AFRL-AFOSR-UK-TR-2011-0005

PrOtocols for WatERmarking (POWER)

Christina Kraetzer
Jana Dittmann

Otto-von-Guericke University of Magdeburg
Advanced Multimedia and Security Lab (AMSL)
Universitätsplatz 2
Magdeburg, Germany 39106

EOARD GRANT 093061

March 2011

Final Report for 01 October 2009 to 01 March 2011

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515 Box 14, APO AE 09421

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE ***(DD-MM-YYYY)*** 24-03-2011	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From – To)* 1 October 2009 – 01 March 2011

4. TITLE AND SUBTITLE **PrOtocols for WatERmarking (POWER)**	**5a. CONTRACT NUMBER** FA8655-09-1-3061
	5b. GRANT NUMBER Grant 09-3061
	5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) Christian Kraetzer Jana Dittmann	**5d. PROJECT NUMBER**
	5d. TASK NUMBER
	5e. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Otto-von-Guericke University of Magdeburg Advanced Multimedia and Security Lab (AMSL) Universitätsplatz 2 Magdeburg, Germany 39106	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD Unit 4515 BOX 14 APO AE 09421	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/AFOSR/RSW (EOARD)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) **AFRL-AFOSR-UK-TR-2011-0005**

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking Otto-von-Guericke University of Magdeburg as follows: Within the work on the POWER project the overall problems of realizing information assurance (trustworthiness in the sense of integrity and confidentiality) and information provenance (specifically authentication of digital data objects and their origin) with digital watermarking as an alternative or complement to cryptography for several application scenarios is addressed. The current problem in the application of watermarking techniques is that, while hundreds of different digital watermarking algorithms (as primitives or building blocks) exist, watermarking protocols (the rules for combining the building blocks into a working data communication system) are still a rather immature research field. But exactly such protocols would be required for the implementation of multi-user multiaccess / hierarchical-access scenarios as they are envisioned within POWER. The main goal for the POWER project has therefore been the introduction of a generalized theoretical framework for watermarking protocol generation and (security) verification based on a protocol life-cycle model. The applicability of the introduced generalized, theoretical framework is shown with an exemplarily selected realization / implementation. The technical solution as exemplary realization for POWER includes three different major blocks: First, for the development of the context model XML (extensible markup language) is used, to achieve a machine interpretable description of: the application scenario with its communication tasks, the communication network considered in the application scenario as well as the algorithm characteristics of the involved watermarking procedures. The second major component covered is the protocol generation mechanism. It takes the context model and transfers it into a protocol in the CASPER (abridgment for: Complier for the Analysis of Security PRotocols) formalization language. This largely automated process allows for manual modifications of certain parts, e.g. the description of application scenario specific attacker assumptions to produce application specific formal communication protocol descriptions. The third, and for the project outcome most significant, part covered is the evaluation procedure performing (semi-) automated syntactic and semantic evaluations of the formulated protocols using the CASPER / CSP (Communicating Sequential Processes) compiler and the model checker FDR (Failures-Divergences Refinement). This evaluation procedure performs the required security evaluations on the protocols and selects the most suitable protocol in case alternatives exist. Besides the syntactic and semantic evaluations, a security evaluation of the used primitives has to be performed in the POWER realization of the aforementioned theoretical framework, which is still an important topic for further research work.

15. SUBJECT TERMS

EOARD, Digital Watermarking, Information Fusion

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** JAMES LAWTON Ph. D.
a. REPORT UNCLAS	**b. ABSTRACT** UNCLAS	**c. THIS PAGE** UNCLAS	SAR	96	**19b. TELEPHONE NUMBER** *(Include area code)* +44 (0)1895 616187

PrOtocols for WatERmarking (POWER)

"Final Report – Protocol Descriptions, Simulation and Evaluation Results"

Date Submitted: 28.02.2011 (Project M17)
Name of Organization: Otto-von-Guericke University of Magdeburg, Germany

Main author: Christian Kraetzer
Contributors: Prof. Dr.-Ing. Jana Dittmann, Maik Schott, Ronny Merkel, Robert Altschaffel, Eric Clausing

This document (which replaces all previous POWER reports) is structured as follows: On page 3, an executive summary gives a brief overview over the motivations, goals and results of the POWER project.
Chapter 1 of the report introduces basics on existing approaches for communication protocol modeling, security aspects addressable by watermarking protocols, an approach for channel modeling for watermarking protocols, some considerations on watermarking data and/or object suitability and existing strategies for multiple watermarking.
In chapter 2 the main contribution of the research work within POWER is found. This contribution consists, on one hand, of a theoretical framework for context modeling as well as (semi-)automated protocol generation and verification for watermarking-based secure communication protocols, and on the other hand, a practical realization of this theoretical framework based on the CASPER modeling language.
Chapter 3 focuses on the application of the practical realization of our framework onto selected watermarking application scenarios.
In chapter 4 a very brief overview over efforts on dissemination and public result verification are described, followed in chapter 5 by a summary of the achieved results and a clear identification of limitations of our framework.

Table of contents:

Executive Summary – Goals and Results of the POWER research project

Within the work on the POWER project the overall problems of realizing information assurance (trustworthiness in the sense of integrity and confidentiality) and information provenance (specifically authentication of digital data objects and their origin) with digital watermarking as an alternative or complement to cryptography for several application scenarios is addressed.

For the illustration of the motivation for this research on alternatives or complementary mechanisms for cryptography we use the following example scenario:
A sensor unit (e.g. a reconnaissance UAV) is taking digital images. These images are combined with metadata (geo-location, timestamp, the UAVs ID, etc) and send to a control unit which automatically segments people in the images and adds the segmentation results as further metadata to the image. An operator receives the image and the accumulated metadata and performs further operations, e.g. identifying some of the persons on the images before he forwards everything to the next processor.
During these processing steps, the amount of information stored or communicated together with the original data (digital signatures, timestamps, annotations, etc.) increases significantly. Encryption can be used to protect the confidentiality of the contents, but it does not prevent an observer to notice the amount of data communicated – an information that might easily reveal e.g. that especially significant data objects are transmitted if they are significantly larger than normal data objects originating from a source system.
In a centralized data storage with system-wide access control mechanisms such information leakage can be easily prevented by assuring that a potential attacker can not observe the communication channels in question. But in ad-hoc or dynamic scenarios such observability can hardly be prevented by cryptographic means.

An easy solution for this problem is data hiding, in particular digital watermarking. Here the accumulating metadata is directly embedded as *watermarking message* into the digital source data (or *cover*). By this targeted modification of the cover (which can be designed in a way that it is imperceptible by a human being and can be completely inverted if necessary) the addition of metadata does not increase the size of the data objects to be transmitted and realizes by this an un-observability with regard to the added metadata. Additionally, other security aspects, like integrity verification or object authentication, can also be realized using digital watermarking.

The current problem in the application of watermarking techniques is that, while hundreds of different digital watermarking algorithms (as primitives or building blocks) exist, watermarking protocols (the rules for combining the building blocks into a working data communication system) are still a rather immature research field. But exactly such protocols would be required for the implementation of multi-user multi-access / hierarchical-access scenarios as they are envisioned within POWER.

The main goal for the POWER project is therefore the introduction of a generalized theoretical framework for watermarking protocol generation and (security) verification based on a protocol life-cycle model. The applicability of the introduced generalized, theoretical framework is shown with an exemplarily selected realization / implementation.

The technical solution as exemplary realization for POWER includes three different major blocks: First, for the development of the context model XML (extensible markup language) is used, to achieve a machine interpretable description of: the application scenario with its communication tasks, the communication network considered in the application scenario as well as the algorithm characteristics of the involved watermarking procedures.
The second major component covered is the protocol generation mechanism. It takes the context model and transfers it into a protocol in the CASPER (abridgment for: Complier for the Analysis of Security PRotocols) formalization language. This largely automatized process allows for manual modifications of certain parts, e.g. the description of application scenario specific attacker assumptions to produce application specific formal communication protocol descriptions.
The third, and for the project outcome most significant, part covered is the evaluation procedure performing (semi-)automatized syntactic and semantic evaluations of the formulated protocols using the CASPER / CSP (Communicating Sequential Processes) compiler and the model checker FDR (Failures-Divergences Refinement). This evaluation procedure performs the required security evaluations on the protocols and selects the most suitable protocol in case alternatives exist.
Besides the syntactic and semantic evaluations, a security evaluation of the used primitives has to be performed in the POWER realization of the aforementioned theoretical framework, which is still an important topic for further research work.

In addition to the work on modeling and realization of the framework, its application to selected, relevant application scenarios is illustrated, including the aforementioned multi-user multi-access / hierarchical-access scenarios.

1. State-of-the-art regarding secure communication protocol modeling, watermarking data and/or object suitability and related topics

In this chapter some required basic definitions and explanations are presented within the POWER context. They include, first, a working definition on the concept of secure communication protocols and corresponding security verification approaches, second, some working definitions on security aspects and security requirements, third, considerations on the integration or watermarking channels into communication protocols, fourth, a brief investigation on which kinds of data objects can be watermarked in general and which cannot, and last, a summary on existing strategies for multiple watermarking.

1.1. Approaches to secure communication protocol modeling

It has to be stated first, that considerations on watermarking protocols are much less common in current literature than considerations on watermarking algorithms and their characteristics.

In contrast to previous work on watermarking protocols (e.g. Dittmann, Katzenbeisser, Schallhart and Veith [Dittmann05]), where manual mathematic proofs are employed to ensure security in watermarking based schemes, in the POWER project we transfer the idea of machine-based verification of the security of communication protocols from cryptography to the domain of digital watermarking based media security protocols.

A protocol, or more precisely a **secure communication protocol**, in the sense of the POWER project is defined here as follows:

> *A secure communication protocol is a sequence of interactions or transactions between entities in a specific communications network with the aim of exchanging data under the assertion of pre-selected security aspects.*

According to Pimentel et al. [Pimentel08], **two general approaches to the security verification of communication protocols** exist in the domain of cryptography, which very well represents the research field of security protocols. The **computational complexity approach** validates the used primitives (a.k.a building blocks or components) themselves by proving that there is no (computational/complexity-acceptable) way to obtain a given secret without the correct key[1]. The alternative, the **formal methods approach**, evaluates the whole protocol by examining flaws in the protocol run, while the primitives are often considered as 'secure' (for this reason it is sometimes also called perfect cryptography approach). Since the work in the POWER project aims at analyzing communication protocols and not the underlying functions/primitives, the formal methods approach is chosen here. One inherent problem of this approach to modeling is, that it generally assumes that the used primitives cannot be broken (the

[1] The computational complexity approach claims that primitives are simply functions on strings of bits. A protocol is considered good in this approach if an oracle cannot guess the (crypto) key, or while consuming the computational power at hand the probability of finding the key is slow-growing under a determined threshold. Although providing strong security guarantees, proofs under this approach are in general hard and difficult to automate.

perfect cryptography assumption mentioned above) and that an attack on the protocol must therefore lie in the protocol itself, where an attacker gains the knowledge (e.g. keys) to achieve his intended result. Even with modifications for this approach proposed in literature (e.g. “After 10 transmitted messages, encrypted with the same key, the attacker has broken this key.”), it cannot overcome the fact that the formalization must rely on **external evaluation regarding the primitives**[2]. Nevertheless, this basic approach is chosen for the work done within POWER.

The formal verification methods introduced above can furthermore be divided into **manual security verification** using mathematic proofs (e.g. the media authentication scheme of Dittmann et al. [Dittmann05], presenting a reversible watermarking scheme using cryptographic signatures and hashes) and **automated verification** using tools such as model-checkers. While examples / tools / languages for the security verification of communication protocols using cryptographic mechanisms are common, such examples for the field of watermarking protocols are rare.

Some selected languages that should be mentioned in this context, which are focused on for security evaluations in cryptographic protocols, are:

- CASPER [Lowe98] (which uses CSP (Communicating Sequential Processes))
- CSP [Roscoe94]
- AVISPA (Automated Validation of Internet Security Protocols and Applications; [Vigano05]) – including formalization and verification mechanisms
- REBECA (Reactive Objects Language; http://ece.ut.ac.ir/FML/rebeca.htm)

Also some stand-alone verification tools for such an approach exist, like the model checker FDR (Failures-Divergences Refinement; [FDR]) for machine-based security verification of CSP files.

1.2. Security aspects addressable by watermarking protocols

In IT-security literature, different definitions/classifications for **security aspects** are found. In the context of POWER we use the following definitions:

- **Confidentiality** (including privacy) signifies the concealment of information or resources. Amongst other mechanisms, access control mechanisms support confidentiality in systems. In the context[3] of POWER we distinguish here between two different aspects regarding confidentiality: the concealment of the contents of a message (classical cryptographic confidentiality) and the un-observability of metadata embedded in watermarking schemes (as a distinguishing feature for watermarking-based approaches that can not generally be achieved by usage of cryptographic mechanisms).
- **Integrity** in computer science and telecommunications normally refers to the quality or condition of data to be consistent, complete and unaltered.
- **Authenticity** can be divided into two different aspects: **data-origin-authenticity** is the proof or verification of the data’s origin, genuineness, originality,

[2] This external evaluation, in combination with well defined interfaces, also allows for exchangeability of components used in the composition of a system (i.e. the exchange of an hash algorithm considered no longer secure).

[3] Pimentel et al. [Pimentel08]: *“Authentication and secrecy* [confidentiality] *are the most common examples of protocol security requirements. These properties have no universal interpretation and are formalized according to the context.”*

truth, accuracy and/or correctness. The second aspect, **entity-authenticity** aims for the proof that an entity, e.g. a specific person or a specific node in a network, has been correctly identified as originator, sender, forwarder or receiver of information. By enforcing entity-authenticity it is ensured that an entity is the one it claims to be.

- **Non-repudiation** is building on the notions of integrity and authenticity and aims at proving to a third party a proof that a transaction or data transmission has happened in a protocol exactly as it is claimed by the participants of the protocol run. It often involves time-stamping alongside the integrity- and authenticity-focused security mechanisms.

Not covered in POWER are considerations on the security aspect of **availability**, i.e. the ability of using an information or resource as desired. Even though availability is an important aspect of reliability of systems, it can not be addressed by cryptographic means or by watermarking, neither by primitives nor by protocols.

Security requirements describe how a security aspect has to be covered in regard to an incident. Three different possibilities exist how this relation can be defined. Security requirements either: describe how a certain security aspect can be ensured for a system (**prevention**), or address the reporting if a security mechanism identifies a violation of a security aspect in an incident (**detection**), or describe how the system/resource/information can be recovered into a consistent state after the handling of an incident (**recovery**).

In general, digital watermarking algorithms address security requirements by means of detection mechanisms. For example, in case of a fragile watermark for integrity verification mechanisms, the violation of the integrity is not prevented but it can be detected afterwards.
With watermarking protocols and the framework introduced for POWER we move from detection into prevention strategies as well. This is done by transfer of a method for communication protocol simulation for security verification from research on cryptographic protocols to watermarking protocols. By this means, flaws in secure communication protocols can be identified and fixed in design, prior to the roll-out into products or systems.

The exemplary realization for the POWER framework described in section 2.2 is limited by our choice of CASPER as protocol description and verification language to the security aspects of confidentiality and entity-authenticity. In general this could be extended also to integrity, data-origin-authenticity and non-repudiation but there exists to our knowledge no language/tool that supports the modeling and verification of those security aspects.
Since our approach is a formal methods approach, we assume here that the underlying cryptographic and watermarking primitives are secure (see section 1.1).
In the complex application examples for the POWER framework presented in section 3.3, we show how our prevention-focused work on watermarking protocols interacts with detection-focused watermarking-based security mechanisms that cover the security aspects that can not be addressed by the current framework realization (e.g. integrity protection).

1.3. *Channel modeling for watermarking protocols*

We define for this report the following terminology: a watermark **cover** is transmitted between the communication partners in a watermark communication scenario via **logical channels**. These covers can be either files (e.g. digital image files), packets in data streams (e.g. TCP packets) or any other digital data object that allows the embedding of a watermark (see section 1.4). Into the cover the **watermark message** is embedded, which consists either of the **payload** or a combination of payload and overhead (e.g. headers or data structures that allow for a hierarchical payload access). The following figure 1 visualizes those four different channel types considered here.

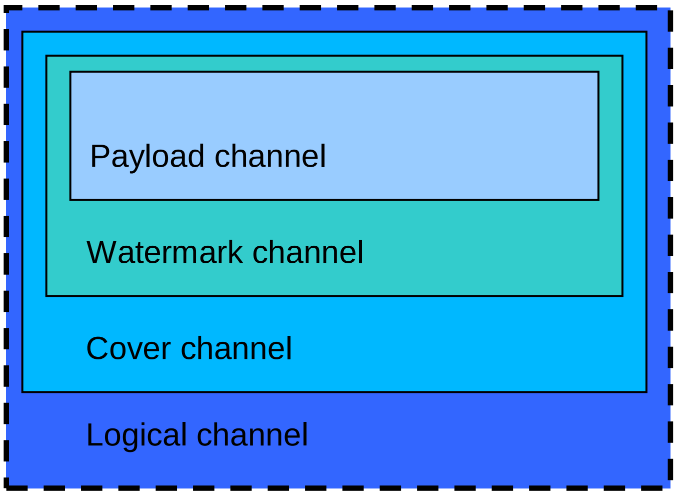


Figure 1: Layers in the channel modeling for watermarking protocols

Logical channels are established between two or more nodes in the underlying communication network. We consider each direction of a communication as a single logical channel (so we need two logical channels to model a bi-directional TCP-connection).
In most cases the communication partners can choose between different potential logical and cover channels, which are assumed to have an associated cost function. Therefore cover channels should be considered as exchangeable or combinable during operation. Nevertheless the actual choice of cover channel influences/limits the possible watermarking algorithms applicable for the construction of the watermarking channel.

In this report we consider the construction of a watermarking channel as a task for a communicating network of nodes or agents (e.g. „Establish a payload channel from Alice to Bob with a capacity of at least 5 kb/hour"). This is a static network, where all existing agents with their characteristics (e.g. access to infrastructure like PKI, known watermarking algorithms, etc.) are known and where the task imposes fixed requirements. Therefore the task can be re-formulated as the following (network graph related) question: *Does a path exist in the communication network from Alice to Bob which fulfills the task requirements?*
Depending on the answer to this graph theory problem different alternative solution strategies have to follow, either aiming at enabling the construction of a path or on finding the optimal existing path(s) or on optimizing existing paths. What shall be mentioned here is that all solution strategies require a specification/description of the network in machine-readable form, a description of the task in machine-readable form and a method for splitting the task into sub-tasks for processing. What is not

strictly required but would be an interesting feature are quality function(s) for nodes/edges in this graph theory problem.
In section 2.2.1.1 we show how this path search is realized within our work for POWER.

1.4. Watermarking data and/or object suitability

An important question, to be answered prior to addressing further conceptual work to be done in POWER, is the question: "What can be watermarked?" Or, to be more precise, the determination which data can be watermarked and which data cannot (e.g. "Can we watermark everything in a networked communication environment?").

There is no definitive answer to these questions, yet, but our observations on these matters within POWER can be summarized as follows:

- Some plain-text data objects can not be watermarked. This is mostly true for protocol data and files/ data streams with an integrity verification mechanism (like a checksum or integrated hash function). If the integrity verification mechanism cannot be updated after watermark embedding then those can obviously not be watermarked.
- For non-plain-text (i.e. encrypted data) different cases have to be distinguished:
  - If algorithms/protocols are used that employ homomorphic encryption then signal processing in the encrypted domain might be used for the watermarking of the considered data objects.
  - If algorithms/protocols are used that work in locally constricted modes (e.g. XOR or block-based ciphers in non-feedback modes like ECB) and have no integrity verification mechanism, watermarking could be applied (e.g. by code-book substitution) but the result of this embedding should be assumed to be visible because the manipulations are performed blindly on the ciphertext without any possibility for transparency preservation for the corresponding plaintext modification.
  - If algorithms/protocols are used that do not work in locally constricted modes (CBC, CFB, OFB, etc) and/or have an integrity verification mechanism (e.g. CBC-MAC) watermarking cannot be applied.
- For non-encrypted data and data without non-modifiable integrity verification checks (which should be everything besides the two points above) the basic assumption so far is that technically everything can be watermarked, again distinguishing between different cases (structured by watermarking algorithm characteristics):
  - Non-blind: everything can be watermarked (e.g. by using code-book substitution) – the question here is the transparency, but that can be optimized e.g. by choosing the code-book entry with the minimum required changes
  - Blind & Invertible: requires enough capacity (white Gaussian noise could not be watermarked in this case since the entropy is already optimal), transparency of the marked object remains a question for further considerations
  - Blind & Non-invertible: trade-of (mainly) between capacity, transparency and robustness

All these observations can be summarized in the fact that the initial question ("What can be watermarked?") has to be extended to: "What can be usefully watermarked?" The answer to this question has to be given cover and application scenario dependent and therefore requires context and task (=application scenario) modeling, as it is applied within the framework presented for POWER. An example of an extreme cover influence is optimally coded data, which by definition would provide an extremely low capacity (if any) and therefore render it unsuitable for most multi-bit watermarking application scenarios. Related investigations found in literature, especially for the question of required capacities in data hiding, are e.g. [Boehme09] and [Comesana07] which perform such observations for steganography.

As mentioned above, the application scenario directly specifies the target watermarking requirements (transparency, robustness, capacity, invertibility, etc). A different approach to address the problem might be the application of the verifier tuple form [Schott10]. By using this approach the requirements for data embedding could be specified for the different syntactic (physical-, Bit- and interpretatory level) and semantic (structural-, functional- and perception) levels of an information object.

1.5. Multiple watermarking for multi-access scenarios

According to Sheppard [Sheppard01] there are **three basic approaches to multiple watermarking**, which represent the basis for the multi-level access protocols desired for POWER: The first and most simple approach to this problem is **re-watermarking**, i.e. there is one watermark for each level and every watermark is simply embedded over all previously embedded watermarks into the same cover. This technique suffers from quite a lot of problems like partial overwriting or mutual elimination of the watermarks. The approach of **segmented watermarking** performs a segmentation of the cover's capacity so that each watermark (and in this case access level) has its own separate embedding area. The problems of re-watermarking therefore do not apply to this scheme, although a certain inflexibility is introduced when segmenting into static partitions. The third (and least likely realized) multiple watermarking approach, according to [Sheppard01] is **composite watermarking**. This technique aims for a composition of all watermarks which are to be embedded into a single watermark. Although this approach does not have to handle any segmentation or overwriting problems, the watermark is embedded here as a whole and therefore also needs to be extracted and re-embedded as a whole, which is not desirable for a multi-level access scenario.

2. Concept and design work for the POWER framework

In this chapter the main contribution of the POWER research project, the methodology and concept of our theoretical framework for watermarking protocol generation and (security) verification based on a protocol life-cycle model, is presented.
The applicability of the introduced generalized, theoretical framework is shown with an exemplarily realization / implementation of the framework.

The applied methodology for the framework for POWER is a formal methods approach based implementation of the design phase in a general protocol life-cycle. The realization concept used by us within the project has the following characteristics:

- It allows for machine-supported context- and task modeling for watermarking protocols,
- It introduces methods for automated solution determination within the context- and task models for communication networks by path search,
- It performs (semi-)automated protocol generation/deviation from the solutions identified,
- It performs a machine-based verification of the security of communication protocols (an approach transferred to the domain of digital watermarking based media security protocols from cryptography).

In the following sub-sections, first the concept for the theoretical framework is presented in section 2.1 and then an example realization, based on the CASPER language and an own context modeling approach, is introduced in detail in section 2.2.

2.1. Concept for a theoretical framework

This document focuses on the design phase in the life-cycle[4] of communication protocols. This perspective is rare in the watermarking domain because most publications in this field focus either on the realization phase, which mainly consists of algorithm development and testing, or the operation phase, which incorporates attacks against deployed watermarking schemes.
The following **exemplary life-cycle for communication protocols** is introduced here, to act as a basis for developing our concept, which is then further précised into the POWER framework for watermarking-based context modeling, protocol generation and -verification.

[4] Here the life-cycle of protocols is derived from the general software development life-cycle model. This concept, with its well known initial waterfall-model ([Royce1970], later succeeded by a spiral-model [Boehm88]), is a general description of software process models and the issues they address.

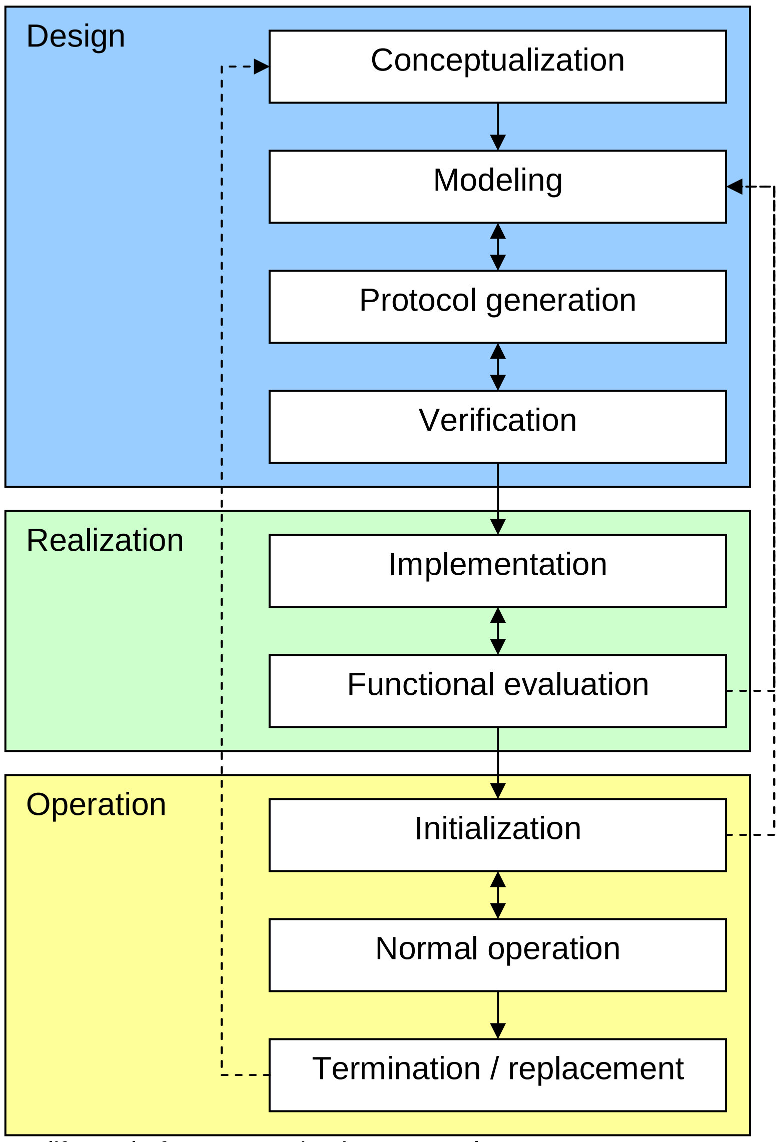


Figure 2: Exemplary software life-cycle for communication protocols

The exemplary protocol life-cycle for communication protocols, shown in figure 2, consists of three main phases: design, realization and operation.
In the **design phase** the application goal is specified (**conceptualization**), specific tasks are derived from the goals, and the communication setup as well as available primitives/ algorithms are modeled. Based on all these **modeling** operations a **protocol generation** is invoked and the result undergoes formal **verification**. We assume here that the modeling, protocol generation and verification directly influence each other, i.e. that problems or certain results in a later step can also result in the necessity to move back into a previous one.
In the **realization phase** the primitives and protocol components are implemented and undergo a functional evaluation before the result is rolled out as a product.
The last phase, the **operation phase**, consists of the initialization of the protocol in the application environment followed by the normal operation or intended usage of the protocol. At some point every protocol implementation will reach the end of its usefulness and will be either terminated or replaced by a successor protocol (which is assumed to undergo its design and realization phases during the normal operation of its predecessor).

The work within POWER is mainly concerned with the design phase in this protocol life-cycle, but this design phase has to provide **hookups to later phases** (e.g. with a counterpart in the realization phase, if the functional evaluation fails and the developers have to move back to the drawing board, or the operation phase if the initialization fails).

The design phase of this exemplary life-cycle for communication protocols is now précised into the **theoretical framework context modeling, protocol generation and -verification for POWER**. The concept used to do so is loosely based on an observation made in [Pimentel08]: "*In the application of formal methods to software verification, both the system and its specification are first expressed as formulas of some (but not necessarily the same) logic. Then, mathematical reasoning is used to prove that the system and the specification are related somehow, for example by logical implication. A state-of-the-art verification tool is capable of yielding either of two outputs: i) OK, indicating that the system is error-free, at least with respect to the coverage analysis of the corresponding tool; and ii) a counterexample, indicating how a system execution violates the specification.*
In the context of security protocol verification, the system is the security protocol under analysis, the specification a protocol security requirement and the counterexample actually is an attack. Authentication and secrecy are the most common examples of protocol security requirements. These properties have no universal interpretation and are formalized according to the context."

