local\_news, national\_news, news, news\_site, weather\_1, weather\_2\} \tag{2.17}$$

$$news\_site < \{weather\_1, weather\_2, news\} \tag{2.18}$$

$$news < \{local\_news, national\_news\} \tag{2.19}$$

Within specific service compositions, the partial order can also be represented using dependencies. In this case, every service s may be dependent on one or more other services d_i with $0 \leq i < n$. In the partial ordering $\leq$, this would give $\forall 0 \leq i < n : d_i \leq s$. Nevertheless, not all partial ordering follows from dependencies, and strictly speaking dependencies should be minimised – or not exist at all – in service oriented systems.

The process structure, being a set of pomsets, contains all possible compositions of a certain set of services Σ_P. This set of services is equal to the union of all sets of services Σ_{p_i} of all the pomsets p_i contained in the process P, as shown in Equation 2.20. Therefore, a process provides a good model of a SOA.

$$\Sigma_P = \bigcup_{i=0}^{n} \Sigma_{p_i} \tag{2.20}$$

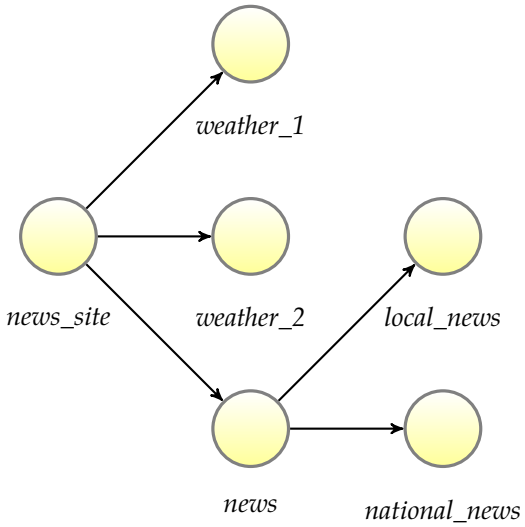


Figure 2.14: Pomset for Figure 2.2

2.4 Differences from Protocols

As stated before, we will continue on the previous work by Datta et al. [18] on secure composition of protocols in Chapter 3. This requires us to ask ourselves what the novelty of this research is. Therefore, we will discuss Research Question 2.

Research Question 2. On what characteristics differ services from protocols such that the work on secure composition of protocols by Datta et al. [18] is not directly applicable to services?

To answer where the differences lie, we need to see what a protocol really is first. A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event [40]. In other words, a protocol can be seen as an algorithm that performs a certain task that is run between multiple entities over a network. For example, FTP – the file transmission protocol – provides a set of procedures for sending files over a network.

It can be easily seen that services make use of protocols, in order to communicate with each other. These protocols form the – commonly open – standards services need to operate. For example, such protocols enable discovery, composition, and invocation of services.

This underlines a key difference between protocols and services. Namely, where services are a means of providing capabilities that can be composed to any number of larger systems, protocols are designed for performing a predefined task. Where a networked algorithm seems an apt analogy for a protocol, a networked process covers

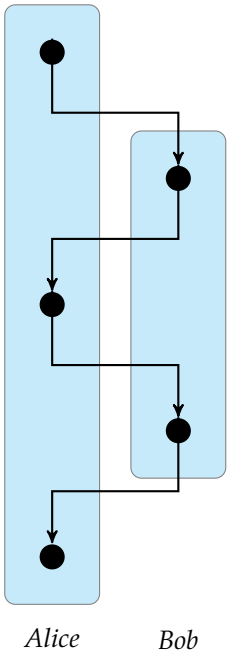


Figure 2.15: Example of a protocol run

the concept of service oriented computing quite well. Please note that, in this statement, the word process is used to refer to its general meaning, i.e. an instance of a computer program. Nevertheless, if we were to model the internals of a service also as a pomset, a service composition would also satisfy Definition 2.7, i.e. it would be a pomset of pomsets. However, as we are not concerned with the internals of services at this point, a service composition is modelled as a pomset, as described in Section 2.3.

Algorithms, and, thus, protocols, are usually modelled as sequences of operations – so-called runs. For example, Datta et al. [18] does this in the form of cord calculus and Jürjens [35] in the form of stream processing functions. For services, we have decided to go with pomsets. This underlines quite well the difference between the strictness of protocols and the looseness of services.

Those protocol runs are commonly presented as shown in Figure 2.15, where two participants – Alice and Bob – perform the several steps of a protocol run. This figure shows that, in a protocol, two or more parties exchange messages following a previously defined format. As stated before, service oriented systems also make use of protocols for the connection between two services.

Actually, when one looks at service oriented computing from a software architectures point of view, one can state that protocols provide connectors between software. Service oriented computing provides the ability to use these connections to run a complete software system in a distributed fashion. As we learned, this software system

is composed of services, which allows for easy replacement, removal and addition of services. Thus, the services are the modular pieces that allow the building of a system, whereas protocols provide infrastructural connectors that can be used to network these pieces.

Furthermore, as already mentioned, protocols have a certain predefined tasks, as opposed to services that are uncoupled and freely composable. As we defined this as the absence of any prior communication between two services or prior knowledge of the other service, this conflicts with the predefined notion of protocols. In the case of protocols, there is prior knowledge of how the participants should react and what they should know. Especially this difference creates extra difficulty when applying the work of Datta et al. [18] to service oriented computing.

To conclude, we can state that services are the uncoupled building blocks of a distributed software system, whereas protocols are strictly defined sequences of communications. Services expose access to computing capabilities, and have, thus, potentially much more possibilities, as opposed to the single goal for which a protocol is defined. Finally, protocols can be seen as infrastructural pieces, whereas services provide architecture and software itself.

3 Secure Composition

Datta et al. [18] provide rules for composing security protocols based on the logic of Hoare [32]. This paper tries to overcome the discussed problems raised by additive combination and non-destructive combination. Additive combination refers to combining security protocols such that security properties are combined. Non-destructive combination means combining security protocols in such a way that they do not void the security properties.

Although additive combination and non-destructive combination sound similar, there is a distinction [18]. For example, if you have a protocol to establish a shared key and a protocol to authenticate the other party, you may want to combine those protocols such that you have a protocol to establish an authenticated shared key. For this, you need additive combination. Namely, you want to combine the security properties of both protocols by constructing a combined protocol.

With non-destructive combination, you want to make sure that a composition does not void the security properties guaranteed by the parts [18]. To illustrate this, non-destructive combination should prevent you from ending up with a protocol that sends encrypted messages and guarantees the confidentiality of those messages with a protocol that would broadcast those messages to anyone listening, as that would void the confidentiality of the first protocol. A typical case for non-destructive combination in the world of protocols can be found in the parallel usage of different versions of the same protocol for the convenience of the user.

Very simply put, additive combination mostly concerns sequential composition, whereas non-destructive combination tends to fit in with parallel composition. To illustrate this, with sequential composition, one commonly wants to combine two protocols such that their functionalities are appended, whereas parallel composition is commonly used to allow users to choose different paths, e.g. to run two versions in parallel. For service oriented systems, this is a bit different, as there the results of services ran in parallel may be combined later on for additive functionality. In this case, the parallel composition is used sequentially with the service that combines those results.

Additive Combination To capture additive combination, Datta et al. [18] introduce a so-called before-after formalism, which revolves around the combination of preconditions, postconditions and invariants. A precondition is a certain logical condition that should hold before the described action. If this precondition is satisfied the logical condition described by the postcondition will hold afterwards. To give an example, I have a function for calculating a sum, as shown in Equation 3.1, and I define a precondition that states that $0 \leq a \leq b$. Now, a postcondition stating $0 \leq f(a, b)$ will

hold when the precondition is satisfied, resulting in a correct logical annotation of the function $f(a, b)$.

$$f(a, b) = a + b \tag{3.1}$$

As our notation, we use $[P]_A$ to denote the actions P of a process A executed by principal $\hat{A}$, as stated in Definition 3.1. Please note that processes are distinguished from the principals that control them. In our notation, we will use a hat to refer to a principal, i.e. $\hat{A}$. The separation of the principal means that it is possible that one entity runs multiple processes. For this reason, the letter used to identify a principal does not have to be equal to the one used for the identification of the process.

Definition 3.1 (Processes [18])**.** The construct $[P]_A$ refers to the actions P executed in a process A controlled by principal $\hat{A}$.

In the logic of Datta et al. [18], the triple $\phi[P]_A\psi$ means that, if the precondition ϕ holds before entity $\hat{A}$ performs actions P, then postcondition ψ will hold afterwards, as shown in Definition 3.2. For example, the precondition ϕ could state that A knows a private key, the actions P that A receives a certain nonce – a random value – and sends this nonce signed using the key, and the postcondition ψ that the nonce is indeed correctly signed using the private key. This triple is in essence a variant of the triple of Hoare [32].

Definition 3.2 (Process Triple [18, 17])**.** The triple $\phi[P]_A\psi$ is true if the postcondition ψ holds for the actions P executed in a process A controlled by principal $\hat{A}$ given the precondition ϕ.

The logic of the triples enables sequential composition of protocols when the postcondition of the first protocol matches the precondition of the second protocol [18]. For example, if we have $\phi[P]_A\psi$ and $\psi[Q]_A\theta$, we can derive $\phi[PQ]_A\theta$. Matching of the conditions does not mean that those have to be equal, but merely that the postcondition of the first protocol has to be stronger than or equal to the precondition of the second protocol.

Non-destructive Combination Non-destructive combination is needed to be able to run parallel processes [18]. In the case of protocols, this can be used to run, for example, two versions of the same protocol concurrently in order to provide a fallback version for older clients. The central assertion in the framework of Datta et al. [18], $\Gamma \vdash \phi[P]_A\psi$, means that the triple $\phi[P]_A\psi$ is true if the invariant Γ is satisfied, as formalised in definition 3.3. Thus, in any protocol run that satisfies Γ, $\phi[P]_A\psi$ will hold, regardless of the actions performed by any possible attacker. This makes parallel execution possible by ensuring that the invariant Γ holds for all those protocols that are ran in parallel.

Definition 3.3 (Process Soundness [18, 17])**.** If we have $\Gamma \vdash \phi[P]_A\psi$, the process triple $\phi[P]_A\psi$ is always true when the environment invariant Γ is satisfied.

An example of an invariant would be that every participant keeps his private key secret. If we would consider the nonce-signing protocol mentioned before, $\Gamma \vdash \phi[P]_A\psi$, means that when every participant keeps his keys to himself, the postcondition ψ of every run will be satisfied when the precondition ϕ holds in that run. Thus, every run will result in a perfectly verifiable signed nonce.

Other examples of invariants could be that honest services act in a specific way. Such an invariant can be used to derive authentication, as it makes it possible to distinguish between honest and dishonest services, as the former should act as stated in the invariant, while the latter does not do so. What is specifically meant by honesty and how such an implementation works is discussed in depth in Section 3.1.

3.1 Secure Composition of Protocols

Given that the work of Datta et al. [18] focusses on protocols, we will start with briefly discussing their work. Afterwards, we will discuss the secure composition of services in Section 3.2.

In this section, we will discuss the formalism used to represent protocols by Datta et al. [18] which is the cord calculus. Additionally, we will discuss their protocol logic and proof system. Due to the nature of protocols, this framework is more tightly bound to the traces of the protocols than a service oriented approach would be.

3.1.1 Cord Calculus

The cord calculus is presented by Durgin, Mitchell and Pavlovic [23] and can be used to represent protocols and their parts. It can be seen as a form of process calculus, which we discussed in Section 2.2. As already mentioned, it is too strict for easy adaptation to a service oriented context.

In this section, we will describe syntax using the extended Backus-Naur Form [34], which has been simplified for readability. In the used notation, a symbol is defined using $::=$ by a sequence of symbols, which are either terminating symbols or non-terminating symbols. The latter sort can be found among the syntactical definitions itself. If a symbol can occur in different forms, the different options are written on different lines, as opposed to the general usage of the symbol | to denote alternatives.

Origin As noted before, the cord calculus of Durgin, Mitchell and Pavlovic [23] is an action calculus based on the π-calculus of Milner, Parrow and Walker [53] and related to spi-calculus of Abadi and Gordon [1]. Most notably, the cord calculus inherits matching and substitution from this family and language. For example, when in such a language on has a constant M and a variable x, the operation (M/x) is the substitution of x by M, i.e. all further occurrences of x are replaced by M. When one refers to matching and substitution, the matching means that both sides of the slash in this operation are of the same form.

Terms In the cord calculus, terms t are built from variables x and constants c [18]. Additionally, basic terms can contain names, commonly denoted by N, and keys, commonly denoted by K. Those names can be either variables $\hat{X}$ or constants $\hat{A}$, for which the syntax is shown in Equation 3.2. Please note that the hats fit in with our notation of principals.

$$\begin{aligned} N ::= \hat{X} \\ \hat{A} \end{aligned} \tag{3.2}$$

Comparably, keys can be either variables y, constants k or names N [18]. The syntax of basic keys is shown in Equation 3.3, whereas Equation 3.4 shows the syntax for a key K, which can either be a public key K_0 or a secret key K_0^{-1} – note that the 0 in the subscript offers differentiation between keys and basic keys in the syntactical definition. In other words, a public key K_0 or secret key K_0^{-1} is, respectively, either a variable y or y^{-1}, constant k or k^{-1}, or name N or N^{-1}. Please note that, at this point, we only consider public key encryption.

$$\begin{aligned} K_0 ::= k \\ y \\ N \end{aligned} \tag{3.3}$$

$$\begin{aligned} K ::= K_0 \\ K_0^{-1} \end{aligned} \tag{3.4}$$

The syntax for basic terms is given by Equation 3.5 [18]. Basic terms consist of the discussed syntactical components, i.e. a basic term is either a variable x, constant c, name N or key K.

$$\begin{aligned} b ::= x \\ c \\ N \\ K \end{aligned} \tag{3.5}$$

In addition, we have role identifiers. A role identifier η can be either a variable role identifier s or a constant role identifier c' [18]. The syntax for role identifiers is given in Equation 3.6. This role identifier is simply used to distinguish roles within the protocols, i.e. the defined participants in the protocol. Therefore, the syntax for a process consists of a name N and a role identifier η, as shown in Equation 3.7. In other words, a process refers to a certain principal – hence the name N – executing a certain role. It should be noted that an attacker does not assume a role, but conforms to the model of Dolev and Yao [22], which we will discuss in Section 3.5.

$$\begin{aligned} \eta &::= s \\ &\quad\ c' \end{aligned} \tag{3.6}$$

$$P ::= N, \eta \tag{3.7}$$

The language of terms is generated by constructors p, which include constructing a tuple, public key encryption and signing [18]. A tuple t, t simply consists of two terms t – please note that those terms do not need to be equal, merely of the same type, i.e. a term t as defined in Equation 3.8. Encryption is denoted by $\{\!|\ t\ |\!\}_K$, where term t is encrypted using public key K. Finally, signing is written as $\{\!|\ t\ |\!\}_{K^{-1}}$ where term t is signed using secret key K^{-1}. This leads to the syntax of normal terms being defined as shown in Equation 3.8.

$$\begin{aligned} t ::= &\ x \\ &\ c \\ &\ N \\ &\ K \\ &\ \eta \\ &\ t, t \\ &\ \{\!|\ t\ |\!\}_K \\ &\ \{\!|\ t\ |\!\}_{K^{-1}} \end{aligned} \tag{3.8}$$

Actions The language of actions consist of the null action, sending a term, receiving something and storing it into a variable, matching a term against a pattern, and generating a new value [18]. The empty action is written as ϵ. The action of sending a term t is denoted by $\langle t \rangle$. With (x) receiving a certain term into variable x is meant. The generation of a new term x is written as (νx).

Furthermore, $(t/q(x_1, \ldots, x_n))$ is used to denoted the matching of the term t to pattern $q(x_1, \ldots, x_n)$ [18]. As stated before, this means that a term t is matched to a term of the same form $q(x_1, \ldots, x_n)$, whereby the variables x_1 through x_n are substituted by constant values from the term t accordingly. Please note that q is used to refer to the constructor, e.g. a tupling constructor $x_1, \ldots, x_n$ as shown in Equation 3.10.

Combined, the discussed actions constitute the syntax as shown in Equation 3.9 [18]. In this syntactical definition a is defined to be either one of the stated actions,, i.e. the empty action, the send action, the receive action, the generation action, and the matching and substitution action.

$$
\begin{aligned}
a ::= \; & \epsilon \\
& \langle t \rangle \\
& (x) \\
& (\nu x) \\
& (t/q(x_1, \ldots, x_n))
\end{aligned}
\tag{3.9}
$$

Patterns Patterns can be used in the matching action, which matches a term t to some pattern $q(x_1, \ldots, x_n)$ and substitutes accordingly [18]. As stated before, in this notation of a pattern, q denotes a constructor and x_1 through x_n variables. There are multiple patterns possible. Of the available patterns, the only basic pattern is the tuple pattern, which is shown in Equation 3.10. In the case of the tuple pattern, $q(x_1, \ldots, x_n)$ would take the form of $x_1, \ldots, x_n$.

$$
p ::= b, \ldots, b \tag{3.10}
$$

The set of additional patterns consists of – in extension to the basic pattern of tupling – the decryption pattern and the signature verification pattern, as shown in Equation 3.12 [18]. The decryption pattern is denoted by $\{\!|\ p\ |\!\}_{K^{-1}}$, where pattern p is of the form of a basic pattern and K^{-1} denotes the secret key. This pattern could be matched against a term of the form $\{\!|\ t\ |\!\}_K$, where t should be of the same form as p. For example, if we have the statement shown in Equation 3.11, the operation is used to denote the decryption of $\{\!|\ M_1, \ldots, M_n\ |\!\}_K$ resulting into the substitution of $x_1, \ldots, x_n$ with $M_1, \ldots, M_n$.

$$
(\{\!|\ M_1, \ldots, M_n\ |\!\}_K / \{\!|\ x_1, \ldots, x_n\ |\!\}_{K^{-1}}) \tag{3.11}
$$

Comparably, the pattern for signature verification is written as $\{\!|\ p\ |\!\}_K$ where p is a basic pattern and K the public key [18]. The inner workings of this pattern are equal to those of decryption. Nevertheless, there is no need for substitutions in the case of signature verification, as the signed variable is directly accessible, whereas an encrypted value needs to be decrypted first. This is commonly depicted by using the same variable names on both sides of the matching operation, e.g. $(\{\!|\ m\ |\!\}_{K^{-1}} / \{\!|\ m\ |\!\}_K)$.

$$
\begin{aligned}
q ::= \; & p \\
& \{\!|\ p\ |\!\}_{K^{-1}} \\
& \{\!|\ p\ |\!\}_K
\end{aligned}
\tag{3.12}
$$

Cords Cords build further on Strands, which were defined by Fabrega, Herzog and Guttman [25]. A strand is a list of actions, which is portrayed by Equation 3.13. Due to this nature, two strands can have the same observable behaviour, although they are formally different [23]. For example, to the outside world generating two values and

sending them as a tuple does not look different depending on the order in which the values are generated.

$$\begin{aligned} S &::= aS \\ &\quad a \end{aligned} \tag{3.13}$$

A cord is an equivalence class of strands that are behaviourally indistinguishable [23]. To give an example, the two strands $(x)\langle t\rangle$ and $\langle t\rangle(x)$ are obviously different. However, they represent the same cord, as the observable behaviour of sending and receiving two independent terms does not require any order. When the strand would be $(x)\langle x\rangle$, it would differ from $\langle x\rangle(x)$, as the variable that is sent in the first strand is equal to the one received moments before, whereas this is not possible the other way round – one cannot send something before receiving it.

Formally, a cord is an equivalence class of strands modulo an equivalence relation $\approx_C$ that allows reordering of actions that do not conflict and renaming of bound variables [23]. This equivalence relation $\approx_C$ is the transitive and reflective closure of the relation $\sim_C$, which is defined on two strands S and T as shown in Equation 3.14, closed under α-equivalence, i.e. bound variables may be renamed as long as this does not result in clashes with other variables. This relation ensures that S and T cannot depend upon each other.

$$\begin{aligned} ST \sim_C TS \Leftrightarrow\ & (\mathit{FreeVariables}(S) \cap \mathit{BoundVariables}(T) = \varnothing) \\ & \wedge\ (\mathit{FreeVariables}(T) \cap \mathit{BoundVariables}(S) = \varnothing) \end{aligned} \tag{3.14}$$

In Equation 3.14, the statements $\mathit{FreeVariables}(S)$ and $\mathit{BoundVariables}(S)$ are used to refer to the sets of, respectively, free and bound variables [23]. A free variable is a variable that has not be bound to a certain value by the substitution operation. When a free variable becomes bound to a value, it follows that this variables becomes a bound variable. Therefore, the relation shown in Equation 3.14 states that ST and TS are similar when the set of free variables of S and the set of bound variables of T, and the set of free variables of T and the set of bound variables of S are distinct.

The set of free variables is defined under mathematical induction in Definition 3.4 and the set of bound variables in Definition 3.5 [23]. In these definitions, the S denotes the tail of the strand the statement is concerned with. For the free variables, it can be seen that for $\langle t\rangle S$ the set of free variables is equal to the union of the sets of free variables of t and S. With $(x)S$ and $(\nu x)S$, the variable x gets bound, which causes its removal from the set of free variables. Comparably, $(t/p(x))S$ substitutes the x with the matching terms in t resulting in the set of free variables being equal to the union of those of t and S minus the variable x. Finally, an empty strand contains no variables and, thus, no free variables. For bound variables, this works the other way around, as can be seen in Definition 3.5.

