Distributed and Embedded Systems
Department of Computer Science
Aalborg University

Master's Thesis

Michael Garde

From Formal to Computational Authenticity

– An approach for reconciling formal and computational authenticity

---

Computer Science 10. semester
Advisor: Hans Hüttel
June 11th, 2013

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg East
Telephone +45 9940 9940
Fax +45 9940 9798
http://www.cs.aau.dk

Title:
From Formal to Computational Authenticity

Subtitle:
– An approach for reconciling formal and computational authenticity

Theme:
Formal and computational security of cryptographic protocols

Project periode:
Computer Science 10. semester
Spring 2013

Project group:
d1013f13

Author:
Michael Garde

Advisor:
Hans Hüttel

Copies:
3

Page count:
32

Submitted
June 11th, 2013

Signature:

Synopsis:

This paper is the product of a study in secrecy and authenticity in cryptographic protocols with respect to the formal and computational paradigms. These paradigms have mainly been separate fields of research, until the first attempt at bridging the gap was presented in 2001. This is still an active research field and though many exciting results have been achieved, we focus on an apparently unsolved issue, namely the relation between formal and computational authenticity. We provide a survey into secrecy and authenticity in both settings and suggest an approach for showing that authenticity in the formal paradigm implies authenticity in the computational paradigm.

Michael Garde
mgarde07@student.aau.dk

Preface

This master's thesis is the product of a 10th semester study in the general field of formal and computational security of cryptographic protocols. It was composed during the spring semester of 2013 at the Department of Computer Science at Aalborg University and submitted on June 11th, 2013. The initial idea for this work was proposed by Hans Hüttel, who had identified the missing relation between formal and computational authenticity.

I would like to thank my advisor Hans Hüttel for his dedication and invaluable help in answering my many questions as well as my family for their patience during this period.

Contents

Introduction 1

In the field of cryptographic protocols there exist at least two main paradigms (or views) for abstract high-level description of such protocols, namely the formal and the computational paradigms. They do not consider the specifics of cryptographic functions and instead considers them as black box functions that provide key generation and performs encryption and decryption etc., in order to focus on the modeling of cryptographic protocols.

This paper studies the theory of the formal and computational paradigms of cryptographic protocols and provides a survey of a small part of the research, that has focused on relating these. A vast amount of research has made an effort of showing how to perform formal analysis of security properties such as secrecy and authenticity.

The following section will discuss these two paradigms in more detail and present some of the related research in these fields.

1.1 Cryptographic Paradigms and Related Work

Inspired by [Kemmerer, 1989] we define a *cryptographic protocol* as follows:

> A cryptographic protocol is a set of rules or procedures for using a cryptographic algorithm to send and receive messages in a secure manner over a network.

1.1.1 The Formal Paradigm

By formal, we mean symbolic in the sense of formal languages. As an example, consider the expression $\{M\}_K$ that may represent the encryption of a plaintext M using the encryption key K, where $\{M\}_K$, M and K are formal expressions. Encryption and key generation algorithms are assumed to exist and the details of these are not considered. Furthermore, within this assumption, it is not possible for an adversary to obtain M given only the encrypted message $\{M\}_K$. Thus, the formal model is used for describing ideal protocols, with respect to security.

The Formal Paradigm has its origin in the work of Dolev and Yao [Dolev and Yao, 1983], which was motivated by the need for semantic models that allowed for a precise account of security issues of cryptographic protocols, in the presence of an adversary. The notion of formal models has come to be synonymous with the Dolev-Yao model. It is an approach for analyzing cryptographic protocols formalized with symbolic expressions in abstract formal languages similar to the π-calculus. The purpose is to give formal verification of the security properties that the protocol provides in the presence of an attacker. This includes the preservation of secrecy in transmitted messages as well as ensuring authenticity.

The groundbreaking work of Dolev and Yao has spawned a considerable amount of research. Woo and Lam [Woo and Lam, 1993] defined a semantic model for authentication protocols, developed a syntax for correspondence and secrecy assertions and defined semantics for their semantic model, as a foundation for analysis techniques. Especially relevant for this paper is the work of Abadi and Gordon [Abadi and Gordon, 1997] who introduced the spi-calculus, an extension of the π-calculus, to include cryptographic primitives, as a tool to describe and analyze authentication protocols. Based on the work of [Woo and Lam, 1993; Abadi and Gordon, 1997], Gordon and Jeffrey [Gordon and Jeffrey, 2003] proposed a method for checking authenticity properties of cryptographic protocols with the means of correspondence assertions and typechecking.

Abadi and Blanchet [Abadi and Blanchet, 2001] developed a process calculus in order to define formal secrecy and they show, under certain conditions, how a process preserves the secrecy of data in the presence of an adversary. In [Blanchet, 2001], Blanchet presents ProVerif[1], a software tool for automatic verification of secrecy properties of cryptographic protocols in formal models. In further work [Blanchet, 2002], Blanchet shows that authenticity can be proved using secrecy. This work relates to Woo and Lam's [Woo and Lam, 1993] notion of correspondence assertions.

However, the formal model does not take into account the complexity or the probability of an attacker to obtain a secret message. Even though this could be considered as a weakness, the simplicity of the formal model provides an ease of use for protocol designers and it is therefore desirable to model protocols in this view.

1.1.2 The Computational Paradigm

As hinted above; one might think that the formal model lacks a real-world perspective of cryptographic protocols. The approach in the *computational paradigm* is to consider protocols as acceptable if an adversary faces a challenge that is computationally infeasible when attempting to "'break"' a protocol. The computational view takes these considerations into account and tries to reason about the probability of a successful attack as well as the computational cost.

As in [Abadi and Rogaway, 2002] and [Bellare et al., 1997] a symmetric encryption scheme is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of three algorithms. The key generator algorithm $\mathcal{K}$ maps a randomly flipped coin to a string k. The encryption algorithm $\mathcal{E}$ uses a random coin flip r to map the strings k and m into a new string $\mathcal{E}_k(m, r)$. The decryption algorithm $\mathcal{D}$ maps strings k and c into $\mathcal{D}_k(c)$ such that $\mathcal{D}_k(\mathcal{E}_k(m, r)) = m$ i.e. decryption of a ciphertext reveals the original message.

An adversary for the encryption scheme above is a Turing machine with the capability of querying an oracle. The oracle may be constructed in two ways.

1. An oracle chooses a fixed random key k, which is used for all queries. For each query x it performs the encryption $\mathcal{E}_k(x)$ using new random coins.
2. An oracle chooses a fixed random key k, which is used for all queries. When given a query x it encrypts a string containing 0 bits of length $|x|$ using new random coins.

The probability that the adversary outputs 1 in the first case minus the probability that the adversary outputs 1 in the second case is called the *adversary's advantage*. If the adversary's advantage grows slowly, with increased computational resources, the encryption scheme is considered to be good.

The formal and the computational views of cryptographic protocols are well-founded research fields, however, they have mainly been separate areas of research. Since both views are methods for describing cryptographic protocols, interest in reconciling these have gained increasing attention. Establishing such a connection is thus a fairly new and active area of research.

[1] *ProVerif: Cryptographic protocol verifier in the formal model*, `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`

Some of the initial work was given by Abadi and Rogaway [Abadi and Rogaway, 2002], whose focus was on relating the two paradigms. They define equivalence of formal expressions, where expressions are data, as used in security protocols, and equivalence is a relation that states whether two pieces of data "look the same". For relating equivalence of formal expressions to the computational view, they introduce the notion of ensembles, that are probability distribution on strings and defines that two ensembles are indistinguishable when the probability, that the adversary "guesses" the two ensembles, is negligible. They then show that equivalent expressions in the formal view implies indistinguishable ensembles in the computational view.

Backes and Pfitzmann [Backes and Pfitzmann, 2005] define computational secrecy (or payload secrecy) by introducing two views of a user: One that includes a machine that randomly replaces messages and another that simply forwards all message. If the view of the user, in a configuration that includes a replacement machine, is indistinguishable from the view of the user, in a configuration that includes a forwarding machine, then computational secrecy is implied for all polynomial-time users and adversaries.

Laud [Laud, 2005] developed a process calculus similar to the spi-calculus with a semantics that makes use of the computational model and further develops a type system for his calculus. Laud was then able to show, that if a protocol typechecks, then it preserves the secrecy of messages and proceeds to show, that if secrecy properties is satisfied by the formal semantics, then it is also satisfied in the computational semantics.

A result which directly relates the work of Laud to that of the formal paradigm, is that of [Abadi et al., 2006]. Abadi, Corin, and Fournet introduce a version of the π-calculus to establish formal secrecy properties. They extend the calculus with a type system, translate their calculus to Laud's calculus and show that their translation is type-preserving. Thus, Laud's language is used as an intermediate language in order to obtain computational secrecy guarantees with respect to their calculus.

As already suggested by [Abadi et al., 2006] (and [Laud, 2005]), we expect that authenticity in the formal paradigm implies authenticity in the computational paradigm. Such a result should be analogous to the result of [Abadi et al., 2006], namely that secrecy properties that are satisfied in the formal paradigm implies secrecy in the computational paradigm. It seems, to the best of our knowledge, that no such result has been achieved. To obtain this, we provide a survey of the theories of formal and computational security of cryptographic protocols and examine how our expected result may be achieved.

1.2 Outline

We divide this paper into two main themes: Secrecy (Chapter 2) and authenticity (Chapter 3).

Section 2.1 presents, based on [Abadi et al., 2006], a process calculus and its reduction semantics to establish a definition of formal secrecy in the presence of an adversary. A type system for the calculus is introduced and we present The Secrecy Theorem by [Abadi and Blanchet, 2001], that shows, how a well-typed process preserves the secrecy of messages in the presence of an adversary. Section 2.2 provides a precise definition of computational secrecy based on the work of [Backes and Pfitzmann, 2005] and in Section 2.3 we present a significant theorem by [Abadi et al., 2006] showing, that a secrecy preserving and well-typed process preserves computational secrecy. Section 3.1 defines formal authenticity as in [Blanchet, 2002] and extends the process calculus with correspondence assertions. We relate these to the notion of non-injective agreement and add to Theorem 1 in [Blanchet, 2002], that if non-injective agreement is satisfied then formal secrecy is preserved. Section 3.3 relates non-injective correspondence to computational authenticity. Finally, in Section 3.4 we propose a method of showing that formal authenticity implies computational authenticity.

