Protecting Individuals' Interests in Electronic Commerce Protocols

Hao Chi Wong

August 2000

CMU-CS-00-160

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Thesis Committee

Jeannette Wing, Chair
Marvin Sirbu
Doug Tygar
Nevin Heintze, Bell Labs

This research is sponsored by the United States Postal Services (USPS) under a series of grants to Carnegie Mellon University. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the USPS or the U.S. government.

Keywords: Protocols, electronic commerce, security, correctness properties, formal methods, models, trust (assumptions), deviation modes, distributed systems.

Abstract

Commerce transactions are being increasingly conducted in cyberspace. We not only browse through on-line catalogs of products, but also shop, bank, and hold auctions on-line.

The general goal of this research is to answer questions such as: What electronic commerce protocols try to achieve? What they must achieve? And how they achieve it? My thesis in this dissertation is that 1) In electronic commerce transactions where participants have different interests to preserve, protection of individual interests is a concern of the participants, and should be guaranteed by the protocols; and 2) A protocol should protect a participant's interests whenever the participant behaves according to the protocol and trusted parties behave as trusted.

In this dissertation, we propose a formal definition of protection of individual interests and a framework in which protocols can be analyzed with respect to this property. Our definition is abstract and general, and can be instantiated to a wide range of electronic commerce protocols. In our framework, we model electronic commerce systems as state machines, make trust assumptions part of protocol specifications, and distinguish executions by deviation modes.

We specify and analyze three protocols using this framework. Our analysis uses standard mathematical techniques. We found protocol weaknesses that have not been found before.

To my dad Tomé (in memoriam).
To Nhá, Zico, and Tica.

Also to Marshall.

Acknowledgments

Working towards a Ph.D. degree is known to be arduous. For me, it was more challenging than anything I had done before. I owe my successful completion to many people, and I want to thank them.

First and foremost, I want to thank Frank Pfenning for being an advisor, a mentor, and a friend. I greatly appreciate his guidance, both technical and non-technical, during the first years of my career at CMU. But, above all, I am grateful for his support later on, when my thesis research stalled and I was discouraged. Frank stood by my side, gave me perspective, reminded me of my strengths, and offered to help however he could. My gratitude is beyond words.

I would also like to thank the members of my thesis committee for sharing their knowledge, insight, and wisdom, specially in earlier stages of this work. I especially would like to thank my thesis advisor, Jeannette Wing, without whom I would not have finished. I am grateful for her patience, fairness, and relentless pursuit of precision and clarity. This dissertation is infinitely better because of her advising.

In the last two years, I was fortunate to be associated with Katia Sycara's research group. Katia is caring and understanding, and her support made finishing this dissertation a much easier task. Her enthusiasm for building real systems has had an important influence on me; I especially thank her for giving me the opportunity of working on concrete cryptographic and security systems. This practice gave me a different and valuable perspective on things that I had previously dealt with only at an abstract level in my thesis work.

Like everybody else, I thank Sharon Burks and Catherine Copetas. They make it possible for us to forget that we are part of an university with rules, calendars, and policies. I felt spoilt in the magic kingdom they have created.

CMU's Computer Science Department is full of interesting, stimulating, and supportive people. I was lucky to have become friends with many of them. I thank Henry Rowley for being an infallible source of hacking expertise and a patient sounding board for my thesis ideas. I treasure our countless dinners and walks, which often had a therapeutic effect. I thank Andrzej Filinski for his friendship, support, and eagerness to discuss all things POP-related. I thank Anurag Acharya for being the wiser, senior officemate. And I thank Juergen Dingel for making the initial coursework fun.

Many other people enriched my graduate school experience. I feel blessed to have enjoyed the support, companionship, and camaraderie of Arup Mukherjee, Bwolen Yang, Siegfried Bocionek, David Tarditi, Manish Pandey, Girija Narlikar, Rujith de Silva, Hank Wang, Yirng-An Chen, Joyoni Dey, Rohini Brahme, Jeff Polakow, Rich Goodwin, Oliver Schulte, Stefan Weiske, Puneet Kumar, Claudson Bornstein, Ekkehard Rohwedder, Peter Dinda, Carsten Schuermann, Joseph O'Sullivan, and Lars Birkedal.

In the last two years, the RETSINA group in the Robotics Institute provided me with a friendly and stimulating home. I thank each and every one of its members. Special thanks to Massimo Paolucci (for his great friendship), Terry Payne (for being a patient and understanding officemate, and especially for providing a quiet environment for my thesis writing), Joe Giampapa, Alan Guisewite, Zhendong Niu, Michelle Agie, Carol Boshears, Gita Suthankar, and Martin van Velsen.

For some mysterious reason (or was it?), I had to go to CMU's Student Health Center more often than I wished. I thank its friendly and competent staff. Special thanks to Diane Dawson, whose attention, friendliness, and warmth always made me feel special.

My development as a researcher did not take place exclusively at CMU. I was lucky to have spent one summer with Amy Felty at (the old) AT&T Bell Labs, and another with Vijay Saraswat and Markus Fromherz at Xerox PARC. I thank them for their mentoring and friendship.

Federal University of Minas Gerais (UFMG) and Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio) were my academic cradles, where my interest in doing computer science research was forged. Several people inspired me and contributed directly or indirectly to my coming to CMU. Special thanks to Jose Nagib Contrim Arabe, Virgilio e Augusto Fernandes Almeida, Nivio Ziviani, Jose Monteiro da Mata, Clarindo da Silva e Padua, Osvaldo de Carvalho, Marcio Bunte de Carvalho, Diogenes da Silva, Luiz Fernando da Costa, Berthier Ribeiro de Neto, Rodolfo Resende from UFMG; and Carlos Lucena, Paulo Veloso, Valeria de Paiva, and Luiz Carlos Pereira from PUC-Rio. I feel fortunate to have them as friends and collegues.

Going for graduate school away from one's home country is not always fun. I thank the Brazilian gang for re-creating a little Brazil here in Pittsburgh. Special thanks to Sergio (for making my transition to Pittsburgh and CMU easier) and Alessandra, Marcel and Clara, Gilberto and Vanessa, Fernando and Helena, Tsen and Ivonne, Alexandre and Ana Paula, Jair and Hiroko, Evandro, and Eduardo.

I came to the United States to attend graduate school, but gained an American family as a bonus. I thank Paula and Joseph for making me actually feel their daughter. I thank my husband, Marshall, for always trying to squeeze fun into my otherwise stressed life, for providing feedback on selected sections of my dissertation, and for never having asked why I was not finished yet.

Obtaining a Ph.D. is as much about mastering and contributing to a field as about perseverance, inner strength, and self-confidence. I could have never obtained my degree without my family. My mother, Nha, has always been a source of inspiration. Her strength and self-reliance, unconditional love and support are in great part responsible for who I am. My brother Zico and my sister Tica are my best friends and are constant reminders of how fortunate I am as a person. From thousands of miles away, they gave me strength, support, and love that I needed to overcome the obstacles. Finally, I am sure that my father was cheering for me from above.

Wong Hao Chi
August, 2000

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Commerce transactions are being increasingly conducted in cyberspace. We not only browse through on-line catalogs of products, but also shop, bank, and hold auctions on-line.

For electronic commerce to sustain its current growth rate, and prevail as a definitive alternative to traditional physical commerce, nothing should happen to undermine consumers' confidence in the new technology. Thus, both electronic transactions and entities should satisfy a minimal set of properties, expected by users of electronic commerce systems. For instance, in an electronic sale transaction, a customer should receive the goods which he or she pays for, and a merchant should be paid for goods delivered. Similarly, electronic cash should always be exchangeable for goods and services. Note that since the electronic world has characteristics of its own, these properties may or may not have a correspondence in the physical world.

These requirements can be hard to meet. First, some intrinsic constraints of the physical world are absent in the electronic world. For instance, due to the resources and knowledge it takes to counterfeit cash, spurious bills and coins are less common in the physical world than they would be. In the electronic setting, however, where copying data items is cheap, fast, and easy to disguise, spurious currency will likely to be much more common.

Second, participants of electronic commerce transactions are often geographically distributed, and informal mechanisms that require face-to-face interactions and that can help bring transactions to successful completion are now unavailable. For example, customers can no longer judge merchants by their storefronts, and different parties of a transaction can no longer easily monitor each other and enforce fulfillment of individual obligations in the transaction. Some types of transactions are more vulnerable than others. For example, in the physical world, a customer does not pay with cash unless he or she is given goods or receipts on the spot. But electronic cash protocols have users send electronic cash over networks. This capability prompts hard-to-settle disputes, such as a merchant and a customer disagreeing over whether or not the electronic cash was sent. Was the cash not sent by the customer? Or was it actually received by a lying merchant? Problems with the network bring in other possibilities: the cash might have been lost during the transmission, due to a network failure.

Third, unlike certain other classes of protocols (e.g., key distribution), where participants share a common goal (e.g., obtaining a secret key for a secure communication) [32, 71, 74], here participants typically have separate and often conflicting goals in mind [42, 28, 89]. For instance, while honest customers will try to pay for goods received, dishonest ones may cheat and try to obtain goods for free. Similarly, while conscientious merchants will strive for customer satisfaction, unscrupulous ones may not care if goods are not delivered, or may deliver something else. In cyberspace, where one is likely to interact with complete strangers, the threat of cheating is real.

To achieve properties they are expected to satisfy, electronic commerce protocols rely on cryptography, trusted parties, and tamper-resistant hardware. These protocols are usually filled with details whose purposes are not immediately clear. To make matters even worse, critical assumptions are often left implicit in their specifications. Given this scenario, it is not always clear what these protocols try to achieve, what they must achieve, and how they achieve it.

This research addresses these questions, and focuses on a requirement called *protection of individuals' interests*.

The rest of this chapter is structured as follows. In Section 1.2, we introduce the notion of protection of individuals' interests starting from the notion of fairness. In Section 1.3, we introduce the notion of trust assumptions, and discuss their importance in protocol design, analysis, implementation, and deployment. In Section 1.4, we give an overview of this thesis work. In Section 1.5, we summarize our contributions. Section 1.6 presents related work, and Section 1.7 gives a roadmap for the rest of this dissertation.

1.2 Protection of Individuals' Interests

The notion of protection of individuals' interests is a generalization of the notion of fairness. In this section, we give an intuition about the former starting from a common interpretation of the latter.

The notion of fairness first appeared in the context of protocols for exchange of secrets, contract signing, and certified electronic mail [10, 11, 7, 27, 38, 65, 85, 75, 83, 88]. More recently, fairness has also been studied in the context of protocols for exchange of documents, electronic sale, and non-repudiation [6, 5, 20, 28, 31, 53, 57, 58, 89, 90].

The most widely used intuitive interpretation of fairness says that a protocol is fair if *no protocol participant can gain any advantage over other participants by misbehaving.* Gaining advantage means different things in different contexts. For example, in a sale transaction, a customer gains advantage if she receives the goods, but the merchant does not receive the payment; and a merchant gains advantage if he receives the payment, but the customer does not receive the goods. In certified electronic mail, the recipient of a message gains advantage if the sender does not receive the proof of delivery; similarly, the sender gains advantage if she has a proof of delivery, even though the recipient did not receive the message.

The usual notion of fairness, however, is not subtle enough. In particular, one party's interests can be compromised even if no one else has gained any advantage. This situation can happen, for example, in sale transactions where customers pay by electronic cash. If the payment sent by the customer gets lost while in transit, and never reaches the merchant, and the merchant does not send the goods, then we are in a scenario where neither the customer nor the merchant receives their shares in the exchange, even though the customer has effectively lost money. Under this scenario, no one has gained any advantage, but the customer's interests have certainly been hurt: she did not receive anything in return for the money she has now lost. Users of ecommerce protocols need to be protected against such losses.

To handle this and other examples, we introduce a new property called *protection of individuals'* *interests*, which is centered on the notion of protection of different participants' interests in a transaction. Intuitively, a protocol protects a participant's interests if *his or her interests cannot be hurt even if everybody else misbehaves.*

Protection of individuals' interests generalizes fairness in two ways. First, it has a wider domain of applicability: while fairness is applicable only in transactions where the zero-sum rule holds, protection of individuals' interests is applicable in any transaction where participants have different interests to preserve. In addition to exchange protocols, which are the only ones that have been studied with respect to fairness, we can now examine an assortment of other types of protocols with respect to protection of individuals' interests. Payment protocols, withdrawal protocols, and auction protocols are some examples. In payment protocols, the payer's interest is protected if the money she gives up is actually received by the payee; and the payee's interest is protected if what he receives is worth real money. In withdrawal protocols, the withdrawer's interest is protected if the amount deducted from her account is never bigger than the amount of cash she receives; the bank's interest is protected if the withdrawer cannot obtain cash without having her account deducted. Finally, in auction protocols, the auctioneer's interest is protected if she always sells her goods at the highest price bidders are willing to pay; and a bidder's interest is protected if he is sold the auctioned goods if his offer is the winning bid, and the sale price was not artificially inflated by bidding frauds.

Second, protection of individuals' interests assumes a less restricted failure model. To analyze a protocol with respect to fairness, one considers scenarios where a participant misbehaves, and checks whether the participant himself can obtain any advantage [3, 42]. To analyze a protocol with respect to protection of individuals' interests, one considers scenarios where everything (other participants and the network) except the participant's local execution can go wrong, and checks whether the same participant's interests can be hurt. This second failure model takes into account threats that are not considered when one is concerned with fairness. They include sabotage attacks where a participant misbehaves not to obtain advantages for herself, but to hurt someone else's interests. (These attacks can sometimes hurt the interests of even the perpetrator herself.) They also include system failures that are out of control of the participants themselves. Network failures are an example. Finally, they include collusions among multiple participants. This failure model, in essence, assumes that one should not rely on anyone other than oneself or anything other than the protocol to protect one's own interests.

This assumption can be too stringent however. When a designer uses trusted parties in a protocol, protection of the various individuals' interests typically rely on certain parties behaving as trusted. We will discuss trusted parties and their roles in protection of individuals' interests in the next section.

1.3 Trusted Parties and Trust Assumptions

Executions of security and electronic commerce protocols are subject to potential disruptions by third party intruders/participants of a protocol themselves. These disruptions can lead to inadmissible outcomes, where basic properties of a transaction are violated. For example, a disrupted execution of an authentication protocol may wrongly authenticate a party A as a different party B.

One way of preventing corruption of outcomes is to use *trusted parties*. In a protocol, trusted parties are parties entrusted with correctly and reliably carrying out protocol steps that are decisive to guaranteeing satisfaction of a targeted property. For example, trusted parties are entrusted with generating fresh keys in authentication protocols to guarantee authentication [71, 74, 81]. In some payment protocols [84], they are entrusted with implementing transaction processing [66] to achieve fairness. This notion of trust is a refinement of the definition of trust adopted by the US Department of Defense in the context of secure systems, which states that "a trusted component is one which, if it breaks, can compromise system security" [1]. It is a refinement because it distinguishes "breaks" that matter and those that do not. For a concrete example, see Example 1.1 below.

A trusted party can be one of the main parties involved in a protocol (for instance, banks in ecash protocols are usually trusted) or someone called in solely to mediate a transaction (like T in Example 1.1).

Trusted parties are entrusted with correctly and reliably carrying out critical steps of a protocol. In the context of protocol design and analysis, entrustment simply translates into assumptions. These assumptions – about how trusted parties behave – are commonly called *trust assumptions*. When reasoning about a protocol, one effectively assumes that ordinary parties can deviate arbitrarily from the protocol, while trusted parties can deviate so long as they do not violate trust assumptions.

To be concrete, consider the following hypothetical fair exchange protocol,

Example 1.1 *A mediated exchange protocol:*

1. $A \rightarrow T : a$
2. $B \rightarrow T : b$
3. $T \rightarrow B : a$
4. $T \rightarrow A : b$

through which A and B trade items a and b, under fair mediation of trusted party T. A is prescribed to release a and wait to receive b. Since he is an ordinary party, we make no assumption about how he will behave in an actual run of the protocol. For example, he may release a, and decide to quit his execution before receiving b. Or he may withhold a, and hope to receive b for free. The same is true of B. T is prescribed first to receive a and b, and then to forward them to their respective destinations. Since T is a trusted party, it is subject to two trust assumptions:

A1: T will not forward the items before it has received them both; and

A2: If T forwards any item, it forwards both.

This set of trust assumptions says that T will not halt its local execution between steps 3 and 4 (this would violate *A2*), but could do so between steps 2 and 3. Note that T's stopping between steps 3 and 4 makes an exchange unfair, while its stopping between steps 2 and 3 does not. Finally, this protocol implements fair exchange, as long as communication channels are reliable and T is a trusted party that satisfies trust assumptions *A1* and *A2*. Without these assumptions, fairness is not guaranteed.

Trust assumptions differ from other system assumptions. System assumptions deal with general characteristics of a system (e.g., networks can go down or processors can halt), whereas trust assumptions specify what trusted parties are trusted to do in the context of a protocol execution. Thus, while system assumptions are protocol-independent, trust assumptions are protocol-specific.

The knowledge of a protocol's trust assumptions is important to those analyzing the design of the protocol. Using these assumptions, analysts can build models that better reflect what the designers had in mind, making the analysis more meaningful. The knowledge of trust assumptions is also important to implementers and deployers, who need to realize these assumptions in the real system. Knowing these assumptions enables them to determine whether a protocol, as conceived by its designers, is realizable; estimate the cost of realizing it; and compare different protocols of a class with respect to their reliance on trusted parties (typically, the less trust a protocol requires, the less vulnerable is the protocol). Finally, the knowledge of trust assumptions is important to prospective users of a protocol. Knowing the trust assumptions allows them to decide whether a prospective participant of a protocol is fit to be a trusted party, and to evaluate the risks of relying on this prospective trusted party.

Even though knowledge of trust assumptions is important in so many contexts, they have mostly remained in the back of the designers' minds, and have rarely been made explicit. In this dissertation, we propose changing this state of practice: we introduce a framework where trust assumptions are made explicit as part of a protocol specification. Explicit trust assumptions are then used in modeling and analyzing protocols. We explain how we specify and use trust assumptions in Section 1.4.2.

1.4 Thesis Overview

> **Thesis Statement:** *Electronic commerce protocols form a distinct class of cryptographic protocols in terms of what it means for a protocol to be correct. Formal methods can help not only in formalization of correctness conditions, but also in protocol analysis.*

To support this thesis, we identify a novel property called protection of individuals' interests that all electronic commerce protocols should satisfy. We then propose a formal definition of protection of individuals' interests and a framework in which protocols can be analyzed with respect to this property. Our definition is abstract and general, and can be instantiated to concrete protocols from different classes of electronic commerce protocols. In our framework, we model electronic commerce systems as state machines, make trust assumptions part of protocol specifications, and distinguish executions by deviation modes.