The first major point in this statement (*"both the system and its specification are first expressed as formulas of some (but not necessarily the same) logic"*) is an equivalent to our conceptualization (definition of a systems specification) and the context and task modeling (summarized here as step 1 in the theoretical framework, called **context modeling**).
Prior to the next point in the statement from Pimentel et al. (which already focuses on verification), we insert here a **protocol generation** block since the protocols for POWER will not be expressed as formulas that can be created and verified at the same point of time. We assume here for POWER that the best way of describing the task (the *"systems specification"* in the statement from Pimentel et al.) as well as the network and algorithms (i.e. the system) is graph-based. In such an approach we can easily describe the network by taking the communicating entities as nodes and the defined connections as links between those nodes. The knowledge about algorithms is described as a characteristic of the nodes. The **network task**[5] for the application scenario is a superposition of all individual **tasks**[6] or desired information flows within the network. Each of those tasks is defined by giving a start and an end node (sender and receiver in the application scenario) and a set of requirements that have to be fulfilled (transmission capacities, complexity constraints, etc.).
If the system and its specification are described in this way, then a **path search** (step 2 in our theoretical framework) from graph theory can be used to determine a set[7] of **solutions** (i.e. paths in the network graph that fulfill the requirements) for each task.

[5] Network task: The superposition of all tasks (i.e. the expression of all goals of the application scenario) within one network is called network task.
[6] Task: One complete communication pipeline from an information source to its destination. By the path search a task is translated into a set of alternative solutions.
[7] This set can also be empty if absolutely no solution can be found.

As a consequence of this modeling approach, there exists for each application scenario exactly one network task, but the path search can generate a set of alternative **network solutions** (as superposition of all solutions) which all fulfill this network task.
From all alternative network solutions, which fulfill the goals of the application scenario, one is chosen in the step 3 of our theoretical framework (the **path selection**) to act as the basis for a translation into a protocol (also called **protocol generation -** step 4 in our theoretical framework). This choice should be made based on a cost or quality scheme for the solutions.

This whole process of protocol generation (the generation of a sequence of interactions or transactions between entities in a specific communications network with the aim of exchanging data; see our protocol definition in section 1.1) is then followed by the protocol **verification**. State-space exploration by model checking (as it is implemented by FDR) is the only applicable approach found in literature to address this problem. There exist some publications on optimizations for this basic approach, mainly by limiting the number of choices explored in the search space, but these hardly found their way into applications[8].
The model checking process consists of different steps: first, the **model generation** (step 5 in our theoretical framework), then the **model verification**[9] by the model checker (step 6) and last, as step 7 in the theoretical framework, a **model selection** (in case multiple models were generated for the verification of different evaluation goals or to compare different alternative protocol solutions).
The output of the verification operation should be what Pimentel et al. [Pimentel08] specify as: "*OK, indicating that the system is error-free, at least with respect to the coverage analysis of the corresponding tool*" […, or] *"a counterexample, indicating how a system execution violates the specification"* for the selected protocol.

The relations between all the mentioned steps in the described theoretical framework for context modeling, protocol generation and -verification for POWER are illustrated in figure 3.

[8] One exception to be mentioned here might be the four model checkers integrated in AVISPA which are focused on specific tasks and granularities and therefore perform such an limitation of the search space.
[9] In case the verification identifies flaws in the generated protocol then the whole procedure moves back into step 3 (path selection) or step 4 (path selection) to try to eliminate that flaw.

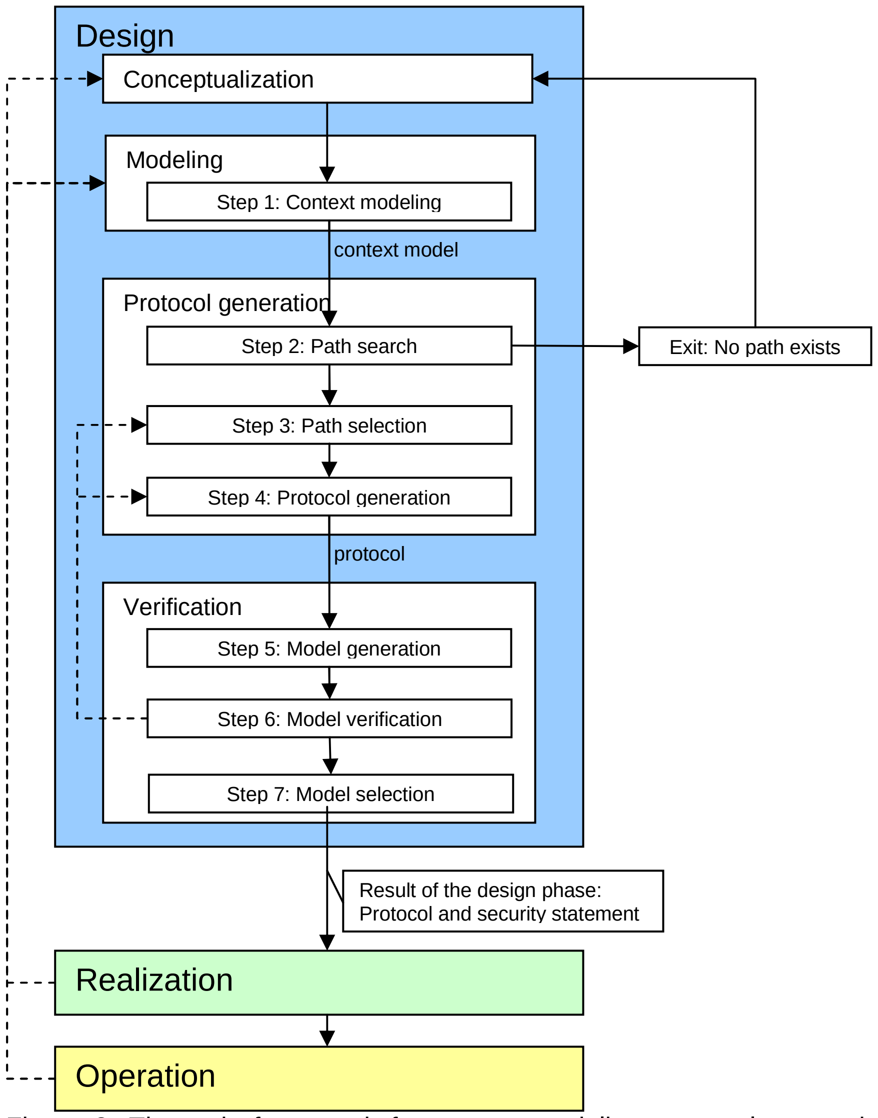


Figure 3: Theoretic framework for context modeling, protocol generation and –verification (derived from the protocol life-cycle shown in figure 2)

2.2. A CASPER-based practical realization of the theoretical framework

Based on the theoretical framework introduced in section 2.1 for context modeling, protocol generation and -verification, here a prototypical realization implemented for POWER is described. This prototypical realization is then used for the application focused investigations in chapter 3.

Early on in our work on POWER we decided to base our practical realization for the project upon the CASPER language ([Lowe98], see section 1.1). CASPER is a simplified version of CSP and in contrast to this more complex language still rather good readable by humans.
Nevertheless, the choice of CASPER imposes some rather severe constraints upon our realization:

- CASPER constructs agent-based communication security models. This implies that the number of communicating entities is limited to a well defined set which allows for no dynamic changes of the communication network.
- All communication protocols in CASPER are strictly sequential.

- The CASPER notation offers a limited number of constructs, which allow only a certain amount of mechanisms to be modeled. Digital watermarking algorithms need to be simulated by the use of similar constructs since they were not directly foreseen by the authors of CASPER.
- The FDR model checker, used in the security evaluation of CASPER protocols, further limits the number of communicating entities in the protocol.

Even though our CASPER (and FDR) based approach faces the aforementioned limitations, CASPER is chosen here for the protocol modeling because the alternatives (e.g. CSP or AVISPA) face similar restrictions and are in contrast to CASPER less human readable.

2.2.1. The processing chain - modeling and verification steps in the protocol framework

Figure 4 compares our prototypical realization in direct contrast to the theoretical framework.

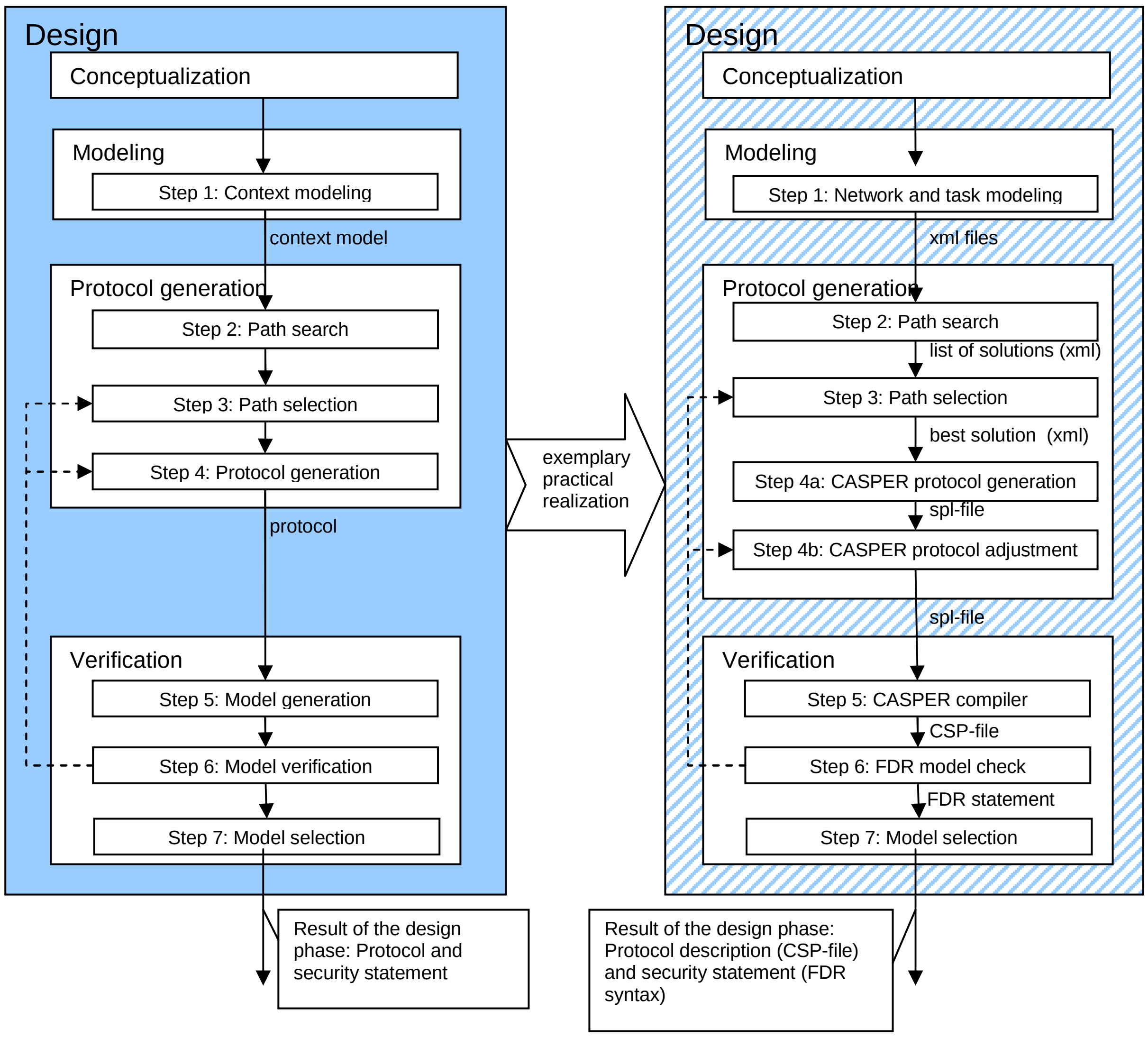


Figure 4: The practical realization of the theoretic framework used in POWER[10]

Besides the detailed definition of the steps in the theoretic framework, our practical realization for POWER has to split the fourth step (the protocol generation) into two parts (see section 2.2.1.2). The following sections describe in detail the steps in our practical realization.

2.2.1.1. Network and task modeling, path search and -selection (steps 1, 2 and 3 in the framework)

The first step in the practical realization of the framework is the **context modeling** for the application scenario[11]. This is done in our exemplary realization by graph based descriptions of the network task and the network in XML structures[12]. The knowledge about existing algorithms / primitives is specified individually for each node (communicating entity in the graph). Furthermore the infrastructure requirements for the used protocol components (e.g. access to PKI for asymmetric watermarking schemes) need to be included into the modeling for the purpose of automatic evaluation.

An example for the XML network **task description** used by us looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<network_task>
      <task>
            <src>CR</src>
            <dst>A</dst>
            <required>
                  <pki>n</pki>
                  <ts>n</ts>
                  <hierarchy>1</hierarchy>
                  <cap>32</cap>
            </required>
      </task>
      <task>
            …
      </task>
</network_task>
```

To improve the path search described in section 2.2 above, each task is split into sequences with identical requirements (called **sub-tasks**) which can be handled as one unit in the search.
The main reason for this division into subtasks is that this allows us to change on processing agents between cover channels and watermarking algorithms[13]. After the

[10] Note: The exit after unsuccessful path-search and hook-ups for input from realization and operation phases (see figure 3) remain. They are just skipped in figure 4 for the sake of readability of the figure.
[11] We assume here that the goals fort he application scenario are predefined, so that the conceptualization is already finished prior to the application of our framework.
[12] This XML-based context modeling allows for an easy adaptation to a wide range of different application scenarios, it is easily processed by computers and at the same point of time it is still human readable.
[13] A brief example can illustrate when to split into subtasks and when not: In a simple watermark signature chain where each station either simply forwards the media file or adds its own ID to the watermark already embedded into a media file, a new subtask is generated each time the channel characteristics change (e.g. the capacity required increases by the adding of an additional ID). In this

path search the **(sub-)solution** for such a sub-task has the form: A path to transmit data from a starting node to a end node, over a connection of nodes with identical or better characteristics than start and end (i.e. on this sub-task and the corresponding sub-solution the watermark algorithm, its characteristics and the used cover channel do not change). The (sub-)solutions for one task are then combined into the solutions.
The shown sub-task with source CR and destination A requires no access to any of the defined infrastructure components, has no hierarchy constraints (the parameter is set to “1” for one embedding level which is identical to a non-hierarchical embedding) but requires a capacity of 32 (Bit/s).

Our corresponding **XML network description** looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
      <nodes>
            <node>
                  <id>CR</id>
                  <cover>
                        <cc>
                              <type>data stream</type>
                              <channel-capacity>1000</channel-capacity>
                              <ctype>bps</ctype>
                        </cc>
                  </cover>
                  <dwm>Alg1</dwm>
                  <node-capacity>5000</node-capacity>
                  …
            </node>
            <node>
                  …
            </node>
      </nodes>

      <lc>
            <connection>
                  <src>CR</src>
                  <dst>A</dst>
            </connection>
            <connection>
                  …
            </connection>
      </lc>
</network>
```

Each of the nodes/agents in this structure is specified with an ID (e.g. `<id>CR</id>`), the cover channels it has access to, the watermarking algorithms (and the corresponding embedding keys) known to the node (e.g. `<dwm>Alg1</dwm>`) and the node communication throughput in Bit/s (`<node-capacity>`). For the accessible cover channels (`<cc>`) the type (in this example a `data stream`), the cover channel capacity and the unit it is measured in (here Bit/s). After these required blocks the infrastructure component accessibility would be specified for each node.

example no new subtask would be generated if an agent just forwards the marked cover without any modifications.

Additionally to the nodes in this XML-structure the connections (logical channels) between the existing nodes are specified as unidirectional source/ destination pairs.

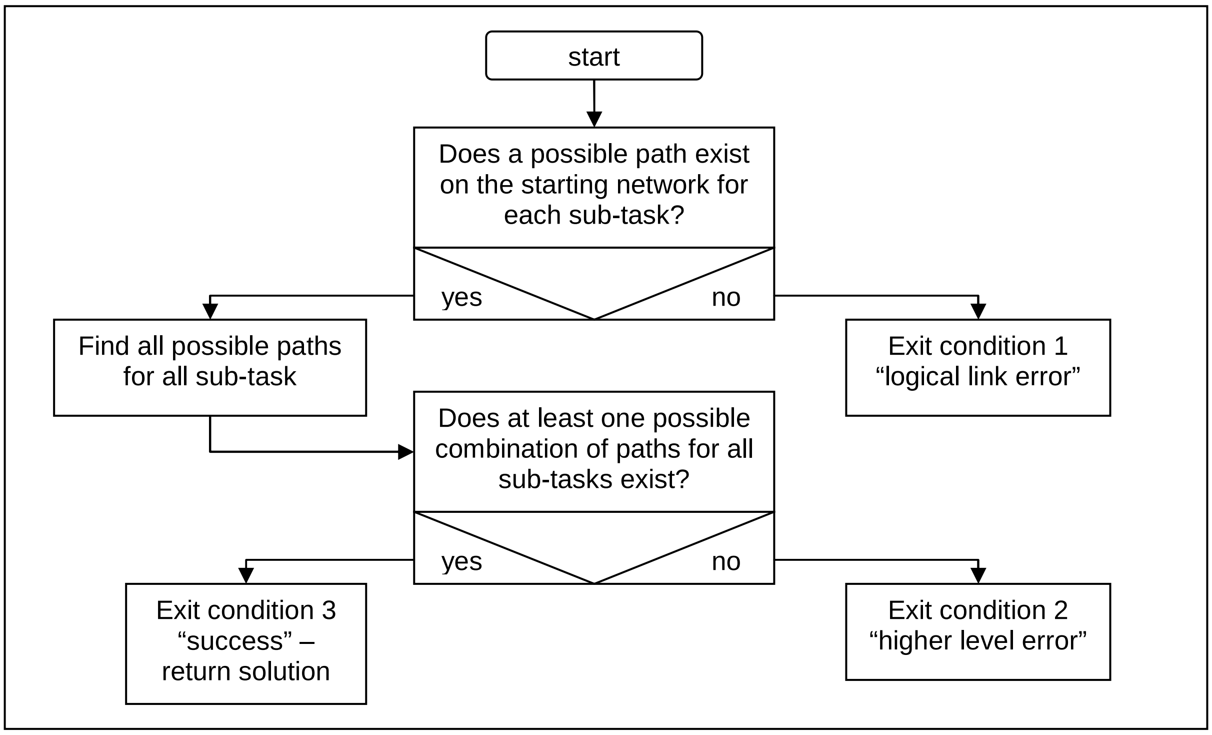
A different XML structure describes the **characteristics of the watermark algorithms available**. The following form is used:

```
<?xml version="1.0" encoding="UTF-8"?>
<dwms>
      <dwm>
            <id>Alg1</id>
            <cover>data stream</cover>
            <dwm-capacity>12.5</dwm-capacity>
            <robustness>low</robustness>
            <transparency>high</transparency>
            <key>symmetric</key>
            <hierarchy>1</hierarchy>
      </dwm>
      <dwm>
            …
      </dwm>
</dwms>
```

Each watermarking algorithm (<dwm>) is specified by an ID, the cover type it can embed into, the embedding capacity offered (in percent of the cover size), robustness and transparency characteristics (for this example simplified to “low”, “medium” and “high” respectively), as well as a key scenario and hierarchy requirements (set to “1” = no hierarchy; embedding on one level only in this example).

The second step in our practical realization of the framework is a classical **path search** that determines all paths in the network that fulfill the requirements specified in all tasks (e.g. capacity, transparency, etc. requirements or the connectivity to required infrastructure like a PKI).
The path-finding process implemented here for POWER is using a simple Depth-First-Search algorithm ([Cormen01]) applied to the complete network with all its logical connections to determine all physical paths from the source to the destination node of the network task. Subsequently the nodes directly involved in each subtask are checked for a common watermarking scheme, i.e. they are checked for whether they have access to at least one common embedding and extracting algorithm plus the corresponding keys. When a common watermarking scheme is found the search for an adequate cover channel can be initiated. So every node on every found path is checked for available cover channels with the cover type defined by the chosen watermarking scheme. A path that satisfies the subtasks requirements (taken from the XML task description) is considered for usage when all nodes on a specific path have knowledge of a matching cover channel with sufficient capacity. Finally it is checked whether the connections between the subtasks and their composition also fulfil the overall task requirements (e.g. the overall channel capacity of a node is not exceeded).
The procedure stops at this point with an error message if the path search can not determine at least one suitable path.

The initialization of the task processing in the path search starts with the **starting network**, which is the modeled network without any influence of preceding processing. A **possible path** for a sub-task is therefore a path that fulfills the given sub-task if this would be the only sub-task in the network.



start
Does a possible path exist on the starting network for each sub-task?
yes
no
Find all possible paths for all sub-task
Exit condition 1 "logical link error"
Does at least one possible combination of paths for all sub-tasks exist?
yes
no
Exit condition 3 "success" – return solution
Exit condition 2 "higher level error"

Figure 5: Task processing

Understanding the path search requires familiarity with our approach to channel modeling for watermarking channels as it is described in section 1.3.
The initial processing of a task (see figure 5) involves the following operations: First, check if it is possible to find at least one path for any given sub-task using the starting network. If the answer is "No" (Exit condition 1): give information about which sub-task failed at which stage. The reasons could be: No logical channel, no cover channel, no access to required infrastructure components (like a PKI), no common watermarking algorithms, etc.
If the answer of this first stage is "Yes": identify all cover-channel-paths for which DWM channels are possible to the next stage.
Second, we check if there is at least one combination of the given cover-channels for each sub-task that does not exceed any capacity or complexity constraint. If the answer to this check is "No" (Exit condition 2): we give information about where the problem occurs (reasons might mainly be capacity or complexity). If the answer is "Yes" (Exit condition 3): we have one possible solution which could in post processing be optimized.

Table 1 shows the sequence which is performed for the path search, i.e. in the comparison between the network/node characteristics. Stages 1 to 3 are performed for each sub-task, stage 4 is the final check performed after all sub-tasks have been evaluated.

Level	**Stage**	**Operations**
Logical connection	1.a)	Check for paths from source `src` to destination `dst`
	1.b)	Check infrastructure requirements `src`
	1.c)	Check infrastructure requirements `dst`
Cover-channels	2.a)	Check source and destination node for common cover channels: `<type>…</type>`
Watermark-channels	3.a)	Check source and destination node for common watermarking algorithms:

		`<dwm>…</dwm>`
	3.b)	Check the watermarking channel capacity: `<channel-capacity>*<dwm-capacity>` must be larger or equal than `<cap>`
	3.c)	Check further watermarking characteristics as required by the scenario, e.g.: • hierarchical access, • transparency, • robustness, • etc.
The result so far is a list of possible paths solving the problems for the defined sub-tasks		
Final checks	4.a)	Check for all sub-tasks the sums of the required capacities per cover channel
	4.b)	Check for all sub-tasks the sums of the required capacities per node
	4.c)	Check for all sub-tasks the sums of secondary restrains if those exist (e.g. complexity per node)
If the result is a list of paths which solve the task, and if weights are assigned to the individual paths then a ranking has to be performed		

Table 1: Path search – operations sequence

The output of the path search is returned in form of a list of (network) **solutions** in XML. The following XML structure combines the relevant information from network-, task- and algorithm descriptions:

```
<?xml version="1.0" encoding="UTF-8" ?>
<solutions>
      <alternative>
            <task>
                  <id>TaskA</id>
                  <subtask>
                        <src>NodeA</src>
                        <dst>NodeB</dst>
                        <meta>
                              <cover>
                                    <id>CoverA</id>
                                    <type>image</type>
                              </cover>
                              <message>
                                    <id>message</id>
                                    <level>1</level>
                                    <content>
                                          <data>DataA</data>
                                    </content>
                              </message>
                        </meta>
                        <dwm>
                              <id>dwm1</id>
                              <cover>image</cover>
                              <robustness>low</robustness>
                              <hierarchy>1</hierarchy>
                              <key>symmetric</key>
                        </dwm>
                        <required>
                              <pki>n</pki>
                              <ts>n</ts>
                              <hierarchy>1</hierarchy>
                        </required>
                  </subtask>
                  <subtask>
```

```
                        …
                  </subtask>
            </task>
      </alternative>
      <alternative>
            …
      </alternative>
  </solutions>
```

If there is no ***<alternative>*** block for a computed ***<solutions>*** there is no path in the network fulfilling the given task requirements and therefore no watermarking protocol can be developed. In any other case a separate CASPER protocol representation can automatically be designed for each alternative and different security aspects can be verified.
Each alternative consists of several ***<task>*** and ***<subtask>*** blocks representing the tasks and subtasks. For each subtask a message ***<message>*** is transferred from a node ***<src>*** to a node ***<dst>*** using a cover ***<cover>*** and the watermarking algorithm ***<dwm>*** under the requirements ***<required>***, being routed over the nodes ***<fwd>*** (which do noting else but passing the marked cover on and in no way change it).
At least one ***<task>*** needs to be present for each alternative.

The third step (the **path selection**) is then used to select one (network) solution for the given network task. This operation, which we did not implement in detail for demonstrators for POWER, should be quality- or cost-function based and would require the definition of those evaluation functions.

2.2.1.2. CASPER protocol generation and -adjustment (steps 4a and 4b)

The fourth step receives the selected solution as a XML structure and translates it into CASPER notation (for an introduction to CASPER focusing on the concepts and constructs required for POWER see Appendix A). Here, first an **automatic translation** is used to generate everything that can be done automatically. For this task an own compiler demonstrator is written by us (see section 3.1.2) to perform a rule based compilation of the XML context model into CASPER. The complete set of translation rules used in this process is described in detail in Appendix B of this document.

Then certain aspects have to be modeled/**adjusted manually** (e.g. the attacker knowledge, which could be initialized with defaults like Kerckhoffs-compliance or be modeled specific to the application scenario).
In this manual adaptation (step 4b in our framework realization example) a normal text editor is used by us to modify mainly the ***#Specification*** and ***#Intruder Information*** blocs of the CASPER descriptions.

2.2.1.3. CASPER compilation, FDR model checking and model selection (steps 5, 6, and 7)

After the CASPER[14] protocol description was generated and adjusted, it undergoes a syntactic evaluation while being compiled into CSP (the fifth step of our framework realization example), followed by a (semi-)automated model-checking based security evaluation using the FDR model checker (step 6). FDR returns as a result a statement on the security of the protocol. If the model checker finds no possible attacks against the security aspects required in the application scenario then the procedure finishes at this point with a CASPER protocol and a positive security statement. If the model checker finds possible attacks then either the protocol has to be adjusted (back to step 4) or the selected path has to be discarded and the next possible solution has to be evaluated (back to step 3). If necessary this has to be repeated until all identified solutions have been checked.
In case different models (e.g. for different solutions) are verified, in a final step the one to be implemented in the realization phase of the protocol life-cycle has to be selected.

2.2.2. Basic functions for the POWER framework

To ease alternative implementations to our framework realization example, in this section the basic functions for the framework, which are described in detail in section 2.2.1) are summarized in a compact version. The basic functions (or methods), for protocol development in the design phase of the protocol life-cycle, with their corresponding input variables and outputs are:

- Step 1) Network and task modeling (context modeling):
    - *model_the_network()*: returns a description of the underlying communication network with infrastructure connections in XML (output: network description *nd* as XML-file)
    - *model_the_watermarking_algorithms():* returns a XML description of watermarking algorithms (output: algorithm descriptions *ad* as XML-file)
    - *model_the_network_task():* returns a XML description of the network task (output: network task *nt* as XML-file)
- Step 2) Path search:
    - *derive_solutions(nd, nt, ad):* performs a path search on the graph described in the network model and identifies with the set of network solutions all alternatives for solving the given network task (output: alternatives *alts* as XML-file)
- Step 3) Path selection:
    - *select_solution(alts):* cost / quality based selection of solutions to find the optimal network solution (output: network solution *nets* as XML-file)
    - if necessary *network_adaptation_steps(nd, nt, ad, weights):* if no solution matches the requirements and if adaptation steps are allowed, then this method gives recommendations on how to modify the network to implement required changes to enable an solution at minimal cost (output: solution *nets* as XML-file)
- Step 4) CASPER Protocol generation and adaptation:

[14] Even though CASPER (and FDR) based approaches face strong practical limitations on the computational complexity required to verify larger systems/protocols, CASPER is chosen here for the protocol modelling because the alternatives for implementing the steps 4 to 6 of our framework (e.g. CSP or AVISPA) face similar restrictions and are in contrast to CASPER less human.

    - *protocol_derivation(nets):* derive a protocol from the selected network solution using a translation template into CASPER (output: *nets_spl* spl-file)
    - *protocol_adjustment(nets_spl):* (manually) adjust the protocol to implement application scenario specific modifications (output: spl-file *nets_spl*)
- Step 5) CASPER compilation:
    - *protocol_verification(nets_spl):* perform a syntactical evaluation of the protocol by using the CASPER compiler (output: CSP-file, *nets_csp*)
- Step 6) FDR model checking:
    - *security_evaluation(nets_csp)*: evaluation on the protocol by FDR (output: FDR security statement *secs*)
- Step 7) Protocol/Model selection:
    - *protocol_selection(secs):* in case different alternatives have been evaluated (output: CSP-file *nets_csp* of the chosen protocol and the corresponding FDR security statement *secs*)

The other two phases of the protocol life-cycle (protocol realization and operation) are not covered by the research work for POWER (and are therefore out of the scope of this document), but we have to acknowledge again that they can have influence on the processes in the design phase. There exist feedback functions from these two protocol life-cycle phases into the design phase covered here (e.g. when the implementation or initialization show flaws in the design, see section 2.1).