Secrecy 2

This chapter studies the notion of formal as well as computational secrecy in terms of cryptographic protocols and present some of the important results obtained. Especially that of [Abadi et al., 2006], which shows that a type system for formal secrecy in fact captures computational secrecy.

2.1 Formal Secrecy

Abadi et al. informally defines secrecy as follows:

> A process P preserves the secrecy of a piece of data M if P never publishes M, or anything that would permit the computation of M, even in interaction with an attacker.

This section introduces a formal process calculus, based on [Abadi et al., 2006], providing the foundation for a definition of formal secrecy. Furthermore, a type system from [Abadi et al., 2006] is introduced and we state an important result – that a process (under certain conditions) preserves the secrecy of secret data from public data and public channels.

2.1.1 The Process Calculus

For establishing the formal secrecy property, we make use of the *Source Process Calculus* of [Abadi et al., 2006]. It is actually similar to the polyadic π-calculus, which means that it allows the communication of multiple terms in a single step, contrary to the monadic nature of the π-calculus. The calculus is asynchronous, thus allowing pending output processes in parallel with other running processes. The syntax of this calculus is given in Table 2.1 on the following page.

The Source Process Calculus defines two categories. The *terms* represent data and can be either variables or names. Note how there is a distinction between names and variables contrary to the traditional π-calculus, which only considers names to represent communication ports, variables or data values. The *processes*, which represent programs, are defined by formation rules similar to those of the π-calculus.

- A process may *output* data $M_1, \ldots, M_n$ through M defined by $\overline{M}\langle M_1, \ldots, M_n \rangle$ and similarly a process P may receive (*input*) data $x_1, \ldots, x_n$ via M defined by $M(x_1, \ldots, x_n).P$, after which it continues as P.
- *Replication*, denoted $!P$, allows an unbounded supply of copies of P in parallel. Thus a process is always able to start a new copy of itself with the same data as input. When replication is optional $! = M(x_1, \ldots, x_n)$ may be used.
- The *nil* process (**0**) is the process that performs no action.

Let a, b, c, k, s be *names* from an infinite set of names $\mathcal{N}$ and let x, y, z be *variables* from an infinite set of variables $\mathcal{V}$, then the syntax is given by

Terms	M, N	::=	x, y, z	Variable
			a, b, c, k, s	Name
Processes	P, Q	::=	$\overline{M}\langle M_1, \ldots, M_n \rangle$	Output
			$M(x_1, \ldots, x_n).P$	Input
			$!M(x_1, \ldots, x_n).P$	Replicated input
			$\mathbf{0}$	Nil
			$P \mid Q$	Parallel composition
			$(\nu a)P$	Restriction
			if $M = N$ then P else Q	Conditional

Table 2.1. Syntax of the Source Process Calculus

- A *parallel composition* $P \mid Q$ allows two processes P and Q to run in parallel, which, combined with input and output rules, enables interaction between P and Q through common channels.
- The *restriction* $(\nu a)P$ creates a new name a that is scoped within P. Thus P is the only process, that is allowed to make use of a.
- Lastly the *conditional* process (if $M = N$ then P else Q) defines how a process may continue depending on the boolean statement $M = N$. If Q is nil, the conditional rule corresponds to a match rule in the π-calculus.

Both the input and the restriction rules bind terms to a process P such that they are local to the process P. For example, in the input process $M(x).P$, x is bound in P and for $(\nu a)P$, the name a is restricted and thus bound in P. In a process P, the set of *bound variables* are denoted by $\mathsf{bv}(P)$ and the set of *bound names* by $\mathsf{bn}(P)$. Conversely, the set of variables and the set of names that are not bound in P are called free and are denoted $\mathsf{fv}(P)$ and $\mathsf{fn}(P)$ respectively. Free names are globally known, also to an adversary, and free variables represent uninstantiated data. In case a process has no free variables, it is said to be *closed*. It may, however, still have free names.

The notation $P\{M/N\}$, performs a substitution in the process P, such that the term M is substituted with the term N. Since the calculus is polyadic, a substitution of multiple terms $\{x_1, \ldots, x_n/y_1, \ldots y_n\}$ substitutes each y_i with x_i for $i \in \{1, \ldots, n\}$.

2.1.2 Reduction Semantics for the Source Process Calculus

A *reduction* $P \longrightarrow Q$ is a transition denoting that P can be reduced to Q in a single step and $P \longrightarrow^* Q$ that P can be reduced to Q in any number of finite steps. I.e. that there exist some natural number $k \geq 0$ such that $P \longrightarrow^k Q$. *Structural congruence*, denoted $P \equiv Q$, means that there is a structural equivalence between P and Q. E.g. $P \mid Q \equiv Q \mid P$ denotes that, even though P and Q may have different behavior, the parallel composition on the left is structurally equivalent to the parallel composition on the right. *Alpha-conversion* means $P \equiv Q$ if Q can be obtained from P by renaming bound names in P

The behavior of the processes in table 2.1 is defined by the reduction semantics given in Table 2.2 on the next page. The semantics for parallel composition (PAR), restriction (RES) and structural congruence (STR) should be straightforward, however, the remaining rules may require elaboration.

Let M and N be terms, $\widetilde{M}$ and $\widetilde{N}$ sequences of n terms, a a name and P, P', Q and Q' processes, then the reduction semantics for the Source Process Calculus is given by the rules:

$$\text{COMM} \quad \overline{M}\langle\widetilde{N}\rangle \mid M(\widetilde{M}).Q \longrightarrow Q\{\widetilde{M}/\widetilde{N}\}$$

$$\text{REP} \quad \overline{M}\langle\widetilde{N}\rangle \mid !M(\widetilde{M}).Q \longrightarrow Q\{\widetilde{M}/\widetilde{N}\} \mid !M(\widetilde{M}).Q$$

$$\text{PAR} \quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$\text{RES} \quad \frac{P \longrightarrow P'}{(\nu a)P \longrightarrow (\nu a)P'}$$

$$\text{STR} \quad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

$$\text{IF TRUE} \quad \frac{P \longrightarrow P'}{\mathsf{if}\ M = M\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow P'}$$

$$\text{IF FALSE} \quad \frac{Q \longrightarrow Q'}{\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow Q'}$$

Table 2.2. Rules of the reduction semantics of the Source Process Calculus

The communication rule (COMM) shows that in a communication between two processes P and Q, Q receives $\widetilde{N}$ from P via the common channel M, which reduces to a substitution in Q such that $\widetilde{N}$ is substituted with $\widetilde{M}$. When a communication between processes, where the receiving process has an unbounded number of copies, the replication (REP) rule results in two processes running in parallel: The output process $\overline{M}\langle\widetilde{N}\rangle$ reduces to nil and is therefore left out, the input process $!M(\widetilde{M}).Q$ reduces to Q, where $\widetilde{M}$ is substituted with $\widetilde{N}$ while the replicated input process persists $!M(\widetilde{M}).Q$. When the terms that are compared are in fact syntactically identical, i.e $M = M$, which is obviously true for the IF TRUE case, the result is P which in turn is reducible to P'. For the IF FALSE rule, the two terms are distinct, i.e. $M = N$, and therefore the conditional process enters the else case, resulting in Q which in turn reduces to Q'.

2.1.3 Definition of Formal Secrecy

Using the process calculus above, we are now able to give a precise definition of secrecy in the formal view.

As informally stated by [Abadi et al., 2006] the concept of secrecy in the formal view is that a process P preserves the secrecy of datum M, if P never discloses M or any other information, that may be used to derive M. Even when communicating with an adversary, this notion of secrecy must hold.

Adopted from [Abadi et al., 2006; Abadi and Blanchet, 2005], we introduce a formal definition of secrecy based on the syntax and semantics of the process calculus described in section 2.1.1. The following definitions give precise notions of output and secrecy.

DEFINITION 2.3
A closed process P outputs M on a if and only if $P \longrightarrow^* \overline{a}\langle M\rangle.R \mid R'$ for some processes R and R'.

Thus, if a process P that has no free variables can be reduced to a process, that reveals an output action on a with M as a parameter, it is said to output M. Conversely, if P outputs M there exists a number of reductions from P to R such that R eventually outputs M.

DEFINITION 2.4
Let S be a finite set of names. The closed process Q is an S-adversary if and only if $\mathsf{fn}(Q) \subseteq S$.

As Q is a closed process, it does not have any free variables. If Q draws its set of names from the finite set of names S, Q is called an S-adversary and conversely if Q is an S-adversary, the free names of Q are contained in S

DEFINITION 2.5
Let S be a finite set of names. The closed process P preserves the secrecy of M from S if and only if $P \mid Q$ does not output M on a for any S-adversary Q and any $a \in S$.

Now the intuition is, that a closed process P preserves the secrecy of M from Q, if P does not output M over any of the names in S. This ensures that Q cannot receive M through any of its free names, since they are contained in S.

2.1.4 A Local Type System

To give a precise definition of formal secrecy, we use the local type system of [Abadi et al., 2006] – a reduced version of type system presented in [Abadi and Blanchet, 2001].

The types are given by the grammar in table 2.3.

$$T ::= D^{Secret} \mid C^{Secret}[T_1, \ldots, T_n] \mid C^{Public}[T_1, \ldots, T_n] \mid \mathit{Public}$$

Table 2.3. The type language of [Abadi et al., 2006]

Hence a type T can be either of four types:

- D^{Secret}: A type where the secrecy of the data is intended to be preserved.
- $C^{Secret}[T_1, \ldots, T_n]$: A *secret channel* for which tuples of n terms of type $T_1, \ldots, T_n$ may be communicated. Adversaries cannot communicate through this channel.
- $C^{Public}[T_1, \ldots, T_n]$: A *public channel* for which tuples of n terms of type $T_1, \ldots, T_n$ may be communicated. Adversaries may send data through, but cannot receive data from from channels that have this type.
- *Public*: A type that represents all public data.