Using this framework, we specify and analyze three protocols [42, 28, 14]. Our analysis uses standard mathematical techniques.

In the rest of this section, we sketch the various components of our framework and our definition of protection of individuals' interests.

1.4.1 Modeling Electronic Commerce Systems

We see electronic commerce systems abstractly as sets of interconnected agents where each agent runs a process. Agents have private local stores where they hold their data. They also have the capability of generating new data, communicating with each other by message passing, and deriving new data from existing ones using cryptography. Communication channels are end-to-end, and there is a channel connecting any pair of agents in a system. Both the communication and process executions are asynchronous.

We consider both process failures and channel failures. Since channel failures manifest themselves in the processes, however, we cast them all in terms of process failures. In this dissertation, processes can fail in two ways: they can *failstop* or they can *deceive*. A process failstops if it terminates abnormally, in a way not prescribed by the algorithm it is running. A process deceives if it follows the algorithm only apparently: at each step, the process executes the type of action prescribed by the algorithm, but possibly with bogus messages.

Both failstopping and deception are simple types of failures. Yet they model realistic attacks to electronic commerce systems. Power outages, computer crashes, and user-initiated interruptions can all cause premature termination of a process; connection failures are common with today's Internet; and deceptions can be easily carried out by malicious users.

We assume secure communication in our model: if a message is received, then it is received by the intended recipient, authenticated, confidential, and intact. Effectively, we assume that processes use the service provided by an underlying layer of cryptographic protocols.

The characterization of electronic commerce systems depicted above differs from the ones traditionally used in investigations of security protocols. In traditional characterizations, participants of a protocol always behave correctly, and there is invariably an intruder that can eavesdrop and corrupt communications among the participants. In our characterization, participants of a protocol, including trusted parties, may fail or misbehave, but communications among them are assumed to be secure. Doing so allows us to abstract away possible spoofings, sniffings, and tamperings of messages by third party intruders, which makes modeling them unnecessary. The goal is to focus on answering the core question: What can participants of a transaction, with the "help" of unreliable communication channels, do to violate other participants' individual interests?

We formalize this abstract model in state machines. Our formalization is standard.

1.4.2 Specifying and Using Trust Assumptions

Traditionally, protocol specifications specify protocol principals, initial conditions, and protocol rules. In our framework, they also specify trust assumptions.

Trust assumptions are specified in linear-time temporal logic [67]. Each trusted party has associated with it a set of logic expressions specifying its trusted behaviors. For example, the specification of the protocol in Example 1.1 (Section 1.3) would include two formulas specifying *A1* and *A2*.

Given that trusted parties are assumed to behave as trusted, executions of trusted parties that violate their corresponding trust assumptions should not appear in the model of the overall system. In our framework, trust assumptions are used to filter out trust-violating executions from the set of all possible executions of a protocol. We explain the filtering mechanism in the next subsection.

1.4.3 Defining Protection of Individuals' Interests

Ideally, protocols should protect the interests of each of its participants. In this subsection, we define the conditions under which they should provide such protection. To do so, we first characterize different types of executions performed by parties in a protocol.

Definition 1.2 A *compliant* execution is one in which the party behaves as prescribed by the protocol. A compliant execution runs to completion, or terminates prematurely because of remotely originated communication disruptions.

Definition 1.3 A *deviant* execution is one in which the party does not behave as prescribed by the protocol. We consider two types of deviant executions. An *abortive* execution is one which terminates prematurely because of local factors, and a *deceptive* execution is one which includes deceptive steps (Section 1.4.1).

Definition 1.4 A *trustworthy* execution is one in which trust assumptions made about the party are satisfied.

Note that the types of executions defined above do not conform to the traditional classification of executions, based on failure modes. The traditional classification is inadequate here for two reasons. First, it does not distinguish between failures that one causes to one's own execution and those caused by other parties of the protocol. Second, it does not distinguish between behaviors that obey trust assumptions and those that violate them.

We now give the primary definition of this subsection:

Definition 1.5 Given a protocol and one of its parties p, the protocol *protects p's interests* (or, for short, *is p-protective*) if p's interests are preserved in all executions where p executes compliantly (p's execution is compliant), and all trusted parties execute as trusted (their executions are trustworthy).

Note that Def. 1.5 implicitly asserts three important policies. First, a party is entitled to protection only if it behaves compliantly. Second, all trusted parties are relied upon to behave as trusted. Third, a party is entitled to protection even when other parties execute deviantly. Depending upon assumptions about how the parties of a protocol can deviate, an analysis can focus on one type of deviation or another.

Def. 1.5 is general and abstract in that *p's protection property*, which describes the conditions under which p's interests are preserved in an execution, is not specified. Protection properties cannot be specified without reference to a protocol, because they vary from protocol to protocol. Individuals' interests in an auction protocol are certainly different from those in a payment protocol. There are variations even within a single class of protocols. For example, receiving payment may mean receiving cash in one protocol and receiving a check in another, and this difference leads to different protection properties.

1.4.4 Case Studies

Using the framework sketched above, we specify and analyze three electronic commerce protocols: Franklin and Reiter's fair exchange protocol with a semi-trusted third party [42]; NetBill (a protocol for purchasing information goods on the Web, proposed by Cox, Sirbu, and Tygar) [28]; and Brands's off-line cash with observers protocol [14]. Each case study includes 1) a description of an abstraction of the protocol, 2) a formalization of the abstract protocol, 3) a specification of protection properties, and 4) protocol analysis.

During the formalization, we identify and specify trust assumptions pertaining to the trusted parties. Because trust assumptions are not given explicitly in the original published versions of these protocols, we had to infer them in some cases, and identify them afresh in others.

Different protocols require different amounts of effort in specification of their protection properties. For Franklin and Reiter's protocol, we simply transcribe the corresponding fair exchange properties. For NetBill, we have to take into account not only what happens on-line, but also dispute resolutions that happen off-line. For Brands's protocol, protection properties had to be formulated from scratch.

In the analysis, we consider protection properties of different parties in turn. Given a party p, we analyze p-protection under three modes: compliance, abortion, and deception. Within each mode, we consider two types of scenarios: one with reliable communication channels and the other with unreliable ones. In each case, we show whether the protocol is p-protective and what would happen if trust assumptions are violated. We also suggest fixes whenever appropriate.

Note that the protocols we analyze were not necessarily designed to function under all the scenarios we consider. (Their designers may have made stronger or weaker assumptions.) We submit them to analysis under all our scenarios, nonetheless, to see how they would do under different conditions.

1.5 Contributions

This dissertation makes several contributions to the fields of electronic commerce, computer security, and formal methods. The main contributions are listed below.

- We include trust assumptions as an explicit part of protocol specifications. We then use them to formalize executions of trusted parties. This formalization allows us to model semi-trustworthiness.

  Explicit trust assumptions also enable comparisons and evaluations of protocols with respect to their trust requirements. They are also an invaluable guide to implementers and deployers.

- We identify protection of individuals' interests as a requirement for electronic commerce protocols, and formalize the conditions under which protocols should provide such protection. The formalization uses several novel types of executions: compliant, abortive, deceptive, and trustworthy.

  Our formalization is abstract and general, and provides a unifying framework for analyzing different classes of protocols.

- We analyze three protocols using our model. The analyses established whether the protocols protect the interests of their participants under different scenarios.

  We found protocol weaknesses that have not been found before, and suggest possible fixes for the weaknesses we found.

1.6 Related Work

Formalization and analysis of cryptographic and security protocols have been topics of research for over twenty years. Authentication and key exchange protocols have by far been the most studied, with secrecy, authentication, and integrity being the properties receiving the most attention. Because we focus on protection of individuals' interests, and assume secrecy, authentication, and integrity as provided by underlying services, this body of research is not as closely related to our work as it might appear. We thus omit it here, and point interested readers elsewhere for surveys [79] and summaries of the history and the state-of-art of formal cryptographic protocol analysis [69].

In what follows, we discuss three different categories of related work, pointing out major differences that distinguish them from our work.

1.6.1 Atomicity, Fairness, and Related Properties

Protection of individuals' interests is a generalization of fairness, and fairness is closely related to atomicity. An introduction to fairness appears in Section 1.2; we do not repeat it here. As for atomicity [84], there are three levels of atomicity. Money-atomic protocols transfer funds from one party to another without creating or destroying money. Goods-atomic protocols are money-atomic, and guarantee that goods are delivered if and only if money is transferred. Finally, certified delivery protocols are goods-atomic, and allow both merchants and customers to prove, to a third party, which goods were delivered.

Atomicity and fairness were both introduced by protocol designers. In [84, 3, 42, 89], an informal definition of atomicity (respectively fairness) is presented, a protocol or a class of related protocols are proposed, and the protocols are shown to satisfy the property. These definitions and analyses are informal and tailored to the specifics of the protocol. Thus, even though they provide good insights and intuitions about a specific protocol, they are ad hoc, not definitive, and cannot be applied straightforwardly to other protocols.

In addition to analyses conducted by the designers themselves, there exist case studies focused on atomicity and fairness, conducted independently by formal methods researchers [51, 80, 78]. Heintze *et al* [51] specified simplified versions of NetBill [28] and Digicash [22] in CSP [52], and analyzed them with respect to money-atomicity and goods-atomicity using the FDR model checker [64]. Shmatikov and Mitchell [80] specified a contract signing protocol by Asokan *et al* [4], and analyzed it with respect to fairness using Murϕ [34]. Schneider [78] specified Zhou and Gollman's non-repudiation protocol [89], and analyzed it using CSP; the proofs are carried out by hand.

The first two case studies share a number of characteristics. Both use a model where trusted parties always behave according to the protocol, while ordinary parties can misbehave. Both specify atomicity (respectively, fairness) as a global property. Neither has a well-defined failure model. Neither is able to explain satisfactorily idiosyncrasies that arise from specifying atomicity and fairness as global properties. In particular, neither formalizes the distinction between interesting and uninteresting violations of the properties. (Uninteresting violations are those where the misbehaving parties are the victims themselves.)

The third case study, in contrast, does not have such weaknesses. Fairness is specified as an agent-centric property, just as our protection of individuals' interests is agent-centric. Its system model closely follows ours, except for its treatment of trusted parties: they always behave according to the protocol [78]. Schneider focuses on a single non-repudiation protocol, and does not propose a general and abstract framework that is applicable for a wider class of protocols.

Other case studies focus on credit card based electronic payment protocols [13, 70]. Bolignano [13] analyzed C-SET [33] using Coq [35]; Meadows and Syverson [70] discussed and formally specified requirements for SET [68] in a version of the NRL Protocol Analyzer Temporal Requirements Language [82]. These case studies differ from the ones discussed above in that, instead of focusing on some specific properties, they address all requirements that seem desirable in the context of their protocols. The requirements are not named, but are loosely separated into groups. There are, for example, customer requirements, merchant requirements, and gateway requirements. Some requirements within these three groups clearly resemble what we call protection properties.

Also in these studies, different hypotheses about how different parties behave are made. For each hypothesis, the properties that should hold are specified. The protocols are then analyzed under different hypotheses with respect to their corresponding properties. Some example scenarios are: everyone behaves according to the protocol; everyone except the customer behaves; no one but the merchant behaves; and so forth. The intent is to verify what conclusions different parties can draw under different circumstances.

Despite their comprehensiveness, these case studies have two major weaknesses. First, because the properties are not named, and the issues they involve are not deeply discussed, it is not straightforward to identify the essence of these properties, which makes applying their approach to other protocols difficult. Second, because the protocols are analyzed with respect to different properties (sometimes weaker or stronger versions of a property) under different scenarios, it is not immediately clear what core properties the protocols satisfy.

1.6.2 Other Properties

Other aspects of electronic commerce protocols have also been studied. Kailar proposed a BAN-like logic [19] for reasoning about accountability, and used it to analyze several protocols [54]. Kessler and Neumann [55] extended AUTOLOG [56] with predicates and rules to model accountability, and proved that the new calculus is correct with respect to the formal semantics given in [86]. They then used this calculus to analyze SET [68] and Payword [77]. Kindred [59] generated automatic checkers for both Kailar's logic and AUTOLOG using Revere. He then applied the resulting checkers to a variety of protocols. Finally, Clarke *et al* [26] proposed a logic of knowledge for specifying security properties of electronic commerce protocols. They then used this logic to specify a few properties, such as privacy and anonymity, for the 1KP protocol [6].

1.6.3 Trust

Also related to this dissertation are research efforts that try to understand and formalize the notion of trust. Trust has long been studied in the context of multi-domain authentication protocols [9, 44, 61, 81]. In this context, trust is typically distributed among a number of key or name servers, and structured in some pre-defined, hierarchical way. To accept an authentication, a user needs to trust only part of the hierarchy.

The notion of trust for distributed authentication has also been studied independently of any protocol or name scheme. For Yahalom *et al* [87], trust is a relationship between two parties, and one party can trust another in any respect. Starting from these two points, they identified and classified different types of trust that may be required in an authentication procedure. Trust with respect to keeping secrets, providing recommendations, and performing correctly algorithmic steps are some examples. They then proposed a language for expressing trust relations and an algorithm for deriving trust relations from recommendations. Finally, they analyzed several authentication protocols with respect to their trust requirements. But because the trust specification language is not embedded in an analysis formalism, all their analyses are informal. This trust specification language was later extended to include valued trust relationships [8]. Using the extended language, one can express to what degree one party trusts another in some specific respect.

Logic based protocol analysis methods [76, 19, 46] have incorporated some notion of trust. For example, BAN [19] has a language construct for formalizing the notion of who can be trusted on what (for example, a key server can be trusted on key generation), and an inference rule for incorporating a trusted party's belief into other parties' belief set. In all these logics, trust specifications are part of the initial assumptions of a system, and used as axioms to draw conclusions about security properties of a protocol.

In commerce transactions, trusted intermediaries can be used to enable exchanges between two mutually distrustful parties. When three or more such parties attempt to exchange multiple items among themselves, and there is no single intermediary that is trusted by them all, relevant questions to ask are: Can such an exchange be feasibly carried out, in the sense that no participant ever risks losing money or goods without receiving everything promised in exchange? And if so, what are the steps [58]? To answer these questions, Ketchpel and Garcia-Molina introduces 1) a graphical notation in which one can represent parties (both exchanging parties and intermediaries) involved in a transaction and the interactions between them; and 2) a method for finding a sequence of pairwise mediated exchanges that will lead to a feasible global exchange, if one exists. Note that this work does not attempt to dissect the notion of trust. Rather, it models trust at a higher level of abstraction, simply as a factor that guarantees successful two-way exchanges.

Finally, Branstad *et al* investigated the role of trust in TMail, a privacy-enhanced electronic mail system [18]. This work is the closest in spirit to our effort in making trust assumptions explicit. They identify functions (both cryptographic and non-cryptographic) that need to execute correctly so that electronic mail remains protected. They then argue that these functions need to be hosted in a trusted computing base so that their correct execution can be assured. They also discuss possible attacks to the mail system if these critical functions are running in an untrusted computing base. The discussion is all informal, and there is no attempt to formalize these ideas in the paper.

1.7 Roadmap

The rest of this dissertation consists logically of three parts. In the first part (Chapter 2), we present our specification, modeling, and analysis framework. We start with a characterization of electronic commerce systems as they are modeled in this dissertation. We then present a standard state machine formalization of such systems and a formalism for protocol specification. Finally, we define different types of executions of a protocol, which we then use to give a formal definition of p-protective protocols.

In the second part (Chapters 3 - 5), we apply the framework from Chapter 2 to specify and analyze three protocols: we specify and analyze Franklin and Reiter's fair exchange protocol using semi-trusted third parties [42] in Chapter 3; NetBill [28] in Chapter 4; and Brands's off-line electronic cash with observers protocol [14] in Chapter 5. These chapters follow the same structure. First we introduce the protocol and justify why it makes an interesting case study. We then present an abstract formulation of mathematical (cryptographic) building blocks used by the protocol. A specification of the protocol and its various protection properties appears next, and is followed by a high-level analysis of the protocol, the main results, and detailed proofs. The chapters conclude with a summary of the findings of our analysis and a discussion of insights gained through the exercise.

The last part (Chapter 6) presents our conclusions, summarizes the dissertation, reflects on our contributions, and proposes directions for future work.

Chapter 2

The Model

In this chapter, we present the model we use to analyze electronic commerce protocols with respect to protection of individuals' interests. In Section 2.1, we characterize electronic commerce systems as they are formalized in this work. This characterization departs from traditional characterizations of security and electronic commerce systems in two ways: in the failure model of the participants of a protocol and in the security of communications. In traditional characterizations, participants of a protocol always behave correctly, and there is invariably an intruder that can eavesdrop and corrupt communications among the participants. In our characterization, participants of a protocol, including trusted parties, may fail or misbehave, but communications among them are assumed to be secure.

In Section 2.2, we present a standard state machine formalization of electronic commerce systems. The formalization models what electronic commerce systems can do in general, independently of any protocol they might be running.

In Section 2.3, we present the formalism we use to specify protocols. Our formalism is state-based in that it uses a set of protocol rules to specify state conditions and protocol steps they enable. In addition to the standard initial condition and protocol rules, our formalism incorporates the trust assumptions of a protocol as a component of protocol specifications.

To analyze a protocol in a model-theoretic setting, one examines the set of all of its executions. In Section 2.4, we define different sets of executions of a protocol, which are then used to give a formal definition of p-protective protocols.

2.1 System Characterization and Assumptions

Electronic commerce systems consist of sets of interconnected agents: banks, shops, individual customers, etc. At any time, these agents may be running multiple processes simultaneously, each process carrying out a different transaction[1] with a different agent. In our model, we assume that each agent runs one process at a time, and identify agents with the processes they run.

In our model, processes have private local stores: the data they hold are not publicly accessible. Processes can *generate* new data – e.g., keys and nonces – and *communicate* with each other by message passing. They also have timing capabilities and can *timeout* while waiting for events to occur. Finally, they have cryptographic capabilities such as encryption and decryption, which allow

[1]Unless otherwise stated, *transaction* is used in an informal, non-technical sense in this dissertation.

them to *derive* new data from existing ones.

The interprocess communication subsystem logically consists of end-to-end, pairwise communication channels. Each channel models all the hardware and software that connect the two end-processes. Communication is connection-oriented (FIFO, at-most-once message delivery), and we assume connections are established at the beginning of transactions.

In our model, both the communication and process executions are *asynchronous*: communication delays are unbounded, and processes can take steps at arbitrary speeds.

In this dissertation, we consider both process failures and channel failures. We cast them all in terms of process failures, however, because in our model communication is connection-oriented and channel failures manifest themselves in the processes. For the purposes of this work, processes can fail in two ways: they can *failstop* or they can *deceive*.