Definition 3.4 (Free variables [23])**.** The set of free variables is written using the operator $\mathit{FreeVariables}(S)$. For a term t, $\mathit{FreeVariables}(t)$ is defined to contain all variables

used in the formation of this term. Furthermore, given a strand S, $FreeVariables(S)$ is defined under mathematical induction as follows:

$$FreeVariables() = \varnothing \tag{3.15}$$
$$FreeVariables(\langle t \rangle S) = FreeVariables(t) \cup FreeVariables(S) \tag{3.16}$$
$$FreeVariables((x)S) = FreeVariables(S) \backslash \{x\} \tag{3.17}$$
$$FreeVariables((t/p(x))S) = FreeVariables(t) \cup FreeVariables(S) \backslash \{x\} \tag{3.18}$$
$$FreeVariables((\nu x)S) = FreeVariables(S) \backslash \{x\} \tag{3.19}$$

Definition 3.5 (Bound variables [23])**.** The set of bound variables is written using the operator $BoundVariables(S)$. Given a strand S, $BoundVariables(S)$ is defined under mathematical induction as follows:

$$BoundVariables() = \varnothing \tag{3.20}$$
$$BoundVariables(\langle t \rangle S) = BoundVariables(S) \tag{3.21}$$
$$BoundVariables((x)S) = BoundVariables(S) \cup \{x\} \tag{3.22}$$
$$BoundVariables((t/p(x))S) = BoundVariables(S) \cup \{x\} \tag{3.23}$$
$$BoundVariables((\nu x)S) = BoundVariables(S) \cup \{x\} \tag{3.24}$$

Besides actions, a cord also contains an input and an output interface [18]. The former of those contains prior knowledge, such as keys, and the latter contains the data the becomes known after executing the cord. These interfaces are, thus, to be used to set up the initial state and returning the data of the final state.

Cord Spaces A cord space can be seen as the multiset of all roles involved with a certain protocol [23]. More formally, a cord space is a multiset of cords that belong to a protocol. For example, in a common client-server protocol, one would have two roles and thus two cords – namely, the client and the server. These two cords combined form the cord space of that specific client-server protocol.

At this point, it should be noted that a principal may perform multiple roles within the same protocol [23]. However, it is not allowed that two different roles ran by the same principal influence each other, which is formalised in Theorem 3.1. Incidentally, this emphasises the uncoupled and stateless characteristic of services, which we discussed in Chapter 2.

Theorem 3.1 (Crosstalk [23])**.** *Given a principal A and a role ρ, if principal A follows role ρ, then all data that is sent by A in role ρ is either generated or received by A in this role.*

The runs of a protocol are built using the reaction sequences of cord spaces [18]. Those reaction sequences define how one or more cords interact and which results this such a reaction yields. The five basic reaction steps are shown in Theorem 3.2. In these reaction steps, C can be regarded as the other cords in the protocol that are left unaffected by the reaction.

Theorem 3.2 (Basic reaction steps for cord spaces [18]). *The following reaction steps define the basic reactions within a cord space:*

$$[S(x)S'] \cup [T\langle t\rangle T'] \cup C \to [SS'(t/x)] \cup [TT'] \cup C \tag{3.25}$$

$$[S(p(t)/p(x))S'] \cup C \to [SS'(t/x)] \cup C \tag{3.26}$$

$$[S(\{\!|\ p(t)\ |\!\}_y/\{\!|\ p(x)\ |\!\}_{y^{-1}})S'] \cup C \to [SS'(t/x)] \cup C \tag{3.27}$$

$$[S(\{\!|\ p(t)\ |\!\}_{y^{-1}}/\{\!|\ p(t)\ |\!\}_y)S'] \cup C \to [SS'] \cup C \tag{3.28}$$

$$[S(\nu x)S'] \cup C \to [SS'(m/x)] \cup C \tag{3.29}$$

The reaction step shown in Equation 3.25 shows how send and receive interact [18]. The second cord sends the term t, which the first cord receives into variable x. Therefore, in the resulting cords, the variable x is substituted by the received term t in the tail of the cord S'. For this reaction step there must be no free variables in t, i.e. $FreeVariables(t) = \emptyset$ should hold.

The latter four reaction steps in Theorem 3.2 are all interactions within one cord [18]. Equation 3.26 shows plain pattern matching, where the pattern $p(x)$ is matched against $p(t)$, thereby substituting x with t. Also for this reaction step, $FreeVariables(t) = \emptyset$ should hold.

Equation 3.27 shows the reaction of the decryption of $\{\!|\ p(t)\ |\!\}_y$, such that the encrypted term t is put into variable x [18]. Comparably, Equation 3.28 verifies the signature on the term $\{\!|\ p(t)\ |\!\}_{y^{-1}}$. As mentioned before, signature verification does not result in extra substitutions, due to the fact that the value is already accessible. For both of these reaction steps, $FreeVariables(t) = \emptyset$ should hold. Additionally, the reaction step of Equation 3.27 requires y to be bound, as this variable represents the used keys.

Finally, Equation 3.29 concerns the fresh generation of a variable x and the substitution of a certain variable m with this value [18]. Obviously, this requires x to be a new variable, since it is generated freshly. Additionally, m should not be used either, as it is used to hold this new variable. Therefore, Equation 3.30 should hold. This equation states that x should not be free in S and m should not be free in S, S' or C. In other words, x and m should not have occurred as free variable, but only as bound variables in the context of the fresh value generation.

$$x \notin FreeVariables(S) \\ \wedge\ m \notin (FreeVariables(S) \cup FreeVariables(S') \cup FreeVariables(C)) \tag{3.30}$$

3.1.2 Protocol Logic

Built upon the cord calculus, Datta et al. [18] use a protocol logic to describe protocols and the predicates attached to it. In Section 3.3, we will discuss our adaptation of this logic, where we will alter the logic to fit services. However, at this point, we will merely discuss the protocol logic as is. This section is divided in a part discussing the syntax and a part concerning the semantics of the protocol logic.

Syntax

This section describes which formulas are available in the protocol logic and how they form a language. Firstly, there are action formulas, which describe the possible actions that can be performed in the protocol logic. Secondly, there are predicate formulas that can be used in the logical statements.

As previously discussed, Datta et al. [18] use the formula $\phi[P]_X\psi$ to reason about a protocol. In this protocol, the actions P are executed in process X, whereby the precondition ϕ is expected to hold before and the postcondition ψ to hold after the execution. The syntax for these modal forms are shown in Equation 3.31. Please note that ϕ is used to denote logical formulas and ρ to denote the action formulas, i.e. ρ is where the discussed cord calculus comes in.

$$\begin{aligned} \Phi ::= \; & \rho\phi \\ & \phi\rho\phi \end{aligned} \tag{3.31}$$

Actions The syntax for action formulas is described by Equation 3.32 [18]. All the defined action formulas mean that the respective action is the last action principal $\hat{P}$ performed in process P. For example, $Decrypt(P,t)$ states that the last action performed by $\hat{P}$ in process P was decrypting term t. Please note that sending and receiving are concerned with a message m and nonce generation, decryption and verification are concerned with a term t. This allows for differentiation between terms that are used in communication and those that stay within a process.

$$\begin{aligned} a ::= \; & Send(P,m) \\ & Receive(P,m) \\ & New(P,t) \\ & Decrypt(P,t) \\ & Verify(P,t) \end{aligned} \tag{3.32}$$

Predicates Additionally to the action formulas, the protocol logic uses predicate formulas [18]. The syntax for those predicate formulas can be found in Equation 3.33. These predicate formulas make a logical statement that can be used in a logical formula.

$$
\begin{aligned}
\phi ::= \; & a \\
& Has(P,t) \\
& Fresh(P,t) \\
& Honest(N) \\
& Contains(t_1,t_2) \\
& \phi \wedge \phi \\
& \neg\phi \\
& \exists x : \phi \\
& \diamondsuit\!\!\cdot \phi \\
& \odot \phi
\end{aligned}
\tag{3.33}
$$

The predicate $Has(P,t)$ states that principal $\hat{P}$ possesses some piece of information t in process P [18]. In this case, possession means that the principal has access to this information. The formula $Fresh(P,t)$ means that the term t has been generated in process P and is not (yet) known to anyone else. Therefore, this term t is fresh. The formula $Honest(N)$ states that principal N is honest, i.e. this principal acts as defined by the protocol. We will discuss honesty in depth in Section 3.1.3.

The terms $\phi \wedge \phi$ and $\neg\phi$ express, respectively, the logical and-relation and the logical negation [18]. With the formula $Contains(t_1,t_2)$, it is expressed that t_2 is a subterm of term t_1. Thus, t_1 contains t_2. Furthermore, $\exists x : \phi$ means that there exists an x such that ϕ.

The temporal operator $\diamondsuit\!\!\cdot\phi$ states that somewhere in the past ϕ was true [18]. This operator is called the once-operator. The temporal operator $\odot\phi$, the so-called previously-operator, states that in the previous state ϕ was true.

Definition 3.6 defines the predicate $After(a,b)$ [18]. This predicate states that action b happened after action a. In the logic of Datta et al. [18], this predicate is a transitive relation for honest principals. This is due to the fact that the messages sent by honest principals are unique – note that this does not necessarily refer to the content of the message, as the uniqueness may be achieved by introducing sequence counters on the messages. Additionally, the statement $ActionsInOrder(a_1,\ldots,a_n)$, as defined in Equation 3.34, can be used to refer to a chain of multiple after-predicates [17].

$$
ActionsInOrder(a_1,\ldots,a_n) = After(a_1,a_2) \wedge \ldots \wedge After(a_{n-1},a_n) \tag{3.34}
$$

Definition 3.6 (After [18])**.** The predicate $After(a,b)$ states that action b happened after action a. This predicate is formalised as:

$$
After(a,b) = \diamondsuit\!\!\cdot(b \wedge \odot\diamondsuit\!\!\cdot a) \tag{3.35}
$$

Semantics

The main semantic relation in the protocol logic of Datta et al. [18] is $Q, R \vDash \phi$, which states that in run R of protocol Q formula ϕ holds. Additionally, $\bar{Q}$ denotes the set of

all initial configurations of protocol Q – including possible intruder cords. $Runs(\bar{Q})$ denotes the set of all runs of protocol Q with attacker. If ϕ has free variables we have $Q, R \vDash \phi$ if we can substitute all free variables in ϕ using substitutions σ such that $Q, R \vDash \sigma\phi$. Additionally, we write $Q \vDash \phi$ if $Q, R \vDash \phi$ holds for all runs $R \in Runs(\bar{Q})$.

To describe the reaction steps of the cord calculus, Datta et al. [18] define the statement $Event(R, X, P, \vec{n}, \vec{x})$, as shown in Equation 3.36, to refer to events. This predicate is used to refer to an event in a certain run R in process X which executes the actions in P that receive the data of $\vec{n}$ into the variables $\vec{x}$. In a comparable fashion, $Last(R, X, P, \vec{n}, \vec{x})$ states that $Event(R, X, P, \vec{n}, \vec{x})$ is the last event in run R. In Equation 3.36, S and S' represent the head and tail of the cord executed by X in which action P is happening. Furthermore, C and C' represent the other cords in run R.

$$Event(R, X, P, \vec{n}, \vec{x}) = (([SPS']_X \cup C \rightarrow [SS'(\vec{n}/\vec{x})]_X \cup C') \in R) \tag{3.36}$$

The semantic for the first modal form is as follows: let $R = R_0 R_1 R_2$ for some R_0, R_1 and R_2 [18]. We have $Q, R \vDash \phi[P]_A\psi$ if P does not match $[R_1]_A$ or if P does match $[R_1]_A$ and $Q, R_0 \vDash \sigma\phi$ implies $Q, R_0 R_1 \vDash \sigma\phi$ with substitution σ matching P to $[R_1]_A$. The definition for the second modal can be reached by setting ϕ in the first modal to true and defining R_0 to be empty. Thus, if we let $R = R_1 R_2$, we have $Q, R \vDash [P]_A\psi$ if P does not match $[R_1]_A$ or if P does match $[R_1]_A$ and $Q, R_1 \vDash \sigma\phi$ with substitution σ matching P to $[R_1]_A$.

The semantic entailment states that if $Q \vDash \Gamma$ implies $Q \vDash \phi$ we have $\Gamma \vDash \phi$ [18]. This means that if the set of environment formulas Γ is true in every run of protocol Q, formula ϕ is also true.

Actions The semantic formulas for the send, receive and new actions can be respectively found in Equation 3.37, Equation 3.38 and Equation 3.39 [18]. Those formulas link the events from the cord calculus to the action formulas discussed in Section 3.1.2. For example, Equation 3.37 states that if the last event in run R in process A was the execution of $\langle m \rangle$, i.e. the sending of variable m, $Q, R \vDash Send(A, m)$ will hold. The semantic formulas for receiving and fresh generation also contain an input of m and output of x. For example, the in Equation 3.39 displayed formula states that, if the last action in run R in process A was (νx), i.e. the fresh generation of x, with input m and output x, $Q, R \vDash New(A, m)$ will hold. In this formula, the input is the new value m which is bound to variable x, i.e. the substitution (m/x) as shown in the reaction step in Equation 3.29 is executed.

$$Last(R, A, \langle m \rangle, \emptyset, \emptyset) \Rightarrow Q, R \vDash Send(A, m) \tag{3.37}$$

$$Last(R, A, (x), m, x) \Rightarrow Q, R \vDash Receive(A, m) \tag{3.38}$$

$$Last(R, A, (\nu x), m, x) \Rightarrow Q, R \vDash New(A, m) \tag{3.39}$$

The semantics for the decryption and signature verification actions are a bit more complex. The semantic for decryption, which is shown in Equation 3.40, does not only require the last event to be $(\{\!|\ m\ |\!\}_K / \{\!|\ x\ |\!\}_{K^{-1}})$, but also states that A should possess

$\{\!|\ m\ |\!\}_K$ [18]. This requirement is needed, as decrypting $\{\!|\ m\ |\!\}_K$ is not possible without possessing it.

$$Last(R, A, (\{\!|\ m\ |\!\}_K / \{\!|\ x\ |\!\}_{K^{-1}}), m, x) \wedge Q, R \vDash Has(A, \{\!|\ m\ |\!\}_K) \Rightarrow Q, R \vDash Decrypt(A, \{\!|\ m\ |\!\}_K) \quad (3.40)$$

Comparable to the semantic definition of decryption, the in Equation 3.41 defined semantic for the signature verification action additionally requires that A possesses $\{\!|\ m\ |\!\}_{K^{-1}}$, m and K. Please note that, due to the nature of signature verification, no input and output variables are defined on the event.

$$Last(R, A, (\{\!|\ m\ |\!\}_{K^{-1}} / \{\!|\ m\ |\!\}_K), \emptyset, \emptyset) \wedge Q, R \vDash Has(A, \{\!|\ m\ |\!\}_{K^{-1}}) \wedge Q, R \vDash Has(A, m) \wedge Q, R \vDash Has(A, K) \Rightarrow Q, R \vDash Verify(A, \{\!|\ m\ |\!\}_{K^{-1}}) \quad (3.41)$$

Predicates The predicate of possession is formalised as shown in Equation 3.42 where $Has_i(A, m)$ is defined using mathematical induction with the base step shown in Equation 3.43 and the induction step shown in Equation 3.45 [18].

$$Has_i(A, m) \Rightarrow Q, R \vDash Has(A, m) \quad (3.42)$$

The base step states shown in Equation 3.43 that $Has_0(A, m)$ is true when the term m was directly received or generated or when it was a free variable of the role [18]. It trivially follows that receiving or generating a variable results in possessing it. Additionally, if m is contained in the set of free variables, it is a previously declared free variable in process A and therefore possessed by A.

$$(m \in FreeVariables([R]_A)) \vee Event(R, A, (vx), m, x) \vee Event(R, A, (x), m, x) \Rightarrow Has_0(A, m) \quad (3.43)$$

The induction step defines that $Has_{i+1}(A, m)$ is true if the term m was obtained in i steps by either decomposition by decryption, decomposition by normal pattern matching, composition by pattern matching, composition by construction of a tuple, composition by encryption, or by computation of a Diffie-Hellman secret [19] and is displayed in Equation 3.45 [18].

Given the complexity of the formula of the induction step, we will discuss it broken down in pieces. First of all, both the decompositions are contained in the part displayed in Equation 3.44, which is true if A has a term m' in i steps and it satisfies either of two forms of decomposition [18]. Namely, the first option requires m' to be $p(t)$ encrypted by K, i.e. $\{\!|\ p(t)\ |\!\}_K$, where t is the message m we are concerned with, and that there was a decryption event concerning m'. In other words, m is possessed

by A when it was obtained in encrypted form and decrypted by A. The second option requires m' to fit a pattern $p(t)$, where t is once again the message m of concern, and a pattern matching event concerning this m' in the past.

$$\begin{aligned}(&Has_i(A, m')\\ &\wedge ((m' = \{|\ p(t)\ |\}_K \wedge m = t \wedge Event(R, A, (m'/\{|\ p(y)\ |\}_{K^{-1}}), t, y))\\ &\vee (m' = p(t) \wedge m = t \wedge Event(R, A, (m'/p(y)), t, y))))\end{aligned} \tag{3.44}$$

The compositions are given by the other parts of Equation 3.45 [18]. When A has a term m' and a term m'', he can obtain both possible tuples of these values, i.e. m', m'' and m'', m'. Additionally, when A has a term m' and a key K, he can obtain the term m' encrypted using K, i.e. $\{|\ m'\ |\}_K$. Finally, when A has a value a and a Diffie-Hellmann exponential g^b, he is able to obtain the Diffie-Hellmann secret g^{ab}, which is equal to g^{ba}.

$$\begin{aligned}(&Has_i(A, m') \wedge ((m' = \{|\ p(t)\ |\}_K \wedge m = t \wedge Event(R, A, (m'/\{|\ p(y)\ |\}_{K^{-1}}), t, y))\\ &\vee (m' = p(t) \wedge m = t \wedge Event(R, A, (m'/p(y)), t, y))))\\ &\vee (Has_i(A, m') \wedge Has_i(A, m'') \wedge ((m = m', m'') \vee (m = m'', m')))\\ &\vee (Has_i(A, m') \wedge Has_i(A, K) \wedge m = \{|\ m'\ |\}_K)\\ &\vee (Has_i(A, a) \wedge Has_i(A, g^b) \wedge m = g^{ab}) \vee (Has_i(A, g^{ab}) \wedge m = g^{ba})\\ &\Rightarrow Has_{i+1}(A, m)\end{aligned} \tag{3.45}$$

Freshness, as shown in Equation 3.46, is defined by the fact that term m was generated by A in the past or was the result of a function applied to a term generated by A in the past [18]. Furthermore, this term m or some term m' containing it should not have been sent. Equation 3.47 states that $Q, R \vDash Honest(A)$ holds when A is contained in $HonestPrincipals(C)$, where C is the initial configuration for R. In Equation 3.48 defines that a term t_1 contains t_2 when this is a visible subterm, i.e. $t_2 \subseteq_v t_1$. A term t_2 is a visible subterm of t_1 when $t_2 \subseteq t_1$ and one of the occurrences of t_2 in t_1 is not encapsulated as a parameter of a one-way function. For example, $n \nsubseteq_v g(n)$ when $g(n)$ is a one-way function.

$$\begin{aligned}Q, R \vDash (&\diamondsuit New(A, m) \vee (\diamondsuit New(A, n) \wedge m = g(n)))\\ &\wedge \neg(\diamondsuit Send(A, m') \wedge m \subseteq m') \Rightarrow Q, R \vDash Fresh(A, m)\end{aligned} \tag{3.46}$$

$$A \in HonestPrincipals(C) \Rightarrow Q, R \vDash Honest(A) \tag{3.47}$$

$$t_2 \subseteq_v t_1 \Rightarrow Q, R \vDash Contains(t_1, t_2) \tag{3.48}$$

Finally, the logic semantics are defined intuitively. Equation 3.49 defines the and-composition as the logical and of the two separate statements [18]. The negation of a statement is defined in Equation 3.50. The exists-operator shown in Equation 3.51 is

slightly more complex. It defines that $Q, R \vDash \exists x : \phi$ is true when $Q, R \vDash (d/x)\phi$ holds for some d, where $(d/x)\phi$ denotes the substitution of x for d in ϕ.

$$(Q, R \vDash \phi_1) \wedge (Q, R \vDash \phi_2) \Rightarrow Q, R \vDash (\phi_1 \wedge \phi_2) \tag{3.49}$$

$$Q, R \nvDash \phi \Rightarrow Q, R \vDash \neg\phi \tag{3.50}$$

$$Q, R \vDash (d/x)\phi \Rightarrow Q, R \vDash \exists x : \phi \tag{3.51}$$

The temporal logic statement $\diamondsuit\!\!\cdot\phi$ is semantically defined in Equation 3.52 where R' is a prefix of R [18]. In other words $\diamondsuit\!\!\cdot\phi$ is true, if in some prefix of that run ϕ was true. Comparably, Equation 3.53 defines the previously-operator where $R = R''e$ for some event e. In other words, $\odot\phi$ is true, if ϕ was true in the previous state.

$$Q, R' \vDash \phi \Rightarrow Q, R \vDash \diamondsuit\!\!\cdot\phi \tag{3.52}$$

$$Q, R'' \vDash \phi \Rightarrow Q, R \vDash \odot\phi \tag{3.53}$$

3.1.3 Proof System

In this section, we will briefly introduce the proof system of Datta et al. [18]. At this point, we will not yet adapt it to a service oriented context, but only familiarise ourselves with it. We will start with discussing the basic axioms and rules of the proof system, followed by those for the temporal ordering and the formalisation of honesty. Finally, we will discuss protocol composition itself.