3. Application and Evaluation Work for POWER

This chapter is dedicated to the practical work done on application and evaluation of the introduced framework within POWER. It contains in section 3.1 descriptions of the demonstration tools as well as in sections 3.2 and 3.3 application examples that show how the framework (in form of its exemplary realization) can be applied in practice and which problems arise during this application.

3.1. Processing chain demonstrators

For the realization of the steps 1, 2, 3 and 4a of our framework, there exist right now no suitable tools which might be applied within POWER. Therefore, we decided to write our own demonstrators that can be used instead of text editors to solve these tasks faster and less error-prone.

Important notice: our software demonstrators are no stable, tested software products! They are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

For step 4b of our framework we use text editors, for the realization of steps 5 to 7 the established tools CASPER and FDR (see section 1.1) are used.

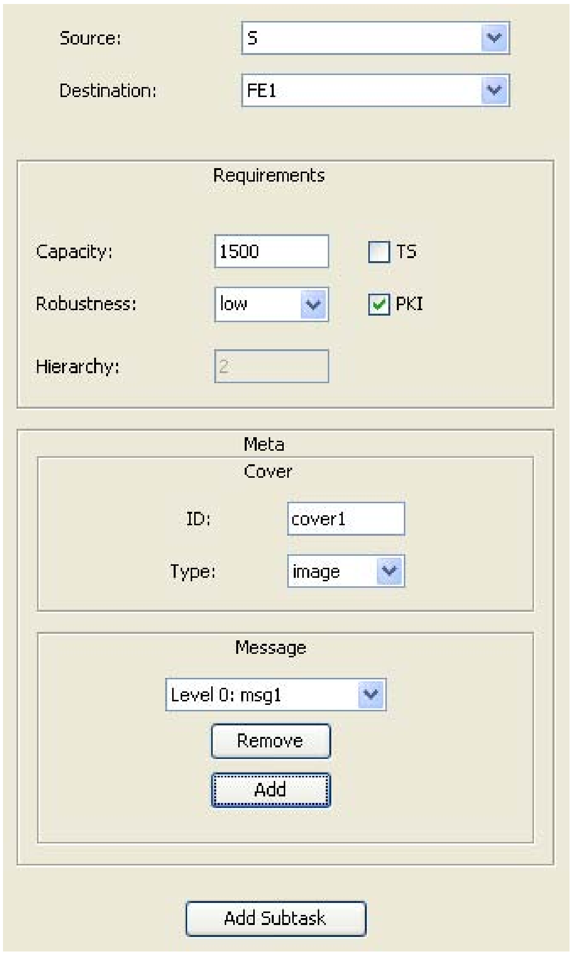
3.1.1. Demonstrator for context modeling and solution determination

A Java-based demonstrator is developed to address step 1, 2 and 3 in the framework for watermark protocol generation and verification introduced in section 2.2.1.1. The main goals of the demonstrator are to act as an authoring tool to make the process of context modeling less error-prone and to allow for a visualization of the solution alternatives after successful path search.

Currently the demonstrator is still in a pre-alpha state, but already the following functionality is provided:

- GUI-supported generation of network descriptions
- Generation of watermarking algorithm descriptions
- GUI-supported generation of network tasks
- Saving and loading projects as XML-files
- Path search as described in section 2.2.1.1
- Visualization of the alternative path search results
- Generation of suggestions for required changes in the specification when no suitable path could be found

The figures 6 and 7 below show screenshots of the demonstrator GUI.


Source: S

Destination: FE1

Requirements

Capacity: 1500 ☐ TS

Robustness: low ☑ PKI

Hierarchy: 2

Meta

Cover

ID: cover1

Type: image

Message

Level 0: msg1

Figure 6: GUI showing the dialog for task descriptions

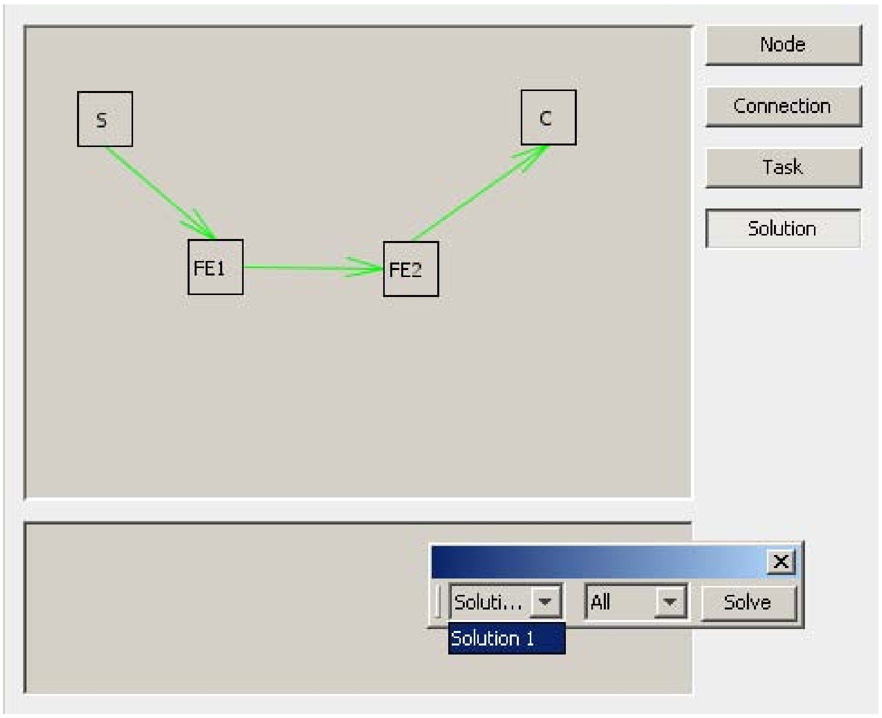


Figure 7: GUI showing the selected solution after successful path search

The most complex of the menus in the tools is the task description, because here we describe a hierarchical construct (the task, consisting of subtasks – see section 2.2.1.1, all with specified requirements and message format information). In our

demonstrator software we try to solve this problem for the defined application examples, nevertheless our solution is still far from being optimal.
The task description consists of three parts, which form together a hierarchical construct of tasks with their corresponding subtasks and the messages which are to be embedded. First, there is a task ID to be specified. After the set of subtasks for this specific task is created by adding one subtask after another, the second part of the task description, the subtask description includes all vital information for the path search, i.e. source, destination and all specified requirements. The third part is the specification of all information which is needed for the later translation into CASPER. Those include the cover properties as well as the structure and content (both editable and visualized in form of a tree) of every message to be embedded on each access level. Detailed examples for the usages of this demonstrator are given in sections 3.2.1.1, 3.3.1.1 and 3.3.2.1.
The source code of this authoring tool demonstrator as an Eclipse[15] project as wall as the resulting JAR-file are found on the CD accompanying this report.

3.1.2. Translator for protocol generation

The translation demonstrator generating CASPER csp-files from the XML output of the authoring / context modeling demonstrator is also implemented in JAVA. In contrast to authoring demonstrator it does not require a GUI, therefore it is implemented as a command-line tool. Essentially, this tool is a compiler, which performs a form of chart parsing[16] of the well structured XML descriptions into the less-well structured csp-file, with its rigid 8-block structure; the translation rules described in Appendix B are applied.
It expects two parameters: the solution-file from the context modeling and a name for the output file of the translation.
An example call of this tool looks as follows:

```
$ java –jar Translator.jar solutions.xml outfile.spl
```

The source code of this demonstrator as an Eclipse project and the executable JAR-file are found on the CD accompanying this report.

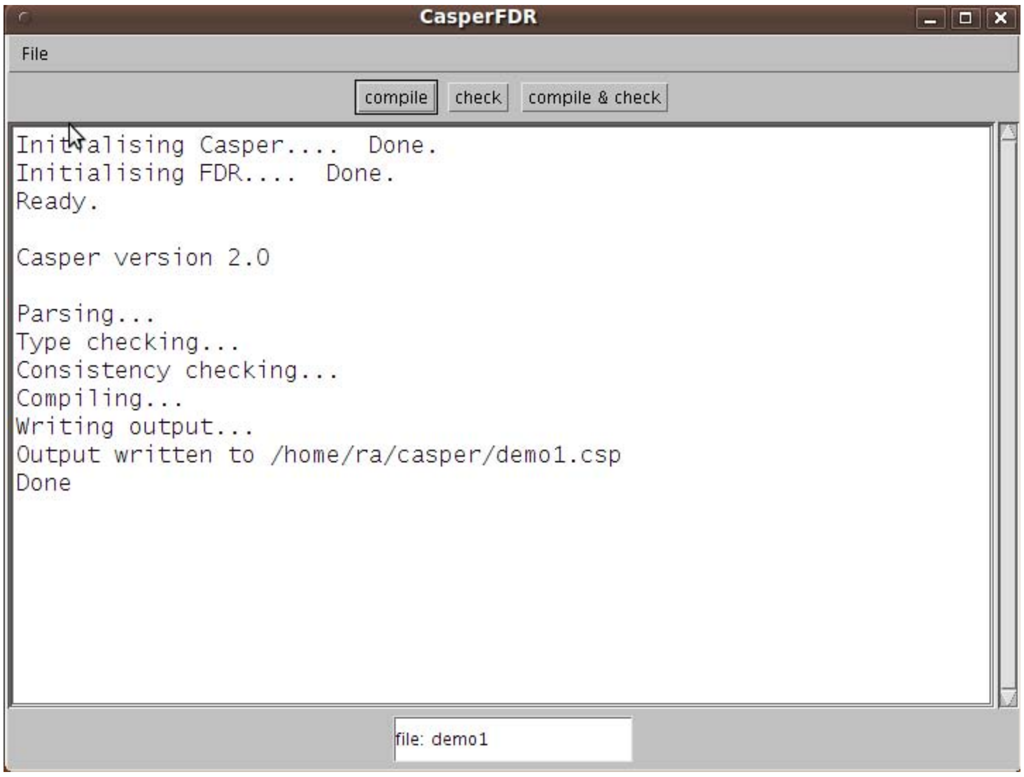
3.1.3. Protocol adaptation and verification (CASPER/CSP/FDR)

For our prototypical framework realization within POWER we used for step 5 and 6 unmodified versions of CASPER and FDR. These tools are available at:

- CASPER (version 2.0) installation repository downloadable from: http://www.comlab.ox.ac.uk/gavin.lowe/Security/Casper/casper-2.0-release.tar.gz
documentation:http://www.comlab.ox.ac.uk/gavin.lowe/Security/Casper/manual.pdf
- FDR (version 2.91): http://web.comlab.ox.ac.uk/projects/concurrency-tools/
With documentation available at:
http://web.comlab.ox.ac.uk/projects/concurrency-tools/download/fdr2manual-2.91.pdf

The easiest way to use these two tools is by running the GUI-based casperFDR interface supplied with the CASPER package. Figure 8 shows the normal operation of this tool with an CASPER input-file being successfully compiled into CSP.

[15] Eclipse Java IDE: http://www.eclipse.org/
[16] Using the dynamic programming approach - partial hypothesized results are stored in a structure called a chart and can be re-used. This eliminates backtracking and prevents a combinatorial explosion.

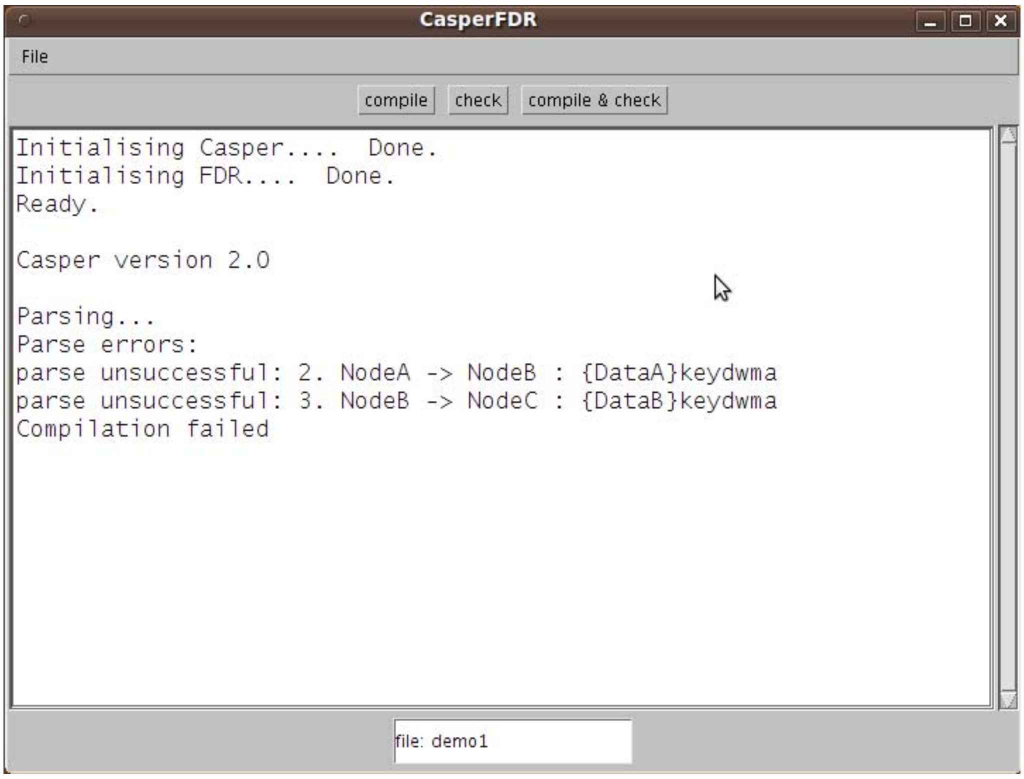


```
Initialising Casper....  Done.
Initialising FDR....  Done.
Ready.

Casper version 2.0

Parsing...
Type checking...
Consistency checking...
Compiling...
Writing output...
Output written to /home/ra/casper/demo1.csp
Done
```

file: demo1

Figure 8: The CasperFDR GUI has compiled an input-file successful

Figure 9 shows an example where the compiler detects an error and fails in the compilation form CASPER to CSP.

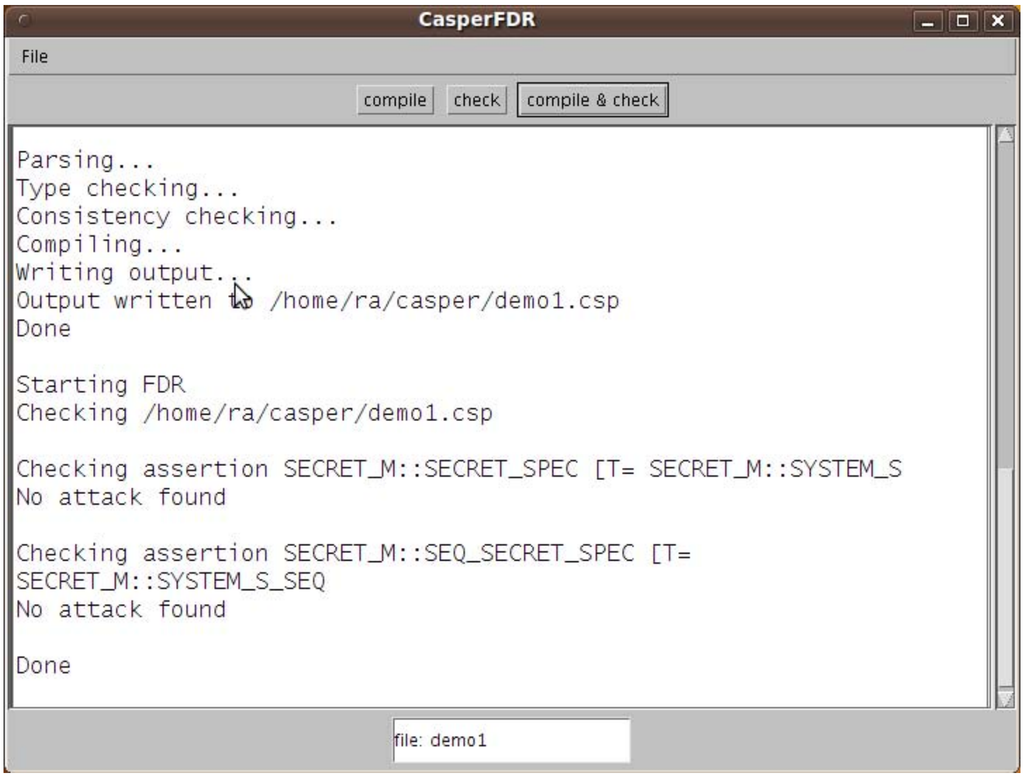


```
Initialising Casper....  Done.
Initialising FDR....  Done.
Ready.

Casper version 2.0

Parsing...
Parse errors:
parse unsuccessful: 2. NodeA -> NodeB : {DataA}keydwma
parse unsuccessful: 3. NodeB -> NodeC : {DataB}keydwma
Compilation failed
```

file: demo1

Figure 9: CasperFDR detects an error while compiling an input-file

Figure 10 shows an example where an input file is first successfully compiled from CASPER into CSP and then successfully verified by FDR.



```
Parsing...
Type checking...
Consistency checking...
Compiling...
Writing output...
Output written to /home/ra/casper/demo1.csp
Done

Starting FDR
Checking /home/ra/casper/demo1.csp

Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T=
SECRET_M::SYSTEM_S_SEQ
No attack found

Done
```

Figure 10: CasperFDR checks assertions (specifications) for an input-file and finds no attack

3.2. Illustration of the application of the framework using a basic scenario

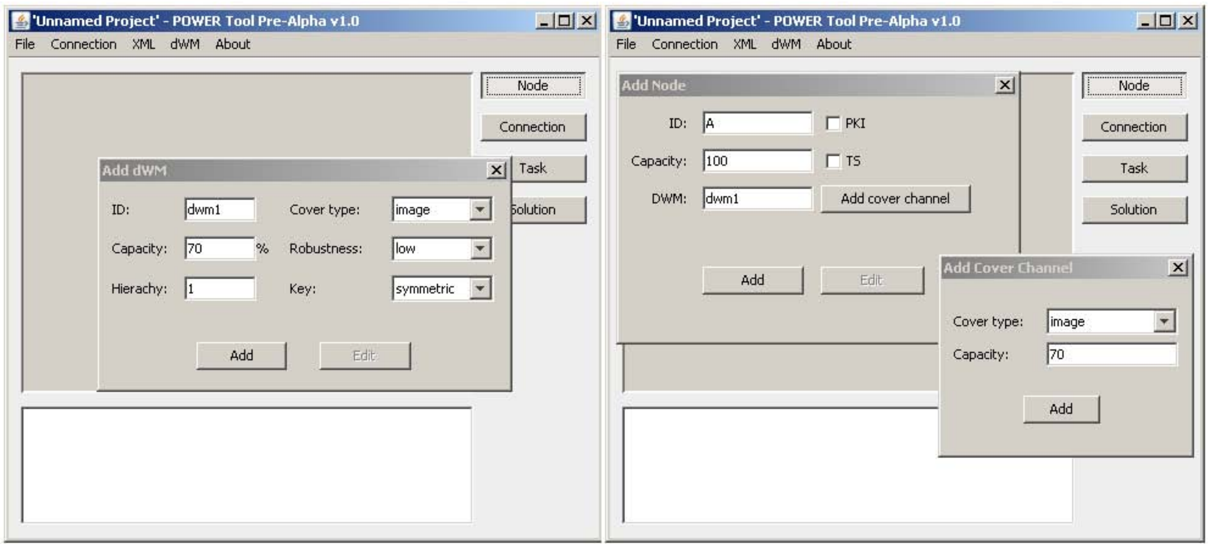
Prior to the application of our practical framework realization to the complex application scenario in section 3.3, here a very simple or basic scenario is processed with detailed descriptions of all processing steps.
This simple example mimics the usage of a watermarking algorithm by two communicating users (e.g. in an integrity verification watermarking scenario).

3.2.1.1. Network and Task description (step 1)

In step 1 of the introduced realization framework, we perform the network and task description for this application scenario (see section 2.2.1.1). This is done by using the authoring demonstrator tool to describe the network (see figures 11 and 12), specifying the characteristics and knowledge of the nodes and describing the task (figure 13).

- **Network description:**

The GUI-based authoring tool demonstrator allows for a very simple definition of the network. Alternatively these descriptions could be also generated in XML using any text editor (which would be slower and more error-prone). Figures 11 and 12 show the definition of nodes (which require a preceding definition of a watermarking algorithm, which is then assigned as required knowledge to the nodes) and the connections between those nodes.



Add dWM

ID: dwm1
Cover type: image
Capacity: 70 %
Robustness: low
Hierachy: 1
Key: symmetric

Add Node

ID: A
PKI [ ]
Capacity: 100
TS [ ]
DWM: dwm1

Add Cover Channel

Cover type: image
Capacity: 70

Figure 11: Algorithm definition and node specification for the basic application scenario in the POWER authoring demonstrator tool

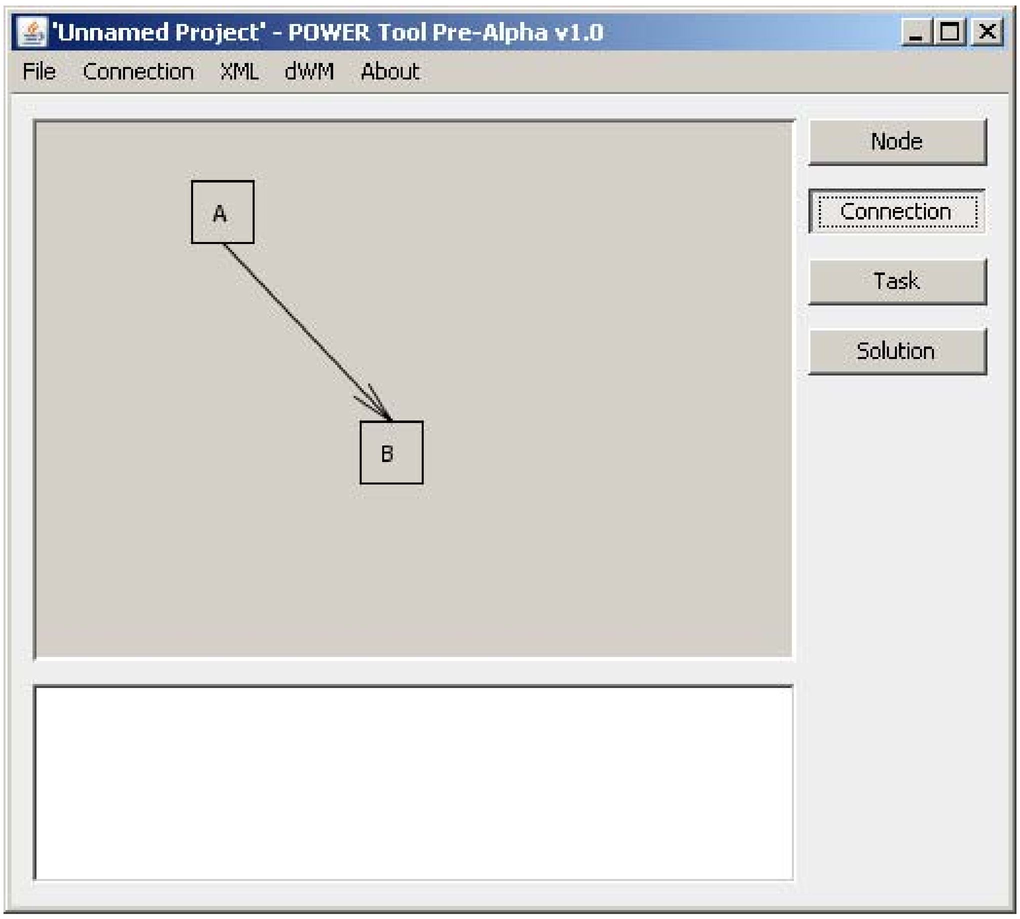

'Unnamed Project' - POWER Tool Pre-Alpha v1.0
File Connection XML dWM About
A
B
Node
Connection
Task
Solution

Figure 12: Network representation for the basic application scenario in the POWER authoring demonstrator tool

The XML description as output of the authoring demonstrator looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
        <nodes>
                <node>
                        <id>A</id>
                        <cover>
                                <cc>
                                        <type>image</type>
                                        <chancap>70</chancap>
                                </cc>
                        </cover>
                        <dwm>dwm1</dwm>
                        <nodecap>100</nodecap>
                        <pki>n</pki>
                        <ts>n</ts>
                </node>
                <node>
                        <id>B</id>
                        <cover>
                                <cc>
                                        <type>image</type>
                                        <chancap>80</chancap>
                                </cc>
                        </cover>
                        <dwm>dwm1</dwm>
                        <nodecap>90</nodecap>
                        <pki>n</pki>
                        <ts>n</ts>
                </node>
        </nodes>
        <lc>
                <connection>
                        <src>A</src>
                        <dst>B</dst>
                </connection>
        </lc>
</network>
```

Both users in this scenario (`A` and `B`) are connected by a unidirectional connection from `A` to `B`. Both know the same watermarking algorithm (`<dwm>dwm1</dwm>`) and can process image covers of the same type.

- **Algorithm description:**

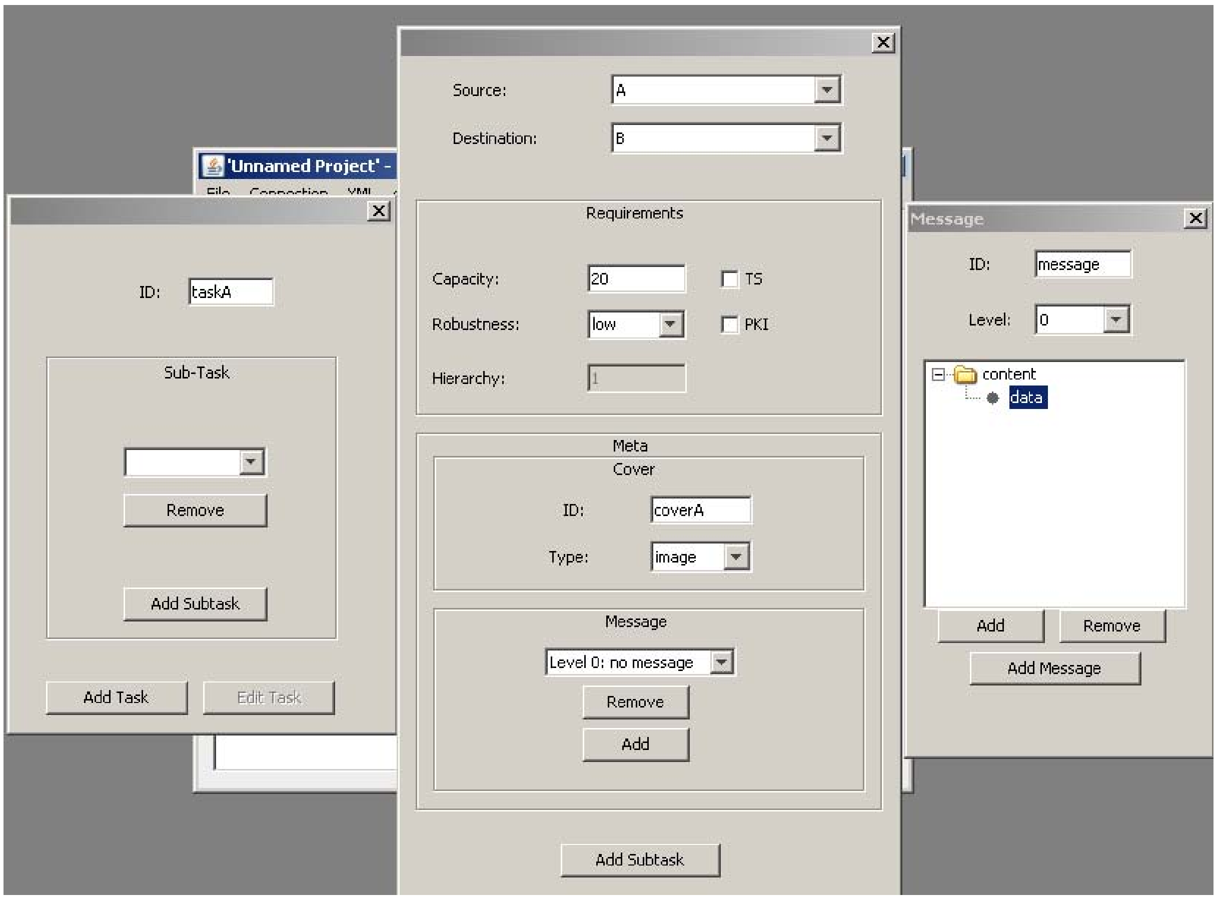
Like the network description for this example, the algorithm description required here is rather simple and looks in XML as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<dwms>
        <dwm>
                <id>dwm1</id>
                <cover>image</cover>
                <dwmcap>70</dwmcap>
                <robustness>low</robustness>
                <hierarchy>1</hierarchy>
                <key>symmetric</key>
        </dwm>
</dwms>
```

This exemplary algorithm works on covers of the type `image`, shows a low robustness (fitting a integrity verification watermarking example), does not allow for hierarchical data embedding and is a symmetric watermarking approach.