We form a relation between the grammar above and the process calculus of Section 2.1.1 by introducing additional notations:

The subtyping relation $T \leq T'$ denotes that any term of type T may safely be used when a term of type T' is expected. The *least reflexive subtyping relation* is the least relation on types which satisfies that $C^{Public}[T_1, \ldots, T_n] \leq Public$ for all types T_i. Thus a type $C^{Public}[T_1, \ldots, T_n]$ can safely be used when a *Public* type is expected. See the type rules for names and variables below. That a term M is of type T, i.e. a term that has been declared as T, is denoted $M : T$.

A *type environment* E is a function that maps terms in processes to types. A *type judgement* $E \vdash P$ denotes the connection between processes and environment which states that a process P is *well-typed* with respect to E. This means that for each term in a process P its type has been declared in E. An environment E is said to be *well-formed* if and only if E is a list of typed terms, $M : T$, where each M is distinct in E. To denote that E is a well-formed environment, we write $E \vdash_\diamond$. Combining the above, we may define M to be a term of type T in the environment E by $E \vdash M : T$ and use $E \vdash_\diamond M : S$ to denote that S is the set of possible types of M in environment E.

The Type Rules

The type rules for the local type system above are defined in Table 2.4.

Let M be a term, u a name or a variable, a a name and P, Q processes. Let any variants of x, E and T be variables, environments and types respectively. *Public* and *Secret* are the types of data that are public and secret respectively and let $L \in \{Public, Secret\}$.

Well-formed environments:

$$\frac{}{\emptyset \vdash_\diamond} \qquad \frac{E \vdash_\diamond \quad u \notin \mathsf{dom}(E)}{E, u : T \vdash_\diamond}$$

Terms:

$$\frac{E \vdash_\diamond \quad (u : T) \in E}{E \vdash u : T} \text{ (Atom)} \qquad \frac{E \vdash M : T \quad T \leq T'}{E \vdash M : T'} \text{ (Subsumption)}$$

Sets of types of terms:

$$\frac{E \vdash_\diamond \quad (x : T) \in E}{E \vdash_\diamond x : \{T' | T' \leq T\}} \text{ (Name)} \qquad \frac{E \vdash_\diamond \quad (x : T) \in E}{E \vdash_\diamond x : \{T' \mid T' \leq T\}} \text{ (Variable)}$$

Processes:

$$\frac{E \vdash M : Public \quad E \vdash M_i : Public, \forall i \in \{1, \ldots, n\}}{E \vdash \overline{M}\langle M_1, \ldots, M_n \rangle} \text{ (Output } Public\text{)}$$

$$\frac{E \vdash M : C^L[T_1, \ldots, T_n] \quad E \vdash M_i : T_i, \forall i \in \{1, \ldots, n\}}{E \vdash \overline{M}\langle M_1, \ldots, M_n \rangle} \text{ (Output } C^L\text{)}$$

$$\frac{(a : Public) \in E \quad E, x_1 : Public, \ldots, x_n : Public \vdash P}{E \vdash\ ! = a(x_1, \ldots, x_n).P} \text{ (Input Public)}$$

$$\frac{(a : C^{Public}[T_1, \ldots, T_m]) \in E \quad E, x_1 : Public, \ldots, x_n : Public \vdash P \quad E, x_1 : T_1, \ldots, x_m : T_m \vdash P \text{ if } m = n}{E \vdash\ ! = a(x_1, \ldots, x_n).P} \text{ (Input } C^{Public}\text{)}$$

$$\frac{(a : C^{Secret}[T_1, \ldots, T_n]) \in E \quad E, x_1 : T_1, \ldots, x_n : T_n \vdash P}{E \vdash\ !\, = a(x_1, \ldots, x_n).P} \qquad \text{(Input } C^{Secret}\text{)}$$

$$\frac{E \vdash_\diamond}{E \vdash 0}\ \text{(Nil)} \qquad \frac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q}\ \text{(Parallel)} \qquad \frac{E, a : T \vdash P \quad T \neq D^{Secret}}{E \vdash (\boldsymbol{\nu} a)P}\ \text{(Restriction)}$$

$$\frac{E \vdash_\diamond M : S_1 \quad E \vdash_\diamond N : S_2 \quad D^{Secret} \notin S_1 \cup S_2 \quad \text{if } S_1 \cap S_2 \neq \emptyset \text{ then } E \vdash P \quad E \vdash Q}{E \vdash \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q} \qquad \text{(Cond)}$$

Table 2.4. Type rules for the local type system in Table 2.3

To provide a better understanding of the typing rules, we give a brief elaboration:

Well-formed environment: The empty environment is well-formed and if a term u is not in the domain of a well-formed environment E, then E extended with u is well-formed.

Atom: A name or variable u is of type T in environment E if E is well-formed and any tuple u of type T is contained in E.

Subsumption: If M is term of type T in an environment E and T is a subtype of T' then M is also a term of type T' in E.

Name: If a name u has type T in E then the set of all types that u may have, subtypes of T.

Variable: If a variable x has type T in E, then the set of all types that x may have, subtypes of T.

Output *Public*: Any set of public terms in E may be sent over a public channel in E.

Output C^L: A list of n terms with expected type $T_1, \ldots, T_n$ contained in E, can be sent over either a public or secret channel also in E. By the subtyping relation and combined with the previous rule, any type of public data can be sent over a public channel.

Input *Public*: An input process (which may be replicated), well-typed in E, can receive a list of public data via a public channel contained in E and an optional number of replications (one or more) of such process is allowed.

Input C^{Public}: In communication via a public channel a where m variables of type $T_1, \ldots, T_m$ is sent, a process that expects $n \neq m$ variables may only receive variables via a, if they are public, or if the number of variables and types in a are the same. In either case, the process is well-typed and it may be replicated an optional number of times.

Input C^{Secret}: If data is communicated via a secret channel, no adversary could have initiated the communication. Thus, if the secret channel accepts n terms of type $T_1, \ldots, T_n$ a process may receive n public or secret variables of matching type.

Nil: The **0** process is well-typed.

Parallel: The parallel composition of two well-typed processes is also well-typed.

Restriction: A process is not well-typed, if it can restrict a name, that is typed as secret data.

Cond: For two terms to be compared, it must be ensured that none of these are typed to D^{Secret}. Otherwise, information about the secret data could be revealed. Furthermore, P is only well-typed in E if the terms to be compared are typed from sets S_1 and S_2 that have common types, $S_1 \cap S_2 \neq \emptyset$. This way, the Cond process may be well-typed even though P is not, as P is never be executed, when S_1 and S_2 are disjoint sets.

Abadi et al. are the first to introduce the (Restriction) and (Cond) rules. They are present as a requirement to payload secrecy, that is presented in section 2.2.

The type system allows us to perform static analysis by type checking a process with respect to computational secrecy. Typechecking a process provides us with valuable information about the

intended usage of the process, enabling us to reject an ill-typed processes that otherwise preserves secrecy. Type checking is inexpensive in terms of time complexity as the type rules are compositional: to type check a composite process, we simply type check its immediate constituents. In this case, this means that type checking can be done in time linear in the size of the process.

To illustrate some applications of the type rules, we finish this section with two examples. Example 2.7 illustrates that a process can preserve secrecy, while the type system rejects it and the finally Example 2.8 presents a poorly designed and ill-typed process, that fails to preserve secrecy.

EXAMPLE 2.7 (An ill-typed process that preserves secrecy)
Consider the process

$$P_1 = (\boldsymbol{\nu} a)a(s).P$$

and assume that the process P has type errors in an environment E. Since a is a new local channel, $a(s)$ will never execute as no reduction can happen. Thus P_1 preserves the secrecy of any datum.

This is an example that shows, how a type system can reject processes that otherwise preserve secrecy. △

EXAMPLE 2.8 (An ill-typed process that does not preserve secrecy)
Cosider the process

$$P_2 = \overline{a}\langle s \rangle$$

where a is a public channel and s is the data intended to be kept secret. Obviously P_2 does not preserve the secrecy of s, as it allows output of s on a. We define the type environment as follows:

$$E = s : D^{Secret},$$
$$c : Public$$

By applying the (Output *Public*) rule we require, that

$$\frac{E \vdash c : Public \quad E \vdash s : Public}{E \vdash \overline{c}\langle s \rangle}$$

which is a contradiction with the fact that s is typed as D^{Secret} in E. Thus, P_1 is rejected by the type system. △

The Secrecy Theorem

The following theorem is a reduced version of the main secrecy theorem of [Abadi and Blanchet, 2001]. We present the theorem, however, the proof is omitted. First two sets of names are introduced:

RW ("Read-Write"): The set of all names declared as type *Public* in E. Using the names of RW an adversary is able to output and input.

W ("Write"): The set of all names that has been declared type $C^{Public}[T_1, \ldots, T_n]$ in E. Using the names of W an adversary is able to output only.

An adversary is a process, that has access to, while also restricted to, the terms in both RW and W. Using these, he is able to perform computations as well as sending and receiving data, that may reveal additional information.

Take a closed process P (a process with no free variables) that is well-typed in an environment E and the set of all names s declared as type D^{Secret} in E.

THEOREM 2.9 (The Secrecy Theorem, [Abadi and Blanchet, 2001])
Let P be a closed process. Suppose that $E \vdash P$ and $E \vdash s : D^{Secret}$ and let

$$RW = \{a \mid E \vdash a : Public\}$$
$$W = \{a' \mid E \vdash a' : C^{Public}\}$$

Then P preserves the secrecy of s from $RW \cup W$.