Protocol Actions In Section 3.1.2, we discussed the protocol actions on a syntactical and semantic level. At this point, we will consider the axioms for those actions. First of all, Equation 3.54 states that if a principal $\hat{X}$ has executed an action a in some role, the predicate b stating that this action occurred in the past is true [18]. For example, if principal $\hat{X}$ executed $\langle m \rangle$ in process X, $Send(X, m)$ would be true, i.e. we would have the situation shown in Equation 3.55.

$$\phi[a]_X \diamondsuit\!\!\cdot(b \wedge \odot\phi) \tag{3.54}$$

$$\phi[\langle m \rangle]_X \diamondsuit\!\!\cdot(Send(X, m) \wedge \odot\phi) \tag{3.55}$$

Equation 3.56 states that if process X generates a new value n and does not execute any other action he possesses this value [18]. Additionally, Equation 3.57 states that only he has this value and Equation 3.58 states that this value is fresh. The axiom of Equation 3.59 defines that if one receives a value m one also possesses this value.

$$\phi[(\nu n)]_X Has(X, n) \tag{3.56}$$

$$\phi[(\nu n)]_X \wedge Has(Y, n) \Rightarrow Y = X \tag{3.57}$$

$$\phi[(\nu n)]_X Fresh(X, n) \tag{3.58}$$

$$\phi[(m)]_X Has(X, m) \tag{3.59}$$

Protocol Predicates As stated before, Datta et al. [18] use the attacker model of Dolev and Yao [22]. To capture this, four axioms are included within the proof system. Firstly, Equation 3.60 states that if X decrypts $\{\!|\ n\ |\!\}_K$, he knows n. Secondly, Equation 3.61 states that if X possesses the tuple (x, y), he also possesses x and y. Thirdly, it is required that one possesses the corresponding secret key in order to decrypt an encrypted message, as shown in Equation 3.62. Please note that the secret key is expected to be possessed only by the owner of this key and that honesty of $\hat{X}$ is required, as only honest participants are expected to keep their secret keys out of the hands of others. Fourthly and finally, the inability to forge signatures is stated in Equation 3.63. This axiom states that a message signed by a honest principal $\hat{X}$ has to be sent by that principal at some point in the past.

$$\diamondsuit Decrypt(X, \{\!|\ n\ |\!\}_K) \Rightarrow Has(X, n) \tag{3.60}$$

$$Has(X, (x, y)) \Rightarrow Has(X, x) \wedge Has(X, y) \tag{3.61}$$

$$\begin{aligned} &Honest(\hat{X}) \\ \wedge &\diamondsuit Decrypt(Y, \{\!|\ n\ |\!\}_X) \Rightarrow \hat{Y} = \hat{X} \end{aligned} \tag{3.62}$$

$$\begin{aligned} &Honest(\hat{X}) \\ \wedge &\diamondsuit Verify(Y, \{\!|\ n\ |\!\}_{X^{-1}}) \Rightarrow \exists X : (\exists m : (\diamondsuit Send(X, m) \wedge Contains(m, \{\!|\ n\ |\!\}_{X^{-1}}))) \end{aligned} \tag{3.63}$$

The uniqueness of nonces is captured in Equation 3.64, which states that a freshly generated value is distinct from any other freshly generated value [18]. Additionally, Equation 3.65 states that this value is also distinct from any Diffie-Hellman exponential [19]. Furthermore, if two processes have fresh values, those two values are also distinct via Equation 3.66.

$$\diamondsuit New(X, n) \wedge \diamondsuit New(Y, n) \Rightarrow X = Y \tag{3.64}$$

$$\diamondsuit New(X, p) \Rightarrow \neg \diamondsuit New(Y, g^p) \tag{3.65}$$

$$\diamondsuit Fresh(X, n) \wedge \diamondsuit Fresh(Y, n) \Rightarrow X = Y \tag{3.66}$$

Finally, Equation 3.67 states the subterm relationship of tuples [18]. This equation defines that the tuple (x, y) contains x and y using the predicate $Contains(t_1, t_2)$.

$$Contains((x, y), x) \wedge Contains((x, y), y) \tag{3.67}$$

Preservation and Loss The axiom in Equation 3.68 states the persistence of the predicate $Has(X, t)$ and the axiom in Equation 3.69 does the same for the predicate $\diamondsuit\phi$ [18]. For example, if one previously had a certain value, one continues to posses this value after executing more actions. Equation 3.70 defines that freshness is persistent, as long as it is not sent. Comparably, Equation 3.71 defines that sole possession of a value is persistent, as long as it is not sent visibly. It should be noted that the predicate $HasAlone(X, n)$, which is used in Equation 3.71, and means that principal X is the sole possessor of value n, is defined in Equation 3.72 [17].

$$Has(X,n)[a]_X\, Has(X,n) \tag{3.68}$$

$$\diamondsuit\phi[a]_X \diamondsuit\phi \tag{3.69}$$

$$Fresh(X,t)[a]_X\, Fresh(X,t) \text{ with } (t \not\sqsubseteq a) \vee (a \neq \langle m \rangle) \tag{3.70}$$

$$HasAlone(X,n)[a]_X\, HasAlone(X,n) \text{ with } (n \not\sqsubseteq_v a) \vee (a \neq \langle m \rangle) \tag{3.71}$$

$$HasAlone(X,n) = Has(X,n) \wedge (Has(Y,n) \Rightarrow X = Y) \tag{3.72}$$

As logically follows from the definition of freshness, it is lost when the term is sent [18]. This loss of freshness is captured in Equation 3.73.

$$Fresh(X,t)[\langle m \rangle]_X \neg Fresh(X,t) \text{ with } t \sqsubseteq m \tag{3.73}$$

Inference Rules The inference rules for the proof system of Datta et al. [18] are shown in Equation 3.74 through Equation 3.77. The rule in Equation 3.74 states that if two formulas exist containing the same precondition and actions with two different postconditions, both those postconditions hold for that prefix. In other words, when one is able to draw two conclusions from the same statement, he can also draw the logical and of those two as conclusion.

$$\frac{\phi[P]_X\psi \qquad \phi[P]_X\theta}{\phi[P]_X(\psi \wedge \theta)} \tag{3.74}$$

Equation 3.75 allows the strengthening of the logical precondition and the widening of the logical postcondition. In this case, strengthening refers to swapping the precondition ϕ with a stronger precondition ϕ' that is at least false in all cases that ϕ is false and widening refers to swapping the postcondition ψ with a weaker precondition ψ' that is at least true in all cases that ψ is true.

$$\frac{\phi[P]_X\psi \qquad \phi \subset \phi' \qquad \psi' \subset \psi}{\phi'[P]_X\psi'} \tag{3.75}$$

Furthermore, Equation 3.76 states that a valid logical statement, i.e. it is true, will also hold when a certain precondition and accompanying actions are placed in front of it [18]. In other words, when something is already true, it will also be true when a precondition is added to it. In essence, this postcondition behaves like an invariant, as it is true regardless of the precondition. To illustrate this, imagine that a certain condition ψ is always true. In this case, you can also state that when another condition ϕ is satisfied ψ is true, which is correct, as ψ is always true.

$$\frac{\psi}{\phi[P]_X\psi} \tag{3.76}$$

Finally, Equation 3.77 describes how the exists-operator can be substituted by a constant. This inference rule allows the mechanism of matching and substitution to propagate through the exists-operator.

$$\frac{\exists x : \phi(x)}{\phi(c_0)} \tag{3.77}$$

Temporal Ordering As stated before, the framework of Datta et al. [18] uses temporal ordering. This is required for their authentication property, which is based upon the notion of matching records of runs of Diffie, Oorschot and Wiener [20]. We will discuss this authentication property and other notions in depth in Section 3.3.

The temporal ordering of actions within a certain role is defined in Equation 3.78. In this axiom, it is shown that the actions in the cord of a role are executed in the order in which they are written. Furthermore, in Equation 3.79, we can see that the $After(a_1, a_2)$ relation is transitive when dealing with honest principals.

$$\phi[a_1 \dots a_n]_X After(a_1, a_2 \wedge \dots \wedge After(a_{n-1}, a_n) \tag{3.78}$$

$$\begin{aligned} &Honest(\hat{X}) \wedge Honest(\hat{Y}) \wedge Honest(\hat{Z}) \\ \Rightarrow (&After(a_1(X), a_2(Y)) \wedge After(a_2(Y), a_3(Z)) \Rightarrow After(a_1(X), a_3(Z))) \end{aligned} \tag{3.79}$$

The temporal axiom in Equation 3.80 states that any action in any other process that uses a nonce freshly generated by X has to happen after this nonce has been sent by X, i.e. any action a_2 by i that involves the term n that is freshly generated by X has to happen after the send-action [18]. Equation 3.81 is similar, but has the roles of X and Y reversed. This axiom states that when a process X uses a term that was generated in and sent by some process Y in some action a_2, this action a_2 has to happen after Y performed the send-action. The proof of Equation 3.81 can be derived using the same reasoning as the proof of Equation 3.80, which is as follows.

$$\frac{Fresh(X, n)[\langle m \rangle]P]_X(\phi \Rightarrow \diamondsuit a_2(Y))}{Fresh(X, n)[\langle m \rangle]P]_X(\phi \Rightarrow After(Send(X, m), a_2))} \text{ with } \begin{array}{l}(X \neq Y) \\ \wedge (n \subseteq m, a_2)\end{array} \tag{3.80}$$

$$\frac{\phi[Pa_2]_X(\psi \Rightarrow \diamondsuit(Send(Y, m) \wedge \odot Fresh(Y, n)))}{\phi[Pa_2]_X(\psi \Rightarrow After(Send(Y, m), a_2))} \text{ with } \begin{array}{l}(X \neq Y) \\ \wedge (n \subseteq m, a_2)\end{array} \tag{3.81}$$

Temporal ordering of fresh values [18]. Given that $X \neq Y$ and $n \subseteq m, a_2$ we need to show that

$$Fresh(X, n)[\langle m \rangle P]_X(\phi \Rightarrow After(Send(X, m), a_2)) \tag{3.82}$$

follows from

$$Fresh(X, n)[\langle m \rangle P]_X(\phi \Rightarrow \diamondsuit a_2(Y)). \tag{3.83}$$

Let $R = R_0 R_1 R_2$ be a run of Q where R_1 matches $\langle m \rangle P$ under substitution σ with $Q, R_0 \vDash Fresh(X, n)$. Now, we need to show that

$$Q, R_0 R_1 \vDash \sigma(\phi \Rightarrow After(Send(X, m), a_2)). \tag{3.84}$$

This holds trivially when $Q, R_0 R_1 \vDash \sigma(\neg\phi)$. When $Q, R_0 R_1 \vDash \sigma\phi$, it follows that $Q, R_0 R_1 \vDash \diamondsuit a_2(Y)$. Given the semantics defined in Equation 3.46 and Equation 3.52, it also follows that Equation 3.84 is true. □

The temporal logic relies on a set of basic axioms. The axioms in Equation 3.85 and Equation 3.86 state that when the once-operator is applied to two formulas combined with, respectively, an and-operator and an or-operator, this statement can be transformed to, respectively, the logical and and the logical or of the two separate formulas under the once-operator [18]. Furthermore, Equation 3.87 shows that the previously-operator and the inverse-operator can be swapped when they occur successively.

$$\diamondsuit(\phi \wedge \psi) \Rightarrow \diamondsuit\phi \wedge \diamondsuit\psi \tag{3.85}$$

$$\diamondsuit(\phi \vee \psi) \Rightarrow \diamondsuit\phi \vee \diamondsuit\psi \tag{3.86}$$

$$\odot\neg\phi \Leftrightarrow \neg \odot \phi \tag{3.87}$$

Finally, we have the temporal generalisation rule, which is shown in Equation 3.88 [18]. This rule states that when a formula ϕ is unconditionally true, the statement $\neg\diamondsuit\neg\phi$ is also true. Basically, this means that if something is always true, it cannot be that it has been false at some point in the past.

$$\frac{\phi}{\neg\diamondsuit\neg\phi} \tag{3.88}$$

Honesty In the proof system of Datta et al. [18], the honesty rule is an essential invariant. Basically, it states that a honest principal will act as described by the protocol and, thus, that properties concerning the protocol can be derived from the assumption of honesty. For example, if Alice communicates with Bob and Bob sends her a certain message, Alice can use information corresponding with the role Bob plays in the protocol to deduce how he generated his reply.

As stated before, honest participants are those participants that act as defined by the protocol [18]. In other words, the honest principals are those that have a role in the protocol and execute the actions defined by that role. An attacker will try to break the system and will thus perform unexpected actions in order to reach his intended goal of unexpected behaviour.

A so-called basic step B of a role ρ in a certain protocol Q is a continuous segment of actions of this role such that B is the empty sequence, B starts at the beginning of the role and continues to the first receive-action, B starts at a receive-action and ends at the next receive-action, or B starts at the last receive-action and continues to the end of the role. We define the statement $BasicSequences(\rho)$ as the set of all basic steps in ρ.

The honesty rule itself states that, if ϕ holds at the start of every role of protocol Q and is preserved by all its atomic sequences, then every honest principal in the protocol satisfies ϕ [18]. If we let $\rho \in Q$ mean role ρ in protocol Q, this can be formalised as shown in Equation 3.89.

$$\frac{[]_X\phi \quad \forall \rho \in Q : (\forall P \in BasicSequences(\rho) : (\phi[P]_X\phi))}{Q \vdash Honest(\hat{X}) \Rightarrow \phi} \tag{3.89}$$

Composition As stated before, sequential composition is possible when the post-condition of the first process is stronger than or equal to the precondition of the second process [18, 17]. Furthermore, the environment invariants need to be weakened, by combining the assumptions of the separate invariants. Finally, one needs to show that the invariants of the first process hold for the second and vice versa. In the case of parallel composition, the first step can be omitted. These methods of composition can be derived from the composition rules shown in this paragraph.

$$\frac{\Gamma \vdash \Theta}{\Gamma \cup \Gamma' \vdash \Theta} \tag{3.90}$$

Equation 3.90, is known as the weakening rule [18]. This rule states that if Θ is provable from the hypothesis Γ, it is also provable from a larger set of hypotheses that contains Γ, i.e. $\Gamma \cup \Gamma'$. The weakening rule is trivially sound, as $\Gamma \vdash \Theta$ already implies $\Gamma \cup \Gamma' \vdash \Theta$.

$$\frac{\Gamma \vdash \phi[P]_A\psi \qquad \Gamma \vdash \psi[Q]_X\theta}{\Gamma \vdash \phi[PQ]_A\theta} \tag{3.91}$$

Furthermore, Equation 3.91 gives the basis for sequential composition, by stating that if the postcondition ψ of the first service matches the precondition of the second service, they can be composed safely, given that the invariant Γ holds in both separate cases [18]. Please note that, in fact, the postcondition of the first service has to be stronger than or equal to the precondition of the second service, due to the inference rule in Equation 3.75. This inference rule can be proved as follows.

Sequential composition rule [18]. To prove the sequential composition rule as displayed in Equation 3.91, we need to show that $\Gamma \vDash \phi[PQ]_A\theta$ can be proven from $\Gamma \vDash \phi[P]_A\psi$ and $\Gamma \vDash \psi[Q]_A\theta$ given a set of formulas Γ. If we assume a protocol T and we have $T \nvDash \Gamma$, this is trivially true. When we have $T \vDash \Gamma$, we have to show that $T \vDash \phi[PQ]_A\theta$ can be proven from $T \vDash \phi[P]_A\psi$ and $T \vDash \psi[Q]_A\theta$. Let $R = R_0R_1R_2$ be a run of T such that R_1 matches $[PQ]_A$ under substitution σ such that formula $T, R_0 \vDash \sigma\phi$. Run R can be written as $R = R_0R_1'R_1''R_2$, where R_1' matches $[P]_A$ under substitution σ and R_1'' matches $[Q]_A$ under substitution σ. It follows that $T, R_0R_1' \vDash \sigma\psi$ holds and, thus, that $T, R_0R_1'R_1'' \vDash \sigma\theta$ holds. □

$$\frac{P \vdash \Gamma \qquad Q \vdash \Gamma}{P \circ Q \vdash \Gamma} \tag{3.92}$$

Finally, the last rule, shown in Equation 3.92, expresses that if the invariant Γ holds for two individual services P and Q, it also holds for the parallel composition of those services $P \circ Q$. In this formula, the operator $\circ$ on two protocols P and Q is defined to return the concatenation of basic sequences of the roles in P and Q. The proof for this inference rule is as follows.

Parallel composition rule [18]. To prove the parallel composition rule as displayed in Equation 3.92, we need to show that $P \circ Q \vDash \Gamma$ can be proven from $P \vDash \Gamma$ and $Q \vDash \Gamma$.

By definition of the honesty rule, as shown in Equation 3.89, every basic sequence of a role in $P \cup Q$ is either a basic sequence B of $P \cup Q$ or a concatenation of two basic sequences in $P \cup Q$. In the first case, it trivially follows that $P \circ Q \vDash \psi[B]_X \phi$, and in the second case, the same follows after the application of the sequential composition rule of Equation 3.91. □

From the discussed composition rules, we can deduce a general methodology for composing two protocols. This methodology is shown in Theorem 3.3. In this methodology, it can be seen that Equation 3.91 is only used in the case of sequential composition. Furthermore, Equation 3.92 is always used, as the protocols need to respect the environment invariants of the other protocol.

Theorem 3.3 (General composition methodology [18])**.** *The composition of two protocols Q_1 and Q_2 is as follows:*

1. *prove the security properties of the two separate protocols, i.e. show that $Q_1 = \phi_1[P_1]_X \psi_1$ and $Q_2 = \phi_2[P_2]_X \psi_2$ are valid;*
2. *identify the environment invariants Γ_1 and Γ_2 such that $\Gamma_1 \vdash Q_1$ and $\Gamma_2 \vdash Q_2$;*
3. *apply the weakening rule, as shown in Equation 3.90, such that we have $\Gamma' = \Gamma_1 \cup \Gamma_2$;*
4. *when concerned with sequential composition, check if the postcondition ψ_1 of Q_1 matches the precondition ϕ_2 of Q_2 – possibly using the axiom in Equation 3.75 – and sequentially compose Q_1 and Q_2 to $\phi_1[P_1 P_2]_X \psi_2$ using the inference rule in Equation 3.91 if they match; and*
5. *prove that Γ' holds for both protocols by proving $Q_1 \vdash \Gamma_2$ and $Q_2 \vdash \Gamma_1$ and applying Equation 3.92 afterwards.*

Adverse and Insecure Protocols Besides the composition of secure protocols, with the goal of creating one large protocol that preserves the security properties, insecure or adverse protocols may be considered. An insecure protocol refers to a protocol that breaks the security properties it was built for, e.g. an authentication protocol that authenticates principals that are not authorised. On the other hand, an adverse protocol is a result of a so-called chosen protocol attack [39]. This is a type of attack where a very powerful attacker writes a protocol using the same key material that breaks the security properties of the protocol under attack. It should be noted that such a chosen protocol attack is theoretically possible on every protocol, but that it probably requires a very powerful attacker and that there are no widely known canonical methods for deriving such an adverse protocol.

In the context of this research, one can imagine that protocols that do not preserve security properties on their own cannot be composed with other protocols that require that those security properties are satisfied. Furthermore, a protocol that intends to break the security properties of a certain protocol cannot be composed with that protocol, merely due to the fact that their invariants conflict in an unresolvable manner.

3.2 Secure Composition of Services

We will adapt the rules for the secure composition of protocols of Datta et al. [18], which we discussed in Section 3.1, to service orientation, whereby the processes P correspond with services S. Simply put, in a general composition every service S can be seen, in the context of the framework discussed in Section 3.1, as a role ρ of a very large protocol Q – please note the differences between services and protocols and the architectures underneath those two still persist. Thus, we will see the execution of a service as the execution of a process. It should be noted that such a service will commonly be an atomic service. Nevertheless, there may be specific conditions under which a composite service is perceived as if it were an atomic service. For example, when a provider offers services that are composites in the internal infrastructure of that provider.

This means that if we have $\Gamma_{S_i} \vdash \phi_{S_i} S_i \psi_{S_i}$, we know that when the invariant Γ_{S_i} is satisfied the postcondition ψ_{S_i} will hold when the precondition ϕ_{S_i} is true. Of course, this is still under the assumptions, and thus the attacker model, of Datta et al. [18]. We will extend our adversarial model in Section 3.5.

Given that services can be seen as interfaces to a remote computing capability, a service consists of an invocation with a set of parameters, the actual capability and the result returned. Thus, if we have a service S that is invoked using parameters $(p_0, \ldots, p_n)$, runs actions A and returns values $(r_0, \ldots, r_m)$, we would have $[(x)P\langle y\rangle]_S$, where x and y are tuples used to. respectively receive the parameters and return the results. In other words, before running the actions P, the service receives the parameters into x, and afterwards, the service sends the results in y.

Service Compositions In order to prove the security of a service composition, we can start with the proof of the smallest parts of this composition, and extent this proof to include more elements. Following this technique, eventually we will have a proof of the complete composition – or not, if the security properties are not satisfied. For this, we can use the proving techniques for parallel and sequential compositions based on the composition rules shown in Section 3.1.3.

Now, if we have a certain service composition, consisting of parallel and sequential composed services. We can prove the security of this composition in several steps. For a parallel composition we can follow the method of Theorem 3.4 and for sequential composition the method of Theorem 3.5. It should be noted that these theorems are based on the general theorem for protocol composition, which can be found in Theorem 3.3.