If a process failstops, it terminates abnormally, i.e., in a way not prescribed by the algorithm it is running. Abnormal terminations can be caused by local or remote factors. Power outages, computer crashes, and user-initiated interruptions are all examples of local factors. Connection failures due to problems at remote processes or with communication channels are examples of remote factors.

If a process deceives, it follows the algorithm only "apparently". Roughly speaking, even though the types of events that occur at each step are those prescribed by the algorithm, the messages they carry may not be. For example, sending a message that is not the one prescribed by the algorithm according to the process's present state is a deception. Deceptions only occur by Byzantine processes. We do not call processes that deceive "Byzantine processes," however, because they do not fail in all the ways that Byzantine processes can, as defined in the literature [36].

We consider these types of failures because they model simple, yet realistic attacks to electronic commerce systems. Power outages, computer crashes, and user-initiated interruptions can all cause premature termination of a process; connection failures are common with today's Internet; and deceptions can be easily carried out by malicious users.

Finally, we assume communication is secure: if a message is received, then it is received by the intended recipient, authenticated, confidential, and intact (untampered). Effectively, we assume that processes use the service of an underlying layer of security protocols. This assumption allows us to abstract away possible spoofings, sniffings, and tamperings of messages by third party intruders and makes modeling of intruders unnecessary.

Discussion

Our model of electronic commerce systems makes simplifying assumptions that have impact on protection properties of a transaction. For example, concurrent execution of two transactions is known to cause problems in traditional distributed systems; it also leads to subtle problems in cryptographic protocols [63]. Message tampering can prevent original messages from reaching their intended destinations, which can also compromise protection of different parties.

In a first step towards formalizing and understanding protection of individuals' interests, however, we purposely abstract away as much detail as possible, and focus on what participants of a transaction with the "help" of unreliable communication channels can do to compromise other participants' interests. This allows us to address core questions, such as "How can we characterize precisely the notion of protection?", "How can we formalize trust assumptions?", and "How do we use trust assumptions in the analyses of protocols?", independently of issues like concurrency and communication security. After addressing these questions, we can then relax our assumptions and investigate how protection interacts with these issues.

For a more complete list of assumptions we make and their implications, see Section 2.5.

2.2 Modeling Electronic Commerce Systems as Transition Systems

In this section, we present a state machine model for electronic commerce systems. We model single processes as simple transition systems and electronic commerce systems as asynchronous compositions of these simple systems. Most of the formalization is standard, and we include it here for self-containment. The few non-standard features will be explained in detail.

We start formalizing some message-related concepts.

2.2.1 Messages

In our model, *messages* can be *basic* messages or *composite* messages. Basic messages are those simplest units of data, such as keys, product ids, and prices, that cannot be further decomposed. Basic messages can be concatenated or used as arguments of cryptographic functions; the result is composite messages.

Cryptographic functions are functions such as encryption, decryption, and signing functions. In our model, they are abstract functions satisfying certain ideal properties. We reserve further discussion and formalization of cryptographic functions for later chapters, when they will be introduced in the context of particular protocols.

In what follows, we formalize message-related concepts. The notion of type is used informally here: a type is simply a domain from which a particular class of messages can be drawn.

Def. 2.1 says that messages are built inductively from basic messages by applying concatenation and cryptographic functions.

Definition 2.1 *Messages are inductively defined as:*

1. *Basic messages m of type t, $m : t$, are messages;*
2. *If $m_1 : t_1$ and $m_2 : t_2$ are messages, then the concatenation $m_1 m_2 : t_1 \times t_2$ of m_1 and m_2 is a message;*
3. *If $f : t_1 \times \ldots \times t_n \rightarrow t$ is a cryptographic function and $m_i : t_i$ are messages, then the function application of f to $m_1, \ldots, m_n$, $f(m_1, ..., m_n) : t$, is a message.*

Definition 2.2 defines when two messages are equal. It assumes that each type of basic messages comes with a definition of identity between messages of that type.

Definition 2.2 *Let m and m' be two messages. m equals m' ($m =_{msg} m'$) if and only if*

1. *m and m' are identical basic messages;*
2. *$m = m_1 m_2$; $m' = m'_1 m'_2$; and $m_i =_{msg} m'_i$, for i = 1, 2; or*
3. *$m = f(m_1, ..., m_n)$; $m' = f(m'_1, \ldots, m'_n)$; and $m_i =_{msg} m'_i$, for i = 1, …, n.*

In the rest of this dissertation, we represent $=_{msg}$ by $=$. It should be clear from the context whether a equality sign denotes equality between messages as defined in Def. 2.2 or is being used informally to denote a mathematical equality.

Once we have defined message equality, we can define associativity. Concatenation of messages is associative. That is, $(m_1m_2)m_3 = m_1(m_2m_3)$. For simplicity, we sometimes represent them by $m_1m_2m_3$.

Message equality will be later extended with conversions between cryptographic terms. A familiar example of conversion says that the result of encrypting and then decrypting a message (with the same symmetric key) is the message itself. That is

$$dec(enc(m,k),k) \doteq m,$$

where $\doteq$ denotes conversion. The conversion relation creates an equivalence class of equal messages. That is,

$$\text{If } m_1 \doteq m_2, \text{ then } m_1 = m_2.$$

From Def. 2.1, we can derive the *submessage* relation between two messages. Any message is a submessage of itself, and any submessage of messages m_i in points 2 and 3 of Def. 2.1 is a submessage of m. We use $\sqsubseteq$ to denote the submessage relation: $m' \sqsubseteq m$ denotes m' is a submessage of m. The notion of submessages of a message will be used later to formalize the notion of freshness of messages generated at random.

Given a set of messages M, we can use concatenation, decomposition, and function application to derive new messages:

Definition 2.3 *Let m be a message and M be a set of messages. m is* constructible from M*, $m \in^* M$, if and only if*

1. *$m \in M$, or*
2. *$m = m_1m_2$, $m_1 \in^* M$ and $m_2 \in^* M$, or*
3. *$m = m_i$, for $i = 1$ or $i = 2$, and $m_1m_2 \in^* M$, or*
4. *$m = f(m_1, \ldots, m_n)$, and $\forall i \in \{1, \ldots, n\}$, $m_i \in^* M$.*

Definition 2.3 says that m is constructible from M if 1) m is a member of M; 2) m is a concatenation of two messages m_1 and m_2, both of which are constructible from M; 3) m is a component of a concatenated message constructible from from M; or 4) m is the result of applying a cryptographic function to inputs $m_1, \ldots, m_n$, all of which are constructible from M.

2.2.2 Modeling processes

We model processes as automata that take steps to go from one state to another. Formally, they are represented by transition systems.

Definition 2.4 *Let $\mathcal{P}$ be a process. Then $\mathcal{P}$ is represented by a tuple $< \Sigma, E, \tau, \Sigma_0 >$ where:*

- *Σ is a (possibly infinite) set of states;*

- *E is a set of events;*

  There are different types of events:

  - *exit: normal process termination;*
  - *failLocal: abnormal process termination, caused by local conditions such as system crashes or deliberate user interruptions;*
  - *failRemote: abnormal process termination caused by communication failures;*
  - *send(p, m): message m is sent to process p;*
  - *receive(p, m): message m is received from process p;*
  - *timeout: handles event delays;*
  - *random(b): Random generation of a basic message b. Messages generated at random are globally fresh and cannot be guessed or otherwise computed by anyone.*

- *τ is a transition relation of type $\Sigma \times E \times \Sigma$;*

  τ can be a partial relation, i.e., for some pairs of states s and s', there may not exist an event e such that (s, e, s') is a valid state transition. There exist system-level constraints that make a tuple (s, e, s') an invalid transition. We formalize these constraints later in this subsection.

- *$\Sigma_0 \subseteq \Sigma$ is a set of initial states.*

Valid State Transitions

In Def. 2.4, states were introduced as abstract entities without any internal structure. To model processes as entities that can accumulate, send, and receive messages, however, we need to keep track of messages available to the processes at each state. We use a state variable `MS` (For Message Set) to model this set of messages. Intuitively, a process starts with a certain set of messages and, as it transitions from one state to another, this set changes according to well-defined rules shown in Def. 2.5. These rules characterize valid state transitions.

In what follows, we use $s(v)$ to denote the value of variable v at state s.

Definition 2.5 *Valid state transitions*
Let $(s, e, s') \in \tau$ be a state transition. Then:

1. *If e = send(p, m), then $m \in^* s(\mathtt{MS})$;*
2. *If e = receive(p, m) or e = random(m), then $s'(\mathtt{MS}) = s(\mathtt{MS}) \cup \{m\}$;*
3. *If e = random(b), then $\nexists m \in s(\mathtt{MS})$ such that $b \sqsubseteq m$;*
4. *If e = exit or e = failLocal or e = timeout or e = failRemote or e = send(p, m), then $s'(\mathtt{MS}) = s(\mathtt{MS})$.*

According to Definition 2.5, processes can send only messages constructible from messages available at s; messages received or randomly generated by a process are available to the process at s'; if a basic message b is randomly generated, then b cannot be a submessage of a message available at s, i.e., b is fresh; and *exit*, *failLocal*, *timeout*, *failRemote*, and *send* preserve the set of available messages. We call condition 3 the *random generation condition.*

Local Computations

The specification of a transition system $\mathcal{S}$ determines the set of $\mathcal{S}$'s *computations*. Def. 2.6 is standard.

Definition 2.6 *Local Computations*

Let $\sigma : s_0\ e_1\ s_1\ \ldots$ be an alternating sequence of states and events and $\mathcal{S} =< \Sigma, E, \tau, \Sigma_0 >$ be a simple transition system. σ is a computation of $\mathcal{S}$ if and only if σ is such that

1. $s_0 \in \Sigma_0$;
2. *For all subsequences $s_i\ e_{i+1}\ s_{i+1}$ in σ, $i = 0, 1, \ldots$, $s_i, s_{i+1} \in \Sigma$, $e_{i+1} \in E$, and $(s_i, e_{i+1}, s_{i+1}) \in \tau$;*
3. *If σ is finite, then the last element of σ is a state;*
4. *Given a subsequence $s_i\ e_{i+1}\ s_{i+1}$ in σ, if $e_{i+1} \in \{$exit, failLocal, failRemote$\}$, then σ is finite and s_{i+1} is the last state of σ.*

2.2.3 Modeling Electronic Commerce Systems

As systems formed by concurrent, asynchronous distributed processes, electronic commerce systems can be modeled by compositions where component state machines – modeling individual processes – take turns in making state transitions. We compose our systems using the standard asynchronous composition [2]. In what follows, we assume the existence of a global synchronized clock. In its absence, we can always resort to Lamport clocks [60], which takes into account causal dependencies among the events in a system.

We now present the composition and computations they determine, starting with some auxiliary definitions.

Auxiliary Definitions

Definition 2.7 *Disjoint union*

Given sets $S_1, \ldots, S_n$, we can define $S = S_1 \uplus \ldots \uplus S_n$, the disjoint union of $S_1, \ldots, S_n$, as follows:

$$a \in S_i \leftrightarrow a^i \in S.$$

Intuitively, the elements of a disjoint union are elements from the component sets tagged with superscripts indicating the set to which they originally belong.

In Def. 2.8, we use the notion of equality of events, whose definition is as expected. Given two events e and e', $e = e'$ if and only if they are the same type of event and their parameters (*send*, *receive*, and *random* have parameters) are equal.

Definition 2.8 *Instances of an event*

Let $\sigma : s_0\ e_1\ s_1\ \ldots$ be an alternating sequence of states and events. e_i and e_j are instances of the same event, if $i \neq j$ and $e_i = e_j$.

Composition

Definition 2.9 *Let $\mathcal{S}_1, \ldots \mathcal{S}_n$ be simple transition systems modeling individual processes $p_1, \ldots, p_n$. If $\mathcal{S}_i =< \Sigma_i, E_i, \tau_i, \Sigma_{0i} >, \forall i \in \{1, \ldots, n\}$, then the system formed by $p_1, \ldots, p_n$ can be modeled by a composite transition system $\mathcal{S} =< \Sigma, E, \tau, \Sigma_0 >$, defined as follows:*

1. $\Sigma = \Sigma_1 \times \ldots \times \Sigma_n$;
2. $E = E_1 \uplus \ldots \uplus E_n$;
3. $\tau : \Sigma \times E \times \Sigma$ *is defined as follows:*

$$(\overline{s}, \overline{e}, \overline{s}') \in \tau \textit{ if and only if } \begin{cases} \exists\ i \textit{ such that} \\ \quad \overline{s} = (s_1, \ldots, s_i, \ldots, s_n) \wedge \overline{s}' = (s_1, \ldots, s_i', \ldots, s_n) \wedge \overline{e} = e^i \wedge \\ \quad (s_i, e^i, s_i') \in \tau_i \wedge \\ \quad e^i = random(b) \rightarrow \not\exists m \in \bigcup_{k=1}^{n} s_k(\mathtt{MS}) \textit{ such that } b \sqsubseteq m. \end{cases}$$

4. $\Sigma_0 = \Sigma_{01} \times \ldots \times \Sigma_{0n}$.

Def. 2.9 (3) says that only one process takes a step in each global state transition, and not all local state transitions can derive global state transitions. If a local state transition (s, e, s') is such that $e = \mathrm{random}(b)$, it cannot derive a global state transition unless it satisfies the random generation condition, which requires b to be globally fresh.

System Computations

As with simple transition systems, specification of a composite transition system $\mathcal{S}$ also determines $\mathcal{S}$'s computations.

Definition 2.10 *System Computations*

Let $\sigma : s_0\ e_1\ s_1\ \ldots$ be an alternating sequence of states and events, and $\mathcal{S} =< \Sigma, E, \tau, \Sigma_0 >$ be a composite transition system. σ is a computation of $\mathcal{S}$, if and only if σ is such that

1. $s_0 \in \Sigma_0$;
2. *For all subsequences $s_i\ e_{i+1}\ s_{i+1}$ in σ, $i = 0, 1, \ldots$, $s_i, s_{i+1} \in \Sigma$, $e_{i+1} \in E$, and $(s_i, e_{i+1}, s_{i+1}) \in \tau$;*
3. *If σ is finite, then the last element of σ is a state;*
4. *Given a subsequence $s_i\ e_{i+1}\ s_{i+1}$ in σ, if $e_{i+1} \in \{exit^p, failLocal^p, failRemote^p\}$, then there does not exist $e_j, j > i+1$, such that $e_j = e^p$;*
5. *If σ is infinite, then it satisfies weak (process) fairness, as defined in [67];*
6. *If $e_i = receive^q(p, m) \in \sigma$, then $\exists e_j \in \sigma, j < i$ such that $e_j = send^p(q, m)$;*
7. *For each instance of $send^p(q, m) \in \sigma$, there exists at most one corresponding $receive^q(p, m)$. And for all corresponding sends and receives, $e_i = send^p(q, m)$ and $e_j = receive^q(p, m)$, $i < j$.*

8. *If* $e_i = receive^q(p, m_1)$, $e_j = receive^q(p, m_2)$, *and* $i < j$, *then there exist events* $e_{i'} = send^p(q, m_1)$, $e_{j'} = send^p(q, m_2)$, *and* $i' < j'$.

Conditions $1 - 3$ are exactly those required of computations of simple transition systems. Condition 4 corresponds to condition 4 in Def. 2.6 and says that a component transition system cannot make further progress once it experiences *exit, failLocal or failRemote* events. Condition 5 requires that computations of composite transition system be *fair interleavings* of computations of the component transition systems. Conditions $6 - 8$ model *interprocess communication*: a message needs to be sent before it can be received; messages are received at most once; and in a FIFO manner. Messages are received at most once because communication is unreliable in our model; there can be transmission delays and channel failures. (A message cannot be received multiple times because the communication is connection-oriented.)

In the rest of this dissertation, we call condition 6 the *communication assumption.*

Definition 2.11 *Projections*
Let $\sigma : s_0\ e_1\ s_1 \ldots$ *be a computation of a system* $\mathcal{S}$, *whose components are* $\mathcal{S}_i =< U_i, \Sigma_i, E_i, \tau_i, q_i >$, $i = 1, \ldots, n$. $\sigma \mid p$, *projection of* σ *along component* S_p, *is the sequence obtained by deleting each pair* $e_r\ s_r$ *for which* e_r *is not an event of* S_p, *and replacing each remaining* s_r *by* s_r^p, *automaton* S_p*'s component of the state* s_r.

Projections are computations. Intuitively, $\sigma \mid p$ captures the computation that takes place at node p, while σ takes place at the system as a whole.

Intuitively, if σ is a system computation, then $\sigma \mid p$ captures the node p's local computation.

2.3 Protocol Specification

Before a protocol can be formally analyzed, it needs to be formally specified. In this dissertation, a formal protocol specification consists of:

- A finite set $P = \{p_1, \ldots, p_n\}$ of principals;
- An initial condition I;
- A principal-indexed collection $R = \{R_{p_1}, \ldots, R_{p_n}\}$ of local protocols; and
- A principal-indexed collection $T = \{T_{p_1}, \ldots, T_{p_n}\}$ of trust assumptions.

All, except trust assumptions, are standard components in protocol specifications.

In what follows, state variables are in `typewriter` font and state formulas are first-order formulas defined in a standard way [67].

2.3.1 Specification of Initial Conditions

Initial conditions determine what needs to hold in the beginning of protocol executions. They determine whether a message is public, private, or shareable among a subset of principals, and how the message relates to other messages in the system. For example, they specify that if a principal A holds a key k, thought of as principal B's public key, and k^{-1} is B's private key, then k is the public counterpart of k^{-1}.

Initial conditions are specified by state formulas. State formulas specifying initial conditions may include a non-standard predicate *shareable* to specify which principals *may* share a message at the initial state.

Formally, given a state s and an initial condition I (or more generally a state formula F), the definition of *s satisfies I* (*s satisfies F*), $s \models I$ ($s \models F$), is standard [67], with the interpretation of *shareable* defined as follows:

Definition 2.3.1 *Let s be a state, $P' \subseteq P$ be a set of principals, and m be a message.*

$$s \models \mathit{shareable}(m, P'), \quad \mathit{if} \;\; \forall p \in P - P', m \notin^* s_p(\mathtt{MS}).$$

Def. 2.3.1 says that m is shareable among principals $p_i \in P'$ if m may be found only in the local stores of principals $p \in P'$. Note that m is not required to be in each p_i's local store; instead, it is required to *not be* in the local stores of processes $p_j \in (P - P')$.

In Example 2.3.2, I specifies that k_c^{-1} is C's private key:

Example 2.3.2 *Let k_c and k_c^{-1} be messages and A, B, C be principals.*

$$I = \mathrm{keyPair}(k_c, k_c^{-1}) \wedge \mathrm{shareable}(k_c^{-1}, \{C\}) \wedge \mathrm{shareable}(k_c, \{A, B, C\})$$

specifies that k_c is a public key whose corresponding private key is k_c^{-1}; k_c^{-1} is private to C; and k_c is shareable by A, B, and C.