- **Task description:**

The most complex of the processes in the context modeling is the task description shown in figure 13, because here we describe a hierarchical construct (the task, consisting of subtasks – see section 2.2.1.1, all with specified requirements and message format information). In our demonstrator software we try to solve this problem for the defined application examples, nevertheless our solution is still far from being optimal.
The task description consists of three parts, which form together a hierarchical construct of tasks with their corresponding subtasks and the messages which are to be embedded. First, there is a task ID to be specified. After the set of subtasks for this specific task is created by adding one subtask after another, the second part of the task description, the subtask description includes all vital information for the path search, i.e. source, destination and all specified requirements. The third part is the specification of all information which is needed for the later translation into CASPER. Those include the cover properties as well as the structure and content (both editable and visualized in form of a tree) of every message to be embedded on each access level.


'Unnamed Project' -

ID: taskA

Source: A
Destination: B

Requirements
Capacity: 20
TS
Robustness: low
PKI
Hierarchy: 1

Meta
Cover
ID: coverA
Type: image

Message
Level 0: no message

Message
ID: message
Level: 0
content
data

Figure 13: Task specification for the basic application scenario in the POWER authoring demonstrator tool

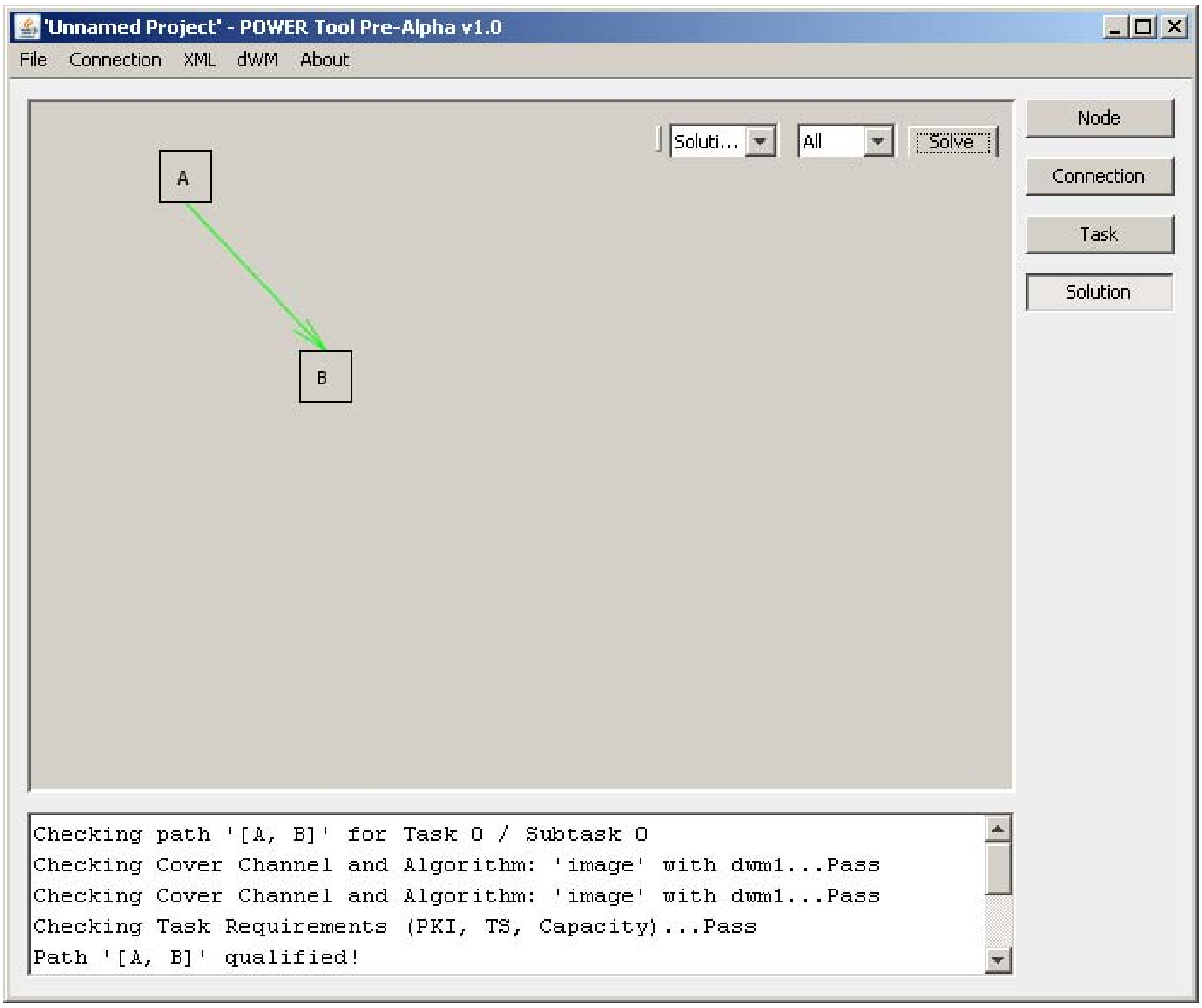
The corresponding XML representation of the task description looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<network_task>
        <task>
                <id>taskA</id>
                <subtask>
                        <src>A</src>
                        <dst>B</dst>
                        <meta>
                                <cover>
                                        <id>coverA</id>
                                        <type>image</type>
                                </cover>
                                <message>
                                        <id>message</id>
                                        <level>0</level>
                                        <content>
                                                <data>data</data>
                                        </content>
                                </message>
                        </meta>
                        <required>
                                <pki>n</pki>
                                <ts>n</ts>
                                <cap>20</cap>
                                <hierarchy>1</hierarchy>
                        </required>
                </subtask>
        </task>
</network_task>
```

The network task in this simple example consists of only one task, which itself consists of only one sub-task in turn. This sub-task requires that a message consisting of the element '`data`' should be send on hierarchy level `0` (which means without any hierarchy) from node `A` to `B` using a cover object `coverA` which is an image. Furthermore the capacity of this message is `20`, so any path needs at least that much capacity. It is stated that no PKI or access to a timeserver is needed for this task.

3.2.1.2. Path search (step 2)

In the second step of our framework, the path search described in section 2.2.1.1 is invoked within the authoring tool. The result of this path search looks for this basic application scenario as shown in figure 14. The corresponding project file for this example is found on the CD accompanying this report. Normally, the user would be able to browse the different alternative solutions identified by the authoring tool demonstrator, but this first small application scenario allows results in only one alternative solution.



```
Checking path '[A, B]' for Task 0 / Subtask 0
Checking Cover Channel and Algorithm: 'image' with dwm1...Pass
Checking Cover Channel and Algorithm: 'image' with dwm1...Pass
Checking Task Requirements (PKI, TS, Capacity)...Pass
Path '[A, B]' qualified!
```

Figure 14: Output of the task search for the basic application scenario in the POWER authoring demonstrator tool

The corresponding XML representation of the solution looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<solutions>
        <alternative>
                <task>
                        <id>taskA</id>
                        <subtask>
                                <src>A</src>
                                <dst>B</dst>
                                <meta>
                                        <cover>
                                                <id>coverA</id>
                                                <type>image</type>
                                        </cover>
                                        <message>
                                                <id>message</id>
                                                <level>0</level>
                                                <content>
                                                        <data>data</data>
                                                </content>
                                        </message>
                                </meta>
                                <dwm>
                                        <id>dwm1</id>
                                        <cover>image</cover>
                                        <robustness>low</robustness>
                                        <hierarchy>1</hierarchy>
                                        <key>symmetric</key>
                                </dwm>
                                <required>
                                        <pki>n</pki>
                                        <ts>n</ts>
                                        <hierarchy>1</hierarchy>
                                </required>
                        </subtask>
                </task>
        </alternative>
</solutions>
```

The XML-structure ***<solutions>*** for this application scenario contains only one alternative (***<alternative>***) and therefore only one possible path in the given network which satisfy the network task requirements.
The message in this small example is directly transferred from the source of the only task in the scenario ***A*** to its destination ***B*** by using images as cover (cover channel ***cover1***) and the watermarking algorithm ***dwm1*** (symmetric, low robustness and no multi-level access). For this simple scenario no PKI or time-server access are required and the messages (and therefore the whole scenario do not require multi-level access).

3.2.1.3. Path selection (step 3)

The path selection step (as the third step in the framework) identifies from all found solutions the one which is the most suitable based on a cost function. How to define suitable cost functions has to be derived from the given application scenario and is considered here a topic for future research.
In this example only one ***<alternative>*** exists so this one is chosen automatically.

3.2.1.4. Automated CASPER modeling using XML-input (step 4a)

The XML to CASPER translator described in section 3.1.2 is now applied to the output solution alternatives from section 3.2.1.3. Hereby the corresponding translation rules (2.2.1.2 and Appendix B) are invoked. The output of the translation tool looks as follows:

```
#Free variables
A, B : Agent
coverA : Cover
data : Data
keydwm1 : sharedKey
InverseKeys = (keydwm1, keydwm1)

#Processes
SENDER(A, keydwm1, coverA, data)
RECEIVER(B, keydwm1)

#Protocol description
0. -> A : B
1. A -> B : {data}{keydwm1}

#Specification

#Actual variables
AVA, AVB, Mallory : Agent
AVcoverA : Cover
```

```
AVdata : Data
AVkeydwm1 : sharedKey
InverseKeys = (AVkeydwm1, AVkeydwm1)

#Functions

#System
SENDER(AVA, AVkeydwm1, AVcoverA, AVdata)
RECEIVER(AVB, AVkeydwm1)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {AVA, AVB}
```

At this step also the ***#Intruder Information*** can be automatically created giving the intruder a standard set of knowledge. This could be - according to Kerckhoffs' law - all the information which is not explicitly specified as secret or a key not supposed to be known by the intruder.

3.2.1.5. CASPER model adjustment (step 4b)

In this simple example no further adjustment to the model itself is needed. However a ***#Specification*** representing the security properties which should be checked needs to be defined. There exist two main constructs for defining specification blocks: The first is a secret-specification, stating that some piece of data is only known to certain nodes after completion of the run. This one is used for evaluations on confidentiality in the protocol run. The second is an agreement which states, that two nodes are authenticated correctly and agree about a certain value. This statement is used for evaluations on authenticity.

In this example it should be checked if the message ***data*** is only known to nodes ***A*** and ***B*** after a complete run. As ***data*** is the element that should stay secret, this would show that the required confidentiality is preserved during the protocol transmissions. This yields the following specification:
#Specification
Secret (A, data, [B])

Checking the data-entity-authenticity between nodes ***A*** and ***B*** is realised by an agreement-statement:
Agreement (A,B,[data])

Which states that ***A*** and ***B*** are sure of each others identity and agree about the current value of the element ***data***.
For ease of demonstration, these two specifications are verified separately using FDR (see files bs_secret.spl, bs_auth.spl on the accompanying CD)

Another adjustment made here would be an altering of the ***#Intruder Information*** representing other possible scenarios. This could be i.e. the intruder learning a secret key by means of social engineering (which is be shown as example is this scenario during Step 6)

3.2.1.6. Compilation into CSP (step 5), checking with FDR (step 6) and protocol selection (step 7)

After compiling the CASPER source code into CSP it can be checked using FDR. In our example (bs_secret.spl) the confidentiality is to be checked using the secret-statement:

Secret (A, mes, [B])

In case there are no flaws in the protocol concerning the confidentiality of the message transmission, FDR answers “No attacks found”. In this case we can assume that no attack is possible in the given system under the given assumptions. As mentioned already depending on the application scenario it might be of importance to test several different attacker scenarios which could occur as well as different security aspects.

To show a possible attack on our example protocol we change the intruder knowledge. In this attacker scenario the intruder has learned the secret key, i.e. by social engineering:

IntruderKnowledge = { ... ,Avkeydwm1}

FDR now reports a possible attack on the protocol (known secret key):

```
Starting FDR
Checking /home/bs_secretattack.csp

Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
Attack found:

Top level trace:
   AVA believes AVdata is a secret shared with AVB
   The intruder knows AVdata

System level:
Casper> 0.     ->  AVA  : AVB
1. AVA -> I_AVB : {AVdata}{AVkeydwm1}
   The intruder knows AVdata

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
Attack found:

Top level trace:
Casper>   The intruder knows AVdata

System level:
Casper> 0.     ->  AVA  : AVB
1. AVA -> I_AVB : {AVdata}{AVkeydwm1}
   The intruder knows AVdata

Done
```

The output shows that a possible intruder ***Mallory*** can intercept the message from ***A*** to ***B*** by claiming to be ***B*** (represented by the term ***I_AVB***) and with the knowledge of the secret key is now able to retrieve the content of the message. This would be a breach of confidentiality.

Also the data-origin authenticity of our example can be checked with the corresponding agreement statement (***Mallory*** being between ***A*** and ***B***, but not knowing the key):
Agreement (A,B,[data])

Example FDR output in case of a possible attack (man-in-the-middle):

```
Checking assertion AUTH1_M::AuthenticateSENDERToRECEIVERAgreement_data
[T= AUTH1_M::SYSTEM_1
Attack found:

Top level trace:
   AVB believes (s)he has completed a run of the protocol, taking role
RECEIVER, with AVA, using data items AVdata

System level:
Casper> 0.       ->    AVA    : Mallory
1.  AVA  -> I_Mallory : {AVdata}{AVkeydwm1}
1. I_AVA ->    AVB    : {AVdata}{AVkeydwm1}

Done
```

The error message of FDR shows a typical “Man-In-The-Middle” attack on a possible watermarking protocol. Even if the attacker (***Mallory***) cannot change the watermark itself (as the data-origin authenticity has been verified before) ***Mallory*** can still try to conduct a replay attack by delaying or resending the watermarked cover.

While all examples given here represent the evaluation of only one solution block from the XML model, it is possible to translate different solutions into protocols, evaluate them and then choose the one that fulfills all given specifications best. As for the path selection in step 3 the specification of the corresponding cost or quality functions for this protocol selection is considered to be outside the scope of our research.

3.3. Illustration of the application of the framework using the complex scenarios CDSC, HAAI and HDSR

Three different abstract, complex application scenarios have been introduced in the first (M6) POWER project report document. These old descriptions of the complex application scenarios have been:

- **Certificate/Digital signature chain in watermarking domain (CDSC):** A card reader or sensor device is recognized by the computer *A*, to which it is attached, by the watermark it embeds into the data stream, which it sends. This device ID of the reader is forwarded (again as watermark) together with the ID of computer *A* (as watermark) and the input from the device to further stations in the network for processing.
- **Hierarchical Access, Authentication & Integrity (HAAI):** Section *A* has to send report material to a section *B*. The report material shall be send without any encryption so that every relay between *A* and *B* can observe its content. The complete section *B* will be able to verify section *A* as origin, the timestamp of watermarking and the integrity of the report from a first level of watermarking. A second level of watermarking will give the section chief of *B* additional info (e.g. which person in *A* was responsible for the generation of the report etc).

- **Hierarchical Digital Signatures for Reproduction of Original (HDSR)** The scenario III enhances the protocol by enabling the reconstruction of the original data stream sent by the card reader to allow access to clear cover data. Here also hierarchical access alternatives motivated from scenario II are further investigated to evaluate and summarize pros and cons. Furthermore the impact of an erroneous communication channel (error due to transmission errors) will be elaborated and protocol mechanisms suggested.

The first two of these abstract, complex application scenarios are substituted in this document by real-world application scenarios, which implement the same basic concepts, but are more realistic and therefore better suited to identify the problems that are encountered when designing watermarking based protocols.

The new **Certificate/Digital signature chain in watermarking domain (CDSC)** application description is taken from a forensics application scenario, where confidentiality (in form of privacy protection), entity-authenticity, data-origin-authenticity and integrity have to be ensured in the watermarking-based protocol.

The new design of a **Hierarchical Access, Authentication & Integrity (HAAI)** watermarking protocol used here combines a multi-level access structure and the assurance of the security aspects of confidentiality, authenticity and integrity. It extends the initial HAAI concept by considerations on data transfer via untrusted communication partners.

3.3.1. CDSC - Scenario implementation

The FRT-MicroProf sensor ([Fries03]) is a high-resolution[17] fingerprint scanner using a chromatic white light (CWL) sensor which can be used to capture biometric fingerprint images on a crime scene. For such high-quality biometric fingerprint data several legal requirements have to be fulfilled in a forensic investigation to be accepted as evidence in a law suit. First, the data privacy of person related biometric data captured at a crime scene (which often includes biometric data of uninvolved people) must be protected to respect data privacy laws. Second, according to [Newman07], for the data to be accepted as evidence in a law suit, all data needs to be protected and documented for the complete path from the generation until the final disposition, which is called the ‘chain of custody’. It has to be proven without any doubts that the data is authentic (meaning it is original) and integer (no unauthorized modification has taken place). Third, all changes made to the data need to be reproducible if requested by a judge.

Designing a protocol to be used in a **real life, forensic application scenario** using biometric data therefore requires these three prerequisites to be fulfilled. These legal requirements have to be mapped to measurable security aspects which need to be included into the design of a proposed protocol. The data privacy (protection against unauthorized revealing of person related data) can be assured by including confidentiality mechanisms, but still it has to be verified in the protocol. Authenticity and integrity are stated in [Newman07] to be the essential requirements to preserve

[17] The sensor can be used to contactlessly scan latent fingerprint traces, e.g. at crime scenes. The CWL sensor head has maximum resolution of about 2μm in the x/y-plane and a resolution of about 6nm in the z-plane.

the chain of custody. For the reproducibility requirement a secure way of documenting data-modifications needs to be implemented.
As a result the three mentioned legal requirements needed for a protocol to be accepted as evidence in a forensic application scenario can be mapped to the measurable security aspects of confidentiality, data-origin-authenticity, entity-authenticity, integrity and the need to enable reproducibility.

Here, we propose a **Certificate/Digital signature chain in watermarking domain (CDSC)** solution for this complex and real-world relevant application scenario, i.e. we propose to use a watermarking protocol as an alternative to present purely cryptographic approaches. This new approach has several advantages. As an example, if an optical scan of a crime scene is made and only the biometric fingerprint area of the image is obfuscated by substitution with a visual watermark to protect the privacy of the biometric data, the surroundings of the fingerprint can still be seen and analyzed by personnel investigating other aspects of the crime. Without having access to the necessary secret key and therefore to the fingerprint area the privacy protection of the data is enhanced (which would not be possible in a purely cryptographic protocol). Furthermore, meta-information about the capture of the fingerprint (such as time, capturing officer or resolution) or changes applied to the image (such as the application of filters, binarization of fingerprints, etc.) as well as additional security measures (such as signatures or hashes) can be easily stored as watermark payload binding the meta-data to the object without the need to create additional objects and implementing the un-observability that distinguished watermarking from encryption.
The **overall setup** using the FRT CWL sensor suits our CDSC scenario very well because of the following characteristics:

- It enables us to evaluate the CDSC scenario in the context of a real project involving highly sensitive person related data.
- Several nodes exist in the processing chain which are changing the data in different ways (some may only sign the data; some may alter the cover or extract/embed data).
- Due to the assumed application scenario in forensics the chain-of-custody needs to be assured, requiring:
    - direct watermarking of the sensor data at the sensor
    - authenticity and integrity verification for each processing and transfer step
    - Mechanisms enabling reproducibility of changes made to the data
- The intended usage in forensic applications requires a second layer of access restrictions for reasons of data privacy protection. The first layer (or hierarchy level; here called “public data”) allows every legitimate user of the system to retrieve metadata (like e.g. a case number) form a public watermarking part, while only case-specific access to the second hierarchy level, containing “private data”, i.e. the potential evidence (here the fingerprint) and information about its processing, together with the required signatures of the processors, can be allowed. Different multi watermarking / hierarchical watermarking approaches introduced by Sheppard (see section 1.5) might be used here to implement these two required hierarchy levels for the watermarking scheme used.

Regarding the watermarking **message generation**, the watermarking approach developed to implement the CDSC scenario is based on an blind, invertible, non-

transparent, fragile secure medial authentication scheme introduced by Dittmann and Katzenbeisser in 2004 / 2005 ([Dit04], [Dittmann05]), which can provably assure the authenticity and integrity of the cover. The scheme is applied to biometric fingerprint data from the FRT CWL sensor and extended to assure data privacy as well as to be suitable for a digital signature chain scenario and hierarchical watermarking. Also the developed approach enables us to distinguish between private and public watermarking data where the former one needs case specific access protection and the latter one can be freely accessed by any legitimate user of the system.
The concept of the scheme is to divide a cover image *O* into an unchangeable area A_O and an embedding area $B_{O,}$ as shown in figure 15.

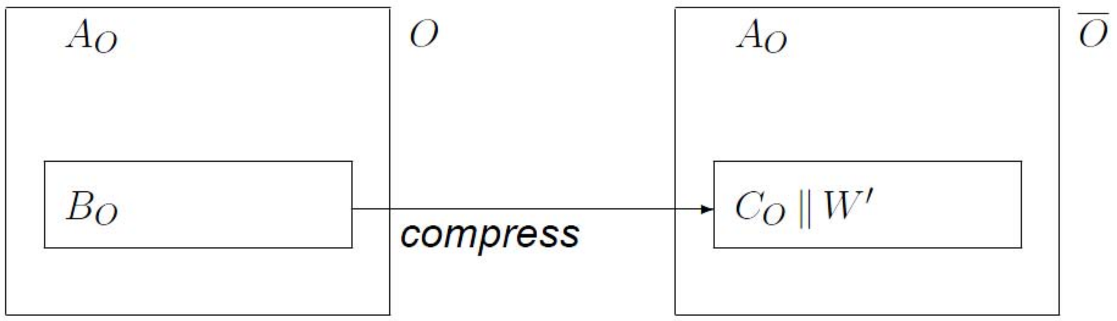


Figure 15: Selection of the embedding area B_O of the cover image which is substituted by the watermark including the compressed original pixel values and payload (taken from [Dittmann05])

The original pixel values of the embedding area B_O are substituted by a compressed and encrypted version of B_O concatenated with the watermark payload which assures the obfuscation of the original embedding area values but which at the same time contains the original pixel values, payload and signature/hash data in form of a watermark (see figure 16).

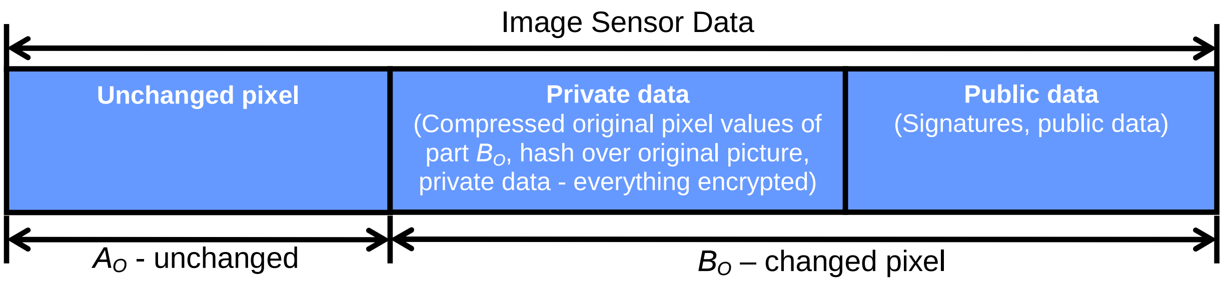


Image Sensor Data
Unchanged pixel
Private data
(Compressed original pixel values of part B_O, hash over original picture, private data - everything encrypted)
Public data
(Signatures, public data)
A_O - unchanged
B_O – changed pixel

Figure 16: Structure of the cover image: Unchanged pixel area A_O and embedding area B_O which is comprised of two parts for private and public data

The location information of the embedding area B_O is later stored in form of a location map in a second watermarking process. This procedure uses a transparent difference expansion annotation watermark approach introduced by Coltuc et. al ([Col07]) to embed the location map without visual artifacts in the watermarked image.
The extended approach can assures the following security aspects:

- Authenticity
- Integrity
- Confidentiality
- Privacy protection (for the person-related Biometric fingerprint data it is designed for)

- Reproducibility (if trusted nodes store changes they made in the private data area)

In the Digital Signature Chain of the CDSC – Scenario each node has the option to apply an arbitrary combination of the following actions to the watermarking object:

- Verifying/adding signatures
- Accessing/writing public data
- Accessing/writing private data (private key required)
- Reconstruction of original image (private key required)
- Selection of new embedding area

With the help of these options also different types of multiple watermarking or hierarchical watermarking described by Sheppard ([Sheppard01]; see section 1.5) could be realized. With the selection of a new embedding area the image can be re-watermarked. By extracting the private data and reconstructing the original image a watermark-decomposition approach can be used. If only the forwarding of a watermarked object is to be logged a forwarding node can sign the object and store the signature in the public area. Each node can then independently check each embedded signature of previous forwarding nodes.

Regarding the **embedding process** used in the watermarking algorithm (or primitive) for our CDSC demonstrator, we employ the following concept: First we context based divide the cover image O into the unchanged area A_O and the embedding area B_O. For this purpose we use the Segmenter module shown in figure 17. We can select B_O either by manually drawing a polygon in a graphical user interface or automatically by content identification with the help of Gabor and median filters. The location information of B_O is stored in the location map LM.

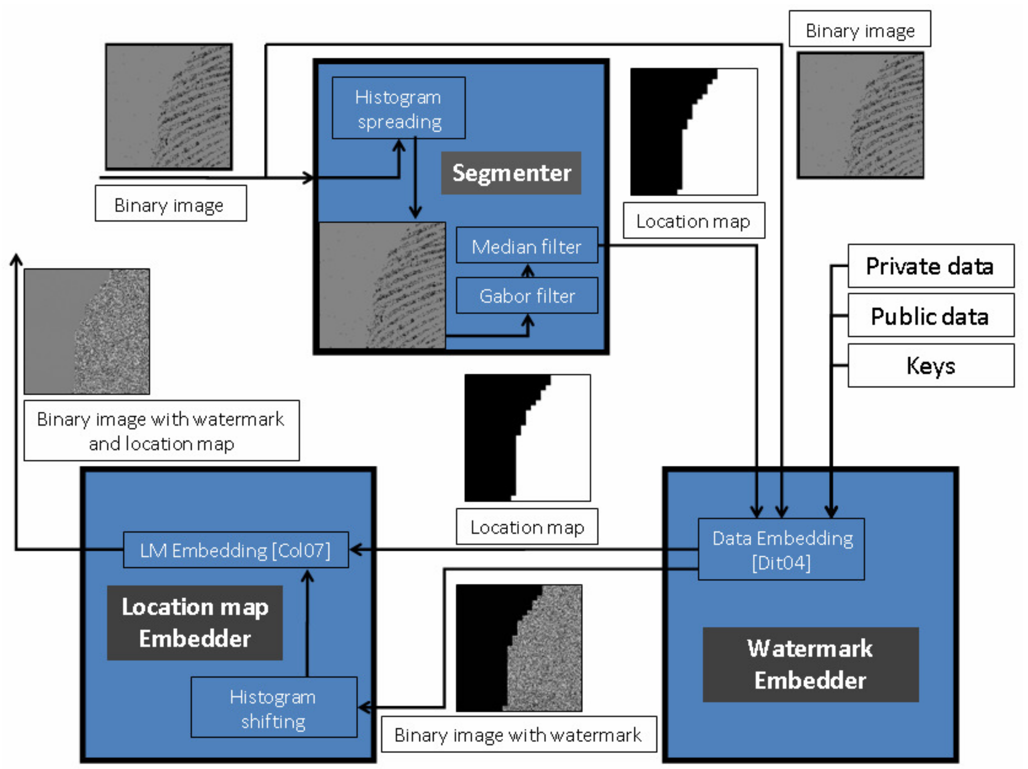


Figure 17: The Structure of the developed watermarking scheme – embedding process

In the Watermark Embedder module the original pixel values of the area B_O are concatenated with the private data D_{priv} and then compressed:

$$c = C(B_O \,||\, D_{priv})$$

A hash value h of the original image is computed using SHA512:

$$h=H(O).$$

The compressed data c and the hash value h are then concatenated and encrypted using DES56:

$$e = DES56_{sk}(c \,||\, h)$$

A signature s is computed over the unchanged pixels A_O and the encrypted data e:

$$s=S(A_O \,||\, e)$$

The final watermark w is then generated by concatenation of the encrypted data e, the signature s and the public data D_{pub} as well as padding (up to full capacity):

$$w = e \,||\, s \,||\, D_{pub} \,||\, padd$$

The created watermark w is then inserted into the image O by substitution of the area B_O with the watermark, the location information is provided by LM:

$$O'=Substitute(LM,w)$$

The Location Map Embedder module then inserts the location map LM together with additional meta information into the watermarked object O'. This is done by an invertible difference expansion approach introduced by Coltuc ([Col07]) using a known *seed* (which acts like a system-wide secret):

$$O''=Emb_{seed}(O',LM)$$

Since the developed approach is fragile and visible by design, the most important watermarking characteristic to analyze is the **capacity**. The compression rates of the embedding area B_O and the location map LM are the major influence factors to the achievable capacity.
Since compression rate of B_O is completely unpredictable also the capacity can vary largely from very low to extremely high embedding rates. For most cases B_O can be compressed to at least some extent. Since the signature and hash sizes to be stored in this CDSC scenario are comparably low, it should be possible in most cases to assure the space needed for integrity and authenticity information.
If the capacity is almost completely used and the fingerprint occupies quite a large area of the image, some bytes of the capacity may be needed for the embedding of the location map.