Theorem 3.4 (Parallel Composition [17])**.** *To prove the parallel composition of two services S_1 and S_2:*

1. *prove the security of S_1 and S_2 separately, i.e. prove that $\phi_{S_1} S_1 \psi_{S_1} \vdash \Gamma_{S_1}$ and $\phi_{S_2} S_2 \psi_{S_2} \vdash \Gamma_{S_2}$ hold;*
2. *weaken the environment invariant of both services to $\Gamma_{S_1} \cup \Gamma_{S_2}$ using the weakening rule of Equation 3.90; and*

3. *prove* $\phi_{S_1} S_1 \psi_{S_1} \vdash \Gamma_{S_1} \cup \Gamma_{S_2}$ *and* $\phi_{S_2} S_2 \psi_{S_2} \vdash \Gamma_{S_1} \cup \Gamma_{S_2}$*, which can, following Equation 3.92, be simplified to proving* $\phi_{S_1} S_1 \psi_{S_1} \vdash \Gamma_{S_2}$ *and* $\phi_{S_2} S_2 \psi_{S_2} \vdash \Gamma_{S_1}$*.*

Theorem 3.5 (Sequential Composition [17])**.** *To prove the sequential composition of two services* S_1 *and* S_2*:*

1. *prove the security of* S_1 *and* S_2 *separately, i.e. prove that* $\phi_{S_1} S_1 \psi_{S_1} \vdash \Gamma_{S_1}$ *and* $\phi_{S_2} S_2 \psi_{S_2} \vdash \Gamma_{S_2}$ *hold;*
2. *weaken the environment invariant of both services to* $\Gamma_{S_1} \cup \Gamma_{S_2}$ *using the weakening rule of Equation 3.90;*
3. *prove that* ψ_{S_1} *is stronger than or equal to* ϕ_{S_2}*, in order to allow sequential composition to* $\phi_{S_1} S_1 S_2 \psi_{S_2}$*; and*
4. *prove* $\phi_{S_1} S_1 \psi_{S_1} \vdash \Gamma_{S_1} \cup \Gamma_{S_2}$ *and* $\phi_{S_2} S_2 \psi_{S_2} \vdash \Gamma_{S_1} \cup \Gamma_{S_2}$*, which can, following Equation 3.92, be simplified to proving* $\phi_{S_1} S_1 \psi_{S_1} \vdash \Gamma_{S_2}$ *and* $\phi_{S_2} S_2 \psi_{S_2} \vdash \Gamma_{S_1}$*.*

Partially Ordered Multisets As we model service compositions as partially ordered multisets (pomsets), we will broaden the discussed methods of compositing two services to the verification of a complete service composition. Presume that we have a pomset $P = [V, \Sigma, \leq, \mu]$, that represents a service composition with services Σ and V as individual invocations. To securely compose this service composition, we need to show that any two service invocations $v, w \in V$ can be composed in parallel. Additionally, for all two service invocations $v, w \in V$ with $v \leq w$, we have to show that sequential composition is possible. If we formalise this method of verifying a service composition, we get Theorem 3.6.

Theorem 3.6 (Service Composition)**.** *Let pomset* $P = [V, \Sigma, \leq, \mu]$ *be a service composition consisting of services* Σ*, individual invocations* V*, and ordering* $\leq$*. To prove the security of this service composition, for every two* $v, w \in V$*:*

1. *prove that* $\phi_v v \psi_v \vdash \Gamma_v$ *and* $\phi_w w \psi_w \vdash \Gamma_w$ *hold;*
2. *apply Equation 3.90 such that* $\Gamma = \Gamma_v \cup \Gamma_w$*;*
3. *if* $v \leq w$*, prove that* ψ_v *is stronger than or equal to* ϕ_w*, or the other way around; and*
4. *prove* $\phi_v v \psi_v \vdash \Gamma_w$ *and* $\phi_w w \psi_w \vdash \Gamma_v$*.*

Following the model of Pavlovic [58], we can transform the original cord spaces of Datta et al. [18] to our pomset-based model. In the theory of Cervesato, Meadows and Pavlovic [16], the reaction steps of cord spaces, which were discussed in Section 3.1.1, are left for the introduction of an implicit ordering between send and receive actions. In other words, in each run, every receive action (x) has a chosen predecessor $\langle t \rangle$ [59]. More formally, in a run of a service composition, each receive action gets assigned a send action, such that receive action does not occur before the coupled send action. This model of ordering originates from the model of Lamport [41].

Opaque Services If we can assume that a honest service will take care of the first step of both composition methods, we still have a problem raised by opaque services. Namely, the fact that the first service needs to be proved to hold under the environment invariant of the second service. For an opaque service, it is not possible to make this proof, as it requires looking inside the service – given that the invariants of the two composed services are not equal. Therefore, we formulate Research Question 3.

Research Question 3. How can limitations raised by opaque services when applying the secure composition technique of Datta et al. [17] be solved?

State Explosion Following our pomset-based view on services, which we discussed in section 2.3, a service S is an entity that has a precondition ϕ_S, a postcondition ψ_S and an invariant Γ_S, thus, a service forms its own cord $\phi_S S \psi_S$ under Γ_S. These services are partially ordered, such that services dependent on other services come after their dependencies in a service composition.

Applying the composition proofs to the complete process, the set of pomsets, will inevitably result in an explosion of states and, thus, in an explosion of proofs. Where for a plain service composition there already is a set of proofs per part required, the complete system needs a set of proofs per possible service composition, given the partial ordering. This raises Research Question 4.

Research Question 4. How can the state explosion when applying the secure composition technique of Datta et al. [18] be prevented?

3.3 Security Properties and Predicates

As it is well known that security properties commonly are not preserved under composition or refinement [48, 52, 35, 73], we have to make sure that the security properties that we introduce do survive under composition. This requires that our security properties are safety properties that can be used to satisfy safety invariants over service traces, which we have built using pomsets [18]. Therefore, composition is possible with these security properties within our framework.

Obviously, in our model, not all invariants will be composable. In these cases, it should be impossible to derive a proof. For example, if we have an invariant stating ϕ and an invariant stating $\neg\phi$, these cannot be combined. However, this does not mean that our security properties cannot be composed in some cases, but merely that they are used in contradictory manners.

In order to build compositionally safe properties, we will start with unfolding a predicate based system for our model. In this model, services are annotated with preconditions, postconditions, and invariants, which make use of constructs that enable us to make claims about the data that is sent, received, or handled by the services.

As we build further upon the framework for secure composition of protocols of Datta et al. [18], which we discussed in Section 3.1, the predicates that we will discuss are also based upon the predicates in that framework. Therefore, we will focus on the differences from and additions to this framework in this section.

To start, the formula $Has(S, x)$, which is shown in Definition 3.7, expresses that a certain service S has some information x [18]. In this formula, possession of a piece of information means that this service has access to this information, e.g. if S can only access it in encrypted form, it is not accessible. The semantics and proof system surrounding the has-predicate are left unchanged.

Definition 3.7 (Has [18])**.** The annotation $Has(S, t)$ states that service S possesses term t, i.e. that service S knows term t.

Freshness, denoted by $Fresh(S, d)$ and defined in definition 3.8, means that the term d is generated by service S and has not been seen by any other service [43]. In other words, no other service possesses this term and it is completely new to the service that generated it. In cryptographic applications, this annotation can be used to describe nonces and randomisation values. As with the has-predicate, the semantics and proof system surrounding the fresh-predicate can be found in Section 3.1.

Definition 3.8 (Fresh [18])**.** The annotation $Fresh(S, t)$ states that service S generated term t freshly, i.e. this term has not been shared with any other service T with $S \neq T$.

Finally, we have honesty, which we already discussed in Section 3.1.3. As stated in Definition 3.9, a honest participant acts as prescribed. In other words, if the service description states which actions a service will perform and which messages it will send, the service will do so if it is honest. An attacker, on the other hand, may perform unexpected actions if this aids him in achieving his – probably malicious – goal.

Definition 3.9 (Honesty [18])**.** The annotation $Honest(S)$ states that the actions of service S are equal to those described in its definition.

Please note that a service that is defined to perform an unwanted action, is not dishonest. However, it will not be possible to compose this service such that this unwanted action is not performed. In such a composition, it is impossible to derive a proof of security, as the requirement of not performing the unwanted action will conflict with the definition of the service that performs this action.

Commonly, security is concerned with the confidentiality, integrity and availability of informations systems [7]. Of those terms, confidentiality regards the absence of disclosure of information to unauthorised principals [71]. Integrity refers to the absence of improper system alterations. Availability concerns the readiness for correct service.

In this paper, we will reason about both confidentiality and integrity. Therefore, we will constitute security properties to capture those two aspects. Confidentiality will be evaluated in terms of secrecy and integrity using authentication.

We do not concern availability, as we reason within a formalised model that the non-determinism required to regard availability. Additionally, some availability aspects can be regarded out of scope, as they happen within the service. It should be noted that this is also the reason why confidentiality and integrity have been specified in terms of, respectively, secrecy and authentication. In practice, there are many scenarios thinkable which also influence the confidentiality or integrity of services, but fall outside of the scope.

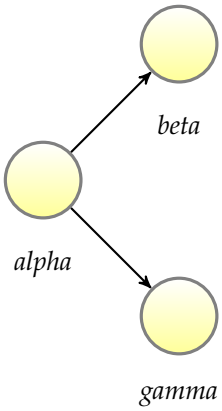


Figure 3.1: Example of a service composition to illustrate non-interference

Secrecy Firstly, we have secrecy annotations. We will define a term to be secret if only authorised principals have access to it [59]. In this report, we will tighten this definition to authorised services, which allows for more fine grained control and compliance with the uncoupled characteristic of services.

The well known notion of confidentiality of Smith [67] states that information from high security variables should never flow into low security variables If we consider this definition, one can understand that secret information should not leak on insecure channels.

Please note that we do not use a definition of strong secrecy or non-interference. Strong secrecy means that an attacker cannot observe any difference when a secret value changes [11]. Non-interference originates from Goguen and Meseguer [28] and means that, when two groups of people perform actions, what the second group sees is not affected by the actions of the first group [7]. Both these security properties relate to information flow analysis and seek to prevent that any detail about secret information is leaked.

To illustrate why non-interference is not feasible in our framework, imagine that we have a service composition consisting of three services: alpha, beta and gamma – see Figure 3.1 for an illustration. Now, suppose that when running this composition, based on a secret value, a choice is made to invoke either *beta* or *gamma* [71]. This would break the property of non-interference, as we could deduce information about the secret value from this run. However, this requires both reasoning about complete compositions and breaking the uncoupled characteristic of services. Therefore, we settle for a security property that requires that the actual value stays within authorised hands.

A real world example of non-interference can be found in Figure 3.2. In this figure, the dashed arrow represents a conditional invocation where the condition is specified by the text under the line. This example shows a travel agency that uses a service for payments. When a payment is successful, it will connect to another service to make

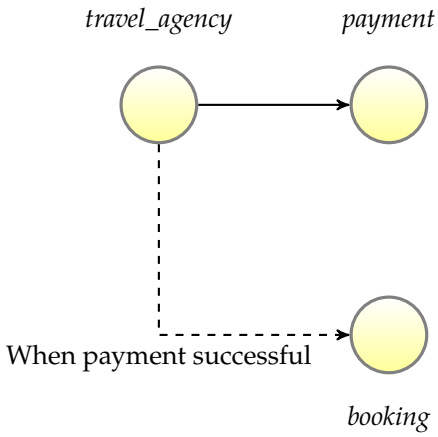


Figure 3.2: Example of a service composition for a travel agency to illustrate non-interference

any bookings. However, an outsider is able to observe whether a connection to the booking service is made and, thus, able to deduce whether the payment was successful. This could disclose information about the amount of money on your banking account.

It should be noted that, commonly, security properties can be defined into two categories [35]. The first category considers properties that are based on the idea that if x is a secret variable in a process P, an adversary should not be able to distinguish between processes $P[x/d_0]$ and $P[x/d_1]$, where x is substituted with d_0 in the first case and d_1 in the second case. The mentioned strong secrecy and non-interference properties fall into this category, which provides rather high security. However, as discussed, this category is very hard to compose, and commonly not composable at all.

The second category of security properties is the category of properties that require that a secret variable is never leaked to an untrusted channel or variable [35]. More specifically, no information from which the secret variable could be derived is to be sent in the clear. Our property and, for example, the property of Smith [67] fall into this category. This category finds its origin with the attacker model of Dolev and Yao [22] and allows much easier for composition.

A formalisation of our secrecy-statement can be found in Definition 3.10, which states that a term t can be asserted to be secret using the predicate $Secret((S_0, \ldots, S_n), t)$. In this predicate the services $(S_0, \ldots, S_n)$ constitute the authorised services. Following the definition of secrecy, it is not allowed for other principals to possess secret values. Therefore, Equation 3.93 states that if a service knows a secret value, he is one of the authorised services.

$$\diamondsuit Secret(\bar{S}, n) \wedge \diamondsuit Has(S, n) \Rightarrow S \in \bar{S} \tag{3.93}$$

Definition 3.10 (Secret terms)**.** A term is secret if only authorised services have access to it. The assertion $Secret((S_0, \ldots, S_n), t)$ means that term t is secret and that all services S_i for $0 \leq i \leq n$ are authorised to access it. Additionally, we have:

$$Secret((S_0, \ldots, S_n), t) = Secret(S_0, t) \wedge \ldots \wedge Secret(S_n, t) \tag{3.94}$$

Besides secret terms, we have public terms. As opposed to secret terms, everyone can access a public term. A piece of data d can be annotated to be public using $Public(d)$, as described in Definition 3.11. When a term is neither defined secret nor public, it defaults to being untyped. It should be noted that data may not change state from secret to public or the other way around.

Definition 3.11 (Public terms)**.** A term is public if all principles are allowed to access it. The assertion $Public(t)$ means that term t is public.

It should be noted that both secrecy and being public are preserved over actions. The preservation of secrecy is shown in Equation 3.95 and the persistence of the public-assertion is shown in Equation 3.96.

$$Secret(S, n)[a]_S Secret(S, n) \tag{3.95}$$

$$Public(n)[a]_S Public(n) \tag{3.96}$$

All data that is sent over an untrusted channel will be rendered public by definition, this is formalised in Equation 3.97. Thus, if I have a service that is called over an untrusted channel with a certain parameter p, we have $Public(p)$. Furthermore, all public keys are expected to be public knowledge.

$$\phi[\langle m \rangle]_X Public(m) \tag{3.97}$$

A secret variable cannot be public and a public value cannot be secret, which is shown in, respectively, Equation 3.98 and Equation 3.99 . Due to the fact that all sent variables are rendered public, if a secret variable s is published, we will get a conflict, as s cannot be both secret and public. Therefore, this situation would be unprovable, which is exactly what we want, as we pursue to prove protection of the confidentiality of this variable.

$$Q, R \vDash Secret(S, n) \Rightarrow Q, R \vDash \neg Public(n) \tag{3.98}$$

$$Q, R \vDash Public(n) \Rightarrow Q, R \vDash \neg Secret(S, n) \tag{3.99}$$

It should be noted that the predicate $HasAlone(X, n)$ defined in Equation 3.72 is different from the secrecy assertion on two points. Firstly, the former is a predicate, whereas the latter is an assertion. That means that the former is a logical formula that can be evaluated, whereas the latter states a condition that may not be violated. Another important difference is that secrecy may be widened to a group of services, whereas only one service can be the sole possessor of information n.

Authentication Secondly, we will discuss authentication. We will use the notion of matching records of runs as notion of authentication [20, 9, 18], which states that two entities A and B accept each other's identities at the end of a run, when their records of the conversation match. In other words, when every message sent is also received, and when the records have the sent messages in equal order, the entities accept each other's identity. Furthermore, in the original description of this notion, Diffie, Oorschot and Wiener [20] note that any public key part of the authentication process will be tied to the accepted identity.

Another possible formalisation of authentication would have been the notion of corresponding statements [74, 29], where authentication is specified using a beginning and an ending statement. The beginning statement marks when the authentication process is started and the ending statement marks when the authentication is performed. In this system, the entities authenticate each other when for every beginning statement there has been at least one matching closing statement. Furthermore, if replay attacks need to be prevented, every beginning statement has to be matched to one and only one ending mark. However, this notion does not allow for clean modularisation, due to the fact that the beginning and ending marks get scattered over services and are only verifiable over the complete block of services and not the atomic services. Therefore, this notion is not viable for our research.

Datta et al. [17] use a predicate which states that a certain set of actions happened in a certain order to provide for the matching records of runs notion of authentication. In other words, they use temporal logic expressed through the statement $After(A_1, 1_2)$ and the statement $ActionsInOrder(a_1, \ldots, a_n)$, which we discussed in Section 3.1.2. If a principal A is sure that a set of actions happened in a certain order, he can be sure of the authentication of those statements.

The predicate for temporal ordering can also be used in a service oriented system. As stated before, communication over the network normally happens at two points: the service invocation and the returning of the results. For example, when we have a service S that invokes a service T with parameter p which results in the return of variable r, we would have the cord shown in Equation 3.100 for S and the one shown in Equation 3.101 for T. It can be seen that the service consumer sends the parameters $\langle p \rangle$ and receives the result (r), whereas the service provider receives the parameters (p), executes its capabilities P_T and sends the result $\langle r \rangle$. This example composition is shown in Figure 3.3.

$$[(m_{S0})P_{S0}\langle p \rangle (r)P_{S1}\langle m_{S1} \rangle]_S \tag{3.100}$$

$$[(p)P_T\langle r \rangle]_T \tag{3.101}$$

If there was an authentication performed between the services S and T, the postcondition for service S would be of the form shown in Equation 3.102. Please note that this postcondition does not constitute a formal authentication, but merely an example of how the temporal ordering would be asserted.

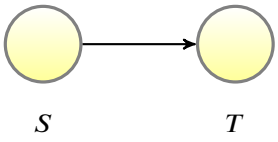


Figure 3.3: Example of a service composition to illustrate invocation

$$\psi_S = Honest(\hat{T}) \Rightarrow ActionsInOrder(Send(S,p), Receive(T,p), Send(T,r), Receive(S,r)) \quad (3.102)$$

Cryptography In our framework, we also use cryptography. This provides one of the basic building blocks to realising secrecy or authentication within service compositions. Cryptography can be seen as a dependency for the realisation of the security properties that we discussed.

In their framework, Datta et al. [18] do not really differentiate between public key and symmetric key cryptography. Although they really only discuss public key cryptography, symmetric key cryptography would be the result of stating that $K = K^{-1}$ for some cryptographic key K. In other words, by defining the encryption and decryption key to be equal and, thus, symmetric.

The syntax, semantics and proof system surrounding cryptography can be found in Section 3.1. This section explicitly discusses public key cryptography and cryptographic signing. As stated before, we have symmetric key cryptography whenever $K = K^{-1}$ holds. It should be noted that the public key is always expected to be known by everyone. Additionally, secret keys and symmetric keys are secret, whereby secret keys are commonly only known by the owner of the key and symmetric keys by the authorised parties.

3.4 Example Cases

To illustrate the proof system, we will discuss some examples. Specifically, we will reason about communicating secrets in a service oriented world and authenticating to a service. The former of those two will be discussed in Section 3.4.1 and the latter in Section 3.4.2.

In these examples, we will use p and r to, respectively, describe the ingoing parameters and the outgoing results of a service. Thus, if a service S consists of actions P, we would have $[(p)P\langle r\rangle]_S$. Additionally, we will use the symbols we already introduced to describe services: namely, ϕ for preconditions, ψ for postconditions, and Γ for invariants.

secure_echo

Figure 3.4: A secure echo service

3.4.1 Sending Secrets

Suppose we have a service that securely echoes your message. This service, which is illustrated in Figure 3.4, retrieves any message encrypted with his public key and returns the same message encrypted with the public key of the sender. If we assume that this service keeps the secret key $K^{-1}_{secure\_echo}$ secret and the original invoker of the service does the same, we can prove that this service keeps the message secret. Thus, we have, where C is the service consumer, the expected input shown in Equation 3.103 and the output shown in Equation 3.104, which results in the cord shown in Equation 3.105. Furthermore, the precondition and the postcondition are defined to be true, and the invariant as shown in Equation 3.106.

$$\{\!|\ m\ |\!\}_{K_{secure\_echo}} \tag{3.103}$$

$$\{\!|\ m\ |\!\}_{K_C} \tag{3.104}$$

$$[(x)(x/\{\!|\ m\ |\!\}_{K_{secure\_echo}})\langle\{\!|\ m\ |\!\}_{K_C}\rangle]_{secure\_echo} \tag{3.105}$$

$$\begin{aligned}\Gamma_{secure\_echo} = {} & Secret(secure\_echo, K^{-1}_{secure\_echo}) \wedge Secret(C, K_C^{-1}) \\ & \wedge Secret(secure\_echo, m) \wedge Secret(C, m)\end{aligned} \tag{3.106}$$

Now, if we add a service that invokes the secure echo service with an encrypted message, we would have a secure composition. This new composition is shown in Figure 3.5. This service can be described using the cord shown in Equation 3.107. As with the service *secure_echo*, this service does not have any precondition or postcondition. Furthermore, the invariant is defined as shown in Equation 3.108.

$$[\langle\{\!|\ m\ |\!\}_{K_{secure\_echo}}\rangle]_{send\_message} \tag{3.107}$$

$$\begin{aligned}\Gamma_{send\_message} = {} & Secret(secure\_echo, K^{-1}_{secure\_echo}) \\ & \wedge Secret(send\_message, K^{-1}_{send\_message}) \\ & \wedge Secret(secure\_echo, m) \wedge Secret(send\_message\}, m)\end{aligned} \tag{3.108}$$

Showing that this composition still guarantees secrecy can be done using some logical steps. When we replace C in Γ_{secure_echo} with $send\_message$, we actually get the same assertion, as shown in Equation 3.110. In addition, since there are no postconditions or preconditions specified, the sequential composition fits easily. For completeness, the resulting cord for *secure_echo* is shown in Equation 3.109.

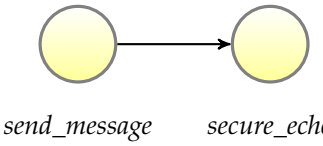


Figure 3.5: The secure echo service of Figure 3.4 in a composition

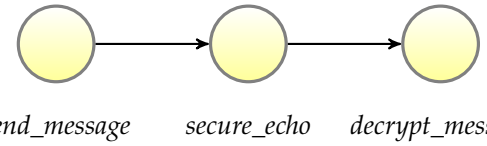


Figure 3.6: The secure echo service of Figure 3.4 in an insecure composition

$$[(x)(x/\{\!|\ m\ |\!\}_{K_{secure\_echo}})\langle\{\!|\ m\ |\!\}_{K_{send\_message}}\rangle]_{secure\_echo} \tag{3.109}$$

$$\begin{aligned}\Gamma_{secure\_echo} = {} & Secret(secure\_echo, K^{-1}_{secure\_echo}) \\ & \wedge Secret(send\_message, K^{-1}_{send\_message}) \\ & \wedge Secret(secure\_echo, m) \wedge Secret(send\_message, m)\end{aligned} \tag{3.110}$$

To make things interesting, we introduce a decryption service that shares its keypair with the *send_message* service, as illustrated in Figure 3.6 – please note that this should not be done in most cases. Additionally, we will also extend the secrecy of the message m to include this service. This service, decrypts any message that it receives and returns the decrypted message, i.e. it announces the messages.