2.3.2 Specification of Local Protocols

A local protocol R_p consists of a set of rules r of the form:

$$r : \eta \Rightarrow \{E_1, \ldots, E_m\},$$

where η is a state formula and E_i's are event-templates – events whose parameters contain variables – of types *exit, timeout, send, receive*, and *random*. *FailLocal* and *failRemote* do not appear in protocol rules because they model abnormal terminations.

Intuitively, η (*enabling condition*) determines the state condition under which r is applicable, and $E_1, \ldots, E_m$ (*protocol steps*) prescribe alternative events a process should go through from states satisfying η. The non-deterministic choice of which event is taken is made by the environment.

A local protocol needs to specify its valid sequences of steps, as well as messages sent/received at each step. A specification for the protocol in Fig. 1.1, for example, needs to specify that T can send a message to A only after T has received messages from both A and B. Furthermore, it needs to specify that the message T sends to A is the one it received from B.

To be able to specify message flows that a protocol prescribes, one should have ways of talking about the sequence of events a process has experienced up to any given state. One way of making this possible is to add another state variable to the processes. This variable should keep track of the sequence of events a process has experienced so far. In our model, this variable is `H` (for History).

Given a state s, $s(\mathtt{H})$ is a sequence $e_1, \ldots, e_m$ of events, e_m being the one most recently experienced by the process. Given a computation $\sigma : s_0\ e_1\ s_1 \ldots$ of a process, $s(\mathtt{H})$ satisfies the following conditions:

$$\begin{cases} s_0(\mathtt{H}) = \epsilon, \text{ and} \\ s_{i+1}(\mathtt{H}) = \mathit{append}(s_i(\mathtt{H}), e_{i+1}) \end{cases} \tag{2.1}$$

We add these conditions to those in Def. 2.6 of computations.

Using variable `H`, local protocol rules typically appear as follows:

Example 2.12 *An example of a protocol rule:*

$$\exists x : t\_Coin \mid last(\mathtt{H}) = receive(M, x) \Rightarrow \{send(N, x), receive(M, y : t\_Receipt), timeout\}$$

It says that if the last event experienced by a principal is receiving a message of type t_Coin from principal M, then any of the following three types of events may occur next. The principal may send the message it received from M to N, it may receive a message of type t_Receipt from M, or it may timeout while waiting for the receive event to occur.

2.3.3 Specification of Trust Assumptions

Given a principal p, its trust assumptions T_p consists of a set $\{\tau_1, \ldots, \tau_k\}$ of formulas in linear time temporal logic [67] over a sequence of states. Effectively, we only make trust assumptions about trusted parties. If p is not a trusted party, $T_p = \mathsf{true}$.

Example 2.13 *The following is an example of a trust assumption:*

$$\forall x,\ \Box(send(M, x) \in \mathtt{H} \rightarrow receive(C, x) \in \mathtt{H}).$$

It says that the principal is trusted to send M only messages it receives from C.

As discussed in Section 1.3, trust assumptions are protocol-specific assumptions about how trusted parties ought to behave. For protocols that we have investigated, they can all be formalized in linear time temporal logic. Not all linear time temporal logic formulas are formalizations of trust assumptions, however. To be a trust assumption, a formula needs to express a condition that can be satisfied by compliant[2] trusted parties and that, if violated, can compromise the individual interests of a second principal who relies on that trusted behavior. What makes a temporal logic formula a trust assumption therefore is its semantics rather than its syntax.

2.4 Formalizing p-Protective Protocols in Our Model

In Section 1.4.3, we introduced an informal characterization of p-protective protocols. In this section, we formalize that characterization in the context of our model. We first give some preliminary definitions (Section 2.4.1). Then we define a number of different types of executions used to characterize p-protective protocols (Section 2.4.2). Finally, we formalize the concept of p-protective protocols itself (Section 2.4.3).

2.4.1 Preliminaries

In this subsection, we give three definitions used in defining different types of executions in Section 2.4.2. Def. 2.14 defines when a state transition $(s\ e\ s')$ *complies with* a protocol rule r. Def. 2.16 defines when $(s\ e\ s')$ *deceptively-complies with* r. And Def. 2.11 defines a projection of a computation along one of its component automata.

In what follows, the definition of instantiation of an event-template E is standard. An event e is an instantiation of E if there is a substitution $[x_1/m_1, \ldots, x_n/m_n]$ for free variables $x_1, \ldots, x_n$ in E such that $e = E[x_1/m_1, \ldots, x_n/m_n]$.

[2] Compliance is being used informally here. See its formal definition in Section 2.4.2.

Definition 2.14 *Let $(s\ e\ s')$ be a state transition and $r : \eta \Rightarrow \{E_1, \ldots, E_n\}$ be a protocol rule. $(s\ e\ s')$ complies with r,*

$$(s\ e\ s') \models r,$$

if and only if:

1. *$s \models \eta$; and*
2. *(a) $\exists E_i \in \{E_1, \ldots, E_n\}$ such that e is an instantiation of $E_i\theta$, where θ is a substitution of bound variables in E_i determined by s; or*
   (b) $\exists E_i \in \{E_1, \ldots, E_n\}$ such that E_i specifies a send or a receive event, and $e = failRemote$.

Intuitively, $(s\ e\ s')$ complies with r, if η is satisfied in s, and (a) e is an instantiation of an event-template E_i whose bound variables are substituted according to the protocol, or (b) e could not be a send or receive event, as specified by r, because of a communication channel failure.

Note that since $(s\ e\ s')$ is a valid state transition, the value of s' cannot be arbitrary (Definition 2.5 and conditions 2.1, page 21). This justifies why we do not have explicit conditions on s' in Definition 2.14.

Example 2.15 *Let $(s\ e\ s')$ be a state transition and $r : \exists x : t \mid \text{receive}(Y, x) \in \mathtt{H} \Rightarrow \{\text{receive}(Z, z : t \times t), \text{send}(Z, x)\}$ be a protocol rule.*

$$(s\ e\ s') \models r$$

if and only if there exists a message $m : t$ such that

1. *$receive(Y, m) \in s(\mathtt{H})$, and*
2. *(a) $e = receive(Z, m')$, for some message $m' : t \times t$; or*
   (b) $e = send(Z, m)$; or
   (c) $e = failRemote$.

Note that, in Example 2.15, the message sent to Z must be received from Y, since the variable x in $\text{send}(Z, x)$ is bound.

Definition 2.16 *Let $(s\ e\ s')$ be a state transition and r be a protocol rule. $(s\ e\ s')$ deceptively-complies with r*

$$(s\ e\ s') \models_d r$$

if and only if:

1. *$s \models \eta$; and*
2. *$\exists E_i \in \{E_1, \ldots, E_n\}$ such that e is an instantiation of E_i, but e is not an instantiation of $E_i\theta$, where θ is a substitution of bound variables in r determined by s.*

Intuitively, $(s\ e\ s')$ deceptively-complies with r if η is satisfied in s and e is an instantiation of an event-template E_i, whose bound variables are not substituted according to the protocol. This is a deceptive compliance because e only appears as an event prescribed by r; it is not an event prescribed by r because its message parameters are not instantiated according to r (the bogus message needs to have the right type, however).

Note that in Definition 2.16 the enabling condition η of r is satisfied in s. This means that the state transition does not violate the conditions for the firing of r, which typically specifies (explicit or implicitly) not only protocol steps that should have happened, but also specific conditions that these previous steps must satisfy.

Deceptive compliance is a form of non-compliance. A second form of non-compliance is introduced in Section 2.4.2.

Example 2.17 *Let* $(s\ e\ s')$ *be a state transition, and*
$r : \exists x : t \mid \text{receive}(Y, x) \in \mathtt{H} \Rightarrow \{\text{receive}(Z, z : t \times t), \text{send}(Z, x)\}$ *be a protocol rule.*

$$(s\ e\ s') \models_d r$$

if and only if there exists a message $m : t$ *such that*

1. *receive*$(Y, m) \in s(\mathtt{H})$, *and*

2. $e =$ *send*(Z, m'), *where* $m' \neq m$.

Note that, in Example 2.17, the message sent in event e is not the one prescribed by r: a different message was sent instead. Deceptive compliance will be used later to characterize deceptive executions, where a process apparently follows the protocol (the order and the format of message exchanges are as prescribed by the protocol), but cheats by using arbitrary messages.

2.4.2 Protocol Executions

In this subsection, we define different types of protocol executions used to define p-protective protocols. We obtain these protocol executions by filtering the set of all computations by protocol specifications. Our classification (Fig. 2.1) is based on the notion of *compliance* and takes both the type and the source of failures into account. We also define *trustworthy executions*, which is based on the trust assumptions of a protocol.

Compliant, Abortive, and Deceptive Executions

Traditionally, protocol executions have been classified according to failure modes. Classifications based on failure modes distinguish executions in terms of whether or not they fail and how they fail. For the purpose of defining p-protective protocols, however, we would like to distinguish executions in terms of whether or not their corresponding processes comply with the protocol, and how they deviate from it. Failure and deviation differ in that all deviations are failures, but not all failures are deviations. The key factor that makes a failure a deviation is the origin of the failure. If the failure is attributed to local causes, then it is a deviation; if the cause is remote, then it is not.

Guided by the notions of compliance and deviation, we propose the classification shown in Fig. 2.1. It is based on our classifying *failRemote* as an event caused by remote failures, and *failLocal* and deceptive steps as results of local problems. Intuitively, this classification captures the notion

$$\text{execution} \begin{cases} \text{compliant: failure-free + abnormal termination with } \textit{failRemote} \\ \text{deviant} \begin{cases} \text{abortive: deviates with a } \textit{failLocal} \\ \text{deceptive: deviates with deceptive steps} \end{cases} \end{cases}$$

Figure 2.1: Classification of local protocol executions.

of compliance and the ways in which one can deviate from compliance. An execution is compliant if and only if it follows the local protocol, deviating from it only if it terminates prematurely because of communication problems. An execution is deviant if and only if it terminates prematurely because of local factors or it deceives.

We formalize the various executions below.

Definition 2.18 *Compliant executions of a local protocol*

Let $\sigma : s_0\ e_1\ s_1 \ldots$ be a computation of a single process and R_p be a local protocol. σ is a compliant execution of R_p if and only if

$$\forall (s_i\ e_{i+1}\ s_{i+1}) \in \sigma, \exists r \in R_p \textit{ such that } (s_i\ e_{i+1}\ s_{i+1}) \models r.$$

Def. 2.18 says that σ is a *compliant* execution of R_p if all its state transitions comply with rules $r \in R_p$. Note that compliant executions can terminate abnormally with a *failRemote* event, or can be failure-free – if it does not terminate with a *failRemote*.

Definition 2.19 *Compliant executions of a protocol*

Given a computation $\sigma : s_0\ e_1\ s_1 \ldots$ for a system of processes and a protocol $< P, I, R, T >$, σ is a compliant execution of the protocol if and only if:

1. *$s_0 \models I$; and*
2. *$\forall p \in P, \sigma \mid p$ is a compliant execution of R_p.*

Def. 2.19 says that a computation is a compliant execution of a protocol if and only if its initial state satisfies the initial condition of the protocol, and its projections (along each of the principals) are compliant execution of its principals' local protocols.

Definition 2.20 *Abortive executions of a local protocol*

Let $\sigma : s_0\ e_1\ s_1 \ldots s_n$ be a computation of a single process and R_p be a local protocol. σ is an abortive execution of R_p if and only if

$$s_0\ e_1\ s_1 \ldots s_{n-1} \textit{ is a compliant execution of } R_p \textit{ and } e_n = \textit{failLocal}.$$

Definition 2.21 *Deceptive executions of a local protocol*

Let $\sigma : s_0\ e_1\ s_1 \ldots$ be a computation of a single process and R_p be a local protocol. σ is a deceptive execution of R_p if and only if

σ is not a compliant execution of R_p only because there are $(s_i\ e_{i+1}\ s_{i+1}) \in \sigma$ such that $(s_i\ e_{i+1}\ s_{i+1}) \models_d r$ for a rule $r \in R_p$.

Def. 2.21 says that σ is a deceptive execution of a local protocol if and only if σ deviates from a compliant execution in that it takes deceptive steps.

Definition 2.22 *Abortive (deceptive) execution of a protocol*

Let $\sigma : s_0\ e_1\ s_1 \ldots$ be a computation of a system and $< P, I, R, T >$ be a protocol. σ is an abortive (deceptive) execution of the protocol if and only if:

1. *$s_0 \models I$;*
2. *There exists at least one $p \in P$ such that $\sigma \mid p$ is an abortive (deceptive) execution of R_p; and*
3. *For the remaining principals p', $\sigma \mid p'$ is a compliant execution of $R_{p'}$.*

Def. 2.22 says that a computation is an abortive (deceptive) execution of a protocol if its initial state satisfies the protocol's initial condition, and at least one of its projections – along each of the principals – is an abortive (deceptive) execution, while all others are compliant executions.

According to our definitions, neither the abortion nor the deception mode subsumes the other. Given a process, its set of compliant, abortive, and deceptive executions are mutually disjoint. This is not necessarily so. We could define executions where one process aborts, while another deceives, and yet another complies with the protocol, for example. For now, we will work with the current set of definitions, for simplicity.

Definition 2.23 *Maximal execution of a protocol*

Let σ be a (compliant/abortive/deceptive) execution of a protocol Π. σ is a maximal (compliant/abortive/deceptive) execution of Π if and only if:

there does not exist a (compliant/abortive/deceptive) execution σ' of Π, such that σ is a proper prefix of σ'.

Def. 2.23 says that σ is a maximal execution if and only if it cannot be further extended.

Trustworthy Executions

In what follows, $\models^*$ denotes the standard satisfaction relation defined for linear time temporal logic formulas [67].

Definition 2.24 *Trustworthy executions of a protocol*

Let σ be a (compliant/abortive/deceptive) execution of a protocol $< P, I, R, T >$, where $T = \{T_1, \ldots, T_k\}$, and $\sigma \setminus s$ be the sequence of states projected from σ. σ is trustworthy if and only if:

$$\sigma \setminus s \models^* \bigwedge_{i \in \{1,\ldots,k\}} \bigwedge_{\tau \in T_i} \tau, \text{ or equivalently, } \sigma \setminus s \models^* T.$$

Def. 2.24 says that trustworthy executions are those whose sequence of states satisfy all the trust assumptions of a protocol.

2.4.3 p-Protective Protocols

In this subsection, we formally define p-protective protocols. Our formalization captures the following insights. First, in commerce transactions, different parties have different interests to preserve. Second, electronic commerce protocols should be designed to protect their user's interests against communication failures and both intentional and unintentional misbehaviors of other users. Lastly, trust assumptions should be taken into account when defining p-protective protocols: often they are what protocols rely on to achieve p-protection.

In what follows, we consider three deviation modes: C (compliance), A (abortion), and D (deception); and use $E(\Pi)^C$, $E(\Pi)^A$, and $E(\Pi)^D$ to denote respectively the set of maximal compliant, abortive and deceptive executions of a protocol Π. Also, let x range over $\{C, A, D\}$ and p over the set of principals of Π, then we use $E(\Pi)^x_p$ to denote the set of maximal x-executions of Π where p's projection is a compliant execution of R_p.

Our definitions below apply only to protocols whose maximal compliant executions are trustworthy. This is an expected restriction: if trust assumptions specify behaviors that trusted parties should exhibit even under deviation modes, then these same behaviors should be exhibited under the compliance mode in the first place.

Definition 2.25 *Let $\Pi =< P, I, R, T >$ be a protocol, $p \in P$ be a principal, P_p be p's protection property, and x be the deviation mode we consider.* Π *protects p's interests, under x deviation mode, if and only if*

1. $\forall \sigma \in E(\Pi)^C, \sigma \models^* P_p$; *and*
2. $\forall \sigma \in E(\Pi)^x_p$, *if* $\sigma \models^* T$ *then* $\sigma \models^* P_p$.

Definition 2.26 *p-protective protocols*

Let $\Pi =< P, I, R, T >$ be a protocol, $p \in P$ be a principal, P_p be p's protection property, and x be the deviation mode we consider. Π *is p-protective under x mode if and only if* Π *protects p's interests under x mode.*

Defs. 2.25 and 2.26 can be used to derive the definition of p-protective protocols under different deviation modes. Under C mode, $E(\Pi)^C$ is the set of executions one needs to examine, since $E(\Pi)^C \cup E(\Pi)^C_p = E(\Pi)^C$ and all executions in $E(\Pi)^C$ are trustworthy[3]. Under A and D modes, $E(\Pi)^C$ plus trustworthy executions in $E(\Pi)^C \cup E(\Pi)^A_p$ and $E(\Pi)^C \cup E(\Pi)^D_p$ are respectively the sets of executions that need to be examined. Intuitively, a protocol is p-protective under compliance mode if and only if for all compliant executions σ, σ satisfies P_p; Π is p-protective under abortion (deception) mode if and only if for all compliant executions and trustworthy abortive (deceptive) executions σ where p behaves compliantly, σ satisfies P_p.

Example: We say that a sale protocol is customer-protective under abortion mode if customer's interests are protected in all $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_{customer}$.

It is often desirable, though not necessary, that a protocol protects the interests of all its participants (or, more precisely, of those that have interests to preserve). We call protocols that do so all-protective protocols.

[3]Section 2.3.3 has a brief justification of why all executions in $E(\Pi)^C$ are trustworthy.

Definition 2.27 *All-protective protocols*

Let $\Pi = < P, I, R, T >$ *be a protocol;* $P' = \{p_1, \ldots, p_k\}$ *be the set of principals that have interests to preserve;* $P_{p_i}, i = 1, \ldots, k$, *be respectively* p_i*'s protection properties; and* x *be the deviation mode we consider.* Π *is all-protective under* x *mode if and only if*

$$\Pi \textit{ is } p_i\textit{-protective, under } x \textit{ mode, for all } p_i \in P'.$$

Example: We say that a sale protocol Π is all-protective under deception mode, if Π is both customer-protective and merchant-protective under deception mode.

Definitions 2.25, 2.26, and 2.27 give a general and abstract framework in which the notion of protection is formalized. The abstract definitions can be refined by instantiating P_p's with more concrete properties. For instance, if Π is a sale protocol, then there are two P_p's of interest, namely, P_C (customer's protection property) and P_M (merchant's protection property), which have respectively the following general forms:

> P_C: *If the merchant receives the payment, then the customer must have received or must eventually receive the goods.*
>
> P_M: *If the customer receives the goods, then the merchant must have received or must eventually receive the payment.*

If Π is a withdrawal protocol, then there are also two P_p's of interest, namely, P_B (bank's protection property) and P_U (user's protection property).