3.3.1.1. CDSC - Scenario – Network and Task description (step 1)

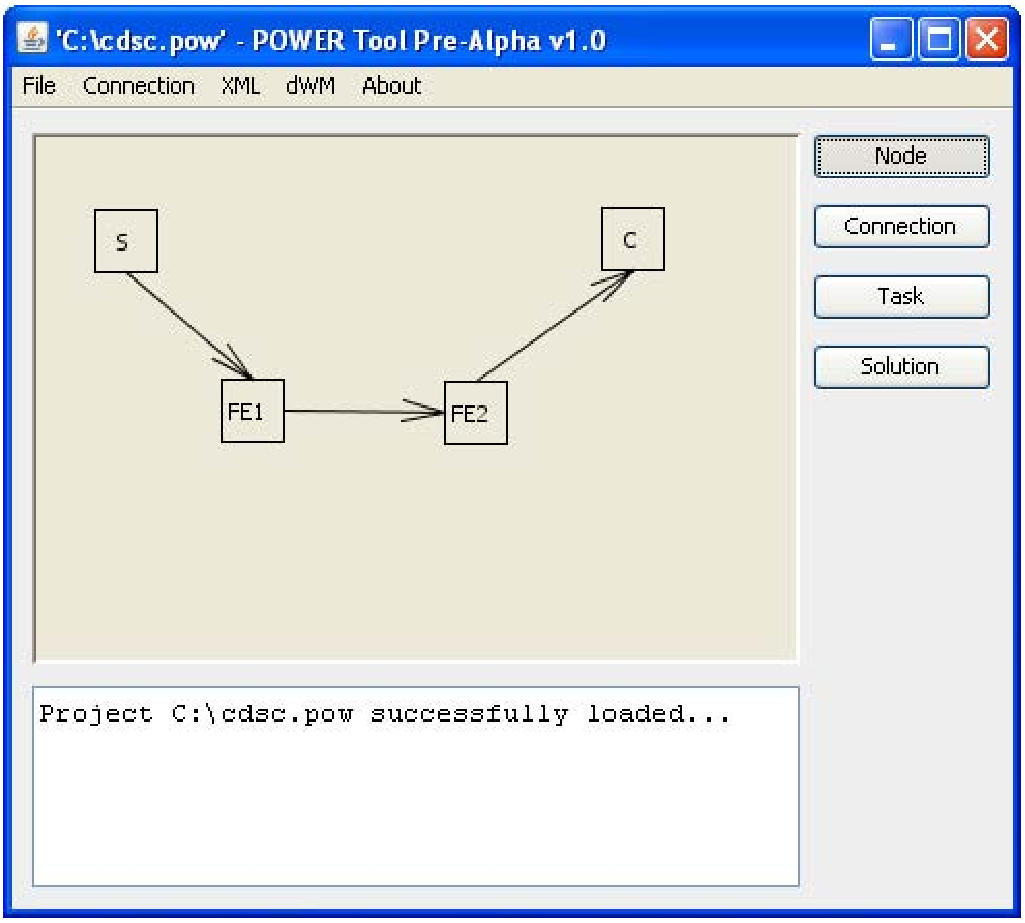
In step 1 of the introduced realization framework, we perform the network and task description for this application scenario (see section 2.2.1.1). For this purpose we

have to give a more specific description of the application scenario at hand. Here the general forensics setup introduced above is limited in the conceptualization to the following data flow protocol:
Our high-resolution fingerprint-scanner is exemplary capturing and watermarking the biometric data at a crime scene, representing the sensor *S*. The traces are then sent to a forensic expert *FE1*: *S*$\rightarrow$*FE1*. The expert processes the data (e.g. enhances the quality of the image) and sends it on to another forensic expert *FE2* who further processes the image or extracts in-formation from it (such as fingerprint verification): *FE1*$\rightarrow$*FE2*. Finally the image is taken to a court hearing *C* where it can be used as evidence: *FE2*$\rightarrow$*C*. This modeled scenario can easily be extended for other forensic application scenarios according to the number of sensors, forensic experts or court-hearings needed.
All four nodes of the proposed scenario need to have access to the biometric data for capturing, processing or reviewing it. They are considered trustworthy therefore having knowledge of the secret key needed to extract the confidential payload and to reconstruct the original biometric image. Therefore, while *S* is creating the image, *FE1*, *FE2* and *C* can reconstruct it, alter the payload and/or the cover image and re-embed data. Additionally, all parties have to check the signatures of sources they received an image from and embed their own signature before sending an image to the next destination. All this information is modeled in the first step by specifying the knowledge of the nodes and the watermarking scheme *dwm* with the help of XML.

The XML context modeling of this contextualization is done by using the authoring demonstrator tool to describe the network (see figure 18), the algorithm (figure 19), the messages required for the two defined access levels and the network task (described here as three tasks each containing one sub-task, for sub-task description *S*$\rightarrow$*FE1* see figure 20).

- **Network description:**

Figure 18 shows the network description of our CDSC application scenario realization in the authoring tool demonstrator.



```
Project C:\cdsc.pow successfully loaded...
```

Figure 18: Contextualizing the information flow in our CDSC-based forensics application scenario using the authoring demonstrator tool

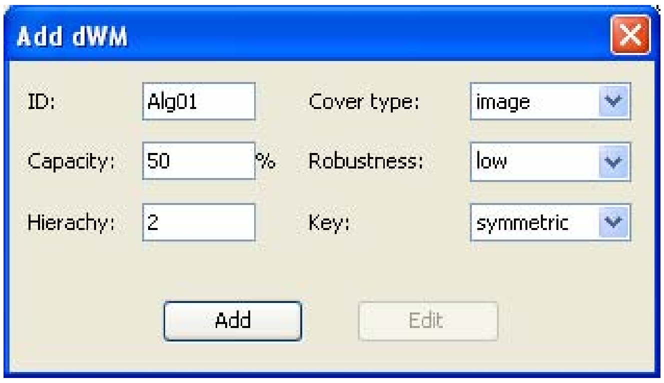
The XML description of the network generated by the authoring tool looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
      <nodes>
            <node>
                  <id>S</id>
                  <cover>
                        <cc>
                              <type>image</type>
                              <chancap>3000</chancap>
                        </cc>
                  </cover>
                  <dwm>Alg01</dwm>
                  <nodecap>3000</nodecap>
                  <pki>y</pki>
                  <ts>n</ts>
            </node>
            <node>
                  <id>FE1</id>
                  <cover>
                        <cc>
                              <type>image</type>
                              <chancap>3000</chancap>
                        </cc>
                  </cover>
                  <dwm>Alg01</dwm>
                  <nodecap>3000</nodecap>
                  <pki>y</pki>
                  <ts>n</ts>
            </node>
            ...
      </nodes>
      <lc>
            <connection>
                  <src>S</src>
                  <dst>FE1</dst>
            </connection>
            <connection>
                  <src>FE1</src>
                  <dst>FE2</dst>
            </connection>
            <connection>
                  <src>FE2</src>
                  <dst>C</dst>
            </connection>
      </lc>
</network>
```

In the network description the node descriptions for `FE2` and `C` are identical (except for the node identifiers) to the ones for `S` and `FE1`, therefore they are omitted here. All nodes in this example possess knowledge of the same cover channels (an image) and the same watermarking algorithm (`<dwm>Alg01</dwm>`). All four nodes have access to the PKI for the scenario (required for the signatures).

- **Algorithm description:**

The algorithm description shown in figure 19 and the corresponding XML structure is defining the two hierarchy levels required in this application scenario to model the two access-levels.



ID:	Alg01	Cover type:	image
Capacity:	50 %	Robustness:	low
Hierachy:	2	Key:	symmetric

Figure 19: Algorithm description for the watermarking algorithm used in the CDSC application example

The authoring tool demonstrator generates the following output algorithm description:

```
<?xml version="1.0" encoding="UTF-8"?>
<dwms>
   <dwm>
      <id>Alg01</id>
      <cover>image</cover>
      <dwmcap>50</dwmcap>
      <robustness>low</robustness>
      <hierarchy>2</hierarchy>
      <key>symmetric</key>
   </dwm>
</dwms>
```

This is a simple algorithm description used for the demonstration of our framework within this report. A more detailed description of algorithms would be necessary in the generation of any commercial- or field-applicable software. A more detailed picture of the information required for close-to-marked prototypes might be the following attempt at algorithm description for our CDSC scenario in pseudo-XML:

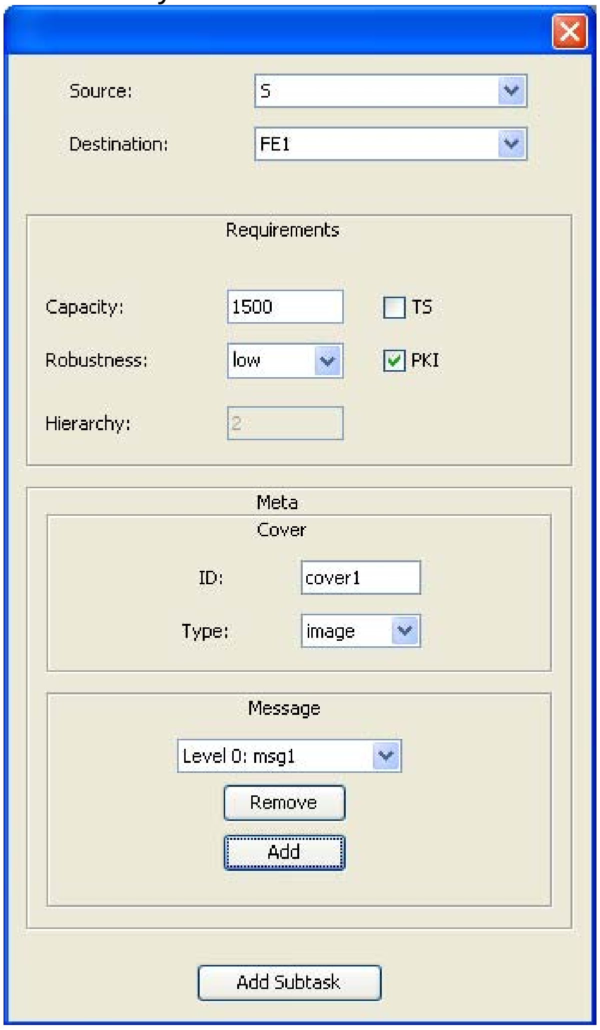
```
<?xml version="1.0" encoding="UTF-8"?>
<dwm>
  <id>MediaAuthScheme</id>
  <cover>image/FTR/16Bit</cover>
  <technique>compression</technique>
  <embedding area determination>
    <context based>yes</context based>
    <manual>yes</manual>
  </embedding area determination>
  <overhead>
    <location-map>yes</location-map>
    <add. overhead>few Bytes</add. overhead>
    <overhead embedding>
      <technique>diff. expansion</technique>
      <addFeatures>histogram shifting</addFeatures>
    </overhead embedding>
  </overhead>
  <data security properties>
      <confidential data>yes</confidential data>
      <public data>yes</public data>
      <integrity>
        <ensured>yes</ensured>
      <technique>hash</technique>
      <technique>signature</technique>
    </integrity>
    <authenticity>
      <ensured>yes</ensured>
      <technique>signature</technique>
    </authenticity>
    <reproducibility>
      <reproducible>optional</reproducible>
      <technique>operation_log</technique>
    </reproducibility>
  </data security properties>
  <watermark properties>
    <dwm-capacity>
      <value>100*(1-K)%</value>
      <metric>byte</metric>
    </dwm-capacity>
    <invertible>yes</invertible>
    <blind>yes</blind>
    <robustness>low</robustness>
    <transparency>none, obfuscation of embedded area</transparency>
    <key>symmetric and asymmetric</key>
  </watermark properties>
</dwm>
```

- **Message description:**

The watermarking message in this application scenario is modeled as two different messages parts - one for each of the two hierarchy levels. Level 0 in this example contains the case-related private data (i.e. the compressed and encrypted fingerprint data required for the inversion of the watermark, the signature chain as well as the secret payload). Due to the complex nature of the message, which changes during the transmission along the processing pipeline by the addition of further processing-related data and signatures, it is not modeled completely here, but the corresponding elements are identified by the flag `complex` (followed by an indicator for the last modifying node, e.g. `complexS` for the private data send by S) This flag is used to signal that after the translation into CASPER a manual adaptation of the message in the CASPER code has to be performed here for each data transmission.

- **Task description:**

The network task for this application scenario is described here as three tasks (each containing one sub-task). In figure 20 the GUI-based description of one of these three nearly identical sub-tasks is shown.



Source: S
Destination: FE1

Requirements
Capacity: 1500 | ☐ TS
Robustness: low | ☑ PKI
Hierarchy: 2

Meta
Cover
ID: cover1
Type: image

Message
Level 0: msg1

Figure 20: Description of one of the sub-tasks in the CDSC application example

Task description in XML is generated by the authoring tool demonstrator (the tasks ***FE1→FE2*** and ***FE2→C*** are omitted here because they are identical to the presented task ***S→FE1***):

```
<?xml version="1.0" encoding="UTF-8"?>
<network_task>
      <task>
            <id>Task1</id>
            <subtask>
                  <src>S</src>
                  <dst>FE1</dst>
                  <meta>
                        <cover>
                              <id>cover1</id>
                              <type>image</type>
                        </cover>
                        <message>
                              <id>msg1</id>
                              <level>0</level>
                              <content>
                                    <signed>
                                          complexS
                                    </signed>
                              </content>
                        </message>
                        <message>
                              <id>msg2</id>
                              <level>1</level>
                              <content>
                                    <data>
                                          PublS
                                    </data>
                              </content>
                        </message>
                  </meta>
                  <required>
                        <pki>y</pki>
                        <ts>n</ts>
                        <cap>1500</cap>
                        <hierarchy>2</hierarchy>
                  </required>
            </subtask>
      </task>
      <task>
            ...
      </task>
</network_task>
```

The network task description automatically integrates the message descriptions described above.

3.3.1.2. CDSC - Scenario – Path search (step 2)

In the second step of our framework, the path search described in section 2.2.1.1 is invoked within the authoring tool. The result of this path search looks for this CDSC scenario as shown in figure 21. The corresponding project file for this example is found on the accompanying CD. Normally, the user would be able to browse the different alternative solutions identified by the authoring tool demonstrator, but our realization of the CDSC scenario results in only one alternative solution due to the sequential setup of the communion flow.

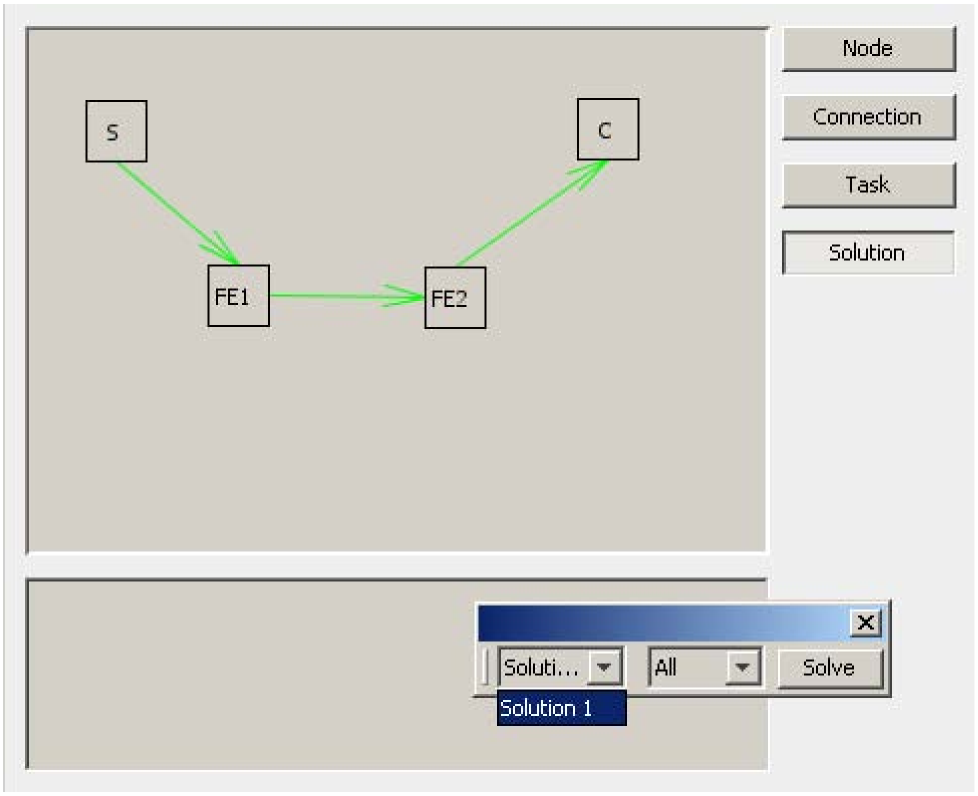


Figure 21: Output of the task search for the CDSC application scenario in the POWER authoring demonstrator tool

The corresponding XML representation of the solution looks as follows (again, the descriptions for the ***FE1→FE2*** and ***FE2→C*** tasks are omitted):

```
<?xml version="1.0" encoding="UTF-8"?>
<solutions>
      <alternative>
            <task>
                  <id>Task1</id>
                  <subtask>
                        <src>S</src>
                        <dst>FE1</dst>
                        <meta>
                              <cover>
                                    <id>cover1</id>
                                    <type>image</type>
                              </cover>
                              <message>
                                    <id>msg1</id>
                                    <level>0</level>
                                    <content>
                                          <signed>
                                                complexS
                                          </signed>
                                          <data>
                                                As
                                          </data>
                                    </content>
                              </message>
                              <message>
                                    <id>msg2</id>
                                    <level>1</level>
                                    <content>
                                          <data>
                                                PublS
```

```
                                        </data>
                                  </content>
                            </message>
                      </meta>
                      <dwm>
                            <id>Alg01</id>
                            <cover>image</cover>
                            <robustness>low</robustness>
                            <hierarchy>2</hierarchy>
                            <key>symmetric</key>
                      </dwm>
                      <required>
                            <pki>y</pki>
                            <ts>n</ts>
                            <hierarchy>2</hierarchy>
                      </required>
                </subtask>
          </task>
          ...
    </alternative>
</solutions>
```

In our scenario the `<solution>` block offers only one `<alternative>`, representing one possible *path* through the network. Using the image ***cover1*** as cover for the first task (*S→FE1*) the source node ***S*** transmits the watermarked cover to the destination node ***FE1***.
The core functionality of the watermarking scheme is coded in the structure of the watermarking messages (`<message>`) `msg1` and `msg2` which are used by the watermarking algorithm (`<dwm>`) `Alg01`. According to our specification of the watermarking scheme an instance of the `<message>` `msg1` holds `signed` content flagged with the `complex` content indicator. As described above in the message composition in section 3.3.1.1, this indicator will result in manual adaptations in the CASPER protocol adjustment (step 4b of the framework).
The `<message>` `msg2` contains the public content (`Publ` followed by an indicator for the last modifying node, e.g. `PublS` for the public data send by `S`) which is neither signed or hashed nor encrypted.

3.3.1.3. CDSC - Scenario – Path selection (step 3)

In the third step of the framework found paths are analyzed using a cost-function and the best path is chosen. Again, for our scenario only one path is possible for each task (the single `<alternative>` described in section 3.3.1.2) and we assume that the capacity (in this case representing the cost) is sufficient.

3.3.1.4. CDSC - Scenario – Automated CASPER modeling using XML-input (step 4a)

Here, the chosen **`<alternative>`** from the generated **`<solutions>`** block is translated into a protocol of the CASPER formalization language. The XML to CASPER translator described in section 3.1.2 is now applied to the output solution alternatives from section 3.2.1.3. Hereby the corresponding translation rules (2.2.1.2) are invoked. For this application scenario special translation rules need to be invoked to be able to also translate the generation and verification of signatures and hash-values.

Modeling the CDSC application scenario in CASPER looks as follows (the ***#Intruder Information*** and ***#Specifications*** blocks required to verify the security aspects of entity-authenticity and confidentiality as well as the resolution of the `complex` flag are manually specified in step 4b – see section 3.3.1.5):

```
#Free variables
S, FE1, FE2, C : Agent
cover1 : Cover
complexS, PublS, complexFE1, PublFE1, complexFE2, PublFE2 : Data
H : HashFunction
keyLevel0, keyLevel1, keyAlg01 : SharedKey
pubS, pubFE1, pubFE2, pubC : PublicKey
secS, secFE1, secFE2, secC : SecretKey
InverseKeys  =  (keyAlg01,  keyAlg01),  (keyLevel0,  keyLevel0),  (keyLevel1,
keyLevel1), (pubS, secS), (pubFE1, secFE1), (pubFE2, secFE2), (pubC, secC)

#Processes
SENDER(S, keyAlg01, cover1, complexS, PublS, secS, keyLevel0, keyLevel1)
FORWARDER(FE1, keyAlg01, complexFE1, PublFE1, secFE1)
FORWARDER(FE2, keyAlg01, complexFE2, PublFE2, secFE2)
RECEIVER(C, keyAlg01, secC, keyLevel0, keyLevel1)

#Protocol description
0. -> S : secS, keyLevel0, keyLevel1
1. -> FE1 : secFE1, keyLevel0, keyLevel1
2. -> FE2 : secFE2, keyLevel0, keyLevel1
3. -> C : secC, keyLevel0, keyLevel1
4. -> S : FE1
5. -> FE1 : FE2
6. -> FE2 : C
7. S -> FE1 : {PublS, {complexS}{secS}}{keyAlg01}
8. FE1 -> FE2 : {PublFE1, {complexFE1}{secFE1}}{keyAlg01}
9. FE2 -> C : {PublFE2, {complexFE2}{secFE2}}{keyAlg01}

#Specification

#Actual variables
AVS, AVFE1, AVFE2, AVC, Mallory : Agent
AVcover1 : Cover
AVcomplexS,  AVPublS,  AVcomplexFE1,  AVPublFE1,  AVcomplexFE2,  AVPublFE2  :
Data
AVkeyLevel0, AVkeyLevel1, AVkeyAlg01 : SharedKey
AVpubS, AVpubFE1, AVpubFE2, AVpubC : PublicKey
AVsecS, AVsecFE1, AVsecFE2, AVsecC : SecretKey
InverseKeys   =   (AVkeyAlg01,   AVkeyAlg01),   (AVkeyLevel0,   AVkeyLevel0),
(AVkeyLevel1,   AVkeyLevel1),   (AVpubS,   AVsecS),   (AVpubFE1,   AVsecFE1),
(AVpubFE2, AVsecFE2), (AVpubC, AVsecC)

#Functions

#System
SENDER(AVS, AVkeyAlg01, AVcover1, AVcomplexS, AVPublS, AVsecS, AVkeyLevel0,
AVkeyLevel1)
FORWARDER(AVFE1, AVkeyAlg01, AVcomplexFE1, AVPublFE1, AVsecFE1)
FORWARDER(AVFE2, AVkeyAlg01, AVcomplexFE2, AVPublFE2, AVsecFE2)
RECEIVER(AVC, AVkeyAlg01, AVsecC, AVkeyLevel0, AVkeyLevel1)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {AVS, AVFE1, AVFE2, AVC}
```

3.3.1.5. CDSC - Scenario – CASPER model adjustment (step 4b)

For our CDSC application scenario in the protocol adjustment step multiple manual adaptations have to be performed. First, the public keys of all users are distributed.

```
#Free Variables
```
```
#Processes
```
```
#Protocol Description
→replace:
0. -> S : secS, keyLevel0, keyLevel1
1. -> FE1 : secFE1, keyLevel0, keyLevel1
2. -> FE2 : secFE2, keyLevel0, keyLevel1
3. -> C : secC, keyLevel0, keyLevel1

by:
0. -> S : secS, keyLevel0, keyLevel1, pubS, pubFE1, pubFE2, pubC
1. -> FE1 : secFE1, keyLevel0, keyLevel1, pubS, pubFE1, pubFE2, pubC
2. -> FE2 : secFE2, keyLevel0, keyLevel1, pubS, pubFE1, pubFE2, pubC
3. -> C : secC, keyLevel0, keyLevel1, pubS, pubFE1, pubFE2, pubC
```
```
#Specification
```
```
#Actual variables
```
```
#Functions
```
```
#System
```
```
#Intruder Information
```

As a second adaptation block the ***complex*** flags (representing a complex data structure containing amongst others the signature chain, the compressed original image part (original biometric data ***B***) as well as the secret payload ***PrivS***) have to be substituted by the real message contents for the corresponding step. For a consistent notation we append the identifier of each node in our scenario to the data items where needed, i.e. changing the private data ***Priv*** embedded by the sensor ***S*** to ***PrivS***, ***PrivFE1*** for the first forensic expert ***FE1*** and ***PrivFE2*** for the second forensic expert ***FE2***, etc.