Intuitively, the *decrypt_message* service breaks the confidentiality of the message m. However, to show this formally, we will have to specify the service first. The input parameter of this service is given in Equation 3.111 and the result is shown in Equation 3.112. In combination, this gives the cord shown in Equation 3.113. Furthermore, the invariant for *decrypt_message* is defined in Equation 3.114.

$$\{\!|\ m\ |\!\}_{K_{send\_message}} \tag{3.111}$$

$$m \tag{3.112}$$

$$[(x)(x/\{\!|\ m\ |\!\}_{K_{send\_message}})\langle m\rangle]_{decrypt\_message} \tag{3.113}$$

$$\begin{aligned}\Gamma_{decrypt\_message} = {} & Secret(secure\_echo, K^{-1}_{secure\_echo}) \\ & \wedge Secret(send\_message, K^{-1}_{send\_message}) \\ & \wedge Secret(decrypt\_message\}, K^{-1}_{send\_message})\end{aligned} \tag{3.114}$$

To compose these services, we have to apply the weakening rule of Equation 3.90 first on both sides. Weakening $\Gamma_{secure\_echo} = \Gamma_{send\_message}$ using $\Gamma_{decrypt\_message}$ is no problem, as we assumed that *send_message* and *decrypt_message* willingly share their keypair. The other way round, we have to add the assertion of Equation 3.115 to the invariant of *decrypt_message*.

$$Secret(decrypt\_message, m) \tag{3.115}$$

However, this service has an implicit postcondition that makes m public, as it executes the action $\langle m \rangle$ which, when Equation 3.97 is applied to it, yields $Public(m)$. On the other hand, we also have the condition shown in Equation 3.116 that yields $\neg Public(m)$, when Equation 3.98 is combined with it. This, of course, results in a conflict, as illustrated in Equation 3.117, which makes it impossible to derive a proof. Therefore, this composition does not preserve secrecy.

$$\begin{aligned} &Secret(secure\_echo, m) \\ \wedge\ &Secret(send\_message, m) \\ \wedge\ &Secret(decrypt\_message, m) \end{aligned} \tag{3.116}$$

$$Public(m) \wedge \neg Public(m) \tag{3.117}$$

It should be noted that, in this example, we have assumed honest services. By honest, we mean that services act as they are specified. Therefore, the introduction of dishonest services in the attacker model, as done in Section 3.5, adds a new dimension of complexity. This calls for the ability to verify the stated proof and assertions of a service, as the possibility of unexpected behaviour means a deviation from these statements. If a service keeps to its definition, the specified proof system is sufficient for secure composition.

3.4.2 Authenticating Services

As we discussed in Section 3.3, for authentication we follow the notion of matching runs of Diffie, Oorschot and Wiener [20]. This means that, in practice, authentication always requires multiple steps and, thus, the use of more than one service invocation. Therefore, authentication in itself will break with the properties of statelessness and uncoupledness.

To cope with the two-steps requirement of authentication, we will see the service composition that constitutes authentication as an atomic service after the security has been proven. This way, the composed service does, once again, comply with the properties, when observed from the new viewpoint.

Lets assume we have two services that want to authenticate against each other, namely *alpha* and *beta*. For this, they use a service oriented adoption of a simple protocol, where they sign nonces to prove their identity. This is illustrated in Figure 3.7. Please note that, in this figure, *alpha* invokes *beta* two times, but does so sequentially – hence the extra arrow between *alpha* and the second appearance of *beta*.

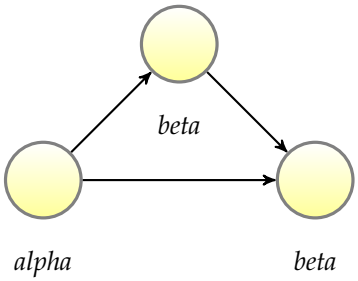


Figure 3.7: A service composition for authentication

The messages sent in the composition start with a nonce n as parameter to *beta* that returns this nonce signed with another nonce m and both names, i.e. the return message of *beta* is $\{\!|\ (n, m, alpha, beta)\ |\!\}_{K_{beta}^{-1}}$. The second invocation of *beta* contains the nonce generated by *beta* and the names signed by *alpha* and results in a final signature on a freshly generated token a and the names. Combined, this gives for *alpha* the cord shown in Equation 3.118, for the first run of *beta* the cord displayed in Equation 3.119, and for the second run the cord written in Equation 3.120.

$$\begin{aligned}&[(\nu n)\langle n\rangle(x)(x/\{\!|\ (n, m, alpha, beta)\ |\!\}_{K_{beta}^{-1}})\\&\langle\{\!|\ (m, alpha, beta)\ |\!\}_{K_{alpha}^{-1}}\rangle(z)(z/\{\!|\ (a, alpha, beta)\ |\!\}_{K_{beta}^{-1}})]_{alpha}\end{aligned} \tag{3.118}$$

$$[(n)(\nu m)\langle\{\!|\ (n, m, alpha, beta)\ |\!\}_{K_{beta}^{-1}}\rangle]_{beta} \tag{3.119}$$

$$[(x)(x/\{\!|\ (m, alpha, beta)\ |\!\}_{K_{alpha}^{-1}})(\nu a)\langle\{\!|\ (a, alpha, beta)\ |\!\}_{K_{beta}^{-1}}\rangle]_{beta} \tag{3.120}$$

As the goal of this service composition is to authenticate *alpha* with *beta*, the postcondition should reflect this. Given the notion of matching records of runs, this is done using temporal ordering, as shown by the postcondition for *alpha*, which is displayed in Equation 3.121. To show the correctness of this formula, we will discuss its proof.

$$\begin{aligned}\psi_{alpha} = Honest(beta) \Rightarrow ActionsInOrder(&New(alpha, n),\\&Send(alpha, n), Receive(beta, n),\\&New(beta, m), Send(beta, \{\!|\ (n, m, alpha, beta)\ |\!\}_{K_{beta}^{-1}}),\\&Receive(alpha, \{\!|\ (n, m, alpha, beta)\ |\!\}_{K_{beta}^{-1}}),\\&Send(alpha, \{\!|\ (m, alpha, beta)\ |\!\}_{K_{alpha}^{-1}}),\\&Receive(beta, \{\!|\ (m, alpha, beta)\ |\!\}_{K_{alpha}^{-1}}))\end{aligned} \tag{3.121}$$

Postcondition alpha. We have the cord shown in Equation 3.118. The first action, (νn), combined with the axioms in Equation 3.54, Equation 3.85 and Equation 3.58 gives us:

$$\begin{aligned}
&[(\nu n)]_\alpha Fresh(\alpha, n) \wedge \diamondsuit new(\alpha, n)[\langle n \rangle]_\alpha \diamondsuit Send(\alpha, n) \\
&\qquad [(\{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}})]_\alpha \diamondsuit Receive(\alpha, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}) \\
&\quad [(\{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}} / \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}})]_\alpha \diamondsuit Verify(\alpha, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}) \\
&\qquad [\langle \{\!|\ (m, \alpha, \beta)\ |\!\}_{K_\alpha^{-1}} \rangle)]_\alpha \diamondsuit Send(\alpha, \{\!|\ (m, \alpha, \beta)\ |\!\}_{K_\alpha^{-1}}) \\
&\qquad [(\{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}})]_\alpha \diamondsuit Receive(\alpha, \{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}) \\
&\quad [(\{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}} / \{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}})]_\alpha \diamondsuit Verify(\alpha, \{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}). \qquad (3.122)
\end{aligned}$$

Please note that α and β are used to respectively describe *alpha* and *beta*. Additionally, when we apply Equation 3.73 to the first part of this equation, this part becomes:

$$[(\nu n)]_\alpha Fresh(\alpha, n) \wedge \diamondsuit new(\alpha, n)[\langle n \rangle]_\alpha \neg Fresh(\alpha, n) \wedge \diamondsuit Send(\alpha, n). \qquad (3.123)$$

After we apply the temporal ordering axioms in Equation 3.78 and Equation 3.79, we get the following suffix:

$$\begin{aligned}
Honest(\beta) \Rightarrow\ & ActionsInOrder(New(\alpha, n), Send(\alpha, n), Receive(\alpha, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}), \\
& Verify(\alpha, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}), Send(\alpha, \{\!|\ (m, \alpha, \beta)\ |\!\}_{K_\alpha^{-1}}), \\
& Receive(\alpha, \{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}), Verify(\alpha, \{\!|\ (a, \alpha, \beta)\ |\!\}_{K_\beta^{-1}})). \qquad (3.124)
\end{aligned}$$

Of course, we are allowed to leave out parts of this, as long as the order is not affected. Thus, we can get:

$$\begin{aligned}
Honest(\beta) \Rightarrow\ & ActionsInOrder(New(\alpha, n), Send(\alpha, n), \\
& Receive(\alpha, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}), Send(\alpha, \{\!|\ (m, \alpha, \beta)\ |\!\}_{K_\alpha^{-1}})). \qquad (3.125)
\end{aligned}$$

Given Equation 3.69 and Equation 3.75, we can put $\neg\diamondsuit Fresh(\alpha, n)$ at the end of the cord. Furthermore, due to the inability to forge signatures, as shown in Equation 3.63, we know that:

$$\begin{aligned}
&Honest(\beta) \wedge \diamondsuit Verify(\alpha, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}) \\
&\qquad \Rightarrow \exists \beta : (\exists l : (\diamondsuit Send(\beta, l) \wedge Contains(l, \{\!|\ (n, m, \alpha, \beta)\ |\!\}_{K_\beta^{-1}}))) \qquad (3.126)
\end{aligned}$$

Through the honesty rule, as defined in Equation 3.89 and the inference rules in Equation 3.74 through Equation 3.75, we can deduced that this signature was sent according to the service description. Furthermore, Equation 3.80 and Equation 3.81 show us that any actions using a freshly generated nonce have to have happened after it was made public. Thus, any sending by β has to have happened in the correct order. Applying this knowledge to the ordering we already found, we can conclude the postcondition as shown in Equation 3.121. □

3.5 On Adversaries

Although we already discussed our formal model for the secure composition of services including properties of the attacker in Section 3.1, Section 3.2 and Section 3.3, we will discuss the attacker model more elaborately. In addition, we will consider special cases raised by services that do not occur when securely composing protocols.

In order to prove any security property, a clear, consistent and complete attacker model is necessary. Otherwise, one does not know against what the system is secured. Even more extreme, without an attacker, anything is secure. This is reflected in Research Question 5.

Research Question 5. What attacker model do we need to reflect all the challenges specific to the secure composition of services?

As already noted, services are more complex than protocols, but still make use of them. Therefore, attackers in a service oriented model have the properties attributed to the attacker of Dolev and Yao [22], but are also more capable. Those properties are:

1. the attacker can obtain any message send between two services;
2. the attacker is a legitimate node in the network; and
3. the attacker will have the opportunity to be a receiver to any user A.

In our context, being a legitimate node in the network does not only mean that the attacker can initiate a conversation with any service in the system, but also that the attacker can deploy as many services as he likes. Furthermore, following both property 2 and property 3, he is able to imitate a service and, for example, perform a man-in-the-middle attack.

An example of a more complex attack would be when a malicious service poisons other services, by returning different results on the same request depending on where it originates. This way, scraping services could be fooled such that they will return wrong results. In other words, services keep state over time and share this state with the other services deployed by the same agent.

Suppose we have an automated stock trading system that is implemented using service orientation. An attacker will not be restricted to the standard attacks of eavesdropping, altering messages or posing as a legitimate node, but can also try to influence those services that give information on the stock indexes. Since these services are active state maintaining nodes in the service oriented architecture (SOA), this could influence all traders.

Additionally, an attacker may be able to compromise services, through social engineering when it is a human agent or through conventional hacking when it is a software agent. This could lead to a service leaking information – comparable to property 1, with the additional possibility to read data encrypted for the compromised service – or the service redirecting to a rogue service – comparable to property 3. However, this is out of scope, as it would require modelling the inner workings of the services.

To return to the example of stock trading, in high frequency trading systems, stock options are traded rapidly based on automated event watchers. Suppose that such a system is built from services that scrape news, stock information, and other comparable information. In that case, a competing trader has a lot to gain by controlling several event generating channels, such as those news or stock information channels, and feeding the trading agents of the competition slightly misleading information. This would be a novel attack, made possible by the concept of service orientation.

In addition to the described novel attack case, traditional attacks from the world of protocols also apply. For example, one may still want to try to eavesdrop on or tamper with messages, which is a viable attack scenario. Therefore, we can see a combination of the problems of authentication – integrity – and secrecy – confidentiality – under composition in an uncoupled world, with other problems, such as the issue of trust. Furthermore, the described attack of misleading competitors does not only carry a trust issue, but also contains the problem of the Byzantine generals [42], which comes down to the problem of having dishonest nodes in your system.

4 The Need for Trust

Up to this point, most cases we discussed assumed honest services. This, of course, simplifies the process, and thereby helps us taking small steps instead of one large jump. Nevertheless, the real world also contains dishonest services, which already led to Research Question 3.

However, besides looking in the direction of coping with opaque services for solutions to the problem of possible dishonesty, we will also consider trust. In addition, trust is also needed to cope with the uncoupled nature, due to the lack of prior communication or knowledge. As trust is an ambiguous term that is under lively debate, we will keep our reasoning on a mathematical and technical level [71].

As we already discussed in Section 3.5, our attacker is more capable than the traditional attacker of Dolev and Yao [22], which leads to additional attack scenarios. Especially what Pavlovic [56] calls the paradox of trust becomes apparent in the described issues, which shows us that if someone has a good reputation it is not necessarily a good decision to award high trust to that person. This paradox states that trust is not transferable, but we do need to transfer trust in order to reason about it. For example, the wife of a Mafioso may trust her husband very much. However, when someone trusts this woman and trust were transferable, this person would trust the Mafioso, which is, for obvious reasons, not a very good idea.

The discussed problems of dishonesty and opaque services gives us reason to formulate Research Question 6 and Research Question 7. Please note that the problem of the Byzantine generals will be explained further in Section 4.3.

Research Question 6. How can we implement trust in our model, thereby keeping in mind the paradox of trust of Pavlovic [56]?

Research Question 7. How can we apply solutions for the problem of the Byzantine generals of Lamport, Shostak and Pease [42] to compensate for dishonest participants?

4.1 Implementing Trust

To reason about trust, we will consider the definition of trust from Verberkt [71], which is shown in definition 4.1. This definition expresses that trust is an unique relation between two persons, in which one expresses trust in the other concerning a certain property. For example, an employee could trust his employer to pay out his salary at the end of every month.

Definition 4.1 (Trust and distrust [71])**.** Trust is a directed relation between a trustor A and trustee B concerning some property P. It can be expressed as a probabilistic

value t, with $0 \leq t \leq 1$, and can be written as a quadruple $Trusts(A, B, P, t)$. Distrust is defined equally.

In the definition of trust, it can be seen that trust is described using a probabilistic value. As the degree of trust persons have in other persons varies from very little to a lot, this enables us to describe the various degrees of trust [46]. Additionally, it enables us to apply probabilistic reasoning to our model of trust.

In order to embed the notion of trust in our model of services, we can augment our system with a network of trust. Thus, we add a set of trust relations T and a set of distrust relations D over the services Σ concerning the validity of the annotations ϕ, ψ and Γ. Thus, for two services S_i and S_j with $i \neq j$, Equation 4.1 shows a logical annotation of service S_j. Service S_i uses this to express a trust of t in S_j, as shown in Equation 4.2.

$$\Theta_{S_j} = \Gamma_{S_j} \vdash \phi_{S_j} S_j \psi_{S_j}) \tag{4.1}$$

$$Trusts(S_i, S_j, \Theta_{S_j}, t) \text{ with } 0 \leq t \leq 1 \tag{4.2}$$

Since we have defined honesty in our formal model, we can generalise this further to the statement shown in Equation 4.3. This predicate shows a trust of S_i in the honesty of S_j, which means that S_i has an expectation of t that S_j will act as defined in its service description. As Θ_{S_j}, as stated in Equation 4.1, is part of this description, this can be used for generalisation. Therefore, we can define our predicate of trust as shown in Definition 4.2.

$$Trusts(S_i, S_j, Honest(S_j), t) \text{ with } 0 \leq t \leq 1 \tag{4.3}$$

Definition 4.2 (Trust predicate)**.** The statement $Trust(S, T, t)$ means that service S trusts T to be honest with a confidence value of t. In other words, S expects $Honest(T)$ to be true with a probability of t.

However, the discussed notion of trust and accompanying predicates only add direct trust relations, which need to be built separately for every two parties that want to trust each other. Commonly, this is done using reputation-based infrastructures or using a certificate-based architecture. In the first case, reputation refers to the common opinion of the trustworthiness of a certain node in the network. The second type refers to the use of certification by trusted authorities that have assessed the trustworthiness of a certain party.

4.1.1 Certification

One of the most well known examples of certificates used to generate trust can be found with X.509 [4, 5]. This refers to the public key infrastructure (PKI) used by, amongst others, transport layer security (TLS) for secure communication on the Internet. A so-called PKI is an infrastructure that is used to generate and transfer trust in public keys. In this combination with TLS, it is used to constitute trust in the public key of the party a certain user is trying to communicate with.

X.509 and Transport Layer Security Suppose that you want to communicate securely with a certain webshop. To do this, TLS is the designated protocol. This protocol, which is the most common protocol for secure communication on the Internet, uses cryptography to encrypt and/or sign the messages that pass between two end-nodes in a specific communication path. However, as it is built on public key cryptography, it requires the use of the public key of this webshop.

In order to be able to trust that the public key you have in front of you is in fact the public key of the digital shop you are trying to communicate with, a PKI is needed. This infrastructure generates trust using certificates. Your computer uses this trust to assure itself that the public key is correct and to enable you to check the details.

In the discussed example, a certificate binds an identity to a public key. For example, it binds a certain webshop that can be found using a certain uniform resource locator (URL) to a public key. This certificate is issued by a trust authority, which is a trusted third party that is supposed to check the details and whether the public key indeed belongs to them. It should be noted that the certification authority makes a certificate by means of a cryptographic signature using its secret key on the data that needs to be authenticated, i.e. the details of the webshop and its public key.

Hierarchical Public Key Infrastructures To give more background on the concept of PKIs, the PKI behind X.509 is a hierarchical one [44]. This means that there is a root certificate authority, which everyone is supposed to trust. This authority issues certificates to sub-authorities. Eventually, at the end of this chain, there is a certificate authority that issues the certificates to certify a public key to be used in, for example, TLS.

An example of a hierarchical PKI is illustrated in Figure 4.1. In this illustration, the orange nodes (CA_0 through CA_3) represent the certification authorities and the purple ones (C_1 through C_4) depict the certificates. Thus, certificate C_3 is issued by CA_3 who is certified by CA_1 who is certified by CA_0 who should be trusted by everyone.

The hierarchical system requires any node in the trust-network to trust the root certificate authority – CA_0 in Figure 4.1. The position of certificate authorities is the position of a trusted third party. In other words, they pose themselves as an objective party that makes good decisions about the trustworthiness of nodes after performing thorough checks.

Service Oriented Applications In our model, we need to ensure the trust in the honesty of a certain service. A certification-based model would mean that we invent a trusted third party, which takes it upon itself to assess the honesty of services. After doing this assessment, the authority is able to publish a certificate that acknowledges the results. For example, such a certificate could state that the honesty of service S is believed to be verified up to a certain level, which can be expressed using the confidence value t that is part of the trust statement.

An infrastructure for the certification of services could be built comparable to how infrastructures for the auditing of organisations or banks are constructed. However, instead of issuing formal documentation to the principal that has been audit, e.g. the

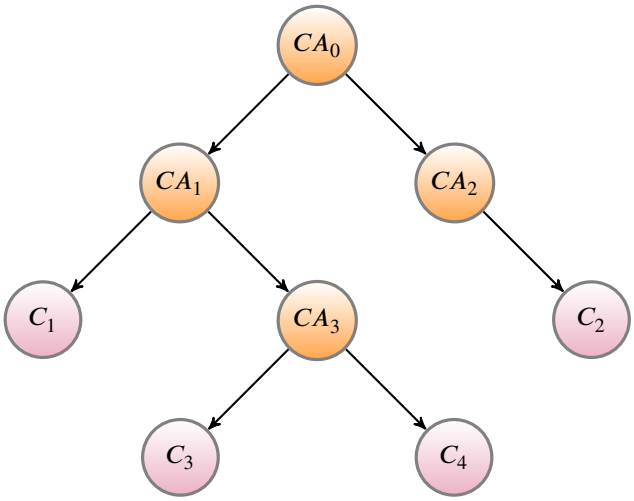


Figure 4.1: A hierarchical public key infrastructure

supervisory board or the directors of the company providing the service under audit, a cryptographic certificate is issued and used in a PKI.

4.1.2 Reputation

A practical example of a reputation-based system can be found with Google's PageRank algorithm [56, 14]. This algorithm is used to calculate the reputation of websites based on the links to that page, which one can see as expressions of trust. Thus, a reputation score is built, which is commonly used by people to evaluate their trust in the website.

In a reputation-based trust-network, all nodes in the network are able to make a statement about the supposed trustworthiness of a certain node. These values are used to compute the reputation that node has. A common way to do this is using a weighted average. In this case, the reputation values are weighted based upon the reputation of those that expressed this statement about the trustworthiness of the node under assessment.

Toy Example To illustrate a simple reputation infrastructure, suppose the network shown in Figure 4.2. In this figure, the arrows represent the statements the nodes made about each others trustworthiness, which is expressed using the probabilistic value displayed above those arrows. Thus, α expressed with a confidence of 0.8 that β is trustworthy.