Thus the theorem states, that P preserves the secrecy of s from names in both RW and W. In other words, a closed process P that is well-typed in an environment E preserves the secrecy of a name s declared as secret data in E from the set of public names and public channels in E.

2.2 Computational Secrecy

The notion of computational secrecy is often related to the notion of *payload secrecy*. By a payload, we here mean a message, that is meant to be secret. An adversary may try to get access to a payload by means of communication with any other participant. Thus, by other routes he may be able to get access to secret data through e.g. communication. Therefore, it must be ensured, that payload secrecy is preserved throughout the system.

2.2.1 Overview

The first general definition of computational secrecy was provided by [Backes and Pfitzmann, 2005], using a model of machines with input and output. Thus, their definition does not build upon the π-calculus and instead builds on a cryptographic library as an abstract view of cryptography. This section studies their definition of computational secrecy in more detail.

Backes and Pfitzmann introduce a *replacement machine* R, running between the system of machines S and the user machine H, that takes parts of secret input messages (*payloads*) sent by H and replaces them with random messages. This means, that in every run, variables with random values are introduced to the system, when messages pass through the replacement machine. Even if an adversary A communicates with user H using secret channels, a system of machines S preserves payload secrecy if A cannot distinguish the system S, that has no replacement machine R, from the system S that has a replacement machine. Thus, if a system does not leak any information about the secret messages, the messages replaced by the replacement machine are indistinguishable from actual messages leaked by a user.

Expanding the idea of indistinguishable messages, [Backes and Pfitzmann, 2005] obtain perfect payload secrecy, by requiring that any user cannot distinguish whether it is communicating with a arbitrary adversary,

- in a system, that includes a replacement machine, or
- in a system with a machine F, that only forwards messages (no replacement or modification occurs) between the user and the system.

Backes and Pfitzmann finally define that computational secrecy is preserved with respect to messages if, for all adversaries, the viewpoint of the user, where R is placed between H and S is indistinguishable to a viewpoint, where F is placed between H and S. What can be observed in a system, is the value of input and output variables as well as the state of the machines. When every value is dependent on the payload and the payload may be replaced by a random payload of the same length, every variable passed through the system, will be a random variable. In this sense,

computational secrecy in a system is established, if there is no observable difference between a real system and a system where payloads are replaced with random data.

Neither [Abadi et al., 2006] nor [Laud, 2005] provides the formal definition of computational secrecy. It seems appropriate, to examine the details of the original definition from [Backes and Pfitzmann, 2005], in order to provide stronger ties between several of the sources used in this paper. The rest of this section will address this and is consequently primarily based on [Backes and Pfitzmann, 2005].

Denote by Sys a *real cryptographic system* represented by a set of *structures* $(\widehat{M}_{\mathcal{H}}, \mathcal{S}_{\mathcal{H}})$ containing the sets of *real cryptographic machines* $\widehat{M}$ and a set of ports $\mathcal{S}$ for which the set of users $\mathcal{H}$ may communicate through. In formal notation, denote the real cryptographic system by

$$\mathsf{Sys} = \left\{ (\widehat{M}_{\mathcal{H}}, \mathcal{S}_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1, \ldots, n\} \right\}$$

where n is the number of users. Note that Sys also includes cryptographic schemes used for asymmetric encryption, signatures, symmetric authentication, and symmetric encryption.

We introduce some additional terminology: The View of a machine holds a snapshot of information about the current state of the machine. This includes the current in- and outputs the machine can see as well as its internal states. A *configuration* Conf is a tuple $(\widehat{M}, \mathcal{S}, H, A)$ of machines, the users service ports, a user and an adversary, respectively. Furthermore, let $\mathsf{View}_{\mathsf{Conf}}(M)$ denote the view of a machine M in a configuration.

2.2.2 Computational Indistinguishability

Both [Abadi and Rogaway, 2002] and [Backes and Pfitzmann, 2005] adopt the definition of computational indistinguishability from [Yao, 1982], with the main difference being that Abadi and Rogaway define computational indistinguishability in terms of probabilistic distributions, while the definition of [Backes and Pfitzmann, 2005] deals with random variables. Since the two definitions are essentially the same, we adopt the earliest of the two and provide the details of the definition in the following.

Let $\eta \in \mathbb{N}$ be a *security parameter*, such that the string of 1 bits 1^η is the actual system input, i.e., the input to the system is of length η. Furthermore, let $\eta^{-\omega(1)}$ be some sufficiently small upper bound. Then we say that a function $\epsilon : \mathbb{N} \to \mathbb{R}$ is *negligible* if $\epsilon(\eta) \in \eta^{-\omega(1)}$. More precisely,

$$\eta^{-\omega(1)} = \left\{ \epsilon(\eta) \mid \forall\, c > 0 \quad \exists\, N_c \quad \text{such that} \quad \epsilon(\eta) \leq \eta^{-c} \quad \forall\, \eta \geq N_c \right\}.$$

An *ensemble* or *probability ensemble* is a collection of probability distributions on strings $D = \{D_\eta\}$ for each security parameter η. Let $x \xleftarrow{\mathcal{R}} D_n$ denote that x is a randomly sampled from D_η. Then the probability of a realization of an event E, given that x is sampled from D_η, is denoted $\Pr[x \xleftarrow{R} D_\eta : E]$.

DEFINITION 2.10 (Computational Indistinguishability [Abadi and Rogaway, 2002])
Let $D = \{D_\eta\}$ and $D' = \{D'_\eta\}$ be ensembles. Then we say that D and D' are *indistinguishable* or *computationally indistinguishable* and write $D \approx D'$ if for every polynomial-time adversary A, the function

$$\epsilon(\eta) := \left| \Pr[x \xleftarrow{\mathcal{R}} D_\eta : A(\eta, x) = 1] - \Pr[x \xleftarrow{\mathcal{R}} D'_\eta : A(\eta, x) = 1] \right|$$

is negligible.

An *adversary's advantage* $\epsilon(\eta)$ is the difference in the probabilities of the adversary "guessing" the sample taken from D_η minus the probabilities of the adversary "guessing" the sample taken from D'_η. Thus, if the adversary's advantage is negligible, then D and D' are said to be indistinguishable. This implies that when $\epsilon(\eta)$ decreases, the computational load of the adversary increases.

Expanding the idea of computational indistinguishability, [Backes and Pfitzmann, 2005] obtains computational secrecy by means of several definitions. The following sections provide an overview of these requirements in order to outline how this important definition is obtained.

2.2.3 Payload Selection Function

As briefly introduced in Section 2.2, parts of messages are replaced by the replacement machine. If the same message is sent over multiple communications we need to make sure that the replacement in this message is consistent. This is managed by a payload selection function that for each message always divides the message into "chunks" (payloads) in exactly the same way.

DEFINITION 2.11 (Payload Selection Function [Backes and Pfitzmann, 2005])
A *payload selection function* is a function that assigns every string l a potentially empty set of non-overlapping substrings of l.

Thus, two selection functions will select the same set of substrings for any given message.

2.2.4 The Replacement and Forwarding Machines

Let $\mathcal{S}$ be the set of ports (or *service ports*) that is available to the user, i.e. these ports are the user interface and let $\mathcal{S}^C$ denote the complement port set of $\mathcal{S}$, i.e. the set of ports the machines use for communication. Finally, in order to establish communication between the machines and the replacement and forwarding machines we consistently redirect the ports in $\mathcal{S}^C$, by renaming, to another set of ports $\mathcal{S}'$. To ensure that the user interface is not changed, when the replacement and forwarding machine are introduced, both of these make use of the set of ports $\mathcal{S}$ and $\mathcal{S}'$.

The idea of the *replacement machine* as well as the *forwarding machine*, is that they are placed between the user and the set of machines in the system, letting them capture all messages sent from the user through ports in $\mathcal{S}$ to the machines through ports in $\mathcal{S}'$ and vice versa. This enables both machines to possibly modify messages before passing them on, while the interface remains unchanged.

In the following two definitions, let $\mathcal{S}$ be a set of ports, f and g be payload selection functions and define a function $L : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$, whose value determines the allowed number of inputs at a certain port as well as the number of bits to be read from each input.

DEFINITION 2.12 (Replacement Machine [Backes and Pfitzmann, 2005])
The *replacement machine* $R_{\mathcal{S},f,g,L}$ for $\mathcal{S}, f, g$ and L has the sets of ports $\mathcal{S}$ and $\mathcal{S}'$, an initially empty set T called a *replacement table* and the following transition rules:

- For an input message l, at a port in $\mathcal{S}$, assign $\{m_1, \ldots, m_n\} := f(l)$. For every tuple $(m_i, n_i) \in T$, replace the payload m_i in l with n_i. For the remaining payloads m_j, set

$$n_j \xleftarrow{\mathcal{R}} \{0,1\}^{\text{len}(m_j)} \setminus \{n \mid \exists\, m : (m,n) \in T\}$$

Set $T := T \cup \{(m_j, n_j)\}$ and replace n_j with m_j in l. Output the resulting string l' on the corresponding port in $\mathcal{S}'$.

- For an input message l, at a port in $\mathcal{S}'$, assign $\{n_1, \ldots, n_m\} := g(l)$. For every tuple $(m_i, n_i) \in T$, replace the payload n_i in l with m_i. Output the resulting string through the corresponding port in $\mathcal{S}$.

We further define that $R_{\mathcal{S},f,g,L}$ accepts $L(\eta)$ inputs at each port in $\mathcal{S} \cup \mathcal{S}'$, with η being the security parameter, and that it reads the first $L(\eta)$ bits of each input.

In the first case, every payload m_i, that the user sends through $\mathcal{S}$, is replaced by a predetermined random payload n_i, defined in the replacement table. If no such rule exists, a new tuple is created containing m_i and a new unique random n_i, which is then added to the replacement table. In the second case, for every tuple (m_i, n_i) in the replacement table, the payload $n_i$that the machines in the system sends though $\mathcal{S}'$ is replaced with its corresponding payload m_i. All other payloads in the message are left unchanged.