```
#Free Variables
→replace:
complexS, PublS, complexFE1, PublFE1, complexFE2, PublFE2 : Data

by:
PublS, PublFE1, PublFE2, PrivS, PrivFE1, PrivFE2, BS, BFE1, BFE2 : Data
```
```
#Processes
→replace:
SENDER(S, keyAlg01, cover1, complexS, PublS, secS, keyLevel0, keyLevel1)
FORWARDER(FE1, keyAlg01, complexFE1, PublFE1, secFE1)
FORWARDER(FE2, keyAlg01, complexFE2, PublFE2, secFE2)

by:
SENDER(S, keyAlg01, cover1, PrivS, BS, PublS, secS, keyLevel0, keyLevel1)
FORWARDERA(FE1,  keyAlg01,  PrivFE1,  BFE1,  PublFE1,  secFE1,  keyLevel0,
keyLevel1)
FORWARDERB(FE2,  keyAlg01,  PrivFE2,  BFE2,  PublFE2,  secFE2,  keyLevel0,
keyLevel1)
```
```
#Protocol Description
```

```
→replace:
7. S -> FE1 : {PublS, {complexS}{secS}}{keyAlg01}
8. FE1 -> FE2 : {PublFE1, {complexFE1}{secFE1}}{keyAlg01}
9. FE2 -> C : {PublFE2, {complexFE2}{secFE2}}{keyAlg01}

by:
7. S -> FE1 : {{PublS,
{BS,PrivS,H(BS),H(PublS)}{keyLevel0}}{secS}}{keyAlg01}
8. FE1 -> FE2 :{{PublFE1,
{BFE1,PrivFE1,H(BFE1),H(PublFE1)}{keyLevel0}}{secFE1}}{keyAlg01}
9. FE2 -> C : {{PublFE2,
{BFE2,PrivFE2,H(BFE2),H(PublFE2)}{keyLevel0}}{secFE2}}{keyAlg01}
```

```
#Specification
```

```
#Actual variables
→replace:
AVcomplexS, AVPublS, AVcomplexFE1, AVPublFE1, AVcomplexFE2, AVPublFE2 :
Data

by:
AVPublS, AVPublFE1, AVPublFE2, AVPrivS, AVPrivFE1, AVPrivFE2, AVBS,
AVBFE1, AVBFE2 : Data
```

```
#Functions
```

```
#System
→replace:
SENDER(AVS, AVkeyAlg01, AVcover1, AVcomplexS, AVPublS, AVsecS,
AVkeyLevel0, AVkeyLevel1)
FORWARDER(AVFE1, AVkeyAlg01, AVcomplexFE1, AVPublFE1, AVsecFE1)
FORWARDER(AVFE2, AVkeyAlg01, AVcomplexFE2, AVPublFE2, AVsecFE2)

by:
SENDER(AVS, AVkeyAlg01, AVcover1, AVPrivS, AVBS, AVPublS, AVsecS,
AVkeyLevel0, AVkeyLevel1)
FORWARDERA(AVFE1, AVkeyAlg01, AVPrivFE1, AVBFE1, AVPublFE1, AVsecFE1,
AVkeyLevel0, AVkeyLevel1)
FORWARDERB(AVFE2, AVkeyAlg01, AVPrivFE2, AVBFE2, AVPublFE2, AVsecFE2,
AVkeyLevel0, AVkeyLevel1)
```

```
#Intruder Information
```

In a third manual adaptation step the security aspects (in the ***#Specifications*** block) to be verified as well as the intruder knowledge (***#Intruder Information*** block) must be manually specified. For the confidentiality we specify the original biometric data as well as the private payload as a secret between the sensor ***S*** and the forensic expert ***FE1*** as: ***Secret(S, PrivS, [FE1,FE2,C])***

For the entity-authenticity verification we include the agreement of ***S*** and ***FE1*** on the original image (original biometric data ***B***) as well as the secret payload ***PrivS*** stored in the image:

Agreement(S,FE1,[BS,PrivS])

The intruder knowledge is specified according to Kerckhoffs' law giving the intruder access to all information which is not explicitly stated as being secret, i.e. everything except the secret keys. Therefore in this example nothing has to be changed for the intruder information, since it is already specified by the translation demonstrator in this way.

3.3.1.6. CDSC - Scenario – Compilation into CSP (step 5), checking with FDR (step 6) and protocol selection (step 7)

The generated CASPER-code is compiled into CSP in step 5 and checked with FDR in step 6.
For being usable in a forensic application scenario our proposed watermarking protocol needs to ensure the privacy protection, preserve the chain of custody and enable reproducibility. For achieving these goals we need to verify the data-origin authenticity, entity-authenticity, integrity and confidentiality of the data and enable reproducibility. Since the data-origin-authenticity and integrity of the underlying watermarking scheme of Dittmann et al. was mathematically proven in [Dit04] and the reproducibility is enabled by storing the applied changes in the private payload area D_{priv} (as described in the beginning of section 3.3.1), the goal of this section is to check the entity-authenticity and confidentiality of the protocol to complete the verification of all needed requirements.
Checking the modeled watermarking protocol quickly shows the limitations of CASPER/FDR concerning the complexity of the networks to be verified. Verifying the CASPER model of our scenario including all three tasks (*S→FE1*, *FE1→FE2*, *FE2→C*) fails with a physical memory size of 3GB not being sufficient main memory. This shows the huge amount of resources needed for the verification of only moderate-sized protocols. We avoid this problem by checking the three tasks separately.
The results of the model-checking for the first task (*S→FE1*) can be seen below:

```
Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T=SECRET_M::SYSTEM_S_SEQ
No attack found

Checking assertion
AUTH1_M::AuthenticateSENDERToPROCESSOR1Agreement_AVBS_AVPrivS
[T=AUTH1_M::SYSTEM_1
Attack found:

Top level trace:
FEone believes (s)he has completed a run of the protocol, taking role
PROCESSOR1, with Scanner, using data items AVBS, AVPrivS
```

The model-checking of our proposed protocol shows no possible attacks on the confidentiality, neither for the original biometric fingerprint data nor for the private payload. For the entity-authenticity a man-in-the-middle attack is found with an intruder claiming to the sensor ***S*** to be the forensic expert ***FE1*** forwarding the image to the real ***FE1*** claiming to him to be ***S***. With such an attack a possible intruder could obtain the cover image as a whole and delete, delay or later replay it. However, since the data-origin-authenticity as well as the integrity of the data can be assured as was manually shown in [Dit04], an intruder is unable to secretly change any part of the data which may eventually reach the intended receiver in the correct form.

For our fingerprint forensics application CDSC scenario this would mean that an attacker could still intercept an image of biometric fingerprints captured at a crime scene and replay it later in another, different investigation. In order to prevent such kind of attack a timestamp can be integrated into the private data section of the watermark, telling the receiver if the image is delayed for a substantial amount of time. In such a scenario a man-in-the-middle-attack is of no concern as long as the data is not delayed longer then a certain amount of time (analog to the transfer of data over an insecure channel). Unfortunately, at the moment CASPER does not offer the possibility to model such kind of scenario.

3.3.1.7. CDSC - Scenario – Summary

Within our work on the CDSC we extended the theoretical, reversible watermarking scheme from [Dit04] to be usable in a forensic application scenario, exemplary using high resolution biometric fingerprint data. We modeled a watermarking protocol based on this scheme to assure privacy protection and chain of custody preservation of the biometric data by verifying data-origin-authenticity, entity-authenticity, integrity and confidentiality of the protocol in a hybrid approach using manual as well as semi-automated verification techniques. We furthermore proposed a way to enable reproducibility and hierarchical two-level access to the data to enable case specific access restrictions to potential evidence in this forensics oriented application scenario.
During the realization of this example we noticed that there are severe restrictions to the semi-automated verification approach using CASPER and FDR, from a resource point of view, regarding the message modeling, as well as considering the syntax provided by CASPER (which in its original focus was only designed for the verification of cryptographic protocols and therefore naturally performs not perfectly in the watermarking protocol domain).

3.3.2. HAAI - Scenario implementation

The goal of this second complex watermarking-based application scenario is the design of a **Hierarchical Access, Authentication & Integrity (HAAI)** watermarking protocol that combines a multi-level access structure and the assurance of the security aspects of confidentiality, authenticity and integrity.

The **communication setup** for the HAAI application scenario foresees three core communication entities: the users/clients *u*, a central delivery infrastructure *D* and a management component *M*.
The roles of these entities are the following: the users *u* perform watermarking-based multi-level data access and modifications to a cover object *o*. The central delivery infrastructure *D* stores *o* and performs integrity verification. All data communication in the application scenario is performed via communication with *D*, i.e. no direct end-user to end-user communication is allowed. The management component *M* handles the key management and user-to-group (access level L_i) binding for the multi-level access.

Due to the problems mentioned in section 1.5 for the re-watermarking and composite watermarking approaches, we chose here the segmented watermarking approach for the implementation of our watermarking protocol. The corresponding **message layout** for the watermarking message is shown in figure 22.

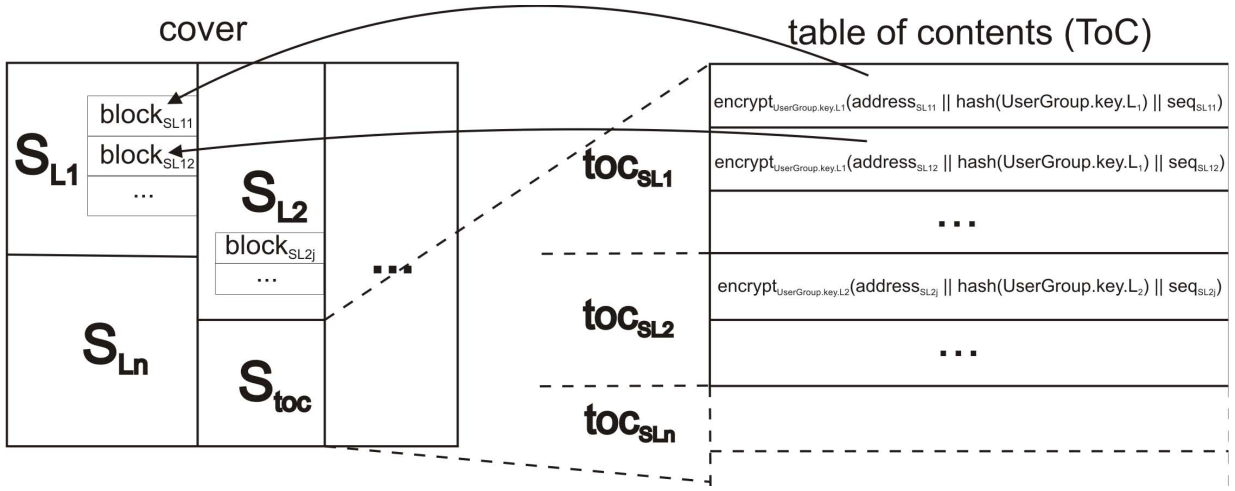

cover
table of contents (ToC)
S_{L1}
block_{SL11}
block_{SL12}
...
S_{L2}
block_{SL2j}
...
S_{Ln}
S_{toc}
...
toc_{SL1}
toc_{SL2}
toc_{SLn}
$\text{encrypt}_{\text{UserGroup.key.L1}}(\text{address}_{SL11} \,||\, \text{hash(UserGroup.key.}L_1) \,||\, \text{seq}_{SL11})$
$\text{encrypt}_{\text{UserGroup.key.L1}}(\text{address}_{SL12} \,||\, \text{hash(UserGroup.key.}L_1) \,||\, \text{seq}_{SL12})$
...
$\text{encrypt}_{\text{UserGroup.key.L2}}(\text{address}_{SL2j} \,||\, \text{hash(UserGroup.key.}L_2) \,||\, \text{seq}_{SL2j})$
...

Figure 22: Exemplary watermark structure incl. the structure of the embedded ToC

The segmented watermarking approach for our proposed watermarking protocol works with a segmentation of the cover into separate embedding segments S_1 – S_n and S_{toc}. Each of the segments S_1 – S_n represents one of n access levels and is further divided into j blocks of random size and with a randomly distributed address $address_{SLji}$ somewhere within the cover. The addresses $address_{SLji}$ belonging to the different access levels L_i are stored in a table of contents (ToC) entry, along with the sequence number seq_{SLi} of the addressed block and a hash h over the group key $Group.key.L_i$ of this certain access level L_i. Such an entry is then encrypted using the group key $Group.key.L_i$ and finally stored into a global table of contents data structure for the considered cover, which is then embedded into the ToC-segment S_{toc}.

Several **assumptions** have to be made before describing the actual functions of the presented system: First, the actual functions described here may be more complex than presented, but are simplified to their core functionality. Second, all primitives which provide a secure network communication in general are considered to be present, i.e. a mutual authentication between both communication partners has to be performed at every step and a secure protocol for the key exchange is used prior to every other action.
The application scenario consists of four **primary system functions**: *init*(), *requestinit*(), *requestObject*() and *returnObject*(). These primary functions are described below.

Function	Description
init(*o*, *n*)	Initialization of an object *o* with *n* predefined access levels by the delivery infrastructure *D*.
Internal operations: 1. *D*: *register*(*o*) returns the unique identifier ID_o of *o* 2. *D*: *segment*(ID_o, *n*) returns a sequence of *n*+1 segments S_{L1}, …, S_{Ln}, S_{toc} 3. *D*: *generateToC*(ID_o, S_{L1}, …, S_{Ln}, S_{toc}) returns the data object ID_o.*ToC* 4. *D*: initialization of the table of contents: for every block $block_{SLi,j}$ in each of the *n* segments S_{L1}, …, S_{Ln} the function ID_o.*ToC*.*init*(*encrypt*().*id*, *decrypt*().*id*, *hash*().*id*, *Init.key*.L_i) writes the initialization value $encrypt_{Init.key.Li}$($address_{SLi,j}$ || *hash*(*Init.key*.L_i) || *random*) with *random*.*size*() = *seq*.*size*() 5. *D*: ID_o.*updateHashes*() returns ID_o.*Hashes* = h_{SL1}, …, h_{SLn} + h_{tocSL1}, …, h_{tocSLn}, + ph_o *with* h_{SLi} = *hash*(&S_{SLi}) , h_{tocSLi} = *hash*(&toc_{SLi}) and ph_o = perceptual hash of *o* 6. *D*: *store*(ID_o, *o*, ID_o.*Hashes*, *u*)	

With the initialization function *init*(*o*,*n*) the delivery infrastructure *D* in the first operation registers a new object *o* to its repository. In the second operation of the initialization the object is segmented into *n+1* segments (*n* segments for the embedding areas for the *n* access levels and one segment for the embedding of the table of contents object $ID_o.ToC$). The third operation generates the ToC object $ID_o.ToC$. The fourth operation performs the ToC initialization in which the basic structure of the ToC and thereby the complete watermark message structure which was presented in figure 2 is created. In the fifth operation, the hash values which ensure the integrity of the ToC (addressed for each access level by $\&toc_{SLi}$) and the embedded data on all segments (addressed for each access level by $\&S_{SLi}$) and a perceptual hash ph_o of *o* are calculated. Finally, in the sixth operation, the cover object with the ToC embedded is stored along with the corresponding hash values from operation 5.

Function	**Description**
requestInit(*o*)	Initialization request by a client for a previously unregistered object *o* by user *u* with level L_i.
Internal operations: 1. *u*: *sendToD*(*request.Init*, *o*) 2. *D*: *init*(*o*, *n*) 3. *D*: if *access.policy* = *single_access_only* then *o.accessAllowed*(*false*) 4. *D*: *sendToUser*(ID_o, *o*, *u*)	

With the *requestInit*(*o*) function an user *u* with level L_i requests the registration of an object *o* into the repository of the delivery infrastructure *D*. After an authentication of *u* at *D* and the transmission of *o* from *u* together with an initialization request, *D* invokes internally the *init*(*o*, *n*) function. Depending on the access policy an access semaphore (or access counter; managed by *o.accessAllowed*()) can be set (or increased) to limit the number of the concurrent accesses to one object. Finally, the registered *o* together with its assigned system-wide object-ID ID_o is returned to *u*.

Function c	**Description**
requestObject(ID_o)	Client request for an object with the identifier ID_o.
Internal operations: 1. *u*: *sendToD*(ID_o) 2. *D*: retrieve *o* belonging to ID_o from internal repository 3. *D*: if *access.policy* = *single_access_only* then check *o.accessAllowed*(). If *o.accessAllowed*() = *true* then *o.accessAllowed*(*false*) and *sendToUser*(ID_o, *o*, *u*), else inform *u* that *o* can not be accessed right now due to an access-lock	

The request for a registered object *o* from a user *u* uses the system-wide object-ID ID_o which is generated by *D* for each object upon registration. Depending on the access policy an access semaphore can prevent the access if the object is currently processed by another user.

Function	**Description**
returnObject(o_{new}, ID_o)	Client commit of an (updated) object o_{new}
1. *u*: *sendToD*(ID_o, o_{new}) 2. *D*: performs an authenticity verification of the committed object o_{new} against its stored version *o* using *isAlteredVersion*(o_{new}, *o*) if the result is *false*: • committed object is no instance of object o → authenticity violation if the result is *true*: • *checkHashes*(o_{new}, L_i) if the result is *false*: o integrity verification failed → integrity violation	

• if the result is *true*: o ID_o.*updateHashes*() o *store*(ID_o, o_{new}, ID_o.*Hashes*, *u*) o *o.accessAllowed*(*true*)

If a user ***u*** returns an object ***o*** after processing it to the delivery infrastructure *D*, an integrity verification (*checkHashes*()) and an authenticity verification (*isAlteredVersion*()) of the new version of the object o_{new} are performed if the object was modified (in case the object was just requested by ***u*** for reading information without any modifying access, ***o*** and o_{new} are identical and the integrity therefore does not have to be verified further). The function *isAlteredVersion*() simply calculates a perceptual hash of o_{new} and compares it to the perceptual hash ph_o of ***o***, which is stored in ID_o.*Hashes* at *D*. Based on the concept of the application scenario, *D* only verifies whether ***u*** has performed modifying access in the segments which belong to ***u***'s access level L_i. If ***u*** modified any other part of o_{new}, *D* signals an integrity violation for that part. If ***u***, as intended by the scheme, just modified the segments belonging to its access level L_i, *D* updates the hashes stored for the object and stores o_{new} into its repository as the new version of ***o***. Depending on the access policy a set access semaphore has to be unset.

The four primary system functions are accompanied by a set of **secondary system functions**. The two most important of these secondary functions are: *checkHashes*() and *updateHashes*().

Function	**Description**
checkHashes(o_{new}, L_i)	Hash verification for one level of the object-version o_{new} submitted to *D* from a user *u* with access level L_i.
1. *D*: Verification of the hashes h_{tocSLi}: • For all ToC segments toc_{SL1},..., toc_{SLn}: computation of the hash value h_i over the complete ToC segment toc_{SLi} • Comparison of the computed h_i with h_{tocSLi} retrieved from ID_o.*Hashes* • if $h_i \neq h_{tocSLi}$ and $i \neq L_i$ of *u*, then *u* modified table-of-content entries which do not belong to his access level □ integrity of *o* is violated 2. *D*: Verification of the hashes h_{SLi}: • For all segments S_{L1}, …, S_{Ln}: computation of the hash value h_i over the complete contents of the segment ($\&S_{Li}$) • Comparison of the computed h_i with h_{SLi} retrieved from ID_o.*Hashes* • if $h_i \neq h_{SLi}$ and $i \neq L_i$ of *u*, then *u* wrote into segments which do not belong to his access level □ integrity of *o* is violated 3. *D*: *checkViolation*(): if one of the two hash verifications implies an integrity violation, then: • the at *D* stored version of *o* is not replaced by o_{new}; o_{new} is discarded • otherwise: ID_o.*updateHashes*()	

The application scenario foresees two integrity verification mechanisms to be used by the delivery infrastructure *D*. They are: a set of hashes h_{tocSLi} protecting the table of contents entries for each hierarchy level L_i, a set of hashes h_{SLi} protecting the data segments for each hierarchy level L_i and a perceptual Hash ph_o, for later authenticity checks of instances of ***o***. Both sets of hashes (h_{tocSLi} and h_{SLi}) along with ph_o are stored in ID_o.*Hashes* at the storage space *D*.

Function	**Description**
updateHashes()	Hash update for object o by delivery infrastructure *D* (e.g. after commit of a modified version of the object by a user).
1. *D*: write hashes h_{tocSLi} to ID_o.*Hashes*: • for all segments toc_{SL1}, …, toc_{SLn}: computation of h_{tocSLi} for the complete ToC entries for the segment S_{Li}	

2. *D*: write hashes h_{SLi} to ID_o.*Hashes*:
• for all segments S_{L1}, …, S_{Ln}: computation of h_{Sli} over $\&S_{Li}$, i.e. the content of segment S_{Li}

This function, which is integral part of the *returnObject*() function described above assumes that a integrity verification has been performed already by *D* using *checkHashes*(). Additional secondary system functions are: *sendToD*(), *sendToUser*(), *getEmbeddingPositions*(), *requestUserLevel*() etc. which essentially perform what their identifiers indicate. Further explanation of their functionality is not necessary for the purpose of our application scenario example.

To show how all these protocol functions interact, an **end-user object access example** is presented in the following. It assumes that a user *u* with access level L_i requests object *o* known to him by the corresponding object identifier ID_o. This example assumes that: the required key exchanges for setup of the application scenario have been performed, that there exists a way (e.g. a central look-up directory) for *u* to learn ID_o, that *u* knows how to address *D* and that *u* knows the ToC embedding address or is capable of determining or receiving it. Furthermore it is assumed, from the operational point of view, that each communication operation is preceded by mutual authentication of the communicating entities.

Operation 1: *u*: *requestObject*(ID_o)
Operation 2: *u*: *getEmbeddingPositions*(ID_o, *o*, *key*):
- this might require two runs of *getEmbeddingPositions*():
  - first, with *key* = *Init.key.*L_i ; if this run fails (i.e. *h* = *hash*(*Init.key.*L_i) cannot be found in any entry) because an other user with the same access level L_i (and therefore group key) has already embed data on L_i and therefore encrypted all entries with *UserGroup.key.*L_i then:
  - second, with *key* = *UserGroup.key.*L_i
- Identification of the segment S_{toc} using ID_o.*ToC.address*
- entry-wise processing of the ToC and decryption of each entry, the result of this decryption has the form: *address* || *h* || *seq*
- if *h* = *hash*(*key*) then *address* is a legitimate embedding address for access level L_i
- return: *meta* = indexes of all ToC-lines and all embedding addresses for L_i

Operation 3: *u*: *embed*(*o*, *data*, *meta*, *UserGroup.key.*L_i):
- *enc_data* = $encrypt_{UserGroup.key.Li}$(*data* || *hash*(*data*))
- the encrypted data *enc_data* is then split into blocks, which are embedded into the segments identified by the addresses $address_{SLi1}$, …, $address_{SLij}$

Operation 4: u: *returnObject*(o_{new}, ID_o)
Operation 5: *D*: authenticity and integrity verification and updating of the hashes h_{SLi} und h_{tocSLi}:
- *isAlteredVersion*(o_{new} , *o*)
- *checkHashes*(o_{new} ,L_i)

The user *u* requests an object and waits for *D* to send the designated object. After determining all ToC entries for the user's level, the embedding positions for the level's segment are known to *u.* Before using the found addresses, the data to be embedded has to be preprocessed in two operations. In the first operation a hash is calculated over the data, which is then concatenated with this hash and encrypted with *UserGroup.key.*L_i. The second operation is a segmentation of the encrypted data into blocks which fit the block-size of the embedding segment. The last action performed on the object is the embedding itself - the changed ToC and the encrypted data are embedded to the assigned segments. After the changed version of *o* has been returned to *D*, it performs all necessary authenticity and integrity checks, including the check of the perceptual hash ph_o in *isAlteredVersion*() and the check of the segment and ToC hash values in *checkHashes*().

Summarizing the addressed security aspects, the end-user object access example given above shows very well the **integrity verification mechanisms** applied by the central delivery infrastructure *D* and the end-user *u*. Each of them embeds its own hash values in places where they are protected from access by others. The delivery infrastructure *D* uses its storage space for copies of its encrypted hashes h_{tocSLi} and h_{SLi}, the original ToC and the original (or last integer instance of the) object. The user appends its hash to the plaintext he wishes to embed and encrypts both. As long as the keys of the other parties are unknown, no participating entity can modify the object outside its allowed embedding areas without violating the integrity of the object, which will at the next access be automatically detected by the other entities. The important point here is therefore the separation of the central delivery infrastructure *D*, whose sole purpose is the data storage, from the management component *M*, which performs the key management and user-to-group (access level L_i) binding. The only point where this separation of duty principle seems to be violated is the initialization of *o* at *D*, where all ToC entries (and with that all potential embedding addresses for all levels) are generated and encrypted using the $Init.key.L_i$ generated by *D*. This problem is solved when a user first embeds data to his access segment and in this process changes the key used for encryption and optionally also the sequence of their own ToC entries to $UserGroup.key.L_i$.
The second security aspect covered inherently by the introduced scheme is the **data confidentiality -** *D* does not gain access to user data and the different user access levels are also protected from each other. The third security aspect covered in the design of the watermarking scheme is the **object authenticity** using the perceptual hash ph_o of the cover object *o* to prevent e.g. watermark copy attacks.
Using with an external audit service *A* one further entity to the communication setup, the security aspect of **non-repudiation** could be easily implemented as an extension of the scheme by time-stamping and system-wide, secure logging.
What is **still missing** in this application scenario so far **is an evaluation of the security aspects** of **communication confidentiality** (no attacker is able to get access to the protected data communicated by the protocol) and **entity-authenticity** in the protocol design. These evaluations are given in the following sections using the methodology and framework summarized in section 2.2.

3.3.2.1. HAAI - Scenario – Network and Task description (step 1)

The first step in the application of our framework is again the modeling of the involved network entities, watermarking algorithms and the task to fulfill. For the application example in this paper we choose a network with three users, of whom two have the same access level. As CASPER doesn't support numbers as identifiers, these users are labeled as ***UAA***, ***UAB*** and ***UB*** of which ***UAA*** and ***UAB*** share the same access level. The task is modeled as that the cover is transferred to ***UAA*** first, who edits it, then to ***UB***, who edits it and finally to ***UAB***. As mentioned above, all those transfers are performed through the central delivery infrastructure ***D.***

The XML context modeling of this contextualization is done by using the authoring demonstrator tool. Screenshots illustrating the usage of this tool in this application scenario as well as the resulting XML files are presented below.

- Network description:

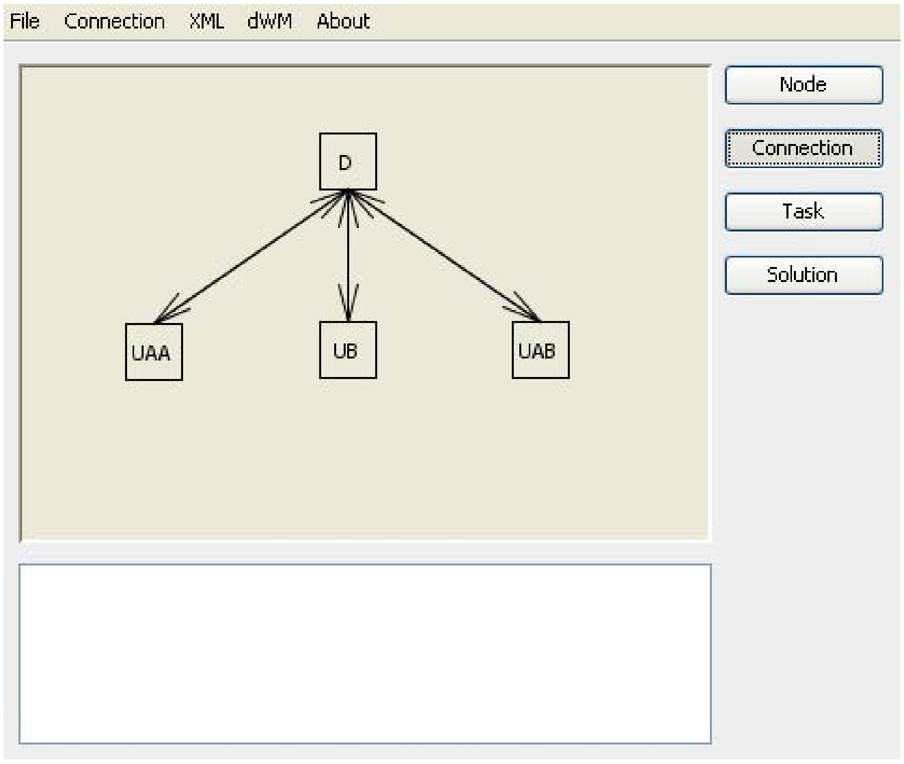


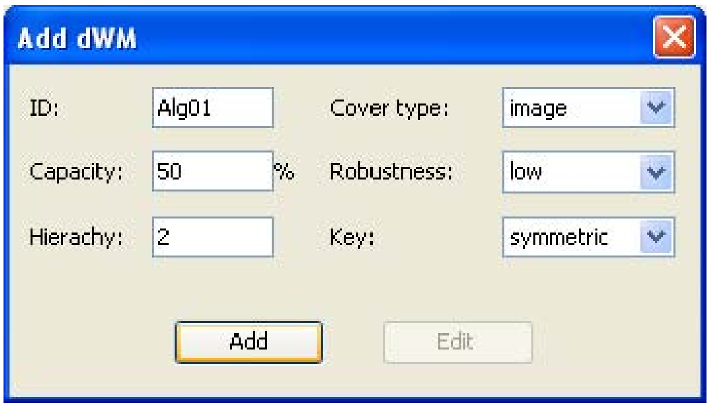
Figure 23: Network description for our HAAI scenario implementation

The network consists of the four defined nodes `D`, `UAA`, `UAB` and `UB`. Between `D` and all other three nodes exists a bi-directional connection modeled as two uni-directional connections. In the context of this scenario direct connections between `UAA`, `UAB` and `UB` are not allowed. All nodes have access to an image cover-channel and a corresponding watermarking algorithm `Alg01`.

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
      <nodes>
            <node>
                  <id>UAA</id>
                  <cover>
                        <cc>
                              <type>image</type>
                              <chancap>10000</chancap>
                        </cc>
                  </cover>
                  <dwm>Alg01</dwm>
                  <nodecap>10000</nodecap>
                  <pki>y</pki>
                  <ts>n</ts>
            </node>
            <node>
                  <id>D</id>
                  <cover>
                        <cc>
                              <type>image</type>
                              <chancap>10000</chancap>
                        </cc>
                  </cover>
                  <dwm>Alg01</dwm>
                  <nodecap>10000</nodecap>
                  <pki>y</pki>
                  <ts>n</ts>
            </node>
            ...
      </nodes>
      <lc>
            <connection>
                  <src>D</src>
                  <dst>UAA</dst>
            </connection>
            <connection>
                  <src>UAA</src>
                  <dst>D</dst>
            </connection>
      ...
      </lc>
</network>
```

- Algorithm description:

For the watermarking algorithm description a symmetric scheme with two hierarchy levels is defined.