For the protocols we analyzed, P_p's were formalizable in linear time temporal logic, which justifies our using $\sigma \models^* P_p$ to formalize preservation of protection property P_p by execution σ.

The informal characterizations above are still general and abstract. When investigating a specific protocol, they need to be further refined. For example, receiving payment may mean receiving cash in one protocol, and receiving a check in another. In other words, the formalization of protection properties requires *protocol-specific* information and cannot be made concrete until after a protocol is given.

We can refine the informal characterizations above in a second dimension. In *closed* characterizations of p-protection, we require protection properties to be satisfied at the end of protocol runs. For example, "receiving an item" is "actually receiving an item during a protocol run". In *open* characterizations, we allow protection properties to be – temporarily – violated, as long as they can be restored later. In this case, "receiving an item" can be translated into "actually receiving an item during a protocol run" *or* "receiving non-repudiable evidence of entitlement during a protocol run", so that one can use it later in court to reclaim the item.

2.5 Assumptions and Their Implications

We make a number of assumptions in our model. In this section, we discuss their implications/limitations.

In our model, we assume that each agent runs one process at a time. This assumption simplifies our model, but prevents us from uncovering problems (for example [63]) that can arise only when multiple processes run concurrently.

We assume communication security: if a message is received, then it is received by the intended recipient, authenticated, confidential, and untampered. Effectively, we assume that the processes we consider use the service of an underlying layer of security protocols. Layering of protocols may lead to unexpected interactions, and unless we can assure ourselves that no undesirable interactions exist, our conclusion about whether a protocol is all-protective is not definitive. Our approach divides the task of analyzing layered protocols into two subtasks: we first analyze individual layers, and then the interactions between the layers. In this dissertation, we focused on the first subtask.

Perfect encryption is another assumption we make. It says, in its most basic form, that the only way to obtain any information from an encrypted message is to have the right decryption key. In our work, this assumption extends to other cryptographic primitives (e.g., blinding and unblinding) as well. In general, however, cryptographic primitives may be amenable to different types of analysis (e.g., timing and probabilistic), and these analysis may have implications at the protocol level. Like communication security assumption, this assumption allows us to decompose the task of analyzing protocols into subtasks: in this case, analysis of protocols themselves, assuming ideal cryptography; and analysis of interactions between the protocols and lower-level properties of cryptographic primitives.

In our specification of protocol rules, we assume that, at reception, recipients of messages only check for typing of messages they receive. Rule R_{M_5} (page 70) in NetBill, for example, prescribes M to receive only messages of type *t_sepo*. We made this assumption because it offers us a reasonable level of abstraction. We can do more or less checks at reception, however, and the change does not bring critical consequences. For example, under deception mode, principals currently receive messages first, and then learn that they are bogus. If more checks are conducted at reception, one can reject receiving bogus messages altogether. This difference is inconsequential because checks are effectively needed only right before certain critical steps, and it does not matter when checks take place, as long as they occur before these steps.

Finally, we assume that protocol executions are finite. In fact, our definition of p-protective protocols takes into account only maximal executions, and our analysis of protocols relies on the fact that all compliant/abortive/deceptive executions of a protocol are finite. Not all protocols have finite executions, however. For example, there can be servers that can respond to requests infinitely often. Our framework is not applicable to this class of protocols.

Chapter 3

Franklin and Reiter's Protocol

In this chapter, we present our first case study, a fair exchange protocol [42] proposed by Franklin and Reiter. We specify the protocol – henceforth referred to as FR's protocol – in our protocol specification formalism (Section 2.3) and analyze it with respect to protection of individuals' interests using our framework (Section 2.4). Since we are interested in protection of all parties equally, we analyze it with respect to all-protection. FR's protocol differs from previous work[1] ([57, 58, 5, 28, 89, 31]) on fair exchange in that it uses *semi-trusted* third parties instead of *fully-trusted* ones.

FR's protocol is the simplest among the three we analyze, in terms of both the protocol's functionality and its protection properties. Our analysis does not reveal any surprises: the protocol is all-protective under all three deviation modes, as long as communication links are reliable.

Our main contribution in this case study is a formalization of semi-trusted third parties. Franklin and Reiter introduce the notion of semi-trustworthiness in electronic commerce protocols [42], but they do not fully develop it. In particular, they do not take it into account in their (informal) analysis of the protocol. In our framework, we can formalize the notions of semi-trustworthiness and conspiracy behaviors using trust assumptions, and provide a clear-cut analysis of the protocol.

The rest of this chapter is structured as follows. In Section 3.1, we introduce the protocol and assumptions made by its designers, and justify why it was chosen for our first case study. In Section 3.2, we present an abstract formulation of the mathematical building blocks used in it. In Section 3.3, we formalize both the protocol and the protection properties. In Section 3.4, we show the main results and their analyses. Detailed proofs are reserved for Section 3.5. Finally, in Section 3.6, we summarize the findings of our analysis and discuss our contributions towards formalizing semi-trustworthiness.

3.1 Introduction

3.1.1 Preliminaries

FR's protocol [42] enables two parties X and Y to exchange documents with the mediation of a third party Z. It was designed to guarantee "fair exchange", i.e., no party should be able to

[1] We have not included here earlier work on "contract signing" [10, 11, 88, 65, 85, 27], because they are inefficient (not practical) and not applicable to electronic commerce protocols.

gain any advantage by quitting prematurely or otherwise misbehaving. Franklin and Reiter take fair exchange in a strict sense: a party engaged in an exchange should not be able to trick the other into accepting a document different from the one that is expected. This view differs from that of NetBill [84], for example. In NetBill, it is possible for a vendor to provide an arbitrary document during an online exchange. The protocol only enables such cheating to be detected after the exchange occurs, using mediation outside the scope of the protocol.

FR's protocol differs from previous solutions to fair exchange in that it assumes semi-trusted – instead of fully-trusted – third-parties. In Franklin and Reiter's definition, a third party is fully-trusted if it does not misbehave at all, and is semi-trusted if it misbehaves on its own, but does not conspire with either of the main parties.

Franklin and Reiter assume a model in which documents are encrypted under a key and encrypted documents are publicly accessible. To get a document, one only needs to get the decryption key from the owner of the document. To guarantee that the owner of a document does not fool interested parties with wrong decryption keys, the authors assume that parties in the system have a one-way hash $f(K)$ of the keys K they desire. During an exchange, they provide these hash values to the intermediary, who uses them to verify whether the keys supplied by document owners are the expected ones. Through this mechanism, exchanges succeed only with desired keys.

The assumption that parties know one-way hashes of keys they desire holds in some existing protocols (e.g., [49, 77, 39, 48]). Franklin and Reiter assume the existence of a public database of tuples, each consisting of a description of the contents of a data file (e.g., the title of a movie); an encryption of the data file (e.g., the movie) under a secret key K ; a one-way hash $f(K)$ of the secret key; and an authority's signature on the preceding information, which serves as the authority's appraisal that the decryption of the encrypted data file using K will indeed produce the described item.

For these hash values to be useful, however, there cannot be a key $K' \neq K$ such that $f(K') = f(K)$; otherwise, the document owner can simply run the protocol with K'. To ensure that this does not happen, f is required to be a collision-free function. Collision-freeness is, in truth, stronger than what is strictly necessary [42]. We adopt it here for simplicity.

Some mathematical details are important to the functioning of FR's protocol. f must be a function from G to G ($f : G \to G$), where G is a algebraic group. Moreover, f must have the property that there exists a computable function $F : G \times G \to G$ such that $F(x, f(y)) = f(xy)$. Franklin and Reiter give a few concrete one-way hash functions with these properties [42].

3.1.2 The Protocol

Fig. 3.1 shows the protocol as it was given [42]. It assumes that X holds a decryption key K_x, Y holds a decryption key K_y, and X and Y are interested in exchanging K_x and K_y. It also assumes that both X and Y know a one-way function $f : G \to G$ on the key space. For the remainder of this chapter, K_x and K_y will be treated abstractly; that is, merely seen as secrets, instead of decryption keys.

To give K_x to Y, X first splits K_x into two shares. One of the shares is x, a member of G generated at random by X. The other is $K_x x^{-1}$, the product of K_x with the inverse of x (We use product to mean the group operation in G). X then sends x to Y (step 1). Y takes an analogous step (step 2). Once X has received y from Y, X sends to Z (step 3) the other share $K_x x^{-1}$ of its secret, the hash value $f(y)$ of y, and the hash value $f(K_y)$ of the secret K_y it desires. $f(K_x)$ is also provided for redundancy. Y takes an analogous step (step 4). Once Z

$$\begin{array}{lll}
1. & X \rightarrow Y: & x \\
2. & Y \rightarrow X: & y \\
3. & X \rightarrow Z: & f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y) \\
4. & Y \rightarrow Z: & f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x) \\
5. & Z \rightarrow X: & K_y y^{-1} \\
6. & Z \rightarrow Y: & K_x x^{-1}
\end{array}$$

The various symbols denote:

$$\begin{array}{rl}
x,\ y: & \text{Randomly generated members of a algebraic group } G; \\
K_x,\ K_y: & X\text{'s and } Y\text{'s secrets, respectively;} \\
f: & \text{a one-way function from } G \text{ to } G.
\end{array}$$

Figure 3.1: FR's protocol.

has received messages from both X and Y, Z can verify the correctness of K_x's splitting using the equality $\forall a \in G, b \in G, F(a, f(b)) = f(ab)$. In this case, $F(K_x x^{-1}, f(x))$ should be equal to $f(K_x)$. The correctness of K_y's splitting can be verified analogously. If both secrets are split correctly, Z forwards $K_y y^{-1}$ to X (step 5) and $K_x x^{-1}$ to Y (step 6). X can then use y and $K_y y^{-1}$ to reconstitute K_y; and analogously with Y. If either secret is split incorrectly, no forwarding takes place.

Franklin and Reiter assume reliable and secure communication networks.

FR's protocol has a number of extensions [42]. In this dissertation we focus on the basic protocol.

3.1.3 Discussion

FR's protocol is an ideal candidate for our first case study because it explores the notion of semi-trusted third parties. A semi-trusted third party, according to Franklin and Reiter, is a trusted party that may misbehave on its own, but does not "conspire" with either of the main parties. They do not precisely distinguish "misbehaviors of one's own" from "conspiracy misbehaviors". Informally, however, they intend "misbehaviors of one's own" to mean misbehaviors that do not bring advantages to one of the main parties, and "conspiracy misbehaviors" to mean those that would. Their interpretation of conspiracy is non-standard in that a misbehaving party can conspire with an honest party without the honest party's consent. They call this type of conspiracy "passive conspiracy".

After the initial discussion, the notion of semi-trustworthiness is not further taken into account however. For example, for their characterization of fair exchange protocols, Franklin and Reiter define two types of parties: honest parties are those that follow the protocol, and misbehaving parties are those that do not. A protocol satisfies fair exchange property if the following hold at the end of an execution:

1. If all three parties are honest, then X learns K_y and Y learns K_x.

2. If X and Z are honest, then Y learns nothing useful about K_x unless X learns K_y.

3. If Y and Z are honest, then X learns nothing useful about K_y unless Y learns K_x.

4. If X and Y are honest, then Z learns nothing useful about K_x or K_y.

Franklin and Reiter's characterization of fair exchange protocols has a number of weaknesses. First, it assumes 1-resilience, i.e., at most one of X, Y, and Z misbehaves. In an open network where one is likely to interact with strangers, this assumption seems too limiting. It is particularly limiting in their protocol because they advocate using random members of the network as intermediaries.

Second, it does not model semi-trustworthiness. In the characterization of fair exchange protocols above, Z either follows the protocol or can misbehave arbitrarily. Note that the notion of a semi-trusted Z that can misbehave on its own, but not conspire with either of the main parties is not captured in this characterization. Arbitrarily-misbehaving Zs pose special challenges to characterizations of fairness. For example, if Z disrupts the exchange in a way that X learns K_y without Y learning K_x, then fairness is certainly compromised. Franklin and Reiter do realize this problem; according to them "It is debatable whether protection against passive conspiracies should be included in our model" [42].

We argue that these weaknesses can be eliminated if semi-trustworthiness is explicitly modeled, and Z is assumed not to display conspiracy behaviors.

Our framework enables explicit modeling of conspiracy behaviors and semi-trustworthiness. Conspiracy behaviors can be modeled as behaviors that violate trust assumptions and semi-trusted parties can be modeled as parties that do not exhibit conspiracy behaviors. In fact, in our characterization of p-protective protocols, we prescribe analyzing protocols under the assumption that trusted parties can misbehave, but cannot violate their trust assumptions. By restricting the type of misbehaviors trusted parties can exhibit, we can eliminate the assumption that only one party misbehaves in each execution, and answer Franklin and Reiter's of question of whether conspiracy behaviors should be considered in the model.

3.2 Abstract Formulation of the Building Blocks

In this section, we present an abstract formulation of the mathematical building blocks used in FR's protocol.

To "split" a key K into Kx^{-1} and x, we first generate x at random, then *mask* K using x. We represent Kx^{-1} abstractly by

$$mask(K, x),$$

where mask corresponds to inverting x, then forming the product of K with the inverse of x. To retrieve K from Kx^{-1}, we apply another product operation: $(Kx^{-1})x$, which converts to K. This retrieval is abstractly represented by

$$\mathrm{unmask}(\mathrm{mask}(K, x), x) \doteq K,$$

where unmask corresponds to the product operation and $\doteq$ denotes a conversion. Henceforth, we call x a *masking key*, and $\mathrm{mask}(K, x)$ a *masked secret*.

We represent the one-way hash function f abstractly by *hash*, and the auxiliary function F by *aux_hash*. Thus, $F(x, f(y)) = f(xy)$ can be mapped into

$$aux\_hash(x, hash(y)) \doteq hash(unmask(x, y)).$$

1. $X \rightarrow Y:\ x$
2. $Y \rightarrow X:\ y$
3. $X \rightarrow Z:\ hash(K_x)\ hash(K_y)\ mask(K_x, x)\ hash(y)$
4. $Y \rightarrow Z:\ hash(K_y)\ hash(K_x)\ mask(K_y, y)\ hash(x)$
5. $Z \rightarrow X:\ mask(K_y, y)$
6. $Z \rightarrow Y:\ mask(K_x, x)$

The various symbols denote:

$x,\ y$: Randomly generated masking keys;
$K_x,\ K_y$: X's and Y's secrets, respectively.

Figure 3.2: An abstract formulation of FR's protocol.

Finally we use an abstract type t to represent groups G.
In what follows, we list the functions and their conversion rules.

Definition 3.1 *Abstract building blocks*

1. *Secrets have type t, i.e., if K is a secret, then $K : t$;*
2. *Masking keys have type t, i.e., if x is a masking key, then $x : t$;*
3. *Functions:*
   - *mask: $t \times t \rightarrow t$;*
   - *unmask: $t \times t \rightarrow t$;*
   - *hash: $t \rightarrow t$;*
   - *aux_hash: $t \times t \rightarrow t$;*
4. *Conversion rules:*
   - $\forall K : t, x : t, y : t,\ unmask(mask(K, x),\ y) \doteq K$, *if and only if* $x = y$;
   - $\forall x : t, y : t,\ aux\_hash(x,\ hash(y)) \doteq hash(unmask(x, y))$.

Note that we make the *Perfect Encryption Assumption* [45, 12] in our framework. In its most basic form, this assumptions says that the only way to obtain any information from an encrypted message is to have the right decryption key. The conversion rule regarding *mask* and *unmask* specifies that a masked secret cannot be retrieved unless it is unmasked by the masking key.

The protocol in Fig. 3.1 can now be represented as in Fig. 3.2.

3.3 Formalizing the Protocol and the Protection Properties

In this section, we specify FR's protocol and protection properties different principals want to have guaranteed.

3.3.1 Specification of the Protocol

Principals of the Protocol

$$\mathcal{P} = \{X, Y, Z\}$$

Initial Conditions

Let κ_x and κ_y be respectively X's and Y's secrets, the initial condition

$$\mathcal{I} : \text{hash}(\kappa_x) = \rho_x \wedge \text{hash}(\kappa_y) = \rho_y \wedge \{\kappa_x, \rho_y\} \subseteq \mathtt{MS}^{\mathtt{X}} \wedge \{\kappa_y, \rho_x\} \subseteq \mathtt{MS}^{\mathtt{Y}}$$

says that ρ_x and ρ_y are respectively κ_x's and κ_y's hashes; principal X has both κ_x and ρ_y in its local store; and principal Y has κ_y and ρ_x in its local store.

Local Protocols

In the specification below, we introduce timeouts in some select places. Intuitively, they are introduced in points where a send or a receive event in an already initiated communication is expected. In these points, timeouts handle communication delays and disruptions. If a send or a receive is critical, however, then no timeout is introduced and the send or the receive is allowed to take place eventually.

In what follows, w, x, y and z are variables.