DEFINITION 2.13 (Forwarding machine [Backes and Pfitzmann, 2005])
The *Forwarding Machine* $F_{\mathcal{S},L}$ for $\mathcal{S}$ and L has the sets of ports $\mathcal{S}$ and $\mathcal{S}'$. For an input message l at a port in $\mathcal{S}$ or $\mathcal{S}'$, it forwards l to the corresponding port in $\mathcal{S}'$ or $\mathcal{S}$ respectively.

$F_{\mathcal{S},L}$ accepts $L(\eta)$ inputs at each port in $\mathcal{S} \cup \mathcal{S}'$, with η being the security parameter and that it reads the first $L(\eta)$ bits if each input.

The forwarding machine F is introduced to the original system, in order to make the operation of such a system, identical to the system that includes a replacement machine. The idea is, that the user should not be able to determine which of the systems it is communicating through.

2.2.5 Payload Secrecy and Computational Secrecy

The following definition defines perfect and computational secrecy of payloads.

DEFINITION 2.14 (Reactive Payload Secrecy [Backes and Pfitzmann, 2005])
Let a system Sys, be given. Let f and g be mappings from the set $\{\mathcal{S} \mid \exists\, (\widehat{M}, \mathcal{S}) \in \mathsf{Sys}\}$ to the set of payload selection functions. Let $f_{\mathcal{S}}$ and $g_{\mathcal{S}}$ replace $f(\mathcal{S})$ and $g(\mathcal{S})$. Let $(\widehat{M}, \mathcal{S}) \in \mathsf{Sys}$ be arbitrary and let $(\widehat{M}', \mathcal{S}')$ be the structure where the port names of ports in $\mathcal{S}$ are consistently replaced on the machines $\widehat{M}$ as for the set of ports $\mathcal{S}'$ in $R_{\mathcal{S},f_{\mathcal{S}},g_{\mathcal{S}},L}$. Then we say that the payload messages selected by $f_{\mathcal{S}}$ and $g_{\mathcal{S}}$ are

1. *perfectly secret* in $(\widehat{M}, \mathcal{S})$, written $(\widehat{M}, \mathcal{S}) = [f_{\mathcal{S}}, g_{\mathcal{S}}](\widehat{M}, \mathcal{S})$, if and only if for all functions $L : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ and for all configurations $\mathsf{Conf} = (\widehat{M}' \cup R_{\mathcal{S},f_{\mathcal{S}},g_{\mathcal{S}},L}, \mathcal{S}, H, A)$ and $\mathsf{Conf}' = (\widehat{M}' \cup \{F_{\mathcal{S},L}\}, \mathcal{S}, H, A)$ (i.e. with the same user H and adversary A), we have $\mathsf{View}_{\mathsf{Conf}}(H) = \mathsf{View}_{\mathsf{Conf}'}(H)$.
2. *computationally secret* in $(\widehat{M}, \mathcal{S})$, written $(\widehat{M}, \mathcal{S})[f_{\mathcal{S}}, g_{\mathcal{S}}](\widehat{M}, \mathcal{S})$, if and only if the above holds for all polynomial bounded functions L, polynomial-time users H, polynomial-time adversaries A and with equality of views replaced by indistinguishability of views.

We say that *the payload messages selected by f and g are perfectly respectively computationally secret in* Sys, written

$$\mathsf{Sys} = [f, g]Sys, \text{ respectively } \mathsf{Sys} \approx [f, g]\mathsf{Sys}$$

if and only if

$$(\widehat{M}, \mathcal{S}) = [f_{\mathcal{S}}, g_{\mathcal{S}}](\widehat{M}, \mathcal{S}) \text{ respectively } (\widehat{M}, \mathcal{S}) \approx [f_{\mathcal{S}}, g_{\mathcal{S}}](\widehat{M}, \mathcal{S})$$

holds for all $(\widehat{M}, \mathcal{S}) \in \mathsf{Sys}$

Thus, we have perfect secrecy of payloads if and only if the view of the user in a configuration, that includes a replacement machine, is the same as the view of the user in a configuration, that includes a forwarding machine. This implies computational secrecy of payloads if and only if the notion of perfect secrecy holds for polynomial-time users and adversaries.

Since the definitions of computational indistinguishability in [Abadi and Rogaway, 2002] and [Backes and Pfitzmann, 2005] are the same, we are able to directly relate computational indistinguishability as defined by [Abadi and Rogaway, 2002] with computational secrecy as in [Backes and Pfitzmann, 2005].

2.3 From Formal to Computational Secrecy

We now present the main result of [Abadi et al., 2006]. We skip the finer details leading up to it and focus on the description of the theorem itself.

Abadi et al. makes use of the process calculus by [Laud, 2005] as an intermediate language by translating their source calculus to his and then rely on the soundness of his type system with respect to a cryptographic library, thus achieving payload secrecy.

Informally [Abadi et al., 2006] defines payload secrecy in the following sense: Let $C_n = \langle \mathsf{S}, \mathsf{H}, \mathsf{A} \rangle$ denote a *configuration*, where

- S is a system of machines $(\mathsf{P}_i)_{i=1,\ldots,n}$, that executes the actual instructions of a protocol. Each machine P_i is connected to a *library machine* $(\mathsf{M}_i)_{i=1,\ldots,n}$, i.e., a machine that manage and run all cryptographic algorithms on behalf of P_i,
- H is a user machine with connections on the unconnected ports in S and
- A is an adversary machine with connections to the remaining pots.

Laud uses input processes $I_{i=1\ldots n}$ for programming the machines $(\mathsf{P}_i)_{i=1,\ldots,n}$. By the definition of payload secrecy by [Backes and Pfitzmann, 2005] as described in Section 2.2.5, Laud's main result states that if each input process I_i is well-typed in some environment, then the configuration C_n preserves payload secrecy of messages sent from the user H to the system S.

Denote by $\tilde{a}_i$ the set of all free names used as input in the process P_i, such that $\tilde{a}$ is the union of all $\tilde{a}_i$ and let $\tilde{b}_i = \mathsf{fn}(P_i) \setminus \tilde{a}_i$, such that $\tilde{b}$ is the set of all $\tilde{b}_i$. The names in $\tilde{b}$ represent data from either the attacker or the user. Let $\tilde{a}_{RW}$ denote the set of all names that an adversary may output and input on, such that $\tilde{a}_{RW} \cap \tilde{a} = \emptyset$ and denote by $\tilde{a}_W \subseteq \tilde{a}$ the set of names, that an adversary only may output on. See Section 2.1.4 for the definition of RW and W. Finally, $\tilde{s}$ denotes the set $\tilde{b} \setminus \tilde{a}_{RW}$ of names, that are user secrets.

The following theorem shows, that the translation presented by Abadi et al. is type-preserving.

THEOREM 2.15 (Computational Secrecy by Local Typing, [Abadi et al., 2006])
Let $P = \prod_{i=1\ldots n} P_i$. Let the machines $(\mathsf{P}_i)_{0\ldots n}$ implement the source processes $(P_i)_{1\ldots n}$ with initialization parameters $\tilde{a}, \tilde{a}_{RW}, \tilde{a}_W$ and $\tilde{s}$.

Let E be the source typing environment that contains

- $(s : D^{Secret})$ for each $s \in \tilde{s}$
- $(a : C^{Secret}[T])$ for each $a \in \tilde{a} \setminus \tilde{a}_W$
- $(a : C^{Public}[T])$ for each $a \in \tilde{a}_W$
- $(b : \mathit{Public})$ for each $b \in \tilde{a}_{RW}$

If $E \vdash P$, then the concrete system $(\mathsf{P}_i, \mathsf{M}_i)_{i=0\ldots n}$ preserves payload secrecy of $\tilde{s}$.

This result is truly significant in relating the formal and computational paradigms, as it states that secrecy preserving processes, that are well-typed in the formal settings, preserves computational secrecy.

Authenticity 3

An informal exemplification of authenticity from [Blanchet, 2002] states:

> Intuitively, a protocol authenticates A to B if, when B thinks he talks to A, then he actually talks to A.

This chapter takes a closer look at authenticity in the formal as well as the computational paradigms and studies a vital theorem, that shows a relation between formal secrecy and formal authenticity.

3.1 Formal Authenticity

This chapter assumes, that the reader is familiar with the syntax and semantics of the process calculus from Section 2.1.1.

In the formal view, authenticity is often formalized in terms of correspondence assertions. These are special begin and end events, such that only when corresponding begin and end events have been emitted, is authenticity satisfied. The initial idea of correspondence assertions was proposed by [Woo and Lam, 1993], who developed a syntax and semantics for these. Based on the work of [Woo and Lam, 1993; Abadi and Gordon, 1997], [Gordon and Jeffrey, 2003] proposed a method for checking authenticity properties of cryptographic protocols represented in the spi-calculus, with the means of correspondence assertions and typechecking.

As many of the following sections are based on [Blanchet, 2002], we also adopt how Blanchet defines authenticity:

> When B thinks he has run the protocol with A, he emits a special event end. When A thinks she runs the protocol with B, she emits another event begin. Authenticity is satisfied when B cannot emit his end event without A having emitted her begin event.

Blanchet shows that "it is easy to prove authenticity by using secrecy" [Blanchet, 2002]. We will pursue this idea and give an overview of his approach.

3.1.1 Correspondence Assertions

To obtain authenticity in the formal sense, Blanchet developed a process calculus with correspondence properties by including four event emitting processes not included in the Source Process Calculus in Table 2.1 on page 6. In fact, our process calculus which is based on [Abadi et al., 2006] is a reduced version of Blanchet's calculus. Thus, we "reintroduce" the event emitting processes in Table 3.1 on the following page.