In this example, we will compute the reputation as a weighted average using the reputation of the expressing nodes as weight. This results in Equation 4.4 for the reputation of α, Equation 4.5 for the reputation of β, and Equation 4.6 for the reputation

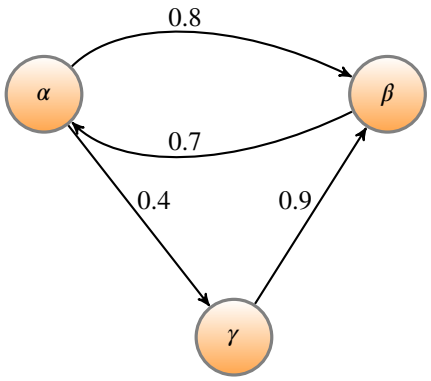


Figure 4.2: A reputation infrastructure

of γ. These formulas show us that r_α is, in fact, 0.7 and r_γ is 0.4. Therefore, the reputation of β is equal to $0.8 \cdot 0.7 + 0.9 \cdot 0.4 = 0.92$ divided by $0.7 + 0.4 = 1.1$, which is approximately 0.84.

$$r_\alpha = \frac{0.7 \cdot r_\beta}{r_\beta} \tag{4.4}$$

$$r_\beta = \frac{0.8 \cdot r_\alpha + 0.9 \cdot r_\gamma}{r_\alpha + r_\gamma} \tag{4.5}$$

$$r_\gamma = \frac{0.4 \cdot r_\alpha}{r_\alpha} \tag{4.6}$$

Normalisation As this is a very simplistic example of a reputation architecture, one can understand that it suffers from some deficiencies. Most noteworthy is the fact that it lacks any normalisation. For example, if a node only gets reputation from one node and this reputation is fairly high, this node becomes a high reputation score, although it probably has not earned it. Therefore, scores need to be normalised in a way that having little arrows means that the reputation value will also stay within a certain conservative interval.

A possible method for normalisation is adding a value of 0.5 with weight 1 to every reputation score. This way, reputations will start out around 0.5, i.e. the neutral position, and get to the extremes only after a lot of nodes have expressed a reputation score. If we were to implement this in our toy example, we would get Equation 4.7 for the reputation of α, Equation 4.8 for the reputation of β, and Equation 4.9 for the reputation of γ. If we solve these equations, we get $r_\alpha \approx 0.58$, $r_\beta \approx 0.68$ and $r_\gamma = 0.46$. As can be seen, nodes with less reputation scores end up closer to the neutral position of 0.5.

$$r_\alpha = \frac{0.7 \cdot r_\beta + 0.5}{r_\beta + 1} \tag{4.7}$$

$$r_\beta = \frac{0.8 \cdot r_\alpha + 0.9 \cdot r_\gamma + 0.5}{r_\alpha + r_\gamma + 1} \tag{4.8}$$

$$r_\gamma = \frac{0.4 \cdot r_\alpha + 0.5}{r_\alpha + 1} \tag{4.9}$$

Paradox of Trust Another important deficiency of the discussed toy example, is that it still suffers from the paradox of trust [56]. As this paradox shows, although that trust is not transferable, we need to do this in reputation-based infrastructures. It is clear that this paradox is not easily solved, given the obvious contradiction. However, we can mitigate it in a way that it affects our trust network less.

Pavlovic [56] notes that in a trust relationship $Trusts(A, B, P, t)$, t quantifies the trust and P qualifies the trust. This qualification makes it possible to bind the trust quantification to the property for which it was accumulated. Due to the fact that this makes the trust value less abstract, a more clear picture about what the trust specifically means is created. This way, unwanted transfers are easier spotted and prevented.

Service Oriented Applications To build trust using a reputation-based infrastructure in our model, one needs to collect opinions on the honesty of the service at hand. Based on this information, a reputation needs to be computed, which should be used by those wanting to invoke the service to decide whether the service is trustworthy enough. It should be noted that this requires a full fledged reputation infrastructure.

The first step in the process of reputation to trust is deciding whether the reputation value can be directly interpreted as trust value, or whether other factors should be consulted or included. Afterwards, a threshold needs to be used to decide whether the resulting trust value is high enough to assume that the service is honest. This threshold may vary between service consumers, based on their security and trust needs.

4.2 High Frequency Trading

In Section 3.5, we briefly introduced high frequency trading systems. This refers to automated stock trading, where algorithms sell and buy trade securities on a very high speed, such that the investments are only held for very brief moments. In order to do this, the systems make use of all sorts of information they can acquire, such as statistical data or news events.

Suppose that we have a high frequency trading agent and several news and stock information services. We could have a service layout as shown in Figure 4.3. In this example, the trading agent could invoke any of the other services to gain information that can be used for the trading process.

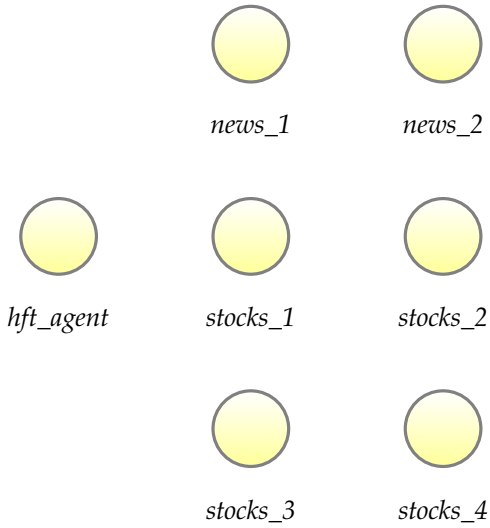


Figure 4.3: Example of a service layout for a high frequency trading system

In a reputation-based trust infrastructure, a stock information site that normally provides reliable information will receive very high ratings and, thus, a very good trust score. When this stock information site gives bogus results when it is accessed by its largest competitor, this trust score stays high. Nevertheless, the trading agent of that competitor should actually grand very little trust to this stock information service. This is due to the fact that trust is, in fact, a very personal relation between two entities concerning a specific property of the trustee, whereas reputation is global and based on a combination of weighted opinions. This is exactly the problem articulated by Pavlovic [56] as the paradox of trust, which we discussed in Section 4.1.

As we already suggested when we formulated Research Question 6 and Research Question 7, there are two possible directions of solving the trust issue, namely solving it or making it irrelevant In other words, we either try to find a reliable way to bootstrap trust or we assume that the majority of the services is trustworthy and find a method to nullify the attempts to break the system by the dishonest participants.

Solving the trust issue requires that one can be sure that the trust relation between two specific parties concerning a specific property is justified. For this, we need a reliable trust infrastructure that can cope with the paradox of trust of Pavlovic [56]. This can be either through an authority, as discussed in Section 4.1.1, or through the network, i.e. by implementing a reputation infrastructure, as discussed in Section 4.1.2, or using a comparable trust network. Of course, it is always possible to assess the service provider yourself, but this is commonly too impractical to be a serious option. Especially given the volatile nature of the usage of services, where vendors are easily and often switched, leaving little time for long assessments.

In order to make the trust issue irrelevant, we depend on a reliable majority. This

direction refers back to the Byzantine generals of Lamport, Shostak and Pease [42], which we will discuss in Section 4.3. When there are enough services performing a comparable task, the solution to that problem can be used to nullify to attempts of the adversarial service to poison another service.

4.3 The Byzantine Generals

The problem of the Byzantine generals tells us about a group of generals that have surrounded a city [42]. Now, they need to decide what their next step should be: attacking or retreating. Given that they reside at all sides of the city, they can only communicate by sending each other messages, which is what they do. For the sending of messages, the Byzantine generals use trustworthy messengers, i.e. they have tamper resistant channels. These messages will be sent to the other generals and state whether this general is intending to attack or to retreat.

However, there may be traitors amongst the generals, that want to lose the battle at any cost. Those traitors may send a message to one general stating that he is going to attack and a message to the other general remarking that he wants to retreat. This could result in half the generals attacking and the other half retreating, giving an easy battle to the defendants of the city under attack. While part of the armies retreats, a minority is starting the fight, at only one side of the town, which allows the guards to focus on that side and lowers the numbers of Byzantine soldiers gravely.

It is clear that the main objective of any traitor is to cause confusion between the generals and thereby impose an inevitably lost battle. In security terms, the traitors want to break the integrity of the decision whether to attack or to retreat. The problem faced by the generals is: how can we unanimously agree upon our next action, given the possibility of traitors amongst us?

The Byzantine Generals Problem To formalise the problem of the Byzantine general, we consider the decision making process of the generals [42]. Suppose that each general makes an observation of the enemy and decides to attack or retreat. Let $v(i)$ be the decision communicated by general g_i, e.g. if general g_i says attack, we have $v(i) = 1$.

Every general needs to combine all the opinions $v(1), \ldots, v(n)$ he received from general g_1 through g_n [42]. The combination method needs to be the same for all generals to make sure that all loyal generals decide upon the same plan. Furthermore, the method needs to be robust, such that a small number of traitors is not able to cause the loyal generals to make a bad decision. In the case of a binary decision between attacking or retreating, this method can be a majority vote, i.e. when the majority of $v(1), \ldots, v(n)$ is 1 it will be an attack, otherwise, when the majority is 0, it will be a retreat.

For the problem to be solved, it is required that the set of opinions every general has, is the same [42]. However, as a traitor may send a different message to any general, we can not simply say that every general uses the message $v(i)$ he received from general g_i. Thus, opinion $v(i)$ does not need to be obtained from general g_i. On the other hand, if a

loyal general g_i sends an opinion $v(i)$ we want this opinion to be used by every general in the combination function. Therefore, we have the following two conditions:

1. any two loyal generals use the same value for $v(i)$; and
2. if general g_i is loyal, the value sent by him must be used by every loyal general as the value for $v(i)$.

The Commander And His Lieutenants As the conditions that should hold for the decision the Byzantine generals make are both written such that they concern opinion $v(i)$ of general g_i, Lamport, Shostak and Pease [42] consider the problem from the viewpoint of one single general. They do this by stating a smaller problem, in which one commanding general sends an order to his $n-1$ lieutenants. In this version of the problem, the following conditions have to be met:

1. all loyal lieutenants obey the same order; and
2. if the commanding general is loyal, every loyal lieutenant obeys the order he sent.

4.3.1 Oral Messages

If we consider oral message, i.e. messages that are fully controlled by the sender and contain either attack or retreat, at most m of the $3 \cdot m + 1$ generals can be dishonest [42] – as we will show in this section. In this case, the Byzantine fault tolerance amounts to one third minus one [15]. This tolerance is the amount of dishonest participants the system can cope with.

Impossibility for $3 \cdot m$ or Less Generals The fact that the problem of the Byzantine commander cannot be solved with 3 participants of which 1 is dishonest can intuitively be shown [42, 60]. If we consider the case of Figure 4.4, we can see that the second lieutenant is a traitor, and sends the wrong message to the first lieutenant. In Figure 4.5, the commanding general is the traitor and the second lieutenant is loyal. Nevertheless, in both illustrations, the first lieutenant receives the same two conflicting messages. Intuitively, the problem cannot be solved, as the first lieutenant should decide to attack in the case of Figure 4.4 and to retreat in the case of Figure 4.5, although the received messages are equal.

If one knows that it is impossible to realise a solution to the problem with 3 nodes, it is easily deduced that the same holds for $3 \cdot m$. Suppose that we have $3 \cdot m$ generals [42]. We can cluster these generals in three groups of m generals. Now, if the three soldiers of the case with 3 participants each control one of these clusters and send out the messages as they would have done in the original setting, we have a multiplication of the number of messages sent, but still the same dynamics underneath it. Namely, the cluster of disloyal generals sends one message to the first loyal cluster, and the opposite message to the other cluster, resulting in the same impossibility as before.

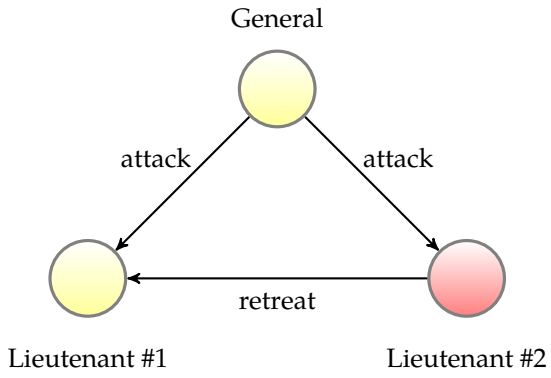


Figure 4.4: Problem of the Byzantine commander with 3 participants and a dishonest lieutenant [42]

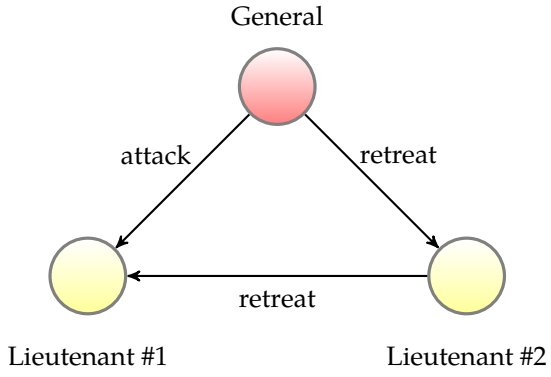


Figure 4.5: Problem of the Byzantine commander with 3 participants and a dishonest general [42]

Oral Message Algorithm To show that this works with $3 \cdot m + 1$ or more generals, with m traitors, we first have to define oral messages more precisely [42]. We require for such messages, that every message is delivered correctly, the receiver knows who sent the message, and that the absence of a message can be detected. This prevents traitors from interfering with other messages than the ones he sends and sending spoofed messages. Furthermore, the traitor cannot try to obstruct the decision by not sending any message.

We define the algorithm $OralMessage(m)$ using mathematical induction for all $m \in \mathbb{N}^0$ in Theorem 4.1 [42]. In this case, the commanding general sends an order to $n-1$ lieutenants. In the discussed definition of this algorithm, we assume a function $Majority(v_1, \dots, v_{n-1})$ that returns the value v if the majority of $v_1, \dots, v_{n-1}$ is equal to v, and otherwise 0.

Theorem 4.1 (Oral message algorithm [42])**.** *The algorithm $OralMessage(m)$ for all $m \in \mathbb{N}^0$ is defined using mathematical induction, with for $m = 0$:*

1. *the commanding general sends his value to every lieutenant; and*
2. *each lieutenant uses the value received from the commanding general or uses the value 0 if no value was received.*

Furthermore, the algorithm is as follows, when $m > 0$:

1. *the commanding general sends his value to every lieutenant;*
2. *for each i, let v_i be the value received by lieutenant i from the commanding general or 0 if no value was received; lieutenant i acts as commanding general in $OralMessage(m-1)$ to send v_i to each of the $n-2$ other lieutenants; and*
3. *for each i and each $j \neq i$, let v_j be the value received by lieutenant i from lieutenant j or 0 if no value was received; lieutenant i uses the value resulting from $Majority(v_1, \dots, v_{n-1})$.*

Possibility for $3 \cdot m+1$ or More Generals Figure 4.6 shows the result of the algorithm $OralMessage(1)$ with 4 participants from the viewpoint of the second lieutenant [42]. In this figure, the third lieutenant is disloyal. Following the algorithm, the loyal commanding general sends his value v_0 to all lieutenants. After receiving this value, each lieutenant sends the received value to all other lieutenants. However, as the third lieutenant is a traitor, he is expected to send a different value. This results in the second lieutenant receiving value v_0 from both the general and the first lieutenant, and value $v_3 \neq v_0$ from the third lieutenant. Therefore, the second lieutenant decides that $v = Majority(v_0, v_0, v_3) = v_0$, which satisfies the conditions of the problem of the Byzantine commander.

As it is also possible that the commanding general is the traitor, we consider this case using Figure 4.7 [42]. In the illustrated situation, the disloyal general sends a different message to each lieutenant. The lieutenants follow the algorithm as they should, which results in every lieutenant receiving the values v_0', v_0'', and v_0'''. This results in every lieutenant executing $Majority(v_0', v_0'', v_0''')$. As every lieutenant invokes the majority function with the same parameters, they all come to the same decision.

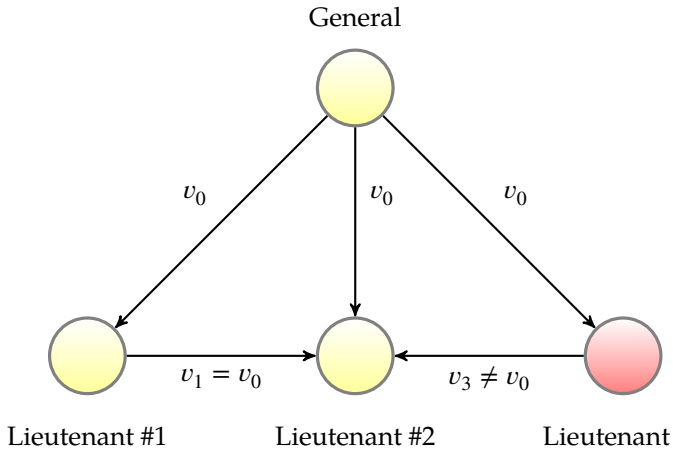


Figure 4.6: The result of *Oral Message*(1) from the viewpoint of the second lieutenant with a dishonest lieutenant [42]

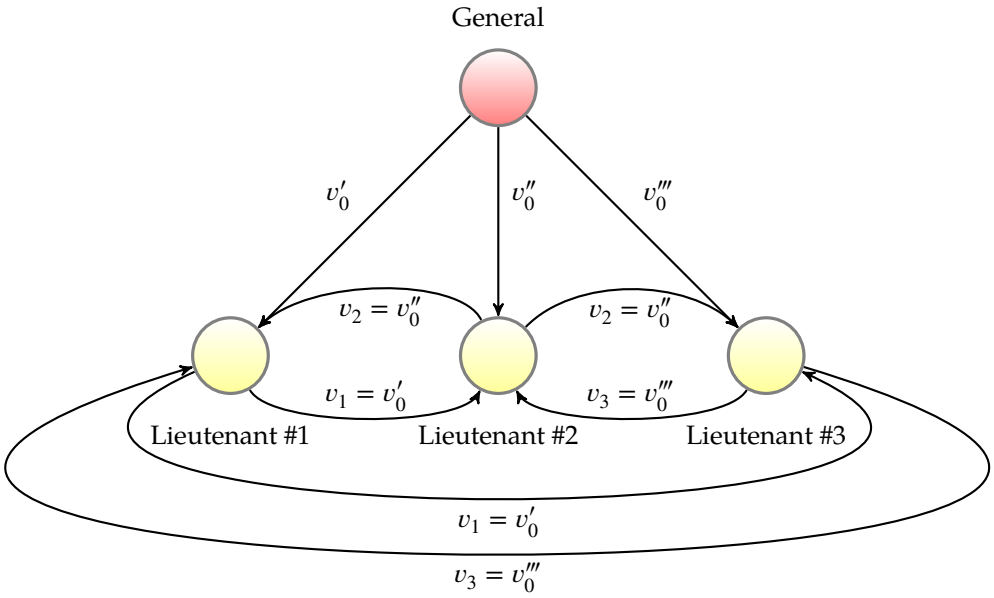


Figure 4.7: The result of *Oral Message*(1) with a dishonest general [42]

Correctness As intuitively shown using Figure 4.6 and Figure 4.7, it is possible to solve the problem of the Byzantine generals for $3 \cdot m + 1$ generals for oral messages. In order to prove that the proposed algorithm $OralMessage(m)$ is correct, we first show that, for any m and k, $OralMessage(m)$ satisfies the condition that, if the commanding general is loyal, all lieutenants obey the order, if there are more than $2 \cdot k + m$ generals, with at most k traitors [42]. This lemma is shown in Theorem 4.2. Furthermore, the proof is as follows.

$OralMessage(m)$ with loyal general [42]. We prove Theorem 4.2 using mathematical induction on m. We consider the case where the commanding general is loyal. As we required that every sent message is delivered correctly, $OralMessage(0)$ is trivially true. For the induction step, we assume that $OralMessage(m-1)$ satisfies Theorem 4.2 with $m > 0$, and prove this for m. In the first step of the algorithm, the commanding general sends a value v_0 to all $n-1$ lieutenants. In the following step, all lieutenants apply $OralMessage(m-1)$ with $n-1$ generals. As Theorem 4.2 stated that $n > 2 \cdot k + m$, we have $n - 1 > 2 \cdot k + (m-1)$. Therefore, we can apply the induction hypothesis to conclude that every loyal lieutenant gets $v_i = v_0$ for every loyal lieutenant i. As there are at most k disloyal lieutenants, and we have $n - 1 > 2 \cdot k + (m-1) \geq 2 \cdot k$, a majority of the lieutenants is loyal. Therefore, every lieutenant has $v_i = v_0$ for a majority of the $n-1$ values i he received, which results in $Majority(v_1, \ldots, v_{n-1})$ evaluating to v_0. □

Theorem 4.2 ($OralMessage(m)$ with loyal general [42])**.** *For any m and k, the algorithm $OralMessage(m)$ satisfies the condition that, if the commanding general is loyal, all lieutenants obey the order he sends, if there are more than $2 \cdot k + m$ generals, with at most k traitors.*

Now, we will prove that $OralMessage(m)$ does, in fact, solve the problem of the Byzantine generals [42]. Thus, we will show that, for any m, $OralMessage(m)$ satisfies both conditions for the Byzantine commander problem, if there are more than $3 \cdot m$ generals and at most m traitors. This is formalised in Theorem 4.3. Furthermore, the proof is as follows.