X's local protocol $\mathcal{R}_X$ consists of the following rules:

R_{X_1}: $\nexists\, x : t \mid \text{random}(x) \in \mathtt{H} \Longrightarrow \{\text{random}(y : t)\}$

R_{X_3}: $\exists\, x : t \mid \text{random}(x) \in \mathtt{H} \wedge (\nexists\, y : t \mid \text{send}(Y, y) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{send}(Y, x), \text{timeout}\}$

R_{X_2}: $(\nexists x : t \mid \text{receive}(Y, x) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow \{\text{receive}(Y, y : t)\}$

R_{X_4}: $(\nexists\, y : t \mid \text{receive}(Y, y) \in \mathtt{H}) \wedge \exists\, x : t \mid \text{send}(Y, x) \in \mathtt{H} \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{receive}(Y, z : t), \text{timeout}\}$

R_{X_5}: $\exists\, y : t \mid \text{receive}(Y, y) \in \mathtt{H} \wedge \exists\, x : t \mid \text{send}(Y, x) \in \mathtt{H} \wedge$
$(\nexists\, z : t \times t \times t \times t \mid \text{send}(Z, z) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{send}(Z, \text{hash}(\kappa_x)\ \rho_y\ \text{mask}(\kappa_x, x)\ \text{hash}(y)), \text{timeout}\}$

R_{X_6}: $\exists\, x : t \times t \times t \times t \mid \text{send}(Z, x) \in \mathtt{H} \wedge (\nexists\, y : t \mid \text{receive}(Z, y) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{receive}(Z, z : t)\}$

R_{X_7}: $\text{last}(\mathtt{H}) = \text{timeout} \vee \exists\, x : t \mid \text{last}(\mathtt{H}) = \text{receive}(Z, x) \Longrightarrow \{\text{exit}\}$

Y's local protocol $\mathcal{R}_Y$ is identical to $\mathcal{R}_X$, and consists of the following rules:

R_{Y_1}: $\nexists\, x : t \mid \text{random}(x) \in \mathtt{H} \Longrightarrow \{\text{random}(y : t)\}$

R_{Y_3}: $\exists\, y : t \mid \text{random}(y) \in \mathtt{H} \wedge (\nexists\, x : t \mid \text{send}(X, x) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{send}(X, y), \text{timeout}\}$

R_{Y_2}: $(\nexists x : t \mid \text{receive}(X,\ x) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow \{\text{receive}(X,\ y : t)\}$

R_{Y_4}: $(\nexists\ x : t \mid \text{receive}(X,\ x) \in \mathtt{H}) \wedge \exists\ y : t \mid \text{send}(X,\ y) \in \mathtt{H} \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{receive}(X,\ z : t),\ \text{timeout}\}$

R_{Y_5}: $\exists\ x : t \mid \text{receive}(X,\ x) \in \mathtt{H} \wedge \exists\ y : t \mid \text{send}(X,\ y) \in \mathtt{H} \wedge$
$(\nexists\ z : t \times t \times t \times t \mid \text{send}(Z,\ z) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{send}(Z,\ \text{hash}(\kappa_y)\ \rho_x\ \text{mask}(\kappa_y,\ y)\ \text{hash}(x)),\ \text{timeout}\}$

R_{Y_6}: $\exists\ x : t \times t \times t \times t \mid \text{send}(Z,\ x) \in \mathtt{H} \wedge (\nexists\ y : t \mid \text{receive}(Z,\ y) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{receive}(Z,\ z : t)\}$

R_{Y_7}: $\text{last}(\mathtt{H}) = \text{timeout} \vee \exists\ x : t \mid \text{last}(\mathtt{H}) = \text{receive}(Z,\ x) \Longrightarrow \{\text{exit}\}$

Finally Z's local protocol $\mathcal{R}_Z$ consists of the following rules:

R_{Z_2}: $(\nexists\ y : t \times t \times t \times t \mid \text{receive}(Y,\ y) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{receive}(Y,\ z\!:\ t \times t \times t \times t),\ \text{timeout}\}$

R_{Z_3}: $(\nexists\ x : t \times t \times t \times t \mid \text{receive}(X,\ x) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{receive}(X,\ w\!:\ t \times t \times t \times t),\ \text{timeout}\}$

R_{Z_4}: $\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \wedge$
$\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid \text{receive}(Y,\ y_1\ y_2\ y_3\ y_4)) \in \mathtt{H} \wedge$
$x_1 = y_2 = \text{aux\_hash}(x_3, y_4) \wedge y_1 = x_2 = \text{aux\_hash}(y_3, x_4) \wedge$
$(\nexists\ y : t \mid \text{send}(X,\ y) \in \mathtt{H}) \wedge (\nexists\ x : t \mid \text{send}(Y,\ x) \in \mathtt{H}) \wedge \text{last}(\mathtt{H}) \neq \text{timeout} \Longrightarrow$
$\{\text{send}(X, y_3),\ \text{send}(Y, x_3),\ \text{timeout}\}$

R_{Z_5}: $\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \wedge$
$\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid \text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \wedge$
$\neg\ (x_1 = y_2 = \text{aux\_hash}(x_3, y_4) \wedge y_1 = x_2 = \text{aux\_hash}(y_3, x_4)) \Longrightarrow \{\text{exit}\}$

R_{Z_6}: $\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \wedge$
$\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid \text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \wedge$
$\exists\ y : t \mid \text{send}(X, y) \in \mathtt{H} \wedge\ \nexists\ x : t \mid \text{send}(Y, x) \in \mathtt{H} \Longrightarrow \{\text{send}(Y, x_3)\}$

R_{Z_7}: $\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid \text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \wedge$
$\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \wedge$
$\exists\ x : t \mid \text{send}(Y, x) \in \mathtt{H} \wedge\ \nexists\ y : t \mid \text{send}(X, y) \notin \mathtt{H} \Longrightarrow \{\text{send}(X, y_3)\}$

R_{Z_8}: $\text{last}(\mathtt{H}) = \text{timeout} \Longrightarrow \{\text{exit}\}$

R_{Z_9}: $\exists\ y : t \mid \text{send}(X,\ y) \in \mathtt{H} \wedge \exists\ x : t \mid \text{send}(Y,\ x) \in \mathtt{H} \Longrightarrow \{\text{exit}\}$

Trust Assumptions

X and Y are not trusted parties. Thus, $\mathcal{T}_X = \textsf{true}$ and $\mathcal{T}_Y = \textsf{true}$. Z is a trusted party. It is trusted:

- T_{Z_1}: To send X's masked secret to Y only if Y's secret is correctly shared by X and Z;

- T_{Z_2}: To send Y's masked secret to X only if X's secret is correctly shared by Y and Z; and
- T_{Z_3}: To forward both X's masked secret to Y and Y's masked secret to X, or neither.

These trust assumptions can be formalized as follows.

$$
\begin{aligned}
T_{Z_1}\text{:}\ & \Box\ (\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \land \text{send}(Y,\ x_3) \in \mathtt{H} \rightarrow \\
& \quad (\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid \text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \land y_1 = x_2 = \text{aux\_hash}(y_3, x_4)))
\end{aligned}
$$

$$
\begin{aligned}
T_{Z_2}\text{:}\ & \Box\ (\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid \text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \land \text{send}(X,\ x_3) \in \mathtt{H} \rightarrow \\
& \quad (\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \land y_1 = x_2 = \text{aux\_hash}(y_3, x_4)))
\end{aligned}
$$

$$
\begin{aligned}
T_{Z_3}\text{:}\ & \Box\ (\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid (\text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \land \text{send}(Y,\ x_3) \in \mathtt{H}) \rightarrow \\
& \quad \Diamond\ ((\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid (\text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \land \text{send}(X,\ y_3) \in \mathtt{H})) \lor \\
& \qquad (\text{failRemote} \in \mathtt{H} \land \nexists y : t \mid \text{send}(X, y) \in \mathtt{H}))) \land \\
& \Box\ (\exists\ y_1\ y_2\ y_3\ y_4 : t \times t \times t \times t \mid (\text{receive}(Y,\ y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \land \text{send}(X,\ y_3) \in \mathtt{H}) \rightarrow \\
& \quad \Diamond\ ((\exists\ x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid (\text{receive}(X,\ x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \land \text{send}(Y,\ x_3) \in \mathtt{H})) \lor \\
& \qquad (\text{failRemote} \in \mathtt{H} \land \nexists x : t \mid \text{send}(Y, x) \in \mathtt{H})))
\end{aligned}
$$

T_{Z_1} says that, at any state of an execution, if Z sends X's masked secret – x_3 – to Y, then Y's secret is correctly shared by X and Z – expressed in our formalization by the equality $y_1 = x_2 = \text{aux\_hash}(y_3, x_4)$. T_{Z_2} is symmetric to T_{Z_1}. T_{Z_3} says that if X's masked secret – x_3 – is sent to Y, then eventually Y's masked secret – y_3 – would have been sent to X, unless a communication failure prevents Z from sending the second message; and symmetrically. Note that y_3 could have been sent before or after x_3 has been sent.

3.3.2 Specification of Protection Properties

In the context of FR's protocol, X's protection property says that

> "If Y gets X's secret, then X gets Y's secret",

while Y's says that

> "If X gets Y's secret, then Y gets X's secret".

These properties can be respectively formalized as P_X and P_Y below:

$$P_X : \Box\ (\phi_1 \rightarrow \Diamond\ \phi_2)$$

$$P_Y : \Box\ (\phi_2 \rightarrow \Diamond\ \phi_1),$$

where

$$\phi_1 = \exists x : t, z : t \mid \text{receive}(X, x) \in \mathtt{H}^{\mathtt{Y}} \land \text{receive}(Z, z) \in \mathtt{H}^{\mathtt{Y}} \land \text{unmask}(z, x) = \kappa_x,$$

$$\phi_2 = \exists y : t, z : t \mid \text{receive}(Y, y) \in \mathtt{H}^{\mathtt{X}} \land \text{receive}(Z, z) \in \mathtt{H}^{\mathtt{X}} \land \text{unmask}(z, y) = \kappa_y.$$

3.4 Analysis of the Protocol

In this section, we analyze FR's protocol under all three deviation modes defined in Section 2.4.3. Subsections 3.4.2, 3.4.3, and 3.4.4 have respectively analyses of the protocol under compliance, abortion and deception modes. Subsection 3.4.1 has some preliminary theorems and lemmas.

In this section, we present only the main results with their high-level analysis. For completeness, detailed proofs appear in Section 3.5.

In what follows, we use she, he, and it to refer to X, Y, and Z respectively.

3.4.1 Preliminaries

In this subsection, we present two results needed in the rest of the analysis. Lemma 3.2 says that each protocol step gets instantiated at most once in executions we consider. Its proof is straightforward, and depends on the fact that the enabling condition of all the protocol rules check for prior occurrences of the types of events they prescribe.

Lemma 3.2 *Let Π be FR's protocol, σ be an execution in $E(\Pi)^C \cup E(\Pi)^A \cup E(\Pi)^D$, e_i be an event in σ, and E be an event-template prescribing a protocol step in Π. If e_i is an instance of E, then there does not exist e_j in σ, $i \neq j$, such that e_j is also an instance of E.*

The following three lemmas are needed in Theorem 3.6.

Lemma 3.3 *Let Π be FR's protocol and $\mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$ be the trust assumptions we make of Z (as given in Section 3.3.1). Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* T_{Z_1}.$$

Lemma 3.4 *Let Π and $\mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$ be as specified in Lemma. 3.3. Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* T_{Z_2}.$$

Lemma 3.5 *Let Π and $\mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$ be as specified in Lemma. 3.3. Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* T_{Z_3}.$$

Theorem 3.6 says that all maximal compliant executions of FR's protocol are trustworthy.

Theorem 3.6 *Let Π be Franklin and Reiter' protocol and $\mathcal{T}$ be its trust assumptions. Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* \mathcal{T}.$$

Proof: Given that Z is the only trusted party, $\mathcal{T} = \mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$, and this theorem follows from Lemmas 3.3, 3.4, and 3.5.

□

3.4.2 Protection Under Compliance Mode

In this subsection, we analyze the protocol under the assumption that all principals behave as prescribed by the protocol, deviating from it only if communication channels fail.

To verify whether the protocol is all-protective, we need to verify whether it is both X-protective and Y-protective. We start with X-protection.

Proposition 3.7 *Let* Π *be FR's protocol,* σ *be an execution in* $E(\Pi)^C$*, and* P_X *be* X*'s protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* P_X.$$

Analysis:

1. Let s_i be a state such that Y has received a message m_1 from X and a message m_2 from Z, and m_1 and m_2 are such that unmask$(m_2, m_1) = \kappa_x$.

   Then, at s_i, X must have received a message m'_1 from Y; and eventually, either Z will have sent a message m'_2 to X, or Z will have failed to do so because of a communication channel failure.

2. We have two scenarios to consider:

   (a) If the communication channel between X and Z does not fail, then, at the end of the execution, Z will have sent m'_2 to X and X will have received it. Since m'_2 was received from Y and Y behaves compliantly, unmask$(m'_2, m'_1) = \kappa_y$, and X will have been able to reconstitute κ_y.

   (b) If the communication channel between X and Z fails, then Z will not have been able to send m'_2 to X, or m'_2 will have been sent but not have been received by X. In either case, X does not receive the second share of κ_y.

□

The analysis of Proposition 3.7 shows that protection of X's interest relies critically on 1) X's receiving a message from Z, and 2) the message X receives from Y and the message she receives from Z form a correct splitting of Y's secret κ_y. If the communication channel between X and Z does not fail, then X receives the message Z forwards – which will allow her to reconstitute κ_y, and her interest is protected. Otherwise, either Z is prevented from sending the message or X does not receive it. In either case, her interest is violated.

From Prop. 3.7 we can derive the following

Corollary 3.8 *FR's protocol is* X*-protective under compliance mode if the communication channel between* X *and* Z *does not fail.*

Proposition 3.9 addresses Y-protection.

Proposition 3.9 *Let* Π *be FR's protocol,* σ *be an execution in* $E(\Pi)^C$*, and* P_Y *be* Y*'s protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* P_Y.$$

Analysis: analogous to that of Proposition 3.7, given that Y's local protocol and protection property are analogous to X's.

□

Corollary 3.10 follows from Prop. 3.9.

Corollary 3.10 *FR's protocol is Y-protective under compliance mode if the communication channel between Y and Z does not fail.*

Theorem 3.11 summarizes the results in this subsection:

Theorem 3.11 *Under the assumption that all principals behave compliantly, Franklin and Reiter's protocol is all-protective if communication channels between X and Z, and Y and Z do not fail.*

Proof: From Corollaries 3.8, and 3.10.

□

Discussion

In FR's protocol, secrets are not released as a whole. Take X's secret K_x for example. It is "split" into two shares by X. One of the shares is given to Y and the other given to Z. Y would be able to reconstitute κ_x later, if Z forwards its share to Y.

If Y is able to reconstitute K_x from messages m_1 and m_2 he receives respectively from X and Z, then Z must have received messages from both X and Y, and must have been able to verify the correctness of both secret splittings. Since a compliant Z sends a message to Y if and only if it sends a message to X, it will try to forward its share of Y's secret K_y to X, which can then be used by X to reconstitute K_y.

The problem arises if the communication channel between X and Z is unreliable: Z may be prevented from sending the message to X, or the message it sends may never be received by X. In either case, X will not be able to reconstitute K_y.

This is a weakness of the protocol, particularly because Franklin and Reiter advocate Z to be a "stranger" – a random party from the network. Strangers on the net can come and go, and nothing in the protocol guarantees that Z can be contacted later, if X does not receive a message from it in a timely manner. X can try to contact Y to get the missing share. But Y may not be reachable himself. Even if he is, Y may not bother to cooperate and re-send the share X wants. After all, this protocol is used because X and Y do not trust each other.

3.4.3 Protection Under Abortion Mode

In this subsection, we analyze the protocol under abortion mode. Like with compliance mode, we verify whether the protocol is both X-protective and Y-protective. We start with X-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, to verify whether Π protects X's interest under abortion mode, we need to examine all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_X$. Here we focus on abortive executions only, since we have analyzed compliant executions in Section 3.4.2.

Proposition 3.12 *Let Π be FR's protocol, σ be an execution in $E(\Pi)^A_X$, $\mathcal{T}$ be the trust assumptions Π makes, and P_X be X's protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_X.$$

Analysis: This analysis is identical to the one for Prop. 3.7, except for what is in *italics* below.

1. Let s_i be a state such that Y has received a message m_1 from X and a message m_2 from Z, and m_1 and m_2 are such that unmask$(m_2, m_1) = \kappa_x$.

   Then, at s_i, X must have received a message m'_1 from Y; and eventually, either Z will have sent a message m'_2 to X, or Z will have failed to do so because of a communication channel failure.

   Note that since σ is a trustworthy execution, Z could not have failed to send m'_2 to X because of local problems – that would have violated the trust assumption T_{Z_3}.

2. We have two scenarios to consider:

   (a) If the communication channel between X and Z does not fail, then, at the end of the execution, Z will have sent m'_2 to X and X will have received it. Since neither Z nor Y deceives, unmask$(m'_2, m'_1) = \kappa_y$, and X will be able to reconstitute κ_y.

   (b) If the communication channel between X and Z fails, then Z will not be able to send m'_2 to X, or m'_2 was sent but not received by X. In either case, X does not receive the second share of κ_y.

□

Jointly, Props 3.7 and 3.12 address X-protection under abortion mode. They show that this protocol offers the same level of protection to X's interests in both compliance and abortion modes. Possible failures in the channel linking X and Z are still what can compromise X's interest.

From Prop. 3.7 and 3.12 we can derive the following

Corollary 3.13 *FR's protocol is X-protective under abortion mode if the communication channel between X and Z does not fail.*

Proposition 3.14 addresses protection of Y's interests in abortive executions.

Proposition 3.14 *Let Π be FR's protocol, σ be an execution in $E(\Pi)^A_Y$, $\mathcal{T}$ be the trust assumptions Π makes, and P_Y be Y's protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_Y.$$

Analysis: analogous to that of Proposition 3.12, given that Y's local protocol and protection property are analogous to X's, and T_{Z_3} is equally applicable here.

□

Jointly, Props 3.9 and 3.14 address Y-protection under abortion mode (Corollary 3.15). Here too, possible failures in the channel linking Y and Z are still what can compromise Y's interest.

$$\begin{array}{lcl} receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) & & receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)) \\ receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)) & or & receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) \\ failLocal & & failLocal \end{array}$$

Figure 3.3: Examples of trustworthy abortive executions of Z.

$$\begin{array}{lcl} receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) & & receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)) \\ receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)) & or & receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) \\ send(Y, K_x x^{-1}) & & send(X, K_y y^{-1}) \\ failLocal & & failLocal \end{array}$$

Figure 3.4: Examples of untrustworthy abortive executions of Z.

Corollary 3.15 *FR's protocol is Y-protective under abortion mode if the communication channel between Y and Z does not fail.*

Theorem 3.16 summarizes the results in this subsection:

Theorem 3.16 *Under abortion mode, Franklin and Reiter's protocol is all-protective if communication channels between X and Z, and Y and Z do not fail.*

Proof: From Corollaries 3.13, and 3.15.

□

Discussion

According to our analysis, the ability of the protocol to guarantee all-protection is not affected under abortion mode; the problem still lies on possible channel failures between Z and the other two principals. This seems counter-intuitive, since depending on when Z terminates its execution, all-protection can actually be compromised. For instance, if Z terminates its execution right after receiving messages from both X and Y, that is, Z's local execution is as shown in Fig. 3.3, neither X-protection nor Y-protection is compromised, since neither of them received the second share of the secret they desire. However, if Z terminates its execution after sending its share of X's secret $(K_x x^{-1})$ to Y, but before sending its share of Y's secret $(K_y y^{-1})$ to X, that is, if Z's local execution is as shown in Fig. 3.4, then X-protection could[2] be violated, since X would never receive the other share of Y's secret from Z.

This argument overlooked one key point: the executions in Fig. 3.4 do not belong to the set of executions we consider in this analysis, because they violate the trust assumption T_{Z_3}.

This discussion shows how trust assumptions limit the set of executions we consider when analyzing a protocol and how critical they are in our conclusion of whether or not a protocol is all-protective.

[2]We use *could*, instead of *would*, because the message Z sent to Y might never get to Y as well. This could happen due to Y's terminating prematurely for example.

Y's deviations do not affect X-protection; the readers should be able to convince themselves straightforwardly.

3.4.4 Protection Under Deception Mode

In this subsection, we analyze the protocol under deception mode.

We first analyze the protocol with respect to X-protection, then with respect to Y-protection.