Let M be a term and P a process. The process calculus is extended with the processes:

Processes	P	$::=$	$\ldots$	
			$\mathsf{begin}(M).P$	Begin event
			$\mathsf{end}(M).P$	End event
			$\mathsf{begin\_ex}(M)$	Executed begin event
			$\mathsf{end\_ex}(M)$	Executed end event

Table 3.1. The extended syntax of the Source Process Calculus that includes event emitting processes

Blanchet extends the process calculus with emitting events to give specifications for authenticity. The processes are divided into two groups:

- The processes $\mathsf{begin}(M).P$ and $\mathsf{end}(M).P$ execute the begin event or the end event respectively and continue as P.
- The processes $\mathsf{begin\_ex}(M)$ and $\mathsf{end\_ex}(M)$ states that a $\mathsf{begin}(M).P$ or an $\mathsf{end}(M).P$ event has been emitted.

Informally, we can consider a begin event as the start of an authentication between processes where $\mathsf{begin}(M).P$ means that a process A within P initiates a session with the parameter M. Another process B within P may then perform computations on M and if it believes that it has initiated a communication with B it emits the end event to indicate that the authentication has been achieved. In practice this may be an initiation of communication using some public- or shared-key cryptographic algorithm If the protocol is correct, B only emits the end event when A has emitted the begin event. .

3.1.2 Reduction Semantics for Correspondence Assertions

The reduction rules of the processes in Table 3.1 are given in Table 3.2.

Let M be a term and P a process, then the reduction semantics for the emitting events is given by the rules:

$$\text{BEGIN} \quad \mathsf{begin}(M).P \longrightarrow \mathsf{begin\_ex}(M) \mid P$$

$$\text{END} \quad \mathsf{end}(M).P \longrightarrow \mathsf{end\_ex}(M) \mid P$$

Table 3.2. Rules of the reduction semantics of the emitting events

Once the begin or end event has been emitted and the processes continue as P, we may emit $\mathsf{begin\_ex}(M)$ or $\mathsf{end\_ex}(M)$, to indicate that a begin or an end event, at some point, has been executed.

3.1.3 Injective and Non-injective Agreement

Recall that a closed process has no free variables, yet, it may have free names. We extend the definition of an S-adversary (see Definition 2.4 on page 8) to include Blanchet's event emitting processes.

DEFINITION 3.3 (*S*-adversary, [Blanchet, 2002])
Let S a finite set of names. The closed process Q is an S-adversary if and only if $\mathsf{fn}(Q) \subseteq S$ and Q does not contain begin, end, begin_ex or end_ex events.

The set S is the set of public names including those that a possible target process has made freely available. Hence, an S-adversary is a process that is able to communicate with other processes using the names in S. Since the names in S are public, none of the free names, that the adversary has access to, are restricted to other processes. To avoid that an S-adversary violates non-injective agreement, he is disallowed from emitting events. Otherwise, he would be able to emit an end event, with some arbitrary parameter, that has no prior begin event. This would not constitute an actual attack, since the adversary would gain no information of value in this way. In other words, we consider the adversary to be an outside process, that has access to the free names in a process that defines the protocol. Since the actual authentication happens within this process, it would make little sense to give the adversary the ability to emit events, as this would have no effect on the execution inside protocol process.

There are two ways in which we consider the correspondence between begin and end events

1. *Non-injective agreement*: Every execution of $\mathsf{end}(M).P$ has a corresponding execution of $\mathsf{begin}(M).P$. Thus, if an end event has been emitted, then the corresponding begin event has been emitted. However, one $\mathsf{end}(M)$ event may be emitted as a response to several $\mathsf{begin}(M)$ events, which means that there may be fewer $\mathsf{end}(M)$ events than there are $\mathsf{begin}(M)$ events. See Figure 3.1.
2. *Injective agreement*: For an injective agreement, the number of end events is smaller than or the same as the number of begin events. Thus, we can assume that each $\mathsf{end}(M)$ is matched by a distinct $\mathsf{begin}(M)$. In addition, if a $\mathsf{begin}(M)$ event has been emitted, there is no guarantee that an $\mathsf{end}(M)$ event will be emitted. See Figure 3.2.

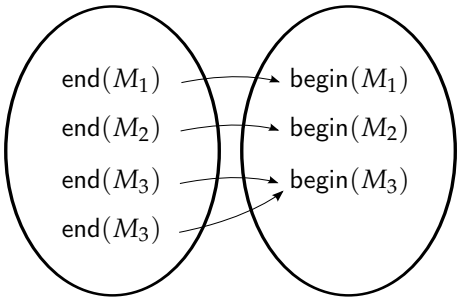

Figure 3.1. Non-injective agreement. Every $\mathsf{end}(M)$ event implies that $\mathsf{begin}(M)$ has been emitted.

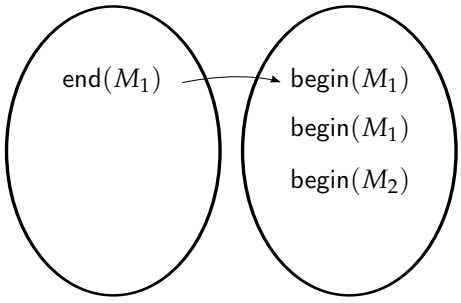

Figure 3.2. Injective agreement. The number of end events are at most the number of begin events.

How a process satisfies non-injective agreement is defined in the following manner:

DEFINITION 3.4 (Satisfying Non-Injective Agreement, [Blanchet, 2002])
The closed starting process P satisfies non-injective agreement with respect to S-adversaries if and only if for any S-adversary Q, for any P' such that $P \mid Q \ (\rightarrow \cup \equiv)^* \ P'$, for any M, if $\mathsf{end\_ex}(M)$ occurs in P', then $\mathsf{begin\_ex}(M)$ also occurs in P'.

Consider the process P to be a process that handles the authorization. Now, for any process P' that has seen both a begin and an end event, and P, by a reduction sequence can reach P', then we know that P' is in a state where authorization has completed. Consequently, P satisfies non-injective agreement. Blanchet's definition is said to be *robust*, since P runs in parallel with any given S-adversary Q.

A process satisfying injective agreement is defined in the following way:

DEFINITION 3.5 (Satisfying Injective Agreement, [Blanchet, 2002])
The closed starting process P satisfies injective agreement with respect to S-adversaries if and only if for any S-adversary Q, for any P' such that $P \mid Q\ (\rightarrow \cup \equiv)^* P'$, for any M, the number of occurrences of $\mathsf{end\_ex}(M)$ in P' is at most the number of occurrences of $\mathsf{begin\_ex}(M)$ in P'.

This definition is similar to Definition 3.4 with the main difference, that if P' does not emit more $\mathsf{end\_ex}(M)$ events than $\mathsf{begin\_ex}(M)$ events, then P satisfies injective agreement. Though, it should be noted that each end event corresponds to only one begin event.

In both of the above definitions, the restrictions in P' are renamed such that they bind pairwise different names, and names different from free names. This simply to avoid name clashes between, for example, a free name x and another restricted name x. This can be done using renaming such that

$$\mathsf{begin}(x).\overline{a}\langle x\rangle \mid (\boldsymbol{\nu} x)x(y).\mathsf{end}(y)$$

by alpha conversion becomes

$$\mathsf{begin}(x).\overline{a}\langle x\rangle \mid (\boldsymbol{\nu} z)z(y).\mathsf{end}(y)$$

3.2 Relating Formal Secrecy and Formal Authenticity

A *context* $C[]$ is a process with "holes", i.e., with placeholders for unknown processes. Contexts allows us to replace a process within the context with another version of the process.

With the following theorem, we extend the theorem by [Blanchet, 2002], that originally showed, that authenticity can be verified by application of formal secrecy. Our version supports the reversed implication, namely that formal authenticity implies formal secrecy.

THEOREM 3.6
Let

$$P = C[\mathsf{begin}(M).P_1,\ \mathsf{end}(M').P_2]$$

be a closed starting process (where $C[]$ is any context) and assume that these are the only begin and end events in P. Let

$$P'(N) = C[\mathsf{if}\ M = N\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ P_1,\ \overline{c}\langle(\mathsf{auth}, M')\rangle.P_2]$$

where auth is a new term. For all closed terms N, $P'(N)$ preserves the secrecy of (auth, N) from S, where $c \in S$ if and only if P satisfies non-injective agreement with respect to S-adversaries.

We give the proof below, but first we study the original theorem itself. The important thing to understand is the transformation from P to P' and especially what is obtained by P'. We start by defining a process P, as a context of two processes, $\mathsf{begin}(M).P_1$ and $\mathsf{end}(M').P_2$, with uninstantiated terms M and M'. If the terms M and M' are instantiated to the same, both $\mathsf{begin}(M).P_1$ and $\mathsf{end}(M').P_2$ is emitted, which implies a successful authentication. P is transformed into the process P', such that P' behaves in the same way as P, if M is not instantiated to N, while preserving the secrecy of N. There are two ways P' can behave for any N:

- If N has been instantiated to the same as M, the conditional process reaches $\mathbf{0}$, implying that a $\mathsf{begin}(M)$ is never emitted by P, since P_1 is never reached in P'. Consequently, P never reaches P_2, since $\mathsf{end}(M')$ cannot be emitted. As Therefore, P' preserves the secrecy of N, as N is never published on c.
- If N is different from M, the conditional process reaches P_1, which means that $\mathsf{begin}(M)$ must have been emitted by P. Since this is the case, an $\mathsf{end}(M')$ is also by P emitted, implying that M and M' have been instantiated to the same term. Thus, P' publishes $\mathsf{auth}(M')$ on c while the secrecy of N is preserved.

In both cases (for any N), P satisfies non-injective agreement with respect to S-adversaries, as there are two cases: Either the both the begin and end event are emitted or none of them are emitted.

PROOF

In the following, we denote by N the payload of the process $P'(N)$. We need to show two cases: 1. If $P'(N)$ preserves the secrecy of N, then P satisfies non-injective agreement and 2. $P'(N)$ preserves the secrecy of N, only if P satisfies non-injective agreement.