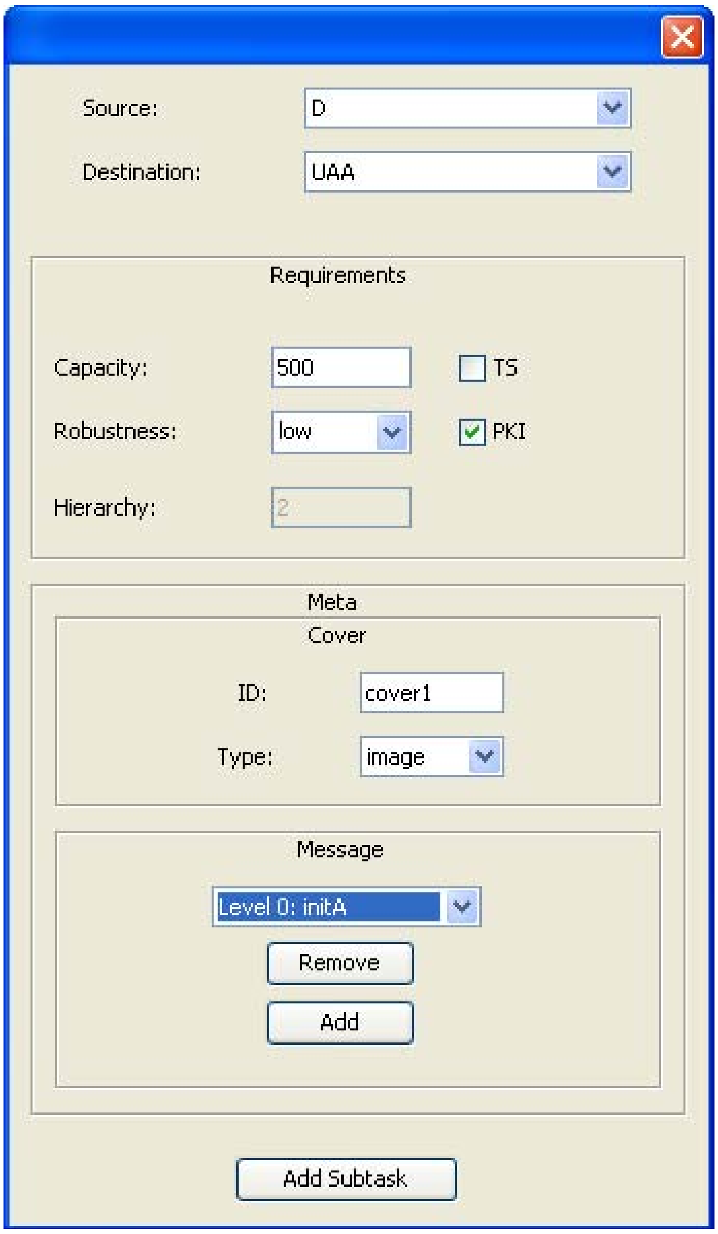
ID:	Alg01	Cover type:	image
Capacity:	50 %	Robustness:	low
Hierachy:	2	Key:	symmetric

Figure 24: Algorithm description in the context modeling demonstrator

```
<?xml version="1.0" encoding="UTF-8"?>
<dwms>
      <dwm>
            <id>Alg01</id>
            <cover>image</cover>
            <dwmcap>50</dwmcap>
            <robustness>low</robustness>
            <hierarchy>2</hierarchy>
            <key>symmetric</key>
      </dwm>
</dwms>
```

- Task description:

In the task description a data transfer along the node sequence ***D→UAA→D→UB→D→UAB*** is defined, with ***D*** being the source in the network task and ***UAB*** being the destination. For the whole path two hierarchy levels are required (one labeled “***A***” for ***UAA*** and ***UAB*** and the second labeled “***B***” for ***UB***). To simulate the initialization performed by ***D*** another two hierarchy levels are added.


Source: D

Destination: UAA

Requirements

Capacity: 500

[ ] TS

Robustness: low

[✓] PKI

Hierarchy: 2

Meta

Cover

ID: cover1

Type: image

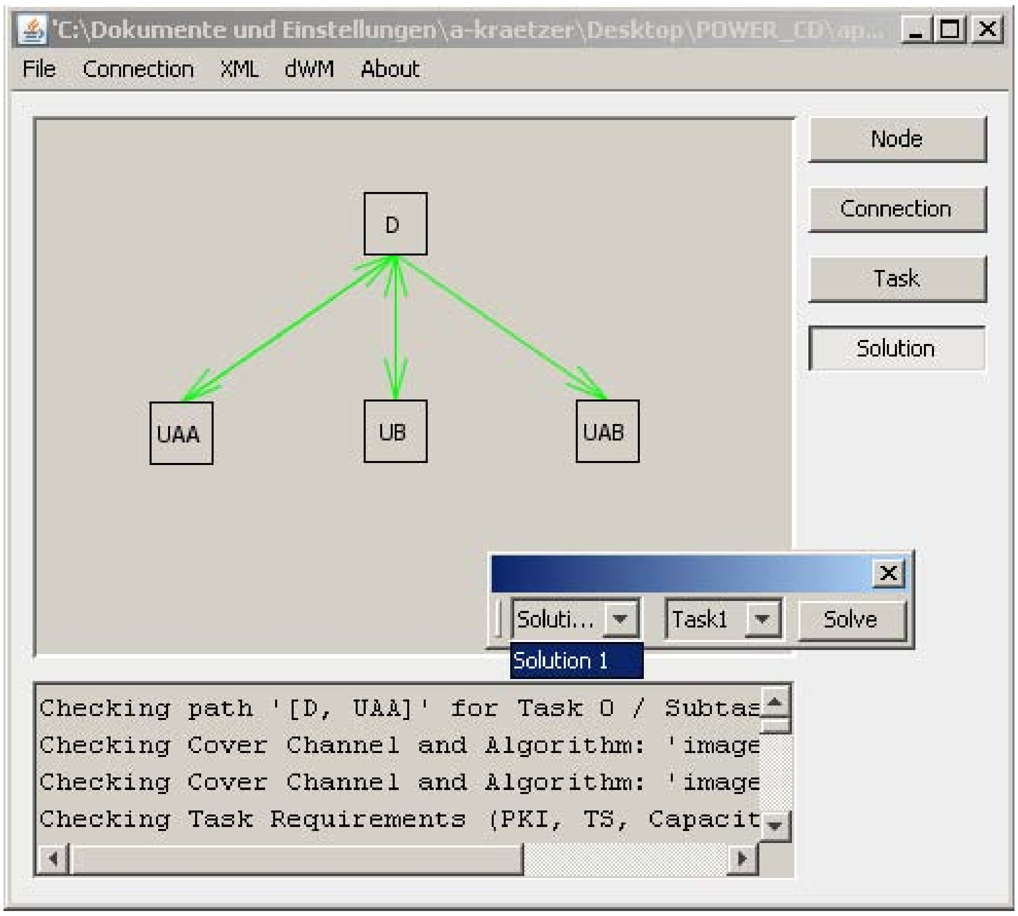
Message

Level 0: initA

Figure 25: Task description in the context modeling demonstrator

3.3.2.2. HAAI - Scenario – Path search (step 2)

In the second step of our framework, the path search described in section 2.2.1.1 is invoked within the authoring tool. The result of this path search looks for this basic application scenario as shown in figure 26. The corresponding project file for this example is found on the CD. Normally, the user would be able to browse the different alternative solutions identified by the authoring tool demonstrator, but this first small application scenario allows results in only one alternative solution.



```
Checking path '[D, UAA]' for Task 0 / Subtas
Checking Cover Channel and Algorithm: 'image
Checking Cover Channel and Algorithm: 'image
Checking Task Requirements (PKI, TS, Capacit
```

Figure 26: Output of the task search for the HAAI application scenario in the POWER authoring demonstrator tool

3.3.2.3. HAAI - Scenario – Path selection (step 3)

As for the CDSC application scenario here the path selection is trivial, because only one alternative path exists.

3.3.2.4. HAAI - Scenario – Automated CASPER modeling using XML-input (step 4a)

The chosen solution block from section 3.3.2.3 is used to generate a CASPER protocol for further verification of desired security aspects in step 4 of the framework. These translations are applied here to the solution block of the HAAI application scenario and yield the following CASPER-code:

```
#Free variables
D, UAA, UB, UAB : Agent
cover1 : Cover
ToCA, OA, ToCB, OB : Data
keyLevel0, keyLevel1, keyLevel2, keyLevel3, keyAlg01 : SharedKey
InverseKeys = (keyAlg01, keyAlg01), (keyLevel0, keyLevel0), (keyLevel1,
keyLevel1), (keyLevel2, keyLevel2), (keyLevel3, keyLevel3)
#Processes
SENDER(D, keyAlg01, cover1, ToCA, OA, ToCB, OB, keyLevel0, keyLevel2)
FORWARDERA(UAA, keyAlg01, keyLevel0, keyLevel1)
FORWARDERB(UB, keyAlg01, keyLevel2, keyLevel3)
RECEIVER(UAB, keyAlg01, keyLevel0, keyLevel1)
#Protocol description
0. -> D : UAA
```

```
1. -> UAA : D
2. -> D : UB
3. -> UB : D
4. -> D : UAB
5. D -> UAA : {{ToCA}{keyLevel0}, {OA}{keyLevel0},
{ToCB}{keyLevel2}%vTocB, {OB}{keyLevel2}%vOB}{keyAlg01}
6. UAA -> D : {{ToCA}{keyLevel1}%vToCA, {OA}{keyLevel1}%vOA,
vTocB%{ToCB}{keyLevel2}, vOB%{OB}{keyLevel2}}{keyAlg01}
7. D -> UB : {vToCA%{ToCA}{keyLevel1}%vToCA, vOA%{OA}{keyLevel1}%vOA,
{ToCB}{keyLevel2}, {OB}{keyLevel2}}{keyAlg01}
8. UB -> D : {vToCA%{ToCA}{keyLevel1}%vToCA, vOA%{OA}{keyLevel1}%vOA},
{ToCB}{keyLevel3}%vToCB, {OB}{keyLevel3}%vOB}{keyAlg01}
9. D -> UAB : {vToCA%{ToCA}{keyLevel1}, vOA%{OA}{keyLevel1},
vToCB%{ToCB}{keyLevel1}, vOB%{OB}{keyLevel1}}{keyAlg01}
#Specification
#Actual variables
AVD, AVUAA, AVUB, AVUAB, Mallory : Agent
AVcover1 : Cover
AVToCA, AVOA, AVToCB, AVOB : Data
AVkeyLevel0, AVkeyLevel1, AVkeyLevel2, AVkeyLevel3, AVkeyAlg01 : SharedKey
InverseKeys = (AVkeyAlg01, AVkeyAlg01), (AVkeyLevel0, AVkeyLevel0),
(AVkeyLevel1, AVkeyLevel1), (AVkeyLevel2, AVkeyLevel2), (AVkeyLevel3,
AVkeyLevel3)
#Functions
#System
SENDER(AVD, AVkeyAlg01, AVcover1, AVToCA, AVOA, AVToCB, AVOB, AVkeyLevel0,
AVkeyLevel2)
FORWARDERA(AVUAA, AVkeyAlg01, AVkeyLevel0, AVkeyLevel1)
FORWARDERB(AVUB, AVkeyAlg01, AVkeyLevel2, AVkeyLevel3)
RECEIVER(AVUAB, AVkeyAlg01, AVkeyLevel0, AVkeyLevel1)
#Intruder Information
Intruder = Mallory
IntruderKnowledge = {AVD, AVUAA, AVUB, AVUAB}
```

This protocol, automatically generated from a XML context model, represents the solution for the task at hand. In the ***#Free variables*** section the CASPER ***Agents*** are directly taken from the nodes in the XML solutions block, while the `Objects` represent the data stored in each access level. The `Toc` objects represent the corresponding ToC entries for the access levels (***A*** and ***B***). Also, the keys used in the protocol and their linkage are specified. The ***#Processes*** section represents the actors and their inherent knowledge. It is assumed here that the communicating nodes already possess all the keys they should have access to in this application scenario.

Representing the data exchange, the ***#Protocol description*** section forms the core of the model. Here can be seen that first all nodes are initialized with the knowledge about which nodes they should be communicating with. The data transfer between the delivery infrastructure ***D***, and the users ***UAA***, ***UAB***, ***UB*** is simple, as everyone receives all ToCs and data objects encrypted with the corresponding keys (every first user on an access level re-encrypts the data exchanging the initialization key with the key used in the further communications on this level – hereby ***D*** is excluded from these confidential communications).

The ***#Actual variables***, ***#Functions*** and ***#System*** sections can be directly created from the precedent blocks. Finally, an ***#Intruder Information*** is automatically generated, which here is provided with the knowledge of all nodes in the network.

This result shows the systematic translation of the solution into CASPER-code but cannot be used in CASPER directly, as CASPER is unable to handle[18] the construct ***vToCA%{ToCA}{keyLevel1}%vToCA***.

3.3.2.5. HAAI - Scenario – CASPER model adjustment (step 4b)

A manual adjustment in step 4b of the framework has to take place here due to the forwarding problem identified at the end of the previous section.
Here in this application scenario we found an easy workaround to solve this problem: The idea here is to not include this relayed message after it has been transferred once. This doesn't influence the knowledge of a malicious attacker since it doesn't matter if he obtains this message in the fifth protocol step, where it is initially send or in the sixth protocol step. So each block is only send when the receiver is a User able to alter the block or when the block has not been send since it has changed the last time.
The required adaptation looks as follows:

```
#Free variables
#Processes
#Protocol description
→Replace :
6. UAA -> D : {{ToCA}{keyLevel1}%vToCA, {OA}{keyLevel1}%vOA,
vTocB%{ToCB}{keyLevel2}, vOB%{OB}{keyLevel2}}{keyAlg01}
7. D -> UB : {vToCA%{ToCA}{keyLevel1}%vToCA, vOA%{OA}{keyLevel1}%vOA,
{ToCB}{keyLevel2}, {OB}{keyLevel2}}{keyAlg01}
8. UB -> D : {vToCA%{ToCA}{keyLevel1}%vToCA, vOA%{OA}{keyLevel1}%vOA},
{ToCB}{keyLevel3}%vToCB, {OB}{keyLevel3}%vOB}{keyAlg01}
9. D -> UAB : {vToCA%{ToCA}{keyLevel1}, vOA%{OA}{keyLevel1},
vToCB%{ToCB}{keyLevel1}, vOB%{OB}{keyLevel1}}{keyAlg01}

by :
6. UAA -> D : {{ToCA}{keyLevel1}%vToCA, {OA}{keyLevel1}%vOA}{keyAlg01}
7. D -> UB : {{ToCB}{keyLevel2}, {OB}{keyLevel2}}{keyAlg01}
8. UB -> D : {{ToCB}{keyLevel3}%vTocB, {OB}{keyLevel3}%vOB}{keyAlg01}
9. D -> UAB : {vToCA%{ToCA}{keyLevel1}, vOA%{OA}{keyLevel1}}{keyAlg01}

#Actual variables
#Functions
#System
#Intruder Information
```

To verify the desired security aspects of confidentiality and entity-authenticity an own CASPER ***#Specification*** block must be formulated for each of the two security aspects. As we plan to verify confidentiality with this protocol we need to check that only certain nodes have knowledge of certain messages. In this case both clients (***UAA*** and ***UAB***) of access level ***A***, which share the key ***keyLevel1***, should have

[18] A scenario including more than one agent in a row forwarding encrypted data without decryption cannot be modeled (the term "***var1%expression%var2***" cannot be compiled in CASPER, preventing the modeling of data which is only forwarded and not processed for more than one step in a row in our example protocol). Here we would need this construct to model the part of the application scenario where a node is relaying a message, which he cannot decrypt himself to another node who is also unable to decrypt this message. See communication step 6 in the initial CASPER ***#Protocol description*** block given in the example, where ***D*** is forwarding the ToC and the data object of access level ***A***, which he is unable to decrypt to user ***UB*** who is also unable to decrypt it. However, as shown for this example, sometimes workarounds are possible, depending on the application scenario at hand.

knowledge about the data of their access level (***A***). Only ***UB*** should have knowledge of the corresponding data ***OB***. The ***#Specification*** for confidentiality therefore contains:
Secret(UAA, OA,[UAA,UAB]) and ***Secret(UB, OB,[UB])***

For the entity-authenticity an set of agreements is defined covering all tuples of directly communicating ***Agents*** and a data object they have to agree upon (here we use the initialization keys known to ***D*** and all users: ***Agreement(D,UAA,[keyLevel0])***, ***Agreement(D,UAB,[keyLevel0])*** and ***Agreement(D,UB,[keyLevel2])***.

3.3.2.6. HAAI - Scenario – Compilation into CSP (step 5), checking with FDR (step 6) and protocol selection (step 7)

The CASPER compiler is used in step 5 to compile the .spl protocol description files from CASPER syntax into CSP. During step 6 of the framework the assertions specified in the ***#Specification*** block (see section 3.3.1.5 above) are checked by FDR to yield information about the security aspects of confidentiality and entity-authenticity. In case of a more complex description of an application example, where multiple ways of modeling are compared, the step 7 of the framework then selects the best model for the protocol, based on the security statements made by FDR and the model complexity.

3.3.2.7. HAAI - Scenario – Summary

We here use a complex watermark-based communication application scenario that combines a multi-level access structure and the assurance of the security aspects of confidentiality, authenticity and integrity and non-repudiation. The application scenario aims at providing a watermarking-based secure central data storage and decentralized data access with full traceability of transactions and modifications as well as the requirement that data embedded by users must remain confidential for the central data storage.
Selected security aspects, namely data confidentiality, integrity and data-origin-authenticity are addressed/solved directly in the watermarking scheme. Others, namely communication confidentiality and entity-authenticity are verified for the resulting protocol using our framework for (semi-)automated protocol verification.
The non-repudiation, as a further security aspect to be covered, could easily added to this list by the inclusion of an external audit service *A* as an additional communication entity to the scenario. This step would allow us to achieve full traceability of transactions and modifications on protocol level, but not all required mechanisms[19] can be modeled in CASPER. .

3.3.3. HDSR - Scenario implementation

The description of this **Hierarchical Digital Signatures for Reproduction of Original (HDSR)** application scenario was initially given as follows: The HDSR scenario enhances the protocol by enabling the reconstruction of the original data stream sent by the card reader to allow access to clear cover data. Here also hierarchical access alternatives motivated from the HAAI scenario are further investigated to evaluate and summarize pros and cons. Furthermore the impact of an

[19] The most problematic mechanisms here would be protocol-level time-stamping and integrity verification, which can not be realized with the set of constructs provided by CASPER.

erroneous communication channel (error due to transmission errors) will be elaborated and protocol mechanisms suggested.
The intention behind the proposal of this third application scenario have been, on one hand, the core functionalities of the other two scenarios (hierarchical access and signature chains) and on the other hand the consideration of erroneous communication channels.

During our work on the first two application scenarios we noticed, that for a practical application of the CDSC scenario we already had to combine the signature chain with hierarchical access. For the second expected innovation in the HDSR scenario – the consideration of erroneous communication channels – we noticed that this can not be done in the protocol generation and verification of a formal methods based approach.
Our approach would be capable of handling this question to some extend in the context modeling by defining a robustness specific quality parameter for each cover channel and watermarking algorithm as well as corresponding requirements in the network task description. Nevertheless, it would require some form of probabilistic simulation instead of model checking to solve this on protocol level, but there exists to our best knowledge no tools in IT-security (neither for CASPER nor any other language we looked into) to realize such probabilistic simulation.
Based on these two observations we decided to discontinue our considerations regarding this HDSR scenario.

4. Dissemination and public result verification

In [Kraetzer10ACM] a first version of the exemplarily implemented practical realization of the framework was introduced on the renown ACM Multimedia and Security Workshop in September 2010 to a watermarking- and media security audience and discussed with experts on this field. The results of these discussions were, next to recommended improvements, the suggestion to submit a paper on additional findings in this context to Information Hiding 2011 (Prague) and to contact Gavin Lowe (developer of CASPER at University of Oxford, UK) and send him a copy of the paper.

Our submission to Information Hiding 2011 is right now in the review process of this conference. Regarding the discussion with Gavin Lowe, it was agreed upon that Chad Heitzenrater will approach Mr. Lowe on the subject of our research work in POWER.

Additionally to the paper submission to Information Hiding 2011 we are right now also preparing a submission for the Communications and Multimedia Security (CMS 2011) conference.

5. Summary of the results for POWER and indications for future work

In the final chapter of this report we summarize briefly our achieved project results (in section 5.1), give a summary and explanations on the limitations we encountered for our methodology, theoretical framework as well as our exemplary CASPER-based practical realization (sections 5.2 and 5.3), and give some indications how future work might extend the research effort described here.

5.1. Summary of the achieved project results

The scientific contribution of our research work can be summarized in the encompassing considerations on the **protocol life cycle** (which are new in the context of secure communication protocols) and the comprehensive considerations on the **design of communication protocols** (here summarized by the design phase of the protocol life cycle). Such a **concept for integrated context modeling, protocol generation and -verification** as it is described here, would be new even for the domain of cryptographic communication protocols[20], but for watermarking research we are significantly extending the current state-of-the-art.

With the **exemplary realization of our framework** using our own context modeling and protocol generation demonstrators, together with CASPER and FDR we implement a very first realization of the complete design process for these protocols.

The practical prospects of the introduced approach of (semi-)automated protocol modeling, generation and security verification can be summarized in its **cost-effectiveness**. In contrast to its alternative, the manual security verification of protocols, it is assumed to be faster and easier adaptable to specific application scenario requirements.

Another important outcome is the **identification of the limitations our methodology** and our theoretical framework as well as our exemplary practical realization of the framework face right now. These limitations are summarized in the following sections.

5.2. Limitations of the methodology and the theoretical framework

Following the basic idea of the formal methods approach (see section 1.1), an approach like the one considered here for POWER relies on **external (security) evaluation regarding the primitives**. This basically means that a model for the security of the used watermarking components is required. Such a model is so far mostly neglected in literature on watermarking evaluation which is in general rather focused on selected robustness attacks than the security of the algorithm.
Additionally, we need definitions for watermarking characteristics which make those characteristics directly comparable (e.g. robustness of algorithm *A1* vs. robustness of algorithm *A2*) for the automated solution determination (in step 3 of our framework). One possible set of definitions that could be used as a basis for such a scheme is

[20] It has to be mentioned here that there exist some tools which solve part of the overall task, e.g. the COSPJ compiler that produces Java implementations of protocols from CASPER-like descriptions (see: http://www.comlab.ox.ac.uk/gavin.lowe/Security/Casper/COSPJ/index.html).

found in the profile based **watermarking benchmarking** approach introduced by Lang in [Lang07]. Nevertheless, it has to be mentioned here that this task is hard for watermarking techniques, especially when there exists right now no agreed upon way to satisfactorily model important watermarking properties like robustness.

A general limitation to automated approaches for protocol verification, whether they are computational complexity-based or based on formal methods, is of course the **computational complexity** required to verify larger systems/protocols and the resulting state space.
For formal method based approaches model checking is the only well established solution method we encountered in literature. This approach has strong problems coping with the concept of **erroneous communication channels**. To address these (as was intended in the HDSR application scenario in section 3.3.3) would rather require a simulation-based solution instead of model checking. But simulation is a technique rather uncommon in security evaluations[21] because it can not prove security.

The work on the practical application work in sections 3.2 and 3.3 made us aware of the following three **major problems in our concept**:

- **Conceptualization:** In our framework description we assume that we are given neat network and task descriptions on which we can then work. In practice this is a rather unlikely situation. More often the overall task will be formulated rather fuzzy (e.g. “we want to implement a watermarking-based forensic tracking protocol for the movie industry”) and the network on which this has to be implemented will be either unknown or simply be identified as “the internet”. There is no way this setting can be completely modeled for protocol generation and -verification. Instead the overall task has to be broken down into use-cases which have to be modeled and evaluated separately. Right now no procedure is known to us to perform this task in an automatable manner.
- **Message modeling:** Especially in our CDSC application scenario we noticed that the modeling of the protocol messages is a rather problematic task, due to the fact that these messages tend to change over time (e.g. signatures of processing nodes are added). To model this changing behavior an own description language for the messages would be required and the context modeling as well as in the XML to CASPER compilation would require evaluations of the message descriptions.
- **Translation rules for the compiler:** The translation rules for the XML to CASPER compiler presented in Appendix B have been generated specifically for the three examples used within this document. So far we found no solution how to generate a universally applicable set of these translation rules which would also be applicable for more complex application scenarios. Every translator tool that is based on use-cases will be over-trained for general purpose application. This translation problem is closely related to the message modeling problem, which would have to be solved before a general purpose compiler could be implemented.

[21] Except for availability evaluations, which are much closer to the domain of safety investigations than any other security aspect.

5.3. *Additional limitations imposed by our CASPER-based realization*

The **CASPER syntax** is limited to a very small set of primitives which are mostly restricted to the area of cryptography and very difficult to adapt/convert to the domain of watermarking (for more details see Appendix A). Some examples for restrictions introduced by the CASPER syntax:

- A situation where an agent stores a hash value (or timestamp) to compare it with another hash value computed a few steps later in the protocol run can not be modeled using CASPER.
- A scenario including more than one agent in a row forwarding encrypted data without decryption cannot be modeled (the term "`var1%expression%var2`" cannot be compiled in CASPER, preventing the modeling of data which is only forwarded and not processed for more than one step in a row in our example protocol). An example for this restriction is found in section 3.3.2, where we would have needed this construct to model the part of the HAAI application scenario where a node is relaying a message, which he cannot decrypt himself to another node who is also unable to decrypt this message. However, as shown for this example, sometimes workarounds are possible, depending on the application scenario at hand.

The most important modifications to CASPER would be from our point of view:

- the introduction of a register set where values like hashes could be stored outside the sequential processing
- a possibility for the definition of data structures for the messages to be transmitted (to avoid problems like the "`var1%expression%var2`" example stated above)
- a step-independent time-stamping (maybe this can also be realized via the register set)
- a function/mechanism realizing digital signatures

Additionally, our approach would benefit from new **watermarking specific primitives in CASPER**, which would allow us to model specific characteristics like e.g. non-blind or non-invertible watermarking schemes.
Currently, only the **security aspects** of entity-authenticity and confidentiality can be modeled/verified using CASPER/FDR on **static networks** a limited number of pre-defined communicating entities and with **strictly sequential data transmissions**.
There exist language constructs and mechanisms in CASPER which also aim at allowing for integrity verification (hashes), but these mechanisms are not working satisfactorily. Also data-origin-authenticity can not be verified using CASPER[22].

[22] To provide evidence a counter-example is given here, showing that data-origin-authenticity cannot be verified using the given approach. In the counter-example scenario an agent *X* sends an unencrypted message *m* to an agent *Y* who forwards it to an Agent *Z*. Lets assume the integrity of such an scenario could be modeled by giving the agents *X* and *Z* predefined knowledge of each other and specifying an agreement between *X* and *Z* on the message *m*. In this case trivially an attack should be found by the model-checker FDR representing a man-in-the-middle attack with an intruder claiming to be node *Y* (which is unknown to *X* and *Z* before the protocol run). However, the results of an FDR run on such an example show that no attack was found, therefore counter-proving this case.
The reason for this behavior seems to be the fact that CASPER is designed to check the authenticity of two nodes exchanging certain messages. Once this authenticity is given, fulfillment of the

Similar to integrity verification mechanisms, CASPER contains language constructs for time-stamping but these language constructs can not be used satisfactory in protocol design for our intended field of applications.
If the result of the security evaluation by FDR is negative (possible attacks have been found) the output of the verification process can be used to **harden the protocol**. So far no methods for the realization (manually or (semi-)automatic) of this process are existing.

As a last practical restriction it has to be mentioned, that the **computational complexity** for the verification of a protocol in FDR seems to be very high in comparison to the size of the protocol[23]. So far no exact figures exist on how many agents and protocol steps FDR can handle, because the state space resulting from the processing of different context models of the same size strongly varies. Own experiments with medium-sized protocols already required sometimes more than the 3GB of RAM available on our test machine. This problem, which is inherent to the model checking approach, could partly be addressed by approaches for protocol simplification or the usage of other model checking approaches like ATHENA [Song1999].

In summary of these restrictions imposed by CASPER and FDR, we have to mention, that so far only a strongly limited number of tools are available that can be adapted for the modeling, generation and/or verification of watermarking-based protocols. Most of those tools (e.g. CSP or AVISPA) have the same severe restrictions mentioned above for the CASPER & FDR combination. Nevertheless, we would expect more of such modeling tools to emerge in the near future for two reasons: on one hand we see the immense advantages automatic protocol verification can have over manual verification, and on the other hand, watermarking-based protocols can offer interesting non-observability options in addition to all the security aspects addressable also by cryptographic primitives and protocols.

agreement-statement is declared (which is the case from the beginning of the protocol run of the above mentioned simple example, because the source and destination node know each other already by predefined knowledge).
Nevertheless, data-origin-authenticity and integrity are two examples of security aspects which would be beneficially for the automated verification in CASPER or similar languages, because the channels in watermarking protocol scenarios are often considered insecure.

[23] The CASPER website (http://www.comlab.ox.ac.uk/gavin.lowe/Security/index.html) states on this fact: "*One weakness with the CSP/FDR approach is that it can only be applied to finite (typically small) instances of the protocol. This means that if no attack is found, there may still be an attack upon a larger instance. We are currently investigating under what circumstances it is enough to analyze only small instances: more precisely, we have discovered sufficient conditions under which if there is no attack on a particular small system, then there is no attack upon any larger system. Many commercial protocols are rather large and complicated. This makes their direct analysis using CSP and FDR infeasible, because of the resulting explosion in the state space and message space sizes; it also makes any other form of analysis more difficult, because of the mass of details. We are therefore investigating safe simplifying transformations for protocols, that is transformations on protocols such that if there is an attack on the original protocol, then there is also an attack on the transformed protocol. The idea is, starting from a large, complicated protocol, to apply as many such transformations as possible, without introducing new attacks; if the resulting protocol is secure, then so is the original*."