According to Def. 2.25 in Section 2.4, to analyze the protocol with respect to X-protection, we need to analyze all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^D_X$. Here we focus on deceptive executions only, since we have analyzed compliant executions in Section 3.4.2.

Proposition 3.17 *Let Π be FR's protocol, σ be an execution in $E(\Pi)^D_X$, $\mathcal{T}$ be the trust assumptions Π makes, and P_X be X's protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_X.$$

Analysis: This analysis is very similar to that for Prop. 3.7, except for some minor details.

1. Let s_i be a state such that Y has received a message m_1 from X and a message m_2 from Z, and m_1 and m_2 are such that $\text{unmask}(m_2, m_1) = \kappa_x$.

   Then, at the same state s_i, X must have received a message m_1' from Y; and eventually, either Z will send a message m_2' to X, or Z will fail to do so because of a communication channel failure.

2. We have two scenarios to consider:

   (a) If the communication channel between X and Z does not fail, then, at the end of the execution, Z will have sent m_2' to X and X will have received it. Since Z behaves as trusted, m_2' is the message X expects, that is $\text{unmask}(m_2', m_1') = \kappa_y$, and X will have been able to reconstitute κ_y.

   (b) If the communication channel between X and Z fails, then Z will not have been able to send m_2' to X, or m_2' will have been sent but not have been received by X. In either case, X does not receive the second share of κ_y.

□

Analogous to the analysis of Proposition 3.7 (regarding compliance mode), the analysis of Proposition 3.17 shows that protection of X's interests relies critically on 1) X receiving a message from Z, and 2) the message X receives from Y and the message it receives from Z forming a correct splitting of Y's secret. Analogous to the compliance mode scenario, if the communication channel between X and Z does not fail, then X receives the message Z sends. Under deception mode, however, this message may not be the one that would allow X to reconstitute Y's secret, unless Z behaves as trusted. If Z behaves as trusted, then Z will make sure that the message it forwards to X is the expected one. If the communication channel between X and Z fails, and either Z is prevented from sending the message or X does not receive the message sent by Z, then X's interest is inevitably compromised.

Jointly Props 3.7 and 3.17 address X-protection under deception mode (Corollary 3.18). They show that, like in the other two modes, this protocol is X-protective only if the communication channel between X and Z does not fail.

Corollary 3.18 *FR's protocol is X-protective under deception mode if the communication channel between X and Z does not fail.*

Proposition 3.19 addresses protection of Y's interest in deceptive executions.

Proposition 3.19 *Let Π be FR's protocol, σ be an execution in $E(\Pi)_Y^D$, $\mathcal{T}$ be the trust assumptions Π makes, and P_Y be Y's protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad implies \quad \sigma \models^* P_Y.$$

Analysis: analogous to that of Proposition 3.17, since Y's local protocol and protection property are analogous to X's, and assumption T_{Z_3} is equally applicable here.

□

Jointly, Props 3.9 and 3.19 address Y-protection under deception mode (Corollary 3.20). Possible failures in the channel linking Y and Z are still what prevents the protocol from being Y-protective.

Corollary 3.20 *FR's protocol is Y-protective under deception mode if the communication channel between Y and Z does not fail.*

Theorem 3.21 summarizes the results in this subsection.

Theorem 3.21 *Under deception mode, Franklin and Reiter's protocol is all-protective if communication channels between X and Z, and Y and Z do not fail.*

Proof: From Corollaries 3.18, and 3.20.

□

Discussion

In this subsection, we focus on what happens under deception mode. Like in sections 3.4.2 and 3.4.3, we focus on X-protection.

According to our proofs, the fact that Y and Z may send out arbitrary messages does not affect the ability of the protocol to guarantee X-protection (all we discussed in section 3.4.2 is still applicable here); the problem still lies on possible channel failures between X and Z. It is straightforward to see that Y's sending arbitrary messages alone cannot compromise X-protection, since in a compliant execution of Z, Z does not forward $K_x x^{-1}$ to Y, unless Z has made sure that jointly X and Z have a correct splitting of K_y.

But what if Z also misbehaves? For example, assume that Z has just received messages from both X and Y, i.e., Z has just experienced the following events:

$$\begin{array}{l} receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) \\ receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)). \end{array}$$

Then R_{Z_4} is applicable, which means that Z can send $K_x x^{-1}$ to Y, $K_y y^{-1}$ to X, or timeout next. In a scenario where Z can deceive, Z can send something other than $K_y y^{-1}$ to X. Fig. 3.5 shows

$$
\begin{array}{l}
receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) \\
receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)) \\
send(X, K_x x^{-1}) \\
send(Y, K_x x^{-1}) \\
exit
\end{array}
$$

Figure 3.5: An untrustworthy deceptive local execution of Z.

$$
\begin{array}{l}
receive(X, f(K_x)\ f(K_y)\ K_x x^{-1}\ f(y)) \\
receive(Y, f(K_y)\ f(K_x)\ K_y y^{-1}\ f(x)) \\
send(X, K_x x^{-1}) \\
send(Y, K_y y^{-1}) \\
exit
\end{array}
$$

Figure 3.6: A trustworthy deceptive local execution of Z.

an example. Under this scenario, X-protection is clearly violated, since what X receives from Z cannot be used to re-constitute K_y.

Here is where trust assumptions come to the rescue: the execution in Fig. 3.5 violates T_{Z_3}, and should not be considered when analyzing X-protection.

Note that Z can deceive without violating the trust assumptions we make of it. The execution in Fig. 3.6 is an example.

These examples illustrate that X-protection is not compromised whenever Z deceives; but it *could* be if Z violates the trust assumptions. We use *could* instead of *would* because even when Z violates the trust assumptions, X-protection might not be violated. In Fig. 3.5, for example, it is violated if the communication channel between Y and Z is reliable and Y receives $K_x x^{-1}$. It is not violated, however, if the communication channel between Y and Z fails, and Y does not receive $K_x x^{-1}$.

3.5 Formal Analysis of the Protocol

This section contains detailed proofs of the results presented in Section 3.4. These proofs use standard proof methods and are presented here for completeness.

3.5.1 Preliminaries

Lemma 3.4 *Let Π be FR's protocol and $\mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$ be the trust assumptions we make of Z (as given in Section 3.3.1). Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* T_{Z_1}.$$

Proof:

1. Let s_i be a state in σ and $a_1\ a_2\ a_3\ a_4$: $t \times t \times t \times t$ be a message such that

$$s_i \models \text{receive}(X, a_1\ a_2\ a_3\ a_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{send}(Y, a_3) \in \mathtt{H}^{\mathtt{Z}}.$$

   Then there exists a state s_k, $k \leq i$, such that

$$s_k \models \text{receive}(X, a_1\ a_2\ a_3\ a_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(Y, a_3).$$

   We know that such a state exists because, according to Z's local protocol, Z receives from both X and Y first, before it sends messages to X or Y.

   At s_k, the last rule applied by Z must have been R_{Z_4} or R_{Z_6}.

2. If s_k^Z resulted from an application of R_{Z_4}, then we know that $\exists j < k$, and $b_1\ b_2\ b_3\ b_4$: $t \times t \times t \times t$ such that

$$s_j \models \text{receive}(Y, b_1\ b_2\ b_3\ b_4) \in \mathtt{H}^{\mathtt{Z}}\ \wedge\ b_1 = a_2 = \text{aux\_hash}(b_3, a_4).$$

3. If s_k^Z resulted from an application of R_{Z_6}, then $\exists j < k, b_3' : t$ such that

$$s_j \models \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(X, b_3'),$$

   which could have resulted only from an application of R_{Z_4}. To be applicable, R_{Z_4} requires that $b_1 = a_2 = \text{aux\_hash}(b_3, a_4)$.

□

Lemma 3.5 *Let* Π *and* $\mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$ *be as specified in Lemma. 3.4. Then*

$$\forall \sigma \in E(\Pi)^C,\ \ \sigma \models^* T_{Z_2}.$$

Proof: Analogous to the proof of Lemma 3.4.

□

Lemma 3.6 *Let* Π *and* $\mathcal{T}_Z = \{T_{Z_1}, T_{Z_2}, T_{Z_3}\}$ *be as specified in Lemma. 3.4. Then*

$$\forall \sigma \in E(\Pi)^C,\ \ \sigma \models^* T_{Z_3}.$$

Proof:

1. Let s_k be a state in σ and $a_1\ a_2\ a_3\ a_4$: $t \times t \times t \times t$ be a message such that

$$s_i \models \text{receive}(X, a_1\ a_2\ a_3\ a_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{send}(Y, a_3) \in \mathtt{H}^{\mathtt{Z}}.$$

   Then there exists a state s_i, $i \leq k$, such that

$$s_i \models \text{receive}(X, a_1\ a_2\ a_3\ a_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(Y, a_3).$$

   From Lemma 3.4, we know that $\exists\ b_1\ b_2\ b_3\ b_4$: $t \times t \times t \times t$ such that

$$s_i \models \text{receive}(Y, b_1\ b_2\ b_3\ b_4) \in \mathtt{H}^{\mathtt{Z}}.$$

   Thus, we only need to prove that

$$s_i \models^* \Diamond(\text{send}(X, b_3) \in \mathtt{H}^{\mathtt{Z}}\ \vee\ (\text{failRemote} \in \mathtt{H}^{\mathtt{Z}} \wedge\ \nexists y : t \mid \text{send}(X, y) \in \mathtt{H}^{\mathtt{Z}})).$$

2. If

$$s_i \models \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(Y, a_3),$$

   then s_i^Z must have resulted from an application of rules R_{Z_4} or R_{Z_6}.

3. If s_i^Z resulted from applying R_{Z_4}, then R_{Z_7} is the only rule applicable from s_i^Z, and in Z's local execution, 'send(Y, a_3)' can be followed only by 'send(Y, b_3)' or 'failRemote'. This means that we have

$$s_i \models^* \Diamond(\text{send}(X, b_3) \in \mathtt{H}^{\mathtt{Z}}\ \vee\ (\text{failRemote} \in \mathtt{H}^{\mathtt{Z}} \wedge\ \nexists y : t \mid \text{send}(X, y) \in \mathtt{H}^{\mathtt{Z}})).$$

   In this case, Z sends messages first to Y, then to X; or terminates its execution after sending the message to Y.

4. If s_i^Z resulted from applying R_{Z_6}, then

$$s_i \models \exists y_3' : t \mid \text{send}(X, y_3') \in \mathtt{H}^{\mathtt{Z}}.$$

   This means that $\exists j < i$, and a message b_3', such that

$$s_j \models \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(X, b_3').$$

   'send(X, b_3')' could have resulted only from applying R_{Z_4}, which prescribes Z to send b_3 to X. Thus, $b_3' = b_3$.

   In this case, Z sends messages first to X, then to Y.

5. From 3 and 4, we have:

$$\begin{aligned} s_i \quad \models^* \quad & \Diamond((\exists y_1 : t, y_2 : t, y_3 : t, y_4 : t \mid (\text{receive}(Y, y_1\ y_2\ y_3\ y_4) \in \mathtt{H}^{\mathtt{Z}} \wedge (\text{send}(X, y_3) \in \mathtt{H}^{\mathtt{Z}}))) \vee \\ & (\text{failRemote} \in \mathtt{H}^{\mathtt{Z}} \wedge\ \nexists y : t \mid \text{send}(X, y) \in \mathtt{H}^{\mathtt{Z}})). \end{aligned}$$

6. The other conjunct of T_{Z_3} can be proven analogously.

$\square$

3.5.2 Protection Under Compliance Mode

Proposition 3.7 *Let Π be FR's protocol, σ be an execution in $E(\Pi)^C$, and P_X be X's protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* P_X.$$

Analysis:

1. Let s_i be a state such that

$$s_i \models \exists x : t, z_x : t \mid \text{receive}(X, x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, z_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(z_x, x) = \kappa_x.$$

   Then, from Lemma 3.22, we deduce that $\exists j > i \mid$

$$s_j \models (\exists x_3 : t, x_4 : t \mid \text{send}(X, x_3) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_4) \in \mathtt{H}^{\mathtt{X}}) \vee \text{failRemote} \in \mathtt{H}^{\mathtt{Z}}.$$

2. If the first conjunct is true, then there exist messages $a_3 : t$ and $a_4 : t$ such that

$$s_j \models \text{send}(X, a_3) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}}.$$

   According to Lemma 3.23, $\text{unmask}(a_3, a_4) = \kappa_y$.

   Two scenarios can be derived from here.

   (a) If the communication channel between X and Z does not fail, then X will receive the message sent by Z, i.e., $\exists k, k > j \mid$

$$s_k \models \text{receive}(Z, a_3) \in \mathtt{H}^{\mathtt{X}} \wedge \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}}.$$

   And from 2) we know that $\text{unmask}(a_3, a_4) = \kappa_y$.

   (b) If the communication channel between X and Z fails before X receives a_3 from Z, then $\forall l, l > j$,

$$s_l \models \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}} \wedge \nexists x_2 : t \mid \text{receive}(Z, x_2) \in \mathtt{H}^{\mathtt{X}}.$$

3. If the first conjunct is false, then

$$s_j \models \text{failRemote} \in \mathtt{H}^{\mathtt{Z}} \wedge \nexists y : t \mid \text{send}(X, y) \in \mathtt{H}^{\mathtt{Z}}.$$

   This is the scenario where Z is prevented from sending messages to X because of a failure in the communication channel. Like 2b), we can conclude that

$$\forall l, l > j, s_l \models \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}} \wedge \nexists x_2 : t \mid \text{receive}(Z, x_2) \in \mathtt{H}^{\mathtt{X}}.$$

4. From 2) and 3), we conclude that this proposition holds if the communication channel between X and Z does not fail. Otherwise, it does not.

$\square$

The next two lemmas are used in the analysis of Proposition 3.7. Lemma 3.22 says that if Y has received a message from Z, then Z will have sent a message to X, and X will have received a message from Y; or Z will have failed to send a message to X because of a communication channel failure.

Lemma 3.22 *Let Π be FR's protocol and σ be an execution in $E(\Pi)^C$.*

$$\begin{aligned}\sigma \quad \models^* \quad & \Box\ (\exists x_1 : t \mid \text{receive}(Z, x_1) \in \mathtt{H}^{\mathtt{Y}} \rightarrow \\ & \Diamond\ ((\exists x_2 : t, x_3 : t \mid \text{send}(X, x_2) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}})\ \vee\ \text{failRemote} \in \mathtt{H}^{\mathtt{Z}})).\end{aligned}$$

Proof:

1. Let s_i be a state in σ and $a_1 : t$ be a message such that

$$s_i \models \text{receive}(Z, a_1) \in \mathtt{H}^{\mathtt{Y}}.$$

By communication assumption (Definition 2.10, condition 6),

$$s_i \models \text{send}(Y, a_1) \in \mathtt{H}^{\mathtt{Z}}.$$

2. $\text{send}^Z(Y, a_1)$ could have been prescribed by rules R_{Z_4} or R_{Z_6}.

3. If $\text{send}^Z(Y, a_1)$ resulted from an application of R_{Z_4}, then $\exists j < i \mid$

$$s_j \models \text{send}(Y, a_1) \notin \mathtt{H}^{\mathtt{Z}} \wedge\ \nexists x_4 : t \mid \text{send}(X, x_4) \in \mathtt{H}^{\mathtt{Z}},$$

and

$$s_{j+1} \models \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(Y, a_1) \wedge\ \nexists x_4 : t \mid \text{send}(X, x_4) \in \mathtt{H}^{\mathtt{Z}}.$$

4. R_{Z_7} is the only rule applicable from s^Z_{j+1}, which implies that

$$s_{j+1} \models^* \Diamond\ (\exists x_4 : t \mid \text{send}(X, x_4) \in \mathtt{H}^{\mathtt{Z}} \vee \text{failRemote} \in \mathtt{H}^{\mathtt{Z}}),$$

and consequently

$$s_i \models^* \Diamond\ (\exists x_4 : t \mid \text{send}(X, x_4) \in \mathtt{H}^{\mathtt{Z}} \vee \text{failRemote} \in \mathtt{H}^{\mathtt{Z}}).$$

5. If $\text{send}^Z(Y, a_1)$ resulted from an application of R_{Z_6}, then Z sent a message to X before it sent messages to Y. That is

$$s_i \models^* \exists x_4 : t \mid \text{send}(X, x_4) \in \mathtt{H}^{\mathtt{Z}},$$

and consequently

$$s_i \models^* \Diamond\ (\exists x_4 : t \mid \text{send}(X, x_4) \in \mathtt{H}^{\mathtt{Z}}).$$

6. Next, let us prove that

$$s_i \models^* \Diamond\ (\exists x_3 : t \mid \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}}).$$

By straightforward backchaining,

$$s_i \models \text{send}(Y, a_1) \in \mathtt{H}^{\mathtt{Z}}$$

implies that $\exists k < i \mid$

$$s_k \models \exists x : t \times t \times t \times t \mid \text{receive}(X, x) \in \mathtt{H}^{\mathtt{Z}},$$

which, by communication assumption, implies that

$$s_k \models \exists x : t \times t \times t \times t \mid \text{send}(Z, x) \in \mathtt{H}^{\mathtt{X}}.$$

7. Since 'send-to-Z' events are only prescribed by R_{X_5}, and R_{X_5}'s enabling condition requires a 'receive-from-Y' event, we can conclude that

$$s_k \models \exists x : t \mid \text{receive}(Y, x) \in \mathtt{H}^{\mathtt{X}},$$

and consequently

$$s_i \models^* \Diamond\ (\exists x_3 : t \mid \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}}).$$

8. Given 4), 5) and 7), the lemma is proved.

□

Lemma 3.23 says that the message sent to X by Z and the message X receives from Y form a correct splitting of Y's secret.