1. **(if)** Assume that P' preserves the secrecy of N for every payload N. We then show, by contradiction, that P satisfies non-injective agreement with respect to S-adversaries Q. Consider a sequence of reductions, where P does not reach a $\mathsf{begin}(M)$. Then, in order to satisfy non-injective correspondence, no $\mathsf{end}(M)$ is allowed to be emitted. If this is the case, it must have been executed by the sequence of reduction:

   $$P \longrightarrow^* \ (\boldsymbol{\nu} n)(\mathsf{end\_ex}(M') \mid Q)$$

   where M' is instantiated to some value N'. This is not possible, since P' satisfies secrecy for any payload N', as P' will either not output anything on $c \in S$ or output something other than the payload on c. However, by choosing the payload to what N' was instantiated to, we reach a contradiction, as it is not possible for P' to output the payload N', while outputting something that is not N'. Hence, P' cannot output anything at all on c. Consequently, the original process P is never able to emit the end event, thus satisfying non-injective agreement with respect to S-adversaries.
2. **(only if)** Let N be an arbitrary payload. Assume that P satisfies non-injective agreement with respect S-adversaries Q and that $P'(N)$, for that sake of contradiction, does not preserve the secrecy of N. Then, for any sequence of reductions, where P at some point will emit an end event:

   $$P \longrightarrow^* \ (\boldsymbol{\nu} n)(\mathsf{end\_ex}(M')) \mid Q$$

   there has previously been emitted a $\mathsf{begin}(M)$ event in P, such that

   $$P \longrightarrow^* \ (\boldsymbol{\nu} m)(\mathsf{begin\_ex}(M')) \mid Q \longrightarrow^* \ (\boldsymbol{\nu} m)(\mathsf{end\_ex}(M')) \mid Q$$

   where M and M' has been instantiated to the same term. However, as $P'(N)$, by assumption, does not preserve the secrecy of N, at the point where P would emit an end event, $P'(N)$ will emit N on c. This implies that M' was instantiated to the same as N. As the reductions for P and P' are equivalent, i.e. P emits an end event, when P' outputs on c, M and M' must have been instantiated to N. This implies that $M' = N$ is true, which means that P

never emits the begin event, followed by P_1, since P' never reaches P_1. This contradicts, that M' was instantiated to the same as N. Thus, our assumption that $P'(N)$ does not preserve the secrecy of N is false. □

The last part of the proof shows, that if non-injective agreement is satisfied by a closed process P, i.e. that P satisfies authenticity then the transformed process $P'(N)$ preserves the secrecy of N.

To further elaborate on Theorem 3.6, we give an example, in which the theorem is applied to a minimal process.

EXAMPLE 3.7
Consider the process

$$\mathsf{begin}(x).\overline{a}\langle x\rangle \mid a(y).\mathsf{end}(y)$$

The left event initiates authentication with parameter x and transmits x on a. The right process receives x on a and completes the authentication. It must be ensured that, if the begin event never takes place, then neither does the end event. By transforming the process above using Theorem 3.6, we get

$$\mathsf{if}\ x = N\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \overline{a}\langle x\rangle \mid a(y).\overline{c}\langle(\mathsf{auth}, y)\rangle$$

for any term N and a new channel c. If begin cannot be emitted, the conditional process ends in a $\mathbf{0}$ state and never outputs N on a. Consequently the secrecy of N is preserved as nothing is transmitted on c, thus the original process satisfies non-injective agreement. △

A much extended work in this area, also composed by Blanchet, is presented in [Blanchet, 2009, Theorem 4], while the proof is provided in [Blanchet, 2008, Theorem 3] only.

3.3 Computational Authenticity

In the computational paradigm we use the same notion of authenticity, as in Section 3.1. Namely that "if some events have been executed, then some other events have been executed" as Blanchet expresses it in [Blanchet, 2007]. In order to achieve computational authenticity, we make use of this particular work, which defines computational authenticity based on *non-injective correspondences* (see Section 3.1.3 for the definition of non-injective agreement). Informally non-injective correspondences means that "if some events have been executed, then some other events have been executed at least once". Computational authenticity is then defined, such that if the probability that a correspondence is not satisfied, is negligible (see Section 2.2.2 on page 13), then a process satisfies the correspondence. This section provides a brief overview of, how this definition is obtained in [Blanchet, 2007].

3.3.1 Non-injective Correspondences

We introduce the notion of an *event* $e(M_1, \ldots, M_m)$, where $M_1, \ldots, M_m$ is a sequence of terms, such that we may denote by $\mathsf{event}(e(M_1, \ldots, M_m))$, the observation that an event has been executed. Denote by $\mathcal{E}$ *a sequence of events* $e(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are bitstrings. To state that a term M evaluates to a bitstring a, we write $\rho, M \Downarrow a$, where ρ is a mapping from terms to bitstrings.

Logic formula are introduced in order to express non-injective correspondence in terms of implication, denoted $\psi \Rightarrow \phi$, where ψ and ϕ are formulae that may contain events. Informally, this notation means, that if a formula ψ evalutes to true, then the formula ϕ also evaluates to true. A logic formula is defined in Table 3.3 on the next page.

Formula	ϕ	::=	M	Term
			$\mathsf{event}(e(M_1, \ldots, M_m))$	Event
			$\phi_1 \wedge \phi_2$	Conjunction
			$\phi_1 \vee \phi_2$	Disjunction

Table 3.3. Logic formulae

Each of the terms $M, M_1, \ldots, M_m$ in a formula represent a single term, i.e., sequences of terms may not be derived from a term. When defined to be variables, they are distinct from variables in processes (variables in processes might be private to the process). We say that the formula $\mathsf{event}(e(M_1, \ldots, M_m))$ holds when the event $e(M_1, \ldots, M_m)$ has been executed and in general denote by $M \Downarrow \mathsf{true}$ that the formula M holds when M evaluates to true.

The satisfaction relation for logic formulae, denoted by $\rho, \mathcal{E} \vdash \phi$ [1], states that a mapping, ρ from terms to bitstrings and a sequence of events $\mathcal{E}$ satisfies a formula ϕ. This s defined as follows:

- $\rho, \mathcal{E} \vdash M$ if and only if $\rho, M \Downarrow \mathsf{true}$. Thus, if the formula M holds $\rho, \mathcal{E}$ evalutes to M.
- $\rho, \mathcal{E} \vdash \mathsf{event}(e(M_1, \ldots, M_m))$ if and only if for all $j \leq m$ such that $\rho, M_j \Downarrow a_j$ and $e(a_1, \ldots, a_m) \in \mathcal{E}$. If a subset of the bitstrings $a_1, \ldots, a_m$ in any sequence of events $\mathcal{E}$ is reduced from corresponding terms by ρ, then ρ and $\mathcal{E}$ satisfies that these events have been executed.
- $\rho, \mathcal{E} \vdash \phi_1 \wedge \phi_2$ if and only if $\rho, \mathcal{E} \vdash \phi_1$ and $\rho, \mathcal{E} \vdash \phi_2$. If a conjunction of formulae is satisfied, then each of the formulae are satisfied.
- $\rho, \mathcal{E} \vdash \phi_1 \vee \phi_2$ if and only if $\rho, \mathcal{E} \vdash \phi_1$ or $\rho, \mathcal{E} \vdash \phi_2$. If a disjunction of formulae is satisfied, then at least one of the formulae are satisfied.

Note that the following definitions only rely on a process calculus with a reduction semantics that captures events, since the notion of correspondence can be defined independently of the process calculus. We define such an extended event capturing semantics as follows:

$$P \xrightarrow{\mathcal{E}} P' \quad \text{where} \quad |\mathcal{E}| > 1, \quad \text{if}$$
$$P \longrightarrow^* P_1 \xrightarrow{e_1} P_2 \quad \cdots \quad P_{k-1} \xrightarrow{e_k} P_k \longrightarrow^* P' \quad \text{where} \quad \mathcal{E} = e_1 \ldots e_k$$

In the following, let ψ denote a conjunction of events and let $\mathsf{var}(\psi)$ denote the set of variables that occurs in ψ.

DEFINITION 3.9 (Satisfying Correspondence for Events, [Blanchet, 2007])
The sequence of events $\mathcal{E}$ *satisfies the correspondence* $\psi \Rightarrow \phi$, denoted $\mathcal{E} \vdash \psi \Rightarrow \phi$, if and only if for all ρ defined on $\mathsf{var}(\psi)$, such that $\rho, \mathcal{E} \vdash \psi$, there exists an extension ρ' of ρ to $\mathsf{var}(\phi)$, such that $\rho', \mathcal{E} \vdash \phi$.

Informally, the definition states that if $\rho, \mathcal{E}$ satisfies ψ and $\rho', \mathcal{E}$ satisfies ϕ then $\mathcal{E}$ satisfies $\psi \Rightarrow \phi$, where ρ' extends the domain of ρ to cover the variables of both ψ and ϕ. The idea is that, if we can construct a function ρ' that extends ρ (for all ρ) such that $\rho, \mathcal{E}$ satisfies ψ and $\rho', \mathcal{E}$ satisfies ϕ then $\mathcal{E}$ satisfies the correspondence $\psi \Rightarrow \phi$.

[1] Please note, that we adopt this particular notation from [Blanchet, 2007] and that it is not to be confused with the notion of type judgment from Section 2.1.4

We denote by $\mathbb{C}$ a *probabilistic reduction relation on semantic configurations*, i.e., a reduction relation where each reduction from one configuration to another is assigned a probability. Denote by $\mathsf{initConfig}(Q)$ the *initial configurations* of the process Q. Let C denote an *evaluation context*, with a "hole" representing an adversary. By replacing the process Q with the hole in C we obtain the process $C[Q]$, to denote a process Q in the presence of an adversary. Both Q and the adversary are processes that run in probabilistic polynomial time. Denote by V a set of variables. We say that C *is acceptable for* (Q, V), where V is the set of variables in the process Q, if and only if C contains no events.