5.4. *Indications for possible future work*

Regarding our framework realization: What is still missing in our realization is a **model for the security of the watermarking components/primitives** introduced as a requirement for all formal method approaches in section 1.1. This model could be included into the watermarking algorithm description and contain information about known vulnerabilities (against attacks), known or estimated security levels, etc. The evaluation of these characteristics could be integrated into two different stages of the framework: either in step 2 and step 3 (the path search and selection) or in step 4 and step 6 (the CASPER modeling and FDR checking). Both alternatives have their specific advantages and drawbacks. If the evaluation of the security of the algorithms is included into the path search, like to other watermarking characteristics, then the number of evaluation runs for the FDR would be much lower. If this evaluation could be included into an extended version of CASPER and be verified with FDR, then all security evaluations (communication security as well as watermarking primitive security) would be performed as one functional block.
A further requirement for efficient implementation of our framework would be a specification of the **cost or quality functions required in the selection steps** (step 3 and 7) where a choice has to be made regarding possible alternative solutions. Here we are still lacking a good concept on how to generalize such functions for a large field of application scenarios.
Our framework would strongly benefit from **extensions of the CASPER language** (see section 5.3). These extensions might strongly decrease the effort required in the modeling and would allow us to include integrity (and partially also non-repudiation) into the set of security aspects verifiable with this approach. Also **alternative realizations based on other languages** (like e.g. AVISPA) might be worth some practical investigations.

Regarding our methodology and theoretical framework: Here the three **major problems in our concept** (conceptualization, message modeling and design of a general purpose compiler – see section 5.2) would have to be solved. Furthermore, research effort might be investigated into the investigation of **probabilistic simulation** strategies that might replace or complement the model checking used in our work for the verification of the protocols. Such a simulation-based approach would also enable moving away from static network descriptions and allow investigations on changing or even ad-hoc networks.

6. Literature

[Boehm88] B. W. Boehm: A Spiral Model of Software Development and Enhancement. IEEE Computer, vol.21, no.5, pp.61-72, doi: 10.1109/2.59, May 1988.

[Boehme09] R. Böhme: An Epistemological Approach to Steganography, in S. Katzenbeisser and A.-R. Sadeghi (Hrsg.): Information Hiding, LNCS 5806, Springer-Verlag, Berlin Heidelberg, 2009, S. 15–30.

[Col07] D. Coltuc, J. Chassery: *Very Fast Watermarking by Reversible Contrast Mapping*. IEEE Signal Processing Letters, Vol. 14, No. 4, IEEE, APRIL 2007.

[Comesana07] P. Comesana and F. Perez-Gonzales: On the capacity of Stego Systems. Proc. ACM MM&Sec'07. ACM, 2007.

[Cormen01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein: Introduction to Algorithms. 2nd ed. MIT Press and McGraw-Hill, ISBN 0-262-03293-7. Section 22.3: Depth-first search, pp. 540-549, 2001.

[Dit04] J. Dittmann, S. Katzenbeisser, C. Schallhart, H. Veith: Provably secure authentication of digital media through invertible watermarks. Cryptology ePrint Archive, Report 2004/293 (2004) http://eprint.iacr.org/

[Dittmann05] J. Dittmann, S. Katzenbeisser, C. Schallhart, H. Veith: *Ensuring Media Integrity on Third-Party Infrastructures*. In Proc. 20th IFIP International Information Security Conference (SEC2005), Springer Verlag, 2005, pp. 493-507.

[FDR] *FDR user manual*. Formal Systems (Europe) Ltd.. http://www.fsel.com/documentation/fdr2/html/

[Fries03] T. Fries: Oberflächenmesstechnik für Labor und Produktion. In: VDI-Berichte 1806, VDI Verlag, 2003, S. 217–225.

[Hoare04] C. A. R. Hoare: Communicating Sequential Processes. Prentice Hall International, 2004.

[Kraetzer10ACM] Christian Kraetzer, Ronny Merkel, Robert Altschaffel, Eric Clausing, Maik Schott, Jana Dittmann: Modelling Watermark Communication Protocols using the CASPER Modelling Language. Proc. ACM Multimedia and Security Workshop 2010 (MMSec'10), Rome, Italy, 09.-10.09.2010, ACM Press, New York, NY, ISBN 978-1-450-30286-9, S. 67-72, 2010.

[Lang07] A. Lang: Audio Watermarking Benchmarking – A Profile Based Approach. PhD Thesis. Otto-von-Guericke-University Magdeburg, Germany, Dept. of Computer Science, ISBN: 978-3-940961-22-8, 2007.

[Lowe98] G. Lowe: *CASPER: A Compiler for the Analysis of Security Protocols.* Journal of Computer Security, Society Press, 1998.

[Lowe09] G. Lowe, P. Broadfoot, C. Dilloway, M. L. Hui: CASPER - A Compiler for the Analysis of Security Protocols. User Manual and Tutorial, Version 1.12, Oxford University Computing Lab, September, 2009.

[Newman07] R. Newman: Computer forensics: evidence, collection, and management. Auerbach, 2007.

[Pimentel08] Juan Carlos López Pimentel and Raúl Monroy: *Formal Support to Security Protocol Development: A Survey*. Computación y Sistemas Vol. 12 No. 1, 2008, pp 89-108, ISSN 1405-5546, 2008.

[Roscoe94] A. W. Roscoe: *Model-checking CSP*. In A classical mind: essays in honour of C. A. R. Hoare, Prentice Hall International (UK) Ltd., 1994.

[Royce1970] W. Royce: Managing the Development of Large Software Systems. Proceedings of IEEE WESCON 26, August 1st-9th, 1970.

[Ryan00] P. Ryan, S. Schneider, M. H. Goldsmith: Modelling and Analysis of Security Protocols. Addison-Wesley Professional, ISBN-13: 978-0201674712, 2000.

[Schott10] M. Schott, C. Kraetzer, J. Dittmann, C. Vielhauer: *Extending the Clark-Wilson Security Model for Digital Long-Term Preservation Use-cases*, Proceedings of Multimedia on Mobile Devices 2010, Electronic Imaging Conference 7542, IS&T/SPIE 22nd Annual Symposium, San José, January 18th-19th, 2010.

[Sheppard01] N.P. Sheppard, R.S. Naini, P. Ogunbona: On multiple watermarking. Proc. ACM Multimedia and Security Workshop 2001, Ottawa, Ontario, Canada, 5 October 2001, pp 3–6, ACM, 2001.

[Song1999] D. X. Song: Athena: a new efficient automatic checker for security protocol analysis. In Proceedings of the Twelth IEEE Computer Security Foundations Workshop, pp. 192-202, IEEE Computer Society Press, 1999.

[Vigano05] L. Viganò: *Automated Security Protocol Analysis with the AVISPA Tool*. Proc. XXI Mathematical Foundations of Programming Semantics (MFPS'05), 2005.

Appendix A: Basic CASPER constructs – a introduction with a focus on watermarking protocols

Since CASPER is not designed to verify watermarking-based security protocols, some effort has to be spent on adapting this language (or more precisely its usage) towards the requirements of POWER.

To do so, this appendix is split into four parts: the first part briefly summarizes basic concepts in CASPER models and a gives a short protocol example, the second part focuses on simulating a watermark message and watermarking algorithms in CASPER, the third part is concerned with watermarking infrastructure modeling and the last part with watermarking protocol security modeling in CASPER.

Basic concepts in CASPER models

CASPER ([Lowe98], [Lowe09], [Ryan00]) as a protocol modeling tool offers the possibility to represent the design of a protocol in a human- and machine-readable format, which can then be compiled into Communication Sequential Processes (CSP; [Hoare04], [Roscoe1994], [Ryan00]) notation to be verified by the FDR (Failures-Divergences Refinement; [FDR]) model checker. If FDR returns a negative statement while running the verification process, a possible attack on the communication scenario is found. With the usage of this procedure, the security aspects of confidentiality, authenticity and integrity of messages communicated in a designed protocol can be evaluated in an automated way.

The protocol description in CASPER is structured into eight well defined blocks. These blocks are named ***#Free variables, #Processes, #Protocol description, #Specification, #Actual variables, #Functions, #System*** and ***#Intruder information***. The right order of these blocks is needed for the compiler but is of no importance for the understanding of the protocol modeling process.

A very simple example of an agent ***A*** sending a message ***mes*** to an agent ***B*** using a symmetric encryption is given in table 1.

Part	***Topology***	***Attack scenario description***
1	***#Free variables*** ***A,B : Agent*** ***mes : Message*** ***Skey : SharedKey*** ***InverseKeys = (Skey,Skey)***	
2	***#Processes*** ***SENDER(A,mes) knows Skey*** ***REICEIVER(B) knows Skey***	
3	***#Protocol Description*** ***0. →A : B*** ***1. A→B : {mes}{Skey}***	
4		***#Specification*** ***Secret (A, mes, [B])***
5	***#Actual variables*** ***AVA, AVB, Mallory : Agent*** ***AVmes: Message***	
6	***#Functions***	

7	***#System***
	 SENDER(AVA,AVmes)
	 RECEIVER(AVB)	

8		***#Intruder Information*** ***Intruder = Mallory*** ***IntruderKnowledge = {AVA,AVB}***

In the ***#Free variables*** section all the variables (including the agents) used in the protocol are specified together with all functions. The given example specifies two variables ***A*** and ***B*** which are of type ***Agent*** and a data-object ***mes*** of type ***Message***. A shared key ***Skey*** is also given representing a symmetric key used in this example. A pair of inverse keys needs to be specified for this shared key to signal that a message encrypted with ***Skey*** will become the original message after being decrypted with ***Skey*** again. If a key is inverse to itself it is a symmetric key. If it is inverse to any other key, it is an asymmetric key. In the current example ***Skey*** is inverse to ***Skey*** representing a shared secret in a symmetric key scenario.

The ***#Processes section*** describes the agents participating in the protocol. Each agent is represented by a process (i.e. ***SENDER***, ***RECEIVER***) including a variable of type ***Agent*** as input and additional variables (i.e. ***mes***). The keyword ***knows*** tells the program which functions are known by this process in addition to the knowledge it gains by the parameters communicated to the process. In the given example both agents know the shared key ***Skey*** due to the assigned parameters. Agent ***A*** also knows the Message ***mes***.

The block ***#Protocol description*** represents the communication steps of the protocol, each line representing one step. These steps follow a simple ***source → destination : message*** – syntax.
In our example, agent ***A*** is told by the environment about the existence of an agent ***B*** in an initialization step 0. He then sends its value of ***mes*** to agent ***B*** in step 1 of this example. Before the sending he encrypts it with the shared key ***Skey***

Security aspects to be checked are defined in the ***#Specification*** section. There exist two main constructs for defining specification. First is a ***secret*** specification, stating, that some piece of data is only known to certain nodes after completion of the run. The keyword ***secret*** in our example states that the message ***mes*** is a secret only known to agent ***A*** and is in addition only allowed to be known by agent ***B***. If any other agent gains access to this secret during the protocol run this is returned as a possible attack.
Another important statement in the ***#Specification*** section is of the form ***Agreement(A,B,[mes])***. It can be used for verifying the entity-authenticity regarding the communication partners. In the case of the communication partners being directly connected to each other (with no additional node in between) FDR checks whether ***A*** and ***B*** completed a protocol run with each other (both agreeing on the value of the transmitted mes) concluding that they are correctly authenticated (here entity-authenticity) to each other.

In the ***#Actual variables*** block certain values are allocated to the variables, i.e. ***AVA*** is the specific value of the variable ***A***. This block can be derived easily from the

#Free variables as the only addition is another ***Agent*** in form of an ***Intruder***, usually named ***Mallory***.

In the ***#Functions section*** functions can be defined. Within the context of our work in POWER we found no functions provided by CASPER which are useful for watermarking protocol modeling.

In the ***#System*** block the specific values of the variables given in this scenario are handed to the agents. Similar to the ***#Actual variables*** this can be derived directly from the ***#Processes*** block.

The block ***#Intruder Information*** defines the intruder and his knowledge. In the given example - according to Kerckhoffs' law - he has all the knowledge except the secret key and the message (which is explicitly defined as being secret in the ***#Specification*** block).

By changing the information specified in the eight CASPER blocks, different topologies, different security aspects as well as different attacker scenarios can be defined. ***#Free Variables***, ***#System***, and ***#Protocol Description*** all govern the topology and will be changed to represent other topologies.
By changing the ***#Specification*** block, different security aspects can be checked, while by changing the ***#Intruder Information***, different attacker scenarios can be modeled.

Simulating a watermark message and watermarking algorithms in CASPER

Since CASPER currently point offers no special constructs for representing digital watermarks and the corresponding embedding and retrieving functions, an approximated representation needs to be developed from the given constructs. We therefore assume that an agent embeds a message ***mes*** with an (optional) overhead oh (which might be needed for certain watermarking schemes, i.e. in hierarchical watermarks) into a cover ***cov*** using a given watermarking algorithm. By sending the message, the overhead and the cover object as three different items to the receiving agent, the transfer of the watermarked cover is emulated in CASPER. The receiver then assumes the message overhead and the cover to be the result of (an emulated) extraction process. This very simple approximation shows similar properties than a real watermarking scheme: A possible intruder can only be prevented from accessing the watermark message by the usage of a key for embedding. This can be approximated in CASPER by the encryption of the watermark message and the overhead by this key. An intruder now still has access to the cover object (since it is not encrypted in our emulation) but he can not read the watermark message or the overhead. Similarly, a non-invertible watermarking scheme can be emulated by sending a changed cover (new cover variable) to the receiver. In this case a reconstruction of the original cover during the detection process won't be possible.

To extend our example from the previous section from sending a simple message from ***A*** to ***B*** to sending an invertibly watermarked cover producing an overhead we have to add a variable ***oh*** to represent the overhead created by the embedding algorithm and a cover ***cov*** to the model. We also change the ***#Protocol description*** block by using the symmetric key the agents know already to encrypt the message and the overhead since they are embedded with a key and not extractable without it:

A→B : {mes,oh}{Skey(A)},cov

Watermarking infrastructure modeling in CASPER

The incorporation of infrastructure mechanisms like a Public Key Infrastructure (PKI) or a Time Server (TS) can be modeled in CASPER in two different ways. The less complex way is to assume that an agent has acquired a timestamp or the public key of a communication partner automatically. In this case he would only add the timestamp as another data item to be transferred to the receiver or respectively use the public key to embed the watermarking message. The second, more complex way, would also allow for the acquisition process of a timestamp or a public key to be checked for security flaws. To do so the acquisition process of a timestamp or a public key would be modeled by using additional agents acting as time- or key-servers. In such a scenario each request to and answer from such a server would be modeled as a separate communication step in the ***#Protocol description*** section.

Watermarking protocol security modeling in CASPER

The definition of the security aspects that should be validated takes place inside the ***#Specification*** block. Casper yields several constructs for such a specification of which a **secret** and an **agreement** are the most common.

CASPER (http://www.comlab.ox.ac.uk/gavin.lowe/Security/Casper/manual.pdf page 31) defines a **secret** as follows:
"***Secret(A, s, [B1,…,Bn])*** specifies that in any completed run, ***A*** can expect the value of the variable ***s*** to be a secret; ***B1 , … ,Bn*** are the variables representing the roles with whom the secret is shared. More precisely, this specification fails if ***A*** can complete a run, where none of the roles ***B1 , … . ,Bn*** is legitimately taken by the intruder, but the intruder learns the value ***A*** gives to ***s***.
Therefore the usage of a secret-Specification lies in the modeling of confidentiality, as it states that the protocol is vulnerable if an intruder can obtain a certain piece of data."

An **agreement** is defined for CASPER as:
"***Agreement(A, B, [v1,…,vn])*** specifies that ***A*** is correctly authenticated to ***B***, and the agents agree upon ***v1 , … , vn***; more precisely, if ***B*** thinks he has successfully completed a run of the protocol with ***A***, then ***A*** has previously been running the protocol, apparently with ***B***, and both agents agreed as to which roles they took, and both agents agreed as to the values of the variables ***v1 , …, vn***, and there is a one-one relationship between the runs of ***B*** and the runs of ***A***."
This means that an agreement is fulfilled if two nodes are authenticated to each other. This specification is used to prove entity-authenticity.

The **#Intruder Information** yields further possibilities for security checking as it allows to model attackers that have gained any chosen amount of knowledge about the system in regard. This block can be automatically created giving the intruder a standard set of knowledge. This could be - according to Kerckhoffs' law - all the information which is not explicitly specified as secret or a key not supposed to be known by the intruder. Other possible scenarios would i.e. include a case where the intruder learned a secret key by means of social engineering.

```
#Intruder Information
Intruder = Mallory
IntruderKnowledge = {AVA,AVB,Mallory}
```

In the given example Mallory does not know ***mes*** because he doesn’t have the symmetric key. If the key ***Skey*** would be added to this knowledge, it would obviously yield other results regarding the security of the protocol.

Appendix B: Translation rules

In this appendix the translation rules required to translate XML context models into CASPER protocols (see section 2.2.1.2) are described in detail. These rules have been designed to translate our application scenarios presented in sections 3.2 and 3.3 – they will not be complete for general purpose translation into CASPER. The rules are implemented by the translator tool for CASPER protocol generation from the XML context models described in section 3.1.2.

Note: As CASPER doesn't support numbers in variable names, each instance of “1” in an identifier is translated to “a”, “2” to “b”, and so on. To allow for easier separation, all XML code is given in `Courier New` font, all CASPER code in italic ***Courier New*** font.
For better clarity the rules of translation are presented here in the order the resulting instructions will appear in the resulting CASPER-Protocol. Therefore this Appendix is structured into the eight blocks of a CASPER file: ***#Free variables***, ***#Processes***, ***#Protocol description***, ***#Specification***, ***#Actual variables***, ***#Functions***, ***#System***, and ***#Intruder information***.

#Free variables

Here the variables used in the protocol are declared, common types and their counterparts in the XML context model are:

<Agents> : Agent
Agents are taken from the `<src>` and `<dst>`-Tags of the `<subtasks>`

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
</subtask>
```

Becomes in CASPER

A,B : Agent

Rule 1

`<subtask>`
`  <src>[NODESRC]</src>`
`  <dst>[NODEDST]</dst>`
→ ***[NODESRC],[NODEDST] : Agent***

<Covers> : Cover
Covers are identified by the `<id>`-Tags of different `<covers>` inside the `<subtasks>`

Example:

```
<cover>
	<id>cover1</id>
	<type>image</type>
</cover>
```

Becomes in CASPER:
cover1 : Cover

Rule 2

`<cover>`
`  <id>[COVER]</id>`
→ ***[COVER] : Cover***

<Data> : Data
Data blocks are taken from the `<content>`-Tag of different `<messages>` inside the `<subtasks>`. This concerns only the raw data blocks, because encryption, hashing or signing are handled separately.

Example:
```
<content>
     mes
</content>
```

Becomes in CASPER:
mes : Data

Rule 3

`<content>`
`  [DATA]`
→ ***[DATA] : Data***

<Keys> : sharedkey
The used keys are a more complex construct, with the necessity to take required information different tags in the XML context model. First, all different <dwm>-blocks that are marked as using a symmetric key (i.e. which contain a `<key>symmetric</key>` statement) get a key assigned.

Example:
```
<dwm>
     <id>dwm1</id>
     ...
     <key>symmetric</key>
</dwm>
```

Becomes in CASPER:
keydwma : sharedkey

Second, keys might also be used by encryption in the `<message>`-blocks. If this encryption is marked as symmetric (`<encrypted-symmetric>`) a key for the corresponding level of the message must be created.

Example:
```
<message>
     <id>message</id>
     <level>0</level>
     <content>
          <encrypted-symmetric>
               Data
          </encrypted-symmetric>
     </content>
</message>
```

Becomes in CASPER:
keylevela : sharedkey

It is also possible that asymmetric encryption is used. In this case we assume that a message is encrypted with the public key of the recipient. So, a pair of keys must be defined in that case.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <content>
          <encrypted-asymmetric>
               Data
          </encrypted-asymetric>
     </content>
     ...
```

Becomes in CASPER:
SkeyB : secretkey
PkeyB: publickey

Furthermore, when using signatures asymmetric keys are also used. In this case we assume that a message is signed with the secret key of the source. A pair of keys has to be specified in this case.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <content>
          <signed>
               Data
          </signed>
     </content>
```

```
    ...
```

Becomes in CASPER:

SkeyA : secretkey
PkeyA : publickey

Rule 4

```
  <id>[DWM]</id>
  ...
  <key>symmetric</key>
</dwm>
```

→

key[DWM] : sharedkey

```
<level>[LEVEL]</level>
<content>
  <encrypted-symetric>
    [DATA]
  </encrypted-symetric>
</content>
```

→

keylevel[LEVEL] : sharedkey

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <content>
    <encrypted-asymetric>
      [DATA]
    </encrypted-asymetric>
  </content>
```

→

Skey[NODEDST] : secretkey
Pkey[NODEDST] : publickey

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <content>
    <signed>
      [DATA]
    </signed>
  </content>
```

→

Skey[NODESRC] : secretkey
Pkey[NODESRC] : publickey

InverseKeys = <Keylist>

From the list of keys generated by the application of rule 4 the relation between those must be specified. For symmetric keys the keys are inverse to themselves. For asymmetric keys the public key is inverse to the secret key of the same node.

keydwma : sharedkey
SkeyA : secretkey
PkeyA: publickey

Is extended by:
InverseKeys = (keydwma,keydwma),(SkeyA,PkeyA)

Rule 5
[KEY[LEVEL]] : sharedkey → ***InverseKeys = ([KEY[LEVEL]],[KEY[LEVEL]])*** ***[SKEY[NODE]] : secretkey*** ***[PKEY[NODE]] : publickey*** ***if [NODE] == [NODE] →*** ***InverseKeys = ([SKEY],[PKEY])***

H: HashFunction
If any part of any message is containing hashes, a hash function `H` has to be added to the CASPER statements.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <content>
          <hashed>
               Data
          </hashed>
     </content>
     ...
```

Becomes in CASPER:
H: HashFunction

Rule 6

`<hashed>`
`  [DATA]`
`</hashed>`
→
H: HashFunction

#Processes

Here the knowledge of the agents partaking in the protocol is modeled. Therefore process names are assigned to all agents, starting with the source of the first subtask which is labeled as `SENDER` to the destination of the last subtask which is labeled as ***`RECEIVER`***. All nodes in between are labeled ***`FORWARDn`***, where ***`n`*** is ***`A`***, ***`B`***,... according to the fact that CASPER doesn't handle numbers in designators.

Example:

`NodeA, NodeB, NodeC : Agent`

Becomes:

```
SENDER(NodeA)
FORWARDERA(NodeB)
RECEIVER(NodeC)
```

The knowledge of a node consists of:

- his own identity
- the keys of the dwms he uses

Example:

```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <dwm>
          <id>dwm1</id>
          ...
```

Becomes in CASPER:

```
SENDER(NodeA, keydwma)
FORWARDER(NodeB,keydwma)
```

Rule 7

`<subtask>`
`  <src>[NODESRC]</src>`
`  <dst>[NODEDST]</dst>`
`  ...`
`  <dwm>`
`    <id>[DWM]</id>`
`    ...`
→
`[NODEPOSITION[NODESRC]]([NODESRC],key[DWM])`
`[NODEPOSITION[NODEDST]]([NODEDST],key[DWM])`

Cover altered by a node

A cover is altered by a node, if that node sends a cover he hasn't received at any earlier step of the protocol.

This is the case when a node is a.) the ***`SENDER`***-node of the network or b.) a ***`FORWARDER`***-node that uses a certain cover while being the source `src` of a

`<subtask>` it hasn't acquired while being the destination `dst` of the preceding `<subtask>`.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     <meta>
          <cover>
               <id>coverA</id>
               <type>image</type>
          </cover>
```

and
SENDER(NodeA)

Are extended to:
SENDER(NodeA, CoverA)

Rule 8

`<subtask>`
`  <src>[NODESRC]</src>`
`  <dst>[NODEDST]</dst>`
`  <meta>`
`    <cover>`
`      <id>[COVER]</id>`
 	
if [NODEPOSITION[NODESRC] == SENDER →
SENDER(NODESRC],[COVER])
if [NODEPOSITION[NODESRC] != SENDER
and [COVER] != [COVER FROM LAST SUBTASK] →
[NODEPOSITION](NODESRC],[COVER])

Data altered by a node
Data is altered by a node if that node sends data he hasn't received at any earlier step of the protocol.
This is the case when a node is a.) the SENDER-node of the network or b.) a ***FORWARDER***-node that send certain data while being the source `src` of a `<subtask>` it hasn't acquired while being the destination `dst` of the preceding `<subtask>`.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <content>
          data
     </content>
```

and

SENDER(NodeA)

Are extended to:
SENDER(NodeA, data)

Rule 9

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <content>
    [DATA]
  </content>
```
 if [NODEPOSITION[NODESRC] == SENDER →
 SENDER(NODESRC],[DATA])
 if [NODEPOSITION[NODESRC] != SENDER
 and [DATA] != [DATA FROM LAST SUBTASK] →
 [NODEPOSITION](NODESRC],[DATA])

The secret key of a node was declared in the *#Free variables*

Example:
SkeyA: SecretKey

Results in:
SENDER(NodeA,SkeyA)

Rule 10
Skey[NODE] : secretkey   ***→***   ***[NODEPOSITION](NODE,Skey[NODE])***

All shared keys a node has access to
A node has access to a shared key if a node sends any message corresponding to the used hierarchy level
This property could also be modeled more complex with a key exchange scenario but this is outside the scope of this work. Instead we use the easier method of modeling the keys as knowledge of the nodes.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <message>
          <id>message</id>
          <level>1</level>
```

Becomes:
SENDER(NodeA, keylevel1)

Rule 11

`<subtask>`
`  <src>[NODESRC]</src>`
`  <dst>[NODEDST]</dst>`
`  ...`
`  <message>`
`    <id>message</id>`
`    <level>[LEVEL]</level>`
→
[NODEPOSITION[NODESRC]]([NODESRC],keylevel[LEVEL])

All public keys of all other agents in the scenario

Example:
PkeyA,PkeyB,PkeyC : PublicKey

Becomes:
SENDER(NodeA,PkeyA,PkeyB, PkeyC)

Rule 12
[PKEYS] : PublicKey → ***[NODEPOSITION](NODE,[PKEYS])***

#Protocol description

Here all data transfers are declared. One step is created for each subtask.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <message>
          <id>message</id>
               <level>0</level>
               <content>
                    <data>data</data>
               </content>
          </message>
```

Becomes in CASPER:
A → B : data

As the message can be more complex, the resulting step becomes more complex. Encryption, signing and hashing are possible with different results:

If the **encryption** is marked as **symmetric** (`encrypted-symmetric`) the key for the corresponding level of the message is used to encrypt.

Example:
```
<message>
     <id>message</id>
     <level>0</level>
     <content>
          <encrypted-symmetric>
               data
          </encrypted-symmetric>
     </content>
</message>
```

Becomes in CASPER:
A → B : {data}{keylevela}

It is also possible that **asymmetric encryption** is used. In this case we conclude that a message is encrypted with the public key of the recipient.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <content>
          <encrypted-asymmetric>
               data
          </encrypted-asymmetric>
     </content>
```

Becomes in CASPER:
A → B : {data}{PkeyB}

Furthermore when using **signatures** we conclude that a message is signed with the secret key of the source.

Example:
```
<subtask>
     <src>A</src>
     <dst>B</dst>
     ...
     <content>
          <signed>
               data
          </signed>
     </content>
```

Becomes in CASPER:
A → B : {data}{SkeyA}

The last possibility is a **hashed** message.

Example:

```
<hashed>
     data
</hashed>
```

Becomes in CASPER:

A → B : H(data)

Rule 13

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <content>
    <encrypted-symmetric>
      [DATA]
    </encrypted-symmetric>
  </content>
→
```

[NODESRC] → [NODEDST] : {[DATA]}{keylevel[LEVEL]}

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <content>
    <encrypted-asymmetric>
      [DATA]
    </encrypted-asymmetric>
  </content>
→
```

[NODESRC] → [NODEDST] : {[DATA]}{Pkey[NODEDST]}

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <content>
    <signed>
      [DATA]
    </signed>
  </content>
→
```

[NODESRC] → [NODEDST] : {[DATA]}{Skey[NODESRC]}

```
<subtask>
  <src>[NODESRC]</src>
  <dst>[NODEDST]</dst>
  ...
  <hashed>
```

```
    [DATA]
  </hashed>
→
[NODESRC] → [NODEDST] : H([DATA])
```

#Specification

A ***#specification*** represents the security properties which should be checked by the model checker. There exist two main constructs for defining a ***#specification***. The first is a ***secret*** statement, stating that some piece of data is only known to certain nodes after completion of the run. This statement is used to model and verify confidentiality. The other statement is an ***agreement*** which states that two nodes are authenticated correctly and agree on a certain value. This statement is used for entity-authenticity.

#Actual variables

Copy the ***#Free variables*** block and add an ***AV*** in front of each variable and add "***Mallory***" as an ***Agent***.

#Functions

This block remains empty for our watermarking-focused modeling and verification operations.

#System

Copy the ***#Processes*** block and add an ***AV*** in front of each variable.

#Intruder Information

This block can be automatically created giving the intruder a standard set of knowledge. This could be - according to Kerckhoffs’ law - all the information which is not explicitly specified as secret i.e. in the general case a key not supposed to be known by the intruder. To model advanced attacker scenarios with different sets of intruder knowledge, manual adaptation of this block is required.