$OralMessage(m)$ [42]. We prove Theorem 4.3 using mathematical induction on m. If there are no traitors, it is trivial to see that $OralMessage(m)$ satisfies both conditions. For the induction step, we assume that $OralMessage(m-1)$ satisfies Theorem 4.3 with $m > 0$, and prove this for m. Firstly, we consider the case where the commanding general is loyal. If we consider k to be equal to m, we can see that the second condition of Theorem 4.3 is satisfied, following Theorem 4.2. Additionally, the first condition follows from the second condition when the general is loyal. Secondly, we consider the case where the commander is disloyal. In this case, the second condition of Theorem 4.3 is trivially satisfied. As there are at most m traitors, and the general is one of them, we have at most $m-1$ traitors between the lieutenants. As there are more than $3 \cdot m$ generals, we have more than $3 \cdot m - 1$ lieutenants. Given that $3 \cdot m - 1 > 3 \cdot (m-1)$, we may apply the induction hypothesis to conclude that $OralMessage(m-1)$ satisfies Theorem 4.3. Thus, for every j, any two lieutenants get the same value v_j in the third step of the algorithm. Therefore, any two lieutenants get the same vector of messages

$v_1, \ldots, v_{n-1}$, which results in every lieutenant obtaining the same value v by computing $Majority(v_1, \ldots, v_{n-1})$. □

Theorem 4.3 ($OralMessage(m)$ [42]). *For any m, the algorithm $OralMessage(m)$ satisfies the following conditions, if there are more than $2 \cdot m$ generals, with at most m traitors:*

1. *all loyal lieutenants obey the same order; and*
2. *if the commanding general is loyal, every loyal lieutenant obeys the order he sent.*

Non-discrete Values It should be noted that, when there is no discrete value to be decided upon, the given solution for oral messages still holds [42]. In this case, the function $Majority(v_1, \ldots, v_{n-1})$ should be implemented differently. For example, to decide upon an approximation of a certain value, this function could return the median of the values $w_1, \ldots, w_{n-1}$, where $w_1, \ldots, w_{n-1}$ is obtained by ordering $v_1, \ldots, v_{n-1}$.

4.3.2 Signed Messages

When we introduce cryptographic signatures to the problem of the Byzantine generals, we can solve the problem for any number of generals with m traitors [42]. In this case, we extend the notion of oral messages to a notion of signed messages by adding a third assumption. Namely, we assume that the signature of a loyal general cannot be forged, that any alteration of a message signed by him will be detected, and that anyone is able to verify the authenticity of the signature of a general. Please note that this does not say anything about the signatures of the traitors, e.g. they could cooperate or forge each others signatures.

Signed Message Algorithm In our algorithm, we will use the notation $\{\!|\ x\ |\!\}_{k_i^{-1}}$ to denote value x signed by general – or lieutenant – i. Additionally, we note that $\{\!|\ \{\!|\ x\ |\!\}_{k_i^{-1}}\ |\!\}_{k_j^{-1}}$ denotes the construction $\{\!|\ x\ |\!\}_{k_i^{-1}}$ signed by general j. Furthermore, each lieutenant i maintains a set V_i of received properly signed orders – not to be confused with received messages [42].

We will also assume a function $Choice(V)$ [42]. This function is applied to a set of orders V to obtain one single order. If the set V consists of only a single element v, we have $Choice(V) = v$. Furthermore, we have that $Choice(\emptyset) = 0$. Lamport, Shostak and Pease [42] mentions the median of the ordered set of arguments V as a possible implementation of $Choice(V)$.

We define the algorithm $SignedMessage(m)$ in Theorem 4.4 [42]. In this algorithm, the commanding general sends all lieutenants a signed order. All these lieutenants sign this message themselves, and pass it on to the other lieutenants, who will do the same thing, until all lieutenants have signed and received the message.

Theorem 4.4 (Signed message algorithm [42]). *The algorithm $SignedMessage(m)$ is defined, with initially $V_i = \emptyset$, as follows:*

1. *the commanding general signs and sends his value to every lieutenant;*

2. *for each* i,
   a) *if lieutenant* i *receives a message of the form* $\{\!|\ v\ |\!\}_{k_0^{-1}}$ *from the commander, and he has not yet received any order,*
      i. *he lets* V_i *equal* $\{v\}$;
      ii. *he sends the message* $\{\!|\ \{\!|\ v\ |\!\}_{k_0^{-1}}\ |\!\}_{k_i^{-1}}$ *to every other lieutenant;*
   b) *if lieutenant* i *receives a message of the form* $\{\!|\ \ldots\ \{\!|\ \{\!|\ v\ |\!\}_{k_0^{-1}}\ |\!\}_{k_{j_1}^{-1}}\ \ldots\ |\!\}_{k_{j_k}^{-1}}$ *and* v *is not in the set* V_i,
      i. *he adds* v *to* V_i;
      ii. *if* $k < m$, *he sends the message* $\{\!|\ \{\!|\ \ldots\ \{\!|\ \{\!|\ v\ |\!\}_{k_0^{-1}}\ |\!\}_{k_{j_1}^{-1}}\ \ldots\ |\!\}_{k_{j_k}^{-1}}\ |\!\}_{k_i^{-1}}$ *to every lieutenant other than* $j_1, \ldots, j_k$;
3. *for each* i, *when lieutenant* i *will not receive any more messages, he obeys order* $\mathbf{Choice}(V_i)$.

As the last step of the algorithm states that the lieutenant will obey the order as soon as he has received all messages to be received, we should note how he should know this [42]. First of all, one can easily show, by mathematical induction on k, that for each sequence of lieutenants $j_1, \ldots, j_k$ with $k \leq m$, a lieutenant can receive at most one message with this sequence of signatures, starting with the signature of the commander. Thus, there are at most $(m-1)^2$ messages to be received. If we would require each lieutenant i to send a report that he is not going to send a message when v is already in his set V_i, it is easily seen when all messages are received.

Example In Figure 4.8, the result of the $SignedMessage(m)$ algorithm is shown for three participants with a traitor as general [42]. In the example, the general sends two different values to the two lieutenants. This results in both lieutenants obtaining the set $V_1 = V_2 = \{v_0', v_0''\}$ and making the decision $Choice(\{v_0', v_0''\})$. Please note that, in this case, the lieutenants can see that the general is dishonest, as his signature appears on two conflicting messages.

We do not show an example of dishonest lieutenant, as a message is required to be signed by the commanding general. Therefore, the only two options a disloyal lieutenant has are sending a malformed message or sending no message. As both options would result in his influence in the decision making process to be ignored, he does not have any effect at all.

Correctness To prove the correctness of the signed messages algorithm, we first formalise our objective [42]. As stated in Theorem 4.5, the algorithm $SignedMessage(m)$ should solve the problem of the Byzantine commander for any m, given at most m traitors. The proof is as follows.

$SignedMessage(m)$ [42]. We prove Theorem 4.5. Firstly, we show that, if the commanding general is loyal, all lieutenants obey his order. In this case, he sends his signed order $\{\!|\ v_0\ |\!\}_{k_0^{-1}}$ to every lieutenant. Thus, every loyal lieutenant will receive

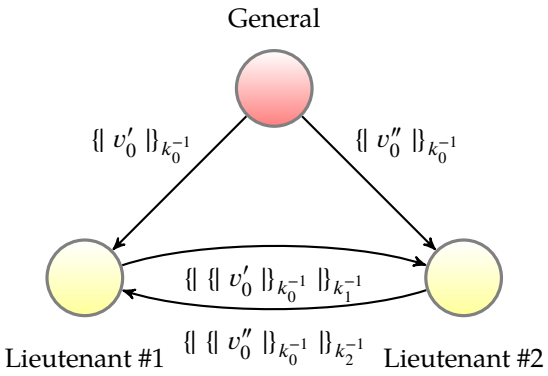


Figure 4.8: The result of $SignedMessage(1)$ with a dishonest general [42]

his order v_0. Because no traitor can forge the signature of the general to create a message of the form $\{\!|\ v_0' \ |\!\}_{k_0^{-1}}$, the loyal lieutenant cannot receive forged messages containing a conflicting order. Therefore, every loyal lieutenant i has $V_i = \{v_0\}$ and will thus obey the order, as $v_0 = Choice(v_0)$. In the case of a loyal commander, the first requirement of the Byzantine commander problem follows from the second, which we just proved. We consider a dishonest general, and show that all loyal lieutenants will obey the same order. Two loyal lieutenants i and j will obey the same order, when the sets of orders V_i and V_j are equal. Thus, we need to show that, if i adds an order v to V_i, lieutenant j will do the same. To do this, we show that j receives a properly signed message containing order v. If i receives order v in step 2.a. of the algorithm, he sends it to j in step 2.a.ii., resulting in j receiving the order. If i adds order v in step 2.b.i. of the algorithm, he received a message signed by lieutenants $j_1, \ldots, j_k$, with $k \leq m$. If j is in the set $j_1, \ldots, j_k$, he already received the v, otherwise, we have two cases: namely, $k < m$ and $k = m$. In the case that $k < m$, i sends the message $\{\!|\ \{\!|\ \ldots \{\!|\ \{\!|\ v \ |\!\}_{k_0^{-1}} \ |\!\}_{k_{j_1}^{-1}} \ldots \ |\!\}_{k_{j_k}^{-1}} \ |\!\}_{k_i^{-1}}$ to j, resulting in him receiving it. In the case that $k = m$, we know that at least one lieutenant in the set $j_1, \ldots, j_m$ is loyal, as we have a disloyal general, and, thus, at most $m - 1$ dishonest lieutenants. This loyal lieutenant must have sent the order v to j when receiving it for the first time. Therefore, j must have received this value. □

Theorem 4.5 ($SignedMessage(m)$ [42])**.** *For any m, $SignedMessage(m)$ solves the problem of the Byzantine commander with at most m traitors.*

4.3.3 Application to Service Oriented Computing

If we presume the high frequency trading system we presented in Figure 4.3. Now, if one of the stock information services, say *stocks_3*, is rogue, we still have three honest services, as shown in Figure 4.9. This enables us to apply one of the existing solutions to the problem of the Byzantine generals in order to obtain the correct result.

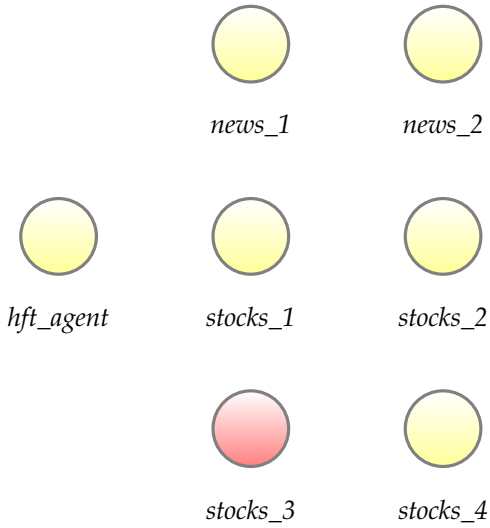


Figure 4.9: The example of Figure 4.3 with a rogue service

Following Section 4.3.1, we can at least achieve a Byzantine fault tolerance of less than one third. As we saw in Section 4.3.2, we can use signatures to recover from an even larger amount of dishonest services in our service chain. This way, we can easily mitigate service poisoning, as long as we have a honest majority.

In the example of high frequency trading systems with a rogue stocks information service, we could impose the algorithm of Theorem 4.4 on the stocks information services in the service composition. As we only want to know the correct information at a single point, we can simplify the algorithm to the viewpoint of a single service, thereby lowering the amount of messages to be sent, i.e. only the message chains that will end up at that single service are continued. This way, we can use the knowledge gained from the Byzantine generals problem to mitigate the need of trust.

5 Related Work

To place the present report in the context of current research, we will discuss noteworthy related work on the discussed topics. For the ease of the reader, this section will divide the topics following the same borders as the report itself does. Therefore, we will start with a discussion of related work on the security of service oriented systems in Section 5.1. This is followed by a discussion of the present research on secure composition in Section 5.2. Finally, we will survey the related work on trust infrastructures in Section 5.3.

5.1 Security in Service Oriented Computing

Currently, systems that protect the quality of service of a service oriented system make heavy use of historical data [e.g. 47, 38] or (third party) trust [e.g. 21]. However, an approach based on historical data is not useful for security for three reasons. Firstly, security is an ongoing game between adversaries and engineers, where previously secure approaches are rendered useless and new secure methods arise [57]. Due to this, historical data shows how well a system could cope with the attacks of yesterday, but gives no information on the issues of today.

Secondly, historical data does not show how many attacks have happened, but how many known attacks have happened. Thus, there may have been more breaches of security than reflected in the measurements. As a lot of attacks make use of exploits of previously unknown vulnerabilities and a successful attack is characterised by being unnoticed, this is a serious issue with the use of statistics for security.

Thirdly, security metrics are commonly infeasible [10]. This is due to the fact that the efforts of an adversary tend to be linear, as opposed to the exponential work needed to be done by those protecting the system [57].

It should be noted, though, that historical data can be used to detect abnormal – and possibly suspicious – behaviour [69, 75, 26]. However, these system do suffer from false positives, when they try to find novel attacks, which results in honest users getting locked out. Furthermore, this still requires the collection of historical data, which does not fit in with the uncoupled nature of services, as we cannot assume any prior contact or knowledge.

In Verberkt [71], an attacker model for service oriented computing was proposed. This model consisted of several layers where an attacker is to be foiled, namely, an overall strategic layer, an organisational layer, and an application layer, with the possibility of extension with an infrastructure layer and a physical layer. The attack surfaces of service oriented systems where shown to be the services themselves, the communication between them, and the compositions as a whole.

5.2 Secure Composition

On the field of secure composition, research started out by showing that certain common security properties cannot be composed. For example, Jürjens [37] proves that the applied flow model of McLean [50] and the probabilistic non-interference model of Gray [30] are not to be composed. On the other hand, some security properties have been proven to hold under composition, such as the hook-up property of information flow security of Varadharajan [70] and the strong non-interference property of Meadows [51] [35]. Another example can be found in Abadi and Lamport [2], where the authors provide a proof method for the composition of security properties. However this method restricts itself to the framework of Alpern and Schneider [6], which leaves most security properties behind, according to McLean [49].

There also has been prior research concerned with secure composition frameworks. A framework for security properties that do hold under composition and refinement has been proposed by Jürjens [35], which specifically concerns the property of secrecy. This secrecy property is also preserved under refinement [36]. However, this work is only a first step on the ladder of modular development of secure software systems as it only concerns one property.

Datta et al. [18] provide us with a means of composing protocols such that they preserve their security protocols. The work of Datta et al. [18] is used as important groundwork in the current research. Nevertheless, due to the fact that, in service oriented computing, we cannot assume any prior communication, we have an additional issue of trust and honesty. Therefore, we need to extent this work to also cover service orientation, and not only protocols. The differences between protocols and service oriented computing, and the way in which this affects the present research, are discussed extensively in Section 2.4.

5.3 Trust Infrastructures

Several trust infrastructures have been discussed and deployed. The most widely known are the X.509 architecture [4, 5] and the web-of-trust implementation of PGP [3]. These models represent the hierarchical and the decentralised method of distributing trust [44]. However, these approaches consider the architecture that could be used to implement certification-based trust, whereas we are concerned with reasonable means to derive trust to cope with the problem of opaque services. In other words, if a certification-based approach where chosen, these results can be used as a complement to this approach.

On the field of reputation-based trust, we merely discussed the basic concepts, as a full discussion of all possibilities is out of scope for the present research. Extensive models of reputation-based trust have been given by Maurer [46], Gutscher [31], Dimitrakos [21]. It should be noted that most of these models suffer from the paradox of trust, as defined by Pavlovic [56]. In the work posing this problem, Pavlovic [56] proposes a direction to coping with this paradox of trust. Nevertheless, the real solution of this paradox is still subject to research.

6 Conclusion and Discussion

To conclude on this research, we will return to the stated research questions and review the answers the research supplied. Afterwards, we will discuss the problem statement itself.

Research Question 1. How can we model services such that their key characteristics, namely uncoupled, composable, and opaqueness, and their distinction from protocols and other modular constructs are honoured?

We model service compositions using partially ordered multisets (pomsets) in Section 2.3. In this model, the uncoupled nature of services is emphasised by the loose structure of a pomset. As there is only a partial order in service compositions, and no order between services that are not composed, this characteristic is fully satisfied. Incidentally, this directly shows the ease with which services can be composed in this model.

Given that the uncoupled nature is one of the characteristics of services that stands out, as we stated in Chapter 2, the distinction from other modular constructs is honoured by definition. In this case, the fact that we use a model based on pomsets is the key to this requirement.

Research Question 2. On what characteristics differ services from protocols such that the work on secure composition of protocols by Datta et al. [18] is not directly applicable to services?

In Section 2.4, it is shown that, where protocols are strictly defined sequences of communications, services constitute uncoupled building blocks for a distributed software system. Additionally, services expose access to computing capabilities, which gives them much more power and possibilities than protocols can offer. Finally, it should be noted that protocols belong in the network infrastructure, whereas services can provide capabilities in a broad range from architecture to software.

Research Question 3. How can limitations raised by opaque services when applying the secure composition technique of Datta et al. [17] be solved?

As shown in Chapter 4, to solve the problem of opaque services, we are in need of trust or a form of insurance. As trust is a rather abstract concept, we pursue this by asking ourselves Research Question 6. Another direction discussed is stated in Research Question 7, where we rely on a majority of honest participants to mitigate the risk of service poisoning by a dishonest service.

Research Question 4. How can the state explosion when applying the secure composition technique of Datta et al. [18] be prevented?

The risk of state explosion is not solved in depth, which is left for future research. However, there are general pointers that should mitigate this risk in practical cases. Most noteworthy, a standardisation of environment invariants ensures that cross-proving of satisfaction of invariants is often not necessary, as those invariants are equal.

Research Question 5. What attacker model do we need to reflect all the challenges specific to the secure composition of services?

The main attacker model is based upon the well-known model of Dolev and Yao [22], which is discussed in Section 3.5. This means that an attacker can intercept, alter and send arbitrary messages. However, an attacker is not able to break the cryptographic constructs used.

For service oriented computing, the attacker model is extended by a notion of service poisoning. This refers to an attack where a dishonest service gives a wrong result to one particular service, in order to poison his computations and results. This attack would not exist in traditional cases, as it results from a dishonest service in your service composition.

Research Question 6. How can we implement trust in our model, thereby keeping in mind the paradox of trust of Pavlovic [56]?

The notion of trust of Verberkt [71] is used as starting point in Section 4.1. For the implementation of trust, two directions are discussed: namely, certification, in Section 4.1.1, and reputation-based trust, in Section 4.1.2.

Certification requires a trusted third party that is able to perform high quality assessments of the honesty of services. If this party decides that a service is honest, he issues a certificate that can be used by other services to verify that this third party is indeed of the opinion that the assessed service is trustworthy. In our model, we would simply require the existence of such a valid certificate to decide whether we should or should not trust a service.

A reputation infrastructure works by collecting opinions about the reputation of a certain service from other nodes in the network. Using some sort of weighing and combining function, these opinions are combined to a reputation score, that can be used to base trust upon.

The paradox of trust does not exist in the first solution, as we expect the trusted third party to perform a thorough assessment, whereby any adversarial tendencies should be found. Reputation-based trust does suffer from the paradox of trust. The main mitigation can be found with making the trust less abstract. Thus, by qualifying the trust and taking into account how it was generated.

Research Question 7. How can we apply solutions for the problem of the Byzantine generals of Lamport, Shostak and Pease [42] to compensate for dishonest participants?

In Section 4.3.3, it is shown that the problem of the Byzantine generals can be used as a model for coping with service poisoning. The best solution requires the usage of cryptographic signatures, as explained in Section 4.3.2, to mitigate the adverse intentions of a dishonest service completely, given a majority of honest services. In this

case, all services act as if they are Byzantine generals, whereby the goal is to get the correct result to the service consumer. For this reason, communications that do not affect the invoking party can be omitted.

Problem Statement. How can we guarantee security properties in a service oriented architecture, thereby taking into account the characteristics of service oriented computing in general and the uncoupled and opaque characteristics specifically?

Firstly, we model services and service compositions using pomsets, after examining them thoroughly. This way, the important characteristics of service oriented computing were identified, and an applicable formalism was identified and put in place. The model for service compositions allows us to reason about services and continue the pursuit of securely composed services.

Secondly, the secure composition framework for protocols of Datta et al. [18] was discussed. This model was, after discussing the important differences, translated to a framework that can be used in a service oriented context. This resulted in a means of securely composing services and proving their security.

However, the proposed model for securely composing services is built under the assumption that the services that are part of the service composition itself are honest. Therefore, we proposed methods for deriving trust in services as one alternative to solve this problem, and a method to mitigate a dishonest service given a honest majority as another alternative.

Given these steps, the current research gives a solution to guaranteeing security properties in a service oriented architecture, by modelling the services fittingly using pomsets, and proposing a framework for the secure composition of services. In addition, methods of deriving or mitigating trust are discussed, such that the framework becomes usable in practical situations.

6.1 Future Research

In future work, a more definitive solution to the state explosion resulting from the application of the secure composition method to a large number of nodes needs to be given. Although the current pointers are helpful in practical situations, this does not suffice as a formal solution to this problem. It is suggested to take non-conflicting variables into account, e.g. if a certain variable is declared secret and never shared, another service can never break the secrecy of this variable.

For easier verification using the proposed framework, an automated verification tool should be developed. An example of such a tool for protocols using process calculus can be found in Blanchet [12], which uses Prolog rules. The introduction of a tool for our framework makes it possible to automate the verification process, thereby generating a much faster and more practical applicable technique for the secure composition of services. Additionally, this makes it possible to consider the effects of multiple concurrent runs of a service composition as attack vector.

In our model, public keys are expected to be publicly known and correctly bound to the corresponding principals, given that those principals are honest. In a practical

situation, a reasonable framework for key management and distribution is required. Of course, there exist many solutions that provide such an infrastructure, which could even be built upon the discussed trust infrastructure. Nevertheless, for practical use, it is important that a means of key management and distribution is incorporated and that the framework is revalidated using this new addition.

Current research concerning the security of services is often focussed at access control. This refers to control over which principals have access to which resources [7]. In our framework, this would concern which principals should have access to secret variables, or which principals should be allowed to authenticate. For future research, it is interesting to review whether access control, which is commonly used in practice, could be considered in the framework, such that the practical implications of current access control solution are reflected by the model.

Bibliography

[1] Martín Abadi and Andrew D. Gordon. 'A calculus for cryptographic protocols: the spi calculus'. In: *Proceedings of the 4th ACM conference on Computer and communications security*. CCS '97. Zurich, Switzerland: ACM, 1997, pp. 36–47. ISBN: 0-89791-912-2. DOI: 10.1145/266420.266432 (cit. on p. 27).