If there exists a semantic configuration $\mathbb{C}$ and a sequence of events $\mathcal{E}$, then

$$\Pr[\exists(\mathbb{C}, \mathcal{E}) \text{ such that } \mathsf{initConfig}(C[Q]) \xrightarrow{\mathcal{E}} \mathbb{C} \wedge \mathcal{E} \nvdash \psi \Rightarrow \phi]$$

is the probability, that the reduction sequence $\xrightarrow{\mathcal{E}}$ (that tracks events as reductions are performed), where the initial configuration of Q, in the presence of an adversary, $\mathsf{initConfig}(C[Q])$ is reduced to $\mathbb{C}$ and $\mathcal{E}$ does not satisfy the correspondence $\psi \Rightarrow \phi$. Informally, this is probability that a process, by a sequence of events from its initial state, does not satisfy the correspondence $\psi \Rightarrow \phi$. The following definition relates non-injective correspondence with computational authenticity.

DEFINITION 3.10 (Satisfying Correspondence for Processes, [Blanchet, 2007])
The process Q *satisfies the correspondence* $\psi \Rightarrow \phi$ with public variables V if and only if for all evaluation contexts C acceptable for (Q, V),

$$\Pr[\exists(\mathbb{C}, \mathcal{E}) \text{ such that } \mathsf{initConfig}(C[Q]) \xrightarrow{\mathcal{E}} \mathbb{C} \wedge \mathcal{E} \nvdash \psi \Rightarrow \phi]$$

is negligible.

Thus, a process Q satisfies the correspondence $\psi \Rightarrow \phi$, if the probability that it does not satisfy $\psi \Rightarrow \phi$, is negligible.

3.4 From Formal to Computational Authenticity

As outlined in Section 1.2, we expect that authenticity properties, that are satisfied in the formal paradigm, implies authenticity in the computational paradigm. From the presented theory in Chapter 2 and 3 we are confident, that this is achievable using the following strategy:

Assume a process P with suitable pairs of begin and end correspondences, such that non-injective agreement is satisfied. Since it is required, by Theorem 3.6, that such pairs must be the only begin and end events in P, reduce each correspondence pair separately by application of Theorem 3.6. Thus, we have terms M and M', implying that $P'(N)$ preserves the secrecy of N. Give $P'(N)$ types by application of the type system in Section 2.1.4 and verify the secrecy of N in $P'(N)$. If the secrecy by typing is preserved, we know by Theorem 2.15 that $P'(N)$ also preserves the secrecy of the payload N i.e. computational secrecy is verified. Then it needs to be shown that the obtained computational secrecy is equivalent to the notion of computational authenticity, for example, by showing that computational authenticity is satisfied if and only if $P'(N)$ preserves the secrecy of N. However, we leave this for future work.

In essence, we reduce from formal authenticity to formal secrecy by application of Theorem 3.6 and from formal secrecy to computational secrecy using Theorem 2.15, which poses the final challenge of showing, that this implies computational authenticity.

Conclusion 4

In Chapter 2 we studied the syntax and semantics of a process calculus in order to give a precise definition of formal secrecy in the presence of an adversary. The process calculus has been extended with a type system and we present a result that states that a closed well-typed process preserves the secrecy of data from public names and public channels accessible by an adversary. The notion of computational indistinguishability provides the foundation for a precise definition of computational secrecy and we present an important result that shows how formal secrecy implies computational secrecy.

In Chapter 3 we present definitions of formal and computational authenticity in terms of correspondence assertions and show how formal authenticity can be verified by a reduction to formal secrecy.

In Section 3.4, we finally present an informal approach for relating formal authenticity to computational authenticity and pave the way for showing a reduction from formal authenticity to computational secrecy, rigorously. As future work, we leave the final challenge of verifying the relation between computational authenticity and computational secrecy, such that a connection between formal and computational authenticity can be established.

4.1 Discussion

The next step would of course be to either verify or reject our hypothesis and we suspect that there is still a substantial amount of work left, in order to reach this goal. Most importantly, an approach for verifying "the missing link" as described above, should be the primary focus.

If our method fails to produce a result, an alternative approach might be to use a similar approach as that of [Abadi et al., 2006]. This would require a formulation of a different type system with respect to authenticity, with the purpose of translating this into the type system of [Laud, 2005]. Then it would be necessary to show, that well-typedness in this new type-system implies well-typedness in Laud's. However, we suspect that this approach would require more work, than our proposed "shortcut".

Appendix A

A.1 The π-calculus

The *π-calculus* is a formal model for describing inter-process communication through channels. Originally presented by [Milner et al., 1992] as an extension of the Calculus of Communicating Systems (CCS), it enables various processes to interchange channels of communication. Thus, a process in the π-calculus is able to send a channel of communication to another process. The other process may thereby interact with a third process directly without having to send the communication through the original process. One of the advantages of the π-calculus is the ability for a process to pass internally scoped channels to other processes. The following definition makes heavy use of [Parrow, 2001].

Agents are usually considered as processes, implying that $P, Q, \ldots$ may perform various actions depending on the their context.

To gain a better understanding of the definition, the following provides an elaboration of the syntax.

Let $\mathcal{N}$ be a set of countably infinite *names* in the range; $a, b, \ldots, z$, representing communication channels, variables or data values. Let A be a set of *agents* with nonnegative arity, where agents range over $P, Q, \ldots$. Then the syntax of π-calculus is defined as

Prefixes	α	::=	$\bar{a}x$	Output
			$a(x)$	Input
			τ	Silent
Agents	P	::=	$\mathbf{0}$	Nil
			$\alpha.P$	Prefix
			$P + P$	Sum
			$P \mid Q$	Parallel
			`if` $x = y$ `then` P	Match
			`if` $x \neq y$ `then` P	Mismatch
			$(\boldsymbol{\nu} x)P$	Restriction
			$A(y_1, \ldots, y_n)$	Identifier
Definitions	$A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$ where $i \neq j \Rightarrow x_i \neq x_j$			

Table A.1. Syntax of the π-calculus

Output: The prefix $\bar{a}x$ defines that an agent may send a name (typically data or a channel) x via the channel a.

Input: The input prefix defines how an agent can receive a name x (typically data or a channel) through the channel a.

Silent: The silent prefix defines how an agent is able to perform a silent action, i.e. it has no communication with the environment and afterwards continues.

Nil: The agent **0** is the empty agent i.e. the agent that does not perform any actions i.e. an agent that has completed.

Prefix: The notation $\alpha.P$ shows how the prefixes are applied to an agent. The output prefix $\bar{a}x.P$ for a process P means that a name x is received via the channel a after which the agent P continues. Similarly for $a(x).P$, the agent P receives x via a and then continues. Finally $\tau.P$ denotes how no names are sent or received and that P thus continues without an form of interaction.

Sum: The sum $P + Q$ represents an agent that can continue as either P or Q.

Parallel: The agent $P \mid Q$ denotes that the agents P and Q run in parallel, where P and Q may interact through a common channel. If no such channel exists, P and Q will continue regardless of each other.

Match: The agent `if` $x = y$ `then` P is a conditional statement that will continue as P if the names x an y are the same. If the conditional statement is false, no actions will be taken.

Mismatch: Similar to the match a mismatch `if` $x \neq y$ `then` P continues as P if x and y are two different names, otherwise no action is taken.

Restriction: A restriction $(\boldsymbol{\nu} x).P$ defines how a name x is restricted to the agent P. This means that x is a local to P or in other words, only within the scope of P. However, with some non-restricted channel, P may send x to another agent, thereby enabling two agents to share a restricted name, sending the name out of scope.

Identifier: An agent may also be an identifier $A(y_1, \ldots, y_n)$ with parameters $y_1, \ldots, y_n$. The identifier is said to have arity n.

Definition: A definition $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$ exists for each identifier where every x_i must be pairwise distinct. The definition states that $A(x_1, \ldots, x_n)$ behaves like P where each x_i is substituted with y_i. Furthermore, A may exist in P allowing recursion.

A *substitution* is a function $\{x/y\}$ that maps a name y to the name x. For renaming multiple names, the notation $\{x_1, \ldots, x_n/y_1, \ldots, y_n\}$, where y_i is pairwise distinct, is used.

The main idea in the π-calculus is to give processes the ability to pass links to other processes. The π-calculus has given rise to other formal languages that extends its capabilities. One of these is the Spi-calculus, that will be discussed in the following section.

Bibliography

Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. In *Foundations of Software Science and Computation Structures*, pages 25–41. Springer, 2001.

Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM (JACM)*, 52(1):102–146, 2005.

Martín Abadi and Andrew D Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 36–47. ACM, 1997.

Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption)*. *Journal of cryptology*, 15(2):103–127, 2002.

Martín Abadi, Ricardo Corin, and Cédric Fournet. Computational secrecy by typing for the pi calculus. In *Programming Languages and Systems*, pages 253–269. Springer, 2006.

Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):109–123, 2005.

Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

Bruno Blanchet. From secrecy to authenticity in security protocols. In *Static Analysis*, pages 342–359. Springer, 2002.

Bruno Blanchet. Computationally sound mechanized proofs of correspondence assertions. In *Computer Security Foundations Symposium, 2007. CSF'07. 20th IEEE*, pages 97–111. IEEE, 2007.

Bruno Blanchet. Automatic verification of correspondences for security protocols. *CoRR*, abs/0802.3444, 2008.

Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, July 2009.

Danny Dolev and Andrew Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

Andrew D Gordon and Alan Jeffrey. Authenticity by typing for security protocols. *Journal of computer security*, 11(4):451–519, 2003.

Richard A Kemmerer. Analyzing encryption protocols using formal verification techniques. *Selected Areas in Communications, IEEE Journal on*, 7(4):448–457, 1989.

Peeter Laud. Secrecy types for a simulatable cryptographic library. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 26–35. ACM, 2005.

Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.

Joachim Parrow. An introduction to the π-calculus. *Handbook of Process Algebra*, pages 479–543, 2001.

Thomas YC Woo and Simon S Lam. A semantic model for authentication protocols. In *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, pages 178–194. IEEE, 1993.

Andrew C Yao. Theory and application of trapdoor functions. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 80–91. IEEE, 1982.