[2] Martín Abadi and Leslie Lamport. 'Composing specifications'. In: *ACM Trans. Program. Lang. Syst.* 15.1 (Jan. 1993), pp. 73–132. ISSN: 0164-0925. DOI: 10.1145/151646.151649 (cit. on p. 82).

[3] Alfarez Abdul-Rahman. 'The PGP Trust Model'. In: *Electronic Commerce*. 1996 (cit. on p. 82).

[4] C. Adams and S. Farrell. *Internet X.509 Public Key Infrastructure Certificate Management Protocols*. RFC 2510 (Proposed Standard). Obsoleted by RFC 4210. Internet Engineering Task Force, Mar. 1999. URL: http://www.ietf.org/rfc/rfc2510.txt (cit. on pp. 64, 82).

[5] C. Adams et al. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. RFC 4210 (Proposed Standard). Internet Engineering Task Force, Sept. 2005. URL: http://www.ietf.org/rfc/rfc4210.txt (cit. on pp. 64, 82).

[6] Bowen Alpern and Fred B. Schneider. 'Defining liveness'. In: *Information Processing Letters* 21.4 (1985), pp. 181–185. ISSN: 0020-0190. DOI: 10.1016/0020-0190(85)90056-0. URL: http://www.sciencedirect.com/science/article/pii/0020019085900560 (cit. on p. 82).

[7] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Second. Wiley Publishing, 2008. ISBN: 978-0-470-06852-6 (cit. on pp. 49, 50, 86).

[8] Boaz Barak et al. 'On the (Im)possibility of Obfuscating Programs'. In: *Advances in Cryptology — CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001, pp. 1–18. ISBN: 978-3-540-42456-7. DOI: 10.1007/3-540-44647-8_1 (cit. on p. 1).

[9] Mihir Bellare and Phillip Rogaway. 'Entity Authentication and Key Distribution'. In: *Advances in Cryptology — CRYPTO' 93*. Ed. by Douglas Stinson. Vol. 773. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1994, pp. 232–249. ISBN: 978-3-540-57766-9. DOI: 10.1007/3-540-48329-2_21 (cit. on p. 53).

[10] Steven M. Bellovin. 'On the Brittleness of Software and the Infeasibility of Security Metrics'. In: *Security Privacy, IEEE* 4.4 (July 2006), p. 96. ISSN: 1540-7993. DOI: 10.1109/MSP.2006.101 (cit. on p. 81).

[11] B. Blanchet. 'Automatic proof of strong secrecy for security protocols'. In: *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. May 2004, pp. 86–100. DOI: 10.1109/SECPRI.2004.1301317 (cit. on p. 50).

[12] Bruno Blanchet. 'An Efficient Cryptographic Protocol Verifier Based on Prolog Rules'. In: *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, June 2001, pp. 82–96 (cit. on p. 85).

[13] Mario Bravetti and Gianluigi Zavattaro. 'Service oriented computing from a process algebraic perspective'. In: *Journal of Logic and Algebraic Programming* 70.1 (2007), pp. 3–14. ISSN: 1567-8326. DOI: 10.1016/j.jlap.2006.05.002. URL: http://www.sciencedirect.com/science/article/pii/S1567832606000270 (cit. on p. 19).

[14] Sergey Brin and Lawrence Page. 'The anatomy of a large-scale hypertextual Web search engine'. In: *Computer Networks and ISDN Systems* 30.1–7 (1998), pp. 107–117. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X. URL: http://www.sciencedirect.com/science/article/pii/S016975529800110X (cit. on p. 66).

[15] Miguel Castro and Barbara Liskov. 'Practical byzantine fault tolerance and proactive recovery'. In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), pp. 398–461. ISSN: 0734-2071. DOI: 10.1145/571637.571640. URL: http://doi.acm.org/10.1145/571637.571640 (cit. on p. 71).

[16] Iliano Cervesato, Catherine Meadows and Dusko Pavlovic. *Deriving Key Distribution Protocols and their Security Properties*. Tech. rep. CMU-CS-06-172. Carnegie Mellon University, 2006 (cit. on p. 47).

[17] Anupam Datta et al. 'A derivation system and compositional logic for security protocols'. In: *Journal of Computer Security* 13 (3 May 2005), pp. 423–482. ISSN: 0926-227X. URL: http://dl.acm.org/citation.cfm?id=1145948.1145952 (cit. on pp. 1, 3, 26, 35, 40, 44, 46–48, 53, 83).

[18] Anupam Datta et al. 'Secure protocol composition'. In: *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*. FMSE '03. Washington, D.C.: ACM, 2003, pp. 11–23. ISBN: 1-58113-781-8. DOI: 10.1145/1035429.1035431 (cit. on pp. III, 1–3, 21–23, 25–30, 32–49, 53, 54, 82, 83, 85).

[19] W. Diffie and M. Hellman. 'New directions in cryptography'. In: *Information Theory, IEEE Transactions on* 22.6 (Nov. 1976), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638 (cit. on pp. 37, 40).

[20] Whitfield Diffie, Paul C. Oorschot and Michael J. Wiener. 'Authentication and authenticated key exchanges'. In: *Designs, Codes and Cryptography* 2 (2 1992), pp. 107–125. ISSN: 0925-1022. DOI: 10.1007/BF00124891 (cit. on pp. 42, 53, 57).

[21] Theo Dimitrakos. 'A Service-Oriented Trust Management Framework'. In: *Trust, Reputation, and Security: Theories and Practice*. Ed. by Rino Falcone et al. Vol. 2631. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, pp. 53–72. DOI: 10.1007/3-540-36609-1_6. URL: http://dx.doi.org/10.1007/3-540-36609-1_6 (cit. on pp. 81, 82).

[22] Danny Dolev and Andrew C. Yao. 'On the security of public key protocols'. In: *Information Theory, IEEE Transactions on* 29.2 (Mar. 1983), pp. 198–208. ISSN: 0018-9448. DOI: 10.1109/TIT.1983.1056650 (cit. on pp. 28, 40, 51, 60, 63, 84).

[23] Nancy Durgin, John Mitchell and Dusko Pavlovic. 'A Compositional Logic for Protocol Correctness'. In: *Computer Security Foundations Workshop, IEEE* (2001), p. 0241. DOI: 10.1109/CSFW.2001.930150 (cit. on pp. 27, 30–32).

[24] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005. ISBN: 0131858580 (cit. on pp. 1, 6, 17).

[25] F.J.T. Fabrega, J.C. Herzog and J.D. Guttman. 'Strand spaces: why is a security protocol correct?' In: *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*. May 1998, pp. 160–171. DOI: 10.1109/SECPRI.1998.674832 (cit. on p. 30).

[26] P. García-Teodoro et al. 'Anomaly-based network intrusion detection: Techniques, systems and challenges'. In: *Computers & Security* 28.1–2 (2009), pp. 18–28. ISSN: 0167-4048. DOI: 10.1016/j.cose.2008.08.003. URL: http://www.sciencedirect.com/science/article/pii/S0167404808000692 (cit. on p. 81).

[27] Jay L. Gischer. 'The equational theory of pomsets'. In: *Theoretical Computer Science* 61.2-3 (1988), pp. 199–224. ISSN: 0304-3975. DOI: 10.1016/0304-3975(88)90124-7. URL: http://www.sciencedirect.com/science/article/pii/0304397588901247 (cit. on pp. 12, 14, 15, 17, 19).

[28] Joseph A. Goguen and José Meseguer. 'Security Policies and Security Models'. In: *IEEE Symposium on Security and Privacy'82*. 1982, pp. 11–20 (cit. on p. 50).

[29] Andrew D. Gordon and Alan Jeffrey. 'Typing Correspondence Assertions for Communication Protocols'. In: *Electronic Notes in Theoretical Computer Science* 45 (2001), pp. 119–140. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(04)80959-9. URL: http://www.sciencedirect.com/science/article/pii/S1571066104809599 (cit. on p. 53).

[30] III Gray J.W. 'Toward a mathematical foundation for information flow security'. In: *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*. May 1991, pp. 21–34. DOI: 10.1109/RISP.1991.130769 (cit. on p. 82).

[31] Andreas Gutscher. 'A Trust Model for an Open, Decentralized Reputation System'. In: *Trust Management*. Ed. by Sandro Etalle and Stephen Marsh. Vol. 238. IFIP International Federation for Information Processing. Springer Boston, 2007, pp. 285–300. ISBN: 978-0-387-73654-9. DOI: 10.1007/978-0-387-73655-6_19. URL: http://dx.doi.org/10.1007/978-0-387-73655-6_19 (cit. on p. 82).

[32] C. A. R. Hoare. 'An axiomatic basis for computer programming'. In: *Commununications of the ACM* 12.10 (Oct. 1969), 576–580. ISSN: 0001-0782. DOI: 10.1145/363235.363259 (cit. on pp. 1, 25, 26).

[33] Yang Hongli et al. 'Exploring the Connection of Choreography and Orchestration with Exception Handling and Finalization/Compensation'. In: *Formal Techniques for Networked and Distributed Systems – FORTE 2007*. Ed. by John Derrick and Jüri Vain. Vol. 4574. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, pp. 81–96. ISBN: 978-3-540-73195-5. DOI: 10.1007/978-3-540-73196-2_6 (cit. on p. 10).

[34] ISO and IEC. *Information technology — Syntactic metalanguage — Extended BNF*. Tech. rep. ISO/IEC 14977:1996. International Organization for Standardization and International Electrotechnical Commission, 1996 (cit. on p. 27).

[35] Jan Jürjens. 'Composability of Secrecy'. In: *Information Assurance in Computer Networks*. Ed. by Vladimir Gorodetski, Victor Skormin and Leonard Popyack. Vol. 2052. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001, pp. 28–38. ISBN: 978-3-540-42103-0. DOI: 10.1007/3-540-45116-1_6 (cit. on pp. 1, 22, 48, 51, 82).

[36] Jan Jürjens. 'Secrecy-Preserving Refinement'. In: *FME 2001: Formal Methods for Increasing Software Productivity*. Ed. by José Oliveira and Pamela Zave. Vol. 2021. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001, pp. 135–152. ISBN: 978-3-540-41791-0. DOI: 10.1007/3-540-45251-6_8 (cit. on p. 82).

[37] Jan Jürjens. 'Secure Information Flow for Concurrent Processes'. In: *CONCUR 2000 — Concurrency Theory*. Ed. by Catuscia Palamidessi. Vol. 1877. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, pp. 395–409. ISBN: 978-3-540-67897-7. DOI: 10.1007/3-540-44618-4_29 (cit. on p. 82).

[38] Alexander Keller and Heiko Ludwig. 'The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services'. In: *Journal of Network and Systems Management* 11 (1 2003), pp. 57–81. ISSN: 1064-7570. DOI: 10.1023/A:1022445108617 (cit. on p. 81).

[39] John Kelsey, Bruce Schneier and David Wagner. 'Protocol interactions and the chosen protocol attack'. In: *Security Protocols*. Ed. by Bruce Christianson et al. Vol. 1361. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1998, pp. 91–104. ISBN: 978-3-540-64040-0. DOI: 10.1007/BFb0028162 (cit. on p. 45).

[40] James F Kurose, Keith W Ross and Bhojan Anand. *Computer networking : a top-down approach*. Boston, MA: Pearson/Addison Wesley, 2008. ISBN: 9780321497703 (cit. on p. 21).

[41] Leslie Lamport. 'Time, clocks, and the ordering of events in a distributed system'. In: *Commun. ACM* 21.7 (July 1978), pp. 558–565. ISSN: 0001-0782. DOI: 10.1145/359545.359563. URL: http://doi.acm.org/10.1145/359545.359563 (cit. on p. 47).

[42] Leslie Lamport, Robert Shostak and Marshall Pease. 'The Byzantine Generals Problem'. In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176 (cit. on pp. III, 3, 61, 63, 70–78, 84).

[43] Zhiyao Liang and Rakesh Verma. 'Complexity of Checking Freshness of Cryptographic Protocols'. In: *Information Systems Security*. Ed. by R. Sekar and Arun Pujari. Vol. 5352. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pp. 86–101. ISBN: 978-3-540-89861-0. DOI: 10.1007/978-3-540-89862-7_6 (cit. on p. 49).

[44] Antonio Lioy et al. 'PKI past, present and future'. In: *International Journal of Information Security* 5 (1 2006), pp. 18–29. ISSN: 1615-5262. DOI: 10.1007/s10207-005-0077-9 (cit. on pp. 65, 82).

[45] C. Matthew MacKenzie et al. 'Reference Model for Service Oriented Architecture 1.0'. In: *Architecture* 2006.October (2006), pp. 1–31. URL: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf (cit. on p. 5).

[46] Ueli Maurer. 'Modelling a public-key infrastructure'. In: *Computer Security — ESORICS 96*. Ed. by Elisa Bertino et al. Vol. 1146. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1996, pp. 325–350. DOI: 10.1007/3-540-61770-1_45. URL: http://dx.doi.org/10.1007/3-540-61770-1_45 (cit. on pp. 64, 82).

[47] E. Michael Maximilien and Munindar P. Singh. 'Toward autonomic web services trust and selection'. In: *Proceedings of the 2nd international conference on Service oriented computing*. ICSOC '04. New York, NY, USA: ACM, 2004, pp. 212–221. ISBN: 1-58113-871-7. DOI: 10.1145/1035167.1035198 (cit. on p. 81).

[48] D. McCullough. 'Noninterference and the composability of security properties'. In: *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on*. Apr. 1988, pp. 177–186. DOI: 10.1109/SECPRI.1988.8110 (cit. on pp. 1, 48).

[49] J. McLean. 'A general theory of composition for a class of ldquo;possibilistic rdquo; properties'. In: *Software Engineering, IEEE Transactions on* 22.1 (Jan. 1996), pp. 53–67. ISSN: 0098-5589. DOI: 10.1109/32.481534 (cit. on p. 82).

[50] J. McLean. 'Security models and information flow'. In: *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*. May 1990, pp. 180–187. DOI: 10.1109/RISP.1990.63849 (cit. on p. 82).

[51] C. Meadows. 'Using traces based on procedure calls to reason about composability'. In: *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*. May 1992, pp. 177–188. DOI: 10.1109/RISP.1992.213262 (cit. on p. 82).

[52] Catherine Meadows. 'Open issues in formal methods for cryptographic protocol analysis'. In: *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*. Vol. 1. 2000, 237–250 vol.1. DOI: 10.1109/DISCEX.2000.824984 (cit. on pp. 1, 48).

[53] Robin Milner, Joachim Parrow and David Walker. 'A calculus of mobile processes, I'. In: *Information and Computation* 100.1 (1992), pp. 1–40. ISSN: 0890-5401. DOI: 10.1016/0890-5401(92)90008-4. URL: http://www.sciencedirect.com/science/article/pii/0890540192900084 (cit. on p. 27).

[54] Michael P. Papazoglou. 'Service-Oriented Computing: Concepts, Characteristics and Directions'. In: *Web Information Systems Engineering, International Conference on* (2003), p. 3. DOI: 10.1109/WISE.2003.1254461 (cit. on pp. 1, 5–7, 17).

[55] Michael P. Papazoglou et al. 'Service-Oriented Computing: State of the Art and Research Challenges'. In: *Computer* 40.11 (Nov. 2007), pp. 38–45. ISSN: 0018-9162. DOI: 10.1109/MC.2007.400 (cit. on p. 1).

[56] Dusko Pavlovic. 'Quantifying and qualifying trust: Spectral decomposition of trust networks'. In: *CoRR* abs/1011.5696 (2010) (cit. on pp. 3, 63, 66, 68, 69, 82, 84).

[57] Dusko Pavlovic. 'The Unreasonable Ineffectiveness of Security Engineering: An Overview'. In: *8th IEEE International Conference on Software Engineering and Formal Methods*. SEFM '10. Sept. 2010, pp. 12–18. DOI: 10.1109/SEFM.2010.10 (cit. on pp. 1, 81).

[58] Dusko Pavlovic. 'Tracing the Man in the Middle in Monoidal Categories'. In: *Coalgebraic Methods in Computer Science*. Ed. by Dirk Pattinson and Lutz Schröder. Vol. 7399. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2012, pp. 191–217. ISBN: 978-3-642-32783-4. URL: http://www.springerlink.com/content/y71ut4187v1k0573/abstract/ (visited on 18/10/2012) (cit. on p. 47).

[59] Dusko Pavlovic and Catherine Meadows. 'Deriving Secrecy in Key Establishment Protocols'. In: *Computer Security – ESORICS 2006*. Ed. by Dieter Gollmann, Jan Meier and Andrei Sabelfeld. Vol. 4189. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, pp. 384–403. ISBN: 978-3-540-44601-9. DOI: 10.1007/11863908_24 (cit. on pp. 47, 50).

[60] M. Pease, R. Shostak and L. Lamport. 'Reaching Agreement in the Presence of Faults'. In: *J. ACM* 27.2 (Apr. 1980), pp. 228–234. ISSN: 0004-5411. DOI: 10.1145/322186.322188. URL: http://doi.acm.org/10.1145/322186.322188 (cit. on p. 71).

[61] Chris Peltz. *Web Services Orchestration*. Tech. rep. Hewlett Packard, 2003 (cit. on p. 8).

[62] Chris Peltz. 'Web Services Orchestration and Choreography'. In: *Computer* 36 (2003), pp. 46–52. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1236471 (cit. on pp. 8, 10).

[63] Vaughan Pratt. 'Modeling concurrency with partial orders'. In: *International Journal of Parallel Programming* 15 (1 1986), pp. 33–71. ISSN: 0885-7458. DOI: 10.1007/BF01379149 (cit. on p. 19).

[64] Vaughan Pratt. 'Some constructions for order-theoretic models of concurrency'. In: *Logics of Programs*. Ed. by Rohit Parikh. Vol. 193. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1985, pp. 269–283. ISBN: 978-3-540-15648-2. DOI: 10.1007/3-540-15648-8_22 (cit. on p. 19).

[65] Vaughan Pratt. 'The pomset model of parallel processes: Unifying the temporal and the spatial'. In: *Seminar on Concurrency*. Ed. by Stephen Brookes, Andrew Roscoe and Glynn Winskel. Vol. 197. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1985, pp. 180–196. ISBN: 978-3-540-15670-3. DOI: 10.1007/3-540-15670-4_9 (cit. on pp. 12, 14).

[66] Jinghai Rao, Peep Kungas and Mihhail Matskin. 'Logic-based Web services composition: from service description to process model'. In: *Web Services, 2004. Proceedings. IEEE International Conference on*. July 2004, pp. 446–453. DOI: 10.1109/ICWS.2004.1314769 (cit. on p. 19).

[67] Geoffrey Smith. 'Principles of Secure Information Flow Analysis'. In: *Malware Detection*. Ed. by Mihai Christodorescu et al. Vol. 27. Advances in Information Security. Springer US, 2007, pp. 291–307. ISBN: 978-0-387-44599-1. DOI: 10.1007/978-0-387-44599-1_13 (cit. on pp. 50, 51).

[68] Apostolos Syropoulos. 'Mathematics of Multisets'. In: *Multiset Processing*. Ed. by Cristian Calude et al. Vol. 2235. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001, pp. 347–358. ISBN: 978-3-540-43063-6. DOI: 10.1007/3-540-45523-X_17 (cit. on p. 12).

[69] Chih-Fong Tsai et al. 'Intrusion detection by machine learning: A review'. In: *Expert Systems with Applications* 36.10 (2009), pp. 11994–12000. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2009.05.029. URL: http://www.sciencedirect.com/science/article/pii/S0957417409004801 (cit. on p. 81).

[70] V. Varadharajan. 'Hook-up property for information flow secure nets'. In: *Computer Security Foundations Workshop IV, 1991. Proceedings*. June 1991, pp. 154–175. DOI: 10.1109/CSFW.1991.151582 (cit. on p. 82).

[71] S.L.C. Verberkt. *A First Step Towards Trustworthy ICT Service Chains*. Tech. rep. IBM Nederland, Nov. 2011 (cit. on pp. 1, 5–8, 10, 17, 49, 50, 63, 81, 84).

[72] Hugo Vieira, Luís Caires and João Seco. 'The Conversation Calculus: A Model of Service-Oriented Computation'. In: *Programming Languages and Systems*. Ed. by Sophia Drossopoulou. Vol. 4960. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pp. 269–283. ISBN: 978-3-540-78738-9. DOI: 10.1007/978-3-540-78739-6_21 (cit. on p. 19).

[73] J.M. Wing. 'A call to action look beyond the horizon'. In: *IEEE Security and Privacy* 1.6 (Nov. 2003), pp. 62–67. ISSN: 1540-7993. DOI: 10.1109/MSECP.2003.1253571 (cit. on pp. 1, 48).

[74] T.Y.C. Woo and S.S. Lam. 'A semantic model for authentication protocols'. In: *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*. May 1993, pp. 178–194. DOI: 10.1109/RISP.1993.287633 (cit. on p. 53).

[75] Shelly Xiaonan Wu and Wolfgang Banzhaf. 'The use of computational intelligence in intrusion detection systems: A review'. In: *Applied Soft Computing* 10.1 (2010), pp. 1–35. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2009.06.019. URL: http://www.sciencedirect.com/science/article/pii/S1568494609000908 (cit. on p. 81).

[76] Hangjung Zo, Derek L. Nazareth and Hemant K. Jain. 'Security and performance in service-oriented applications: Trading off competing objectives'. In: *Decision Support Systems* 50.1 (2010), pp. 336–346. ISSN: 0167-9236. DOI: 10.1016/j.dss.2010.09.002. URL: http://www.sciencedirect.com/science/article/pii/S0167923610001715 (cit. on p. 1).

Acronyms

API application programming interface. III

PKI public key infrastructure. 64–66
pomset partially ordered multiset. III, VII, 2, 12–22, 47, 48, 83, 85

SOA service oriented architecture. III, 1, 6, 20, 60

TLS transport layer security. 64, 65

URL uniform resource locator. 5, 65