Lemma 3.23 *Let* Π *be FR's protocol and* σ *be an execution in* $E(\Pi)^C$. *Then* $\forall\ x_2 : t,\ x_3 : t,$

$$\sigma \quad \models^* \quad \Box\ (\text{send}(X, x_2) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}} \rightarrow \text{unmask}(x_2, x_3) = \kappa_y).$$

Proof: Let s_n be a state such that there exists messages a_2 and a_3 such that

$$s_n \models \text{send}(X, a_2) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, a_3) \in \mathtt{H}^{\mathtt{X}}.$$

Then, we can prove the lemma as follows:

1. Given

$$s_n \models \text{send}(X, a_2) \in \mathtt{H}^{\mathtt{Z}},$$

we can conclude, by straightforward backchaining, that

$$s_n \models \exists x_1\ x_2\ x_3\ x_4 : t \times t \times t \times t \mid \text{receive}(Y, x_1\ x_2\ x_3\ x_4) \in \mathtt{H}^{\mathtt{Z}},$$

and that the third component of this concatenated message is a_2, i.e.,

$$s_n \models \exists x_1 : t, x_2 : t, x_4 : t \mid \text{receive}(Y, [x_1, x_2, a_2, x_4]) \in \mathtt{H}^{\mathtt{Z}}.$$

2. By communication assumption,

$$s_n \models \exists x_1 : t, x_2 : t, x_4 : t \mid \text{receive}(Y, [x_1, x_2, a_2, x_4]) \in \texttt{H}^{\texttt{Z}}$$

implies that

$$s_n \models \exists x_1 : t, x_2 : t, x_4 : t \mid \text{send}(Z, [x_1, x_2, a_2, x_4]) \in \texttt{H}^{\texttt{Y}}.$$

And since R_{Y_5} is the only rule that prescribes a "send-to-Z" event, we can conclude that

$$a_2 = \text{mask}(\kappa_y, a_1),$$

for a message $a_1 : t$, and

$$s_n \models \text{send}(X, a_1) \in \texttt{H}^{\texttt{Y}}.$$

3. Also, by communication assumption,

$$s_n \models \text{receive}(Y, a_3) \in \texttt{H}^{\texttt{X}}$$

implies that

$$s_n \models \text{send}(X, a_3) \in \texttt{H}^{\texttt{Y}}.$$

Given that there can be only one send-to-Z event in $\texttt{H}^{\texttt{Y}}$, we conclude that $a_1 = a_3$.

4. We are now ready to show that $\text{unmask}(a_2, a_3) = \kappa_y$. From 2), we know that $a_2 = \text{mask}(\kappa_y, a_3)$. Thus, $\text{unmask}(a_2, a_3) = \text{unmask}(\text{mask}(\kappa_y, a_3), a_3) = \kappa_y$.

□

3.5.3 Protection Under Abortion Mode

Proposition 3.12 *Let* Π *be FR's protocol,* σ *be an execution in* $E(\Pi)_X^A$, $\mathcal{T}$ *be the trust assumptions* Π *makes, and* P_X *be* X*'s protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_X.$$

Analysis: The structure of this analysis is identical to the structure of the analysis of Proposition 3.7; the lemmas it uses are in one-to-one correspondence to those used by Proposition 3.7. We repeat the analysis here for self-containment.

1. Let s_i be a state such that

$$s_i \models \exists x : t, z_x : t \mid \text{receive}(X, x) \in \texttt{H}^{\texttt{Y}} \land \text{receive}(Z, z_x) \in \texttt{H}^{\texttt{Y}} \land \text{unmask}(z_x, x) = \kappa_x.$$

Then from Lemma 3.24, we deduce that $\exists j > i \mid$

$$\begin{aligned} s_j \quad \models \quad & (\exists x_3 : t, x_4 : t \mid \text{send}(X, x_3) \in \texttt{H}^{\texttt{Z}} \land \text{receive}(Y, x_4) \in \texttt{H}^{\texttt{X}}) \lor \\ & \text{failRemote} \in \texttt{H}^{\texttt{Z}}. \end{aligned}$$

2. If the first conjunct is true, then there exist messages $a_3 : t$ and $a_4 : t$ such that

$$s_j \models \text{send}(X, a_3) \in \texttt{H}^{\texttt{Z}} \land \text{receive}(Y, a_4) \in \texttt{H}^{\texttt{X}}.$$

According to Lemma 3.25, $\text{unmask}(a_3, a_4) = \kappa_y$.

Two scenarios can be derived from here.

(a) If the communication channel between X and Z does not fail, then X will receive the message sent by Z, i.e., $\exists k, k > j \mid$

$$s_k \models \text{receive}(Z, a_3) \in \mathtt{H}^{\mathtt{X}} \wedge \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}}.$$

And from 2) we know that $\text{unmask}(a_3, a_4) = \kappa_y$.

(b) If the communication channel between X and Z fails before X receives a_3 from Z, then $\forall l, l > j$,

$$s_l \models \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}} \wedge \not\exists x_2 : t \mid \text{receive}(Z, x_2) \in \mathtt{H}^{\mathtt{X}}.$$

3. If the first conjunct is false, then

$$s_j \models \text{failRemote} \in \mathtt{H}^{\mathtt{Z}} \wedge \not\exists y : t \mid \text{send}(X, y) \in \mathtt{H}^{\mathtt{Z}}.$$

This is the scenario where Z is prevented from sending messages to X because of a failure in the communication channel. Like 2b), we can conclude that

$$\forall l, l > j, s_l \models \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}} \wedge \not\exists x_2 : t \mid \text{receive}(Z, x_2) \in \mathtt{H}^{\mathtt{X}}.$$

4. From 2) and 3), we conclude that this proposition holds if the communication channel between X and Z does not fail. Otherwise, it does not.

□

Note that we did not refer to trust assumptions in the analysis of Prop. 3.12. In fact, we only need them in the proof of Lemma 3.24. This fact is reflected in the statements of the lemmas used by Proposition 3.12 (Lemmas 3.24 and 3.25): Lemma 3.24 is the only one whose statement mentions trust assumptions.

Lemma 3.24 *Let Π be FR's protocol and σ be an execution in $E(\Pi)^A_X$. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \Box\, (\exists x_1 : t \mid \text{receive}(Z, x_1) \in \mathtt{H}^{\mathtt{Y}} \rightarrow \Diamond\, ((\exists x_2 : t, x_3 : t \mid \text{send}(X, x_2) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}}) \vee \text{failRemote} \in \mathtt{H}^{\mathtt{Z}}).$$

Proof:

1. This proof uses some of the reasoning steps used in the proof of Lemma 3.22. More specifically,

$$\sigma \models^* \Box\, (\exists x_1 : t \mid \text{receive}(Z, x_1) \in \mathtt{H}^{\mathtt{Y}} \rightarrow \exists x_3 : t \mid \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}})$$

can be proven using steps 1, 6, and 7 in that proof. Steps 1, 2, and 5 prove that a $\text{send}^Z(X, x_2)$ event could have preceded $\text{send}^Z(Y, a_1)$ in σ.

2. But if $\text{send}^Z(Y, a_1)$ was not preceded by a $\text{send}^Z(X, x_2)$ event, and σ is an arbitrary member of $E(\Pi)^A_X$, then an occurrence of $\text{send}^Z(Y, a_1)$ may be followed by an occurrence of 'failLocal'. That is, steps 3 and 4 (in the proof of Lemma 3.22) does not always hold of an execution $\sigma \in E(\Pi)^A_X$.

3. However, let s_j be the state in σ such that

$$s_j \models \text{send}(Y, a_1) \notin \mathtt{H}^{\mathtt{Z}} \quad \text{and} \quad s_{j+1} \models \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(Y, a_1).$$

   (a) By straightforward backchaining,

$$s_{j+1} \models \text{send}(Y, a_1) \in \mathtt{H}^{\mathtt{Z}}$$

   implies that $\exists k \leq j$ and $b_1\ b_2\ b_3\ b_4$: $t \times t \times t \times t$ such that

$$s_k \models \text{receive}(X, b_1\ b_2\ b_3\ b_4) \in \mathtt{H}^{\mathtt{Z}} \quad \wedge \quad a_1 = b_3.$$

   Thus,

$$s_{j+1} \models \text{receive}(X, b_1\ b_2\ b_3\ b_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{last}(\mathtt{H}^{\mathtt{Z}}) = \text{send}(Y, b_3).$$

   (b) Given that $\sigma \models^* T_{Z_3}$,

$$\begin{aligned}\sigma \models^* \Box(&\exists x_1 : t, x_2 : t, x_3 : t, x_4 : t \mid (\text{receive}(X, x_1\ x_2\ x_3\ x_4) \in \mathtt{H} \wedge \text{last}(\mathtt{H}) = \text{send}(Y, x_3) \rightarrow \\ &\Diamond((\exists y_1 : t, y_2 : t, y_3 : t, y_4 : t \mid \\ &\quad (\text{receive}(Y, y_1\ y_2\ y_3\ y_4) \in \mathtt{H} \wedge (\text{send}(X, y_3) \in \mathtt{H}))) \vee \\ &\quad (\text{failRemote} \in \mathtt{H} \wedge \nexists y : t \mid \text{send}(X, y) \in \mathtt{H})))).\end{aligned}$$

   This implies that $\exists i > j + 1$ such that

$$s_i \models \exists y_3 : t \mid \text{send}(X, y_3) \in \mathtt{H}^{\mathtt{Z}} \vee \text{failRemote} \in \mathtt{H}^{\mathtt{Z}}.$$

□

The proof above shows that the liveness condition specified in Lemma 3.24 does not hold in all executions $\sigma \in E(\Pi)^A_X$. It is violated in executions where local factors cause Z to terminate its execution after it sends its share of X's secret to Y, but before it sends its share of Y's secret to X. If we consider only trustworthy executions, however, the liveness condition is never violated because of T_{Z_3}.

Lemma 3.25 corresponds to Lemmas 3.23. Unlike Lemma 3.24, it holds in all executions $\sigma \in E(\Pi)^A_X$.

Lemma 3.25 *Let Π be FR's protocol and σ be an execution in $E(\Pi)^A_X$. Then $\forall\ x_1 : t,\ x_2 : t,\ x_3 : t$,*

$$\sigma \quad \models^* \quad \Box\ (\text{send}(X, x_2) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}} \rightarrow \text{unmask}(x_2, x_3) = \kappa_y).$$

Proof: Identical to that of Lemma 3.23.

□

3.5.4 Protection Under Deception Mode

Proposition 3.17 *Let* Π *be FR's protocol,* σ *be an execution in* $E(\Pi)^D_X$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* P_X *be X's protection property as specified in Section 3.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_X.$$

Analysis: The structure of this analysis is identical to the structure of the analysis of Proposition 3.7; the lemmas it uses are in one-to-one correspondence to those used by Proposition 3.7. We repeat the analysis here for self-containment.

1. Let s_i be a state such that

$$s_i \models \exists x : t, z_x : t \mid \text{receive}(X, x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, z_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(z_x, x) = \kappa_x.$$

   Then, from Lemma 3.26, we deduce that $\exists j > i \mid$

$$s_j \models (\exists x_3 : t, x_4 : t \mid \text{send}(X, x_3) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_4) \in \mathtt{H}^{\mathtt{X}}) \vee \text{failRemote} \in \mathtt{H}^{\mathtt{Z}}.$$

2. If the first conjunct is true, then there exist messages $a_3 : t$ and $a_4 : t$ such that

$$s_j \models \text{send}(X, a_3) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}}.$$

   According to Lemma 3.27, $\text{unmask}(a_3, a_4) = \kappa_y$.

   Two scenarios can be derived from here.

   (a) If the communication channel between X and Z does not fail, then X will receive the message sent by Z, i.e., $\exists k, k > j \mid$

$$s_k \models \text{receive}(Z, a_3) \in \mathtt{H}^{\mathtt{X}} \wedge \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}}.$$

   And from 2) we know that $\text{unmask}(a_3, a_4) = \kappa_y$.

   (b) If the communication channel between X and Z fails before X receives a_3 from Z, then $\forall l, l > j$,

$$s_l \models \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}} \wedge \not\exists x_2 : t \mid \text{receive}(Z, x_2) \in \mathtt{H}^{\mathtt{X}}.$$

3. If the first conjunct is false, then

$$s_j \models \text{failRemote} \in \mathtt{H}^{\mathtt{Z}} \wedge \not\exists y : t \mid \text{send}(X, y) \in \mathtt{H}^{\mathtt{Z}}.$$

   This is the scenario where Z is prevented from sending messages to X because of a failure in the communication channel. Like 2b), we can conclude that

$$\forall l, l > j, s_l \models \text{receive}(Y, a_4) \in \mathtt{H}^{\mathtt{X}} \wedge \not\exists x_2 : t \mid \text{receive}(Z, x_2) \in \mathtt{H}^{\mathtt{X}}.$$

4. From 2) and 3), we conclude that this proposition holds if the communication channel between X and Z does not fail. Otherwise, it does not.

$\square$

Lemma 3.26 corresponds to Lemmas 3.22 and 3.24. It holds in all executions $\sigma \in E(\Pi)^D_X$.

Lemma 3.26 *Let Π be FR's protocol and σ be an execution in $E(\Pi)^D_X$.*

$$\begin{array}{lll} \sigma & \models^* & \Box\, (\exists x_1 : t \mid \text{receive}(Z, x_1) \in \mathtt{H}^{\mathtt{Y}} \rightarrow \\ & & \quad \Diamond\, ((\exists x_2 : t, x_3 : t \mid \text{send}(X, x_2) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, x_3) \in \mathtt{H}^{\mathtt{X}}) \;\vee\; \text{failRemote} \in \mathtt{H}^{\mathtt{Z}})). \end{array}$$

Proof: Identical to that of Lemma 3.22.

□

Lemma 3.27 corresponds to Lemmas 3.23 and 3.25. It is, however, more complex. The safety condition that appears in Lemmas 3.23 and 3.25 is now specified to hold in trustworthy executions where Y has received the two shares of X's secret. This should have been expected, given that under the deception mode, there is no guarantee whatsoever of what the principals may send to each other. Unless Z forwards what it should to Y and behaves according to the trust assumptions, no guarantee can be given about the message X receives from Y and the one Z sends to X.

Lemma 3.27 *Let Π be FR's protocol and σ be an execution in $E(\Pi)^D_X$. Then $\forall y : t, z_y : t$,*

$$\begin{array}{lll} \sigma & \models^* & \mathcal{T} \text{ implies} \\ \sigma & \models^* & \Box\, (\exists x : t, z_x : t \mid (\text{receive}(X, x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, z_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(z_x, x) = \kappa_x) \rightarrow \\ & & \quad (\text{receive}(Y, y) \in \mathtt{H}^{\mathtt{X}} \wedge \text{send}(X, z_y) \in \mathtt{H}^{\mathtt{Z}}) \rightarrow \text{unmask}(z_y, y) = \kappa_y). \end{array}$$

Proof: Let s_j be a state such that

$$s_j \models \exists x : t, z_x : t \mid \text{receive}(X, x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, z_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(z_z, x) = \kappa_x.$$

And assume that $\exists z_y : t, y : t$ such that

$$s_j \models \text{send}(X, z_y) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, y) \in \mathtt{H}^{\mathtt{X}}.$$

If c_x, d_x, c_y, and d_y are messages such that

$$\begin{array}{lll} s_j & \models & \text{receive}(X, c_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, d_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \\ & & \text{send}(X, d_y) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, c_y) \in \mathtt{H}^{\mathtt{X}}, \end{array}$$

then we want to prove that $\text{unmask}(d_y, c_y) = \kappa_y$.

1. According to Lemma 3.28,

   $$s_j \models \text{receive}(X, c_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, d_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(d_x, c_x) = \kappa_x$$

   implies that

   $$s_j \models \exists x_1 \ldots x_4 : t \times t \times t \times t \mid (\text{receive}(X, x_1 \ldots x_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{send}(Y, x_3) \in \mathtt{H}^{\mathtt{Z}}),$$

   which implies, according to rules R_{Z_4} and R_{Z_6}, that

   $$s_j \models \exists y_1 \ldots y_4 : t \times t \times t \times t \mid (\text{receive}(Y, y_1 \ldots y_4) \in \mathtt{H}^{\mathtt{Z}} \wedge y_1 = x_2 = \text{aux\_hash}(y_3, x_4).$$

   In what follows, let $a_1 \ldots a_4$ and $b_1 \ldots b_4$ be such that

   $$s_j \models \text{receive}(X, a_1 \ldots a_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, b_1 \ldots b_4) \in \mathtt{H}^{\mathtt{Z}}.$$

   Thus, $s_j \models b_1 = a_2 = \text{aux\_hash}(b_3, a_4)$.

2. From
$$s_j \models \text{receive}(X, a_1 \ldots a_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{send}(Y, a_3) \in \mathtt{H}^{\mathtt{Z}},$$
$$s_j \models \text{send}(X, d_y) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{receive}(Y, b_1 \ldots b_4) \in \mathtt{H}^{\mathtt{Z}},$$
T_{Z_3}, and the fact that there can be only one 'receive-from-Y' event and only one 'send-to-X' event at Z, we can conclude that $d_y = b_3$.

3. From
$$s_j \models \text{receive}(X, a_1 \ldots a_4) \in \mathtt{H}^{\mathtt{Z}}$$
we can conclude, by communication assumption, that
$$s_j \models \text{send}(Z, a_1 \ldots a_4) \in \mathtt{H}^{\mathtt{X}}.$$
And, according to Lemma 3.29,
$$s_j \models (a_2 = \text{hash}(\kappa_y)) \;\wedge\; (\exists k : t \mid \text{receive}(Y, k) \in \mathtt{H}^{\mathtt{X}} \wedge \text{hash}(k) = a_4).$$
Since there can be only one 'receive-from-Y' event at X, and
$$s_j \models \text{receive}(Y, c_y) \in \mathtt{H}^{\mathtt{X}},$$
we conclude that $\text{hash}(c_y) = a_4$.

4. We are now ready to prove that $\text{unmask}(d_y, c_y) = \kappa_y$.
$$\begin{aligned} \text{hash}(\kappa_y) \;=\; a_2 &= \text{aux\_hash}(b_3, a_4) \\ &= \text{aux\_hash}(b_3, \text{hash}(c_y)) \\ &= \text{hash}(\text{unmask}(b_3, c_y)) \\ &= \text{hash}(\text{unmask}(d_y, c_y)). \end{aligned}$$

□

Lemma 3.28 is an auxiliary lemma used in the proof of Lemma 3.27. It establishes a straightforward result about sequencing of events and holds for all executions $\sigma \in E(\Pi)^D_X$.

Lemma 3.28 *Let* Π *be FR's protocol and* σ *be an execution in* $E(\Pi)^D_X$. *Then*

$$\begin{aligned} \sigma \;\models^*\; & \text{shareable}(\kappa_x, \{X\}) \rightarrow \\ & \Box\, (\exists x : t, z_x : t \mid \\ & \qquad (\text{receive}(X, x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, z_x) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(z_x, x) = \kappa_x) \rightarrow \\ & \quad \exists x_1 \ldots x_4 : t \times t \times t \times t \mid \\ & \qquad \text{receive}(X, x_1 \ldots x_4) \in \mathtt{H}^{\mathtt{Z}} \wedge \text{send}(Y, x_3) \in \mathtt{H}^{\mathtt{Z}}). \end{aligned}$$

Proof:

1. Let s_n be a state and m_1 and m_2 be messages such that
$$s_n \models \text{receive}(X, m_1) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{receive}(Z, m_2) \in \mathtt{H}^{\mathtt{Y}} \wedge \text{unmask}(m_2, m_1) = \kappa_x.$$

2. By communication assumption,

$$s_n \models \text{receive}(Z, m_2) \in \mathtt{H}^{\mathtt{Y}}$$

implies that

$$s_n \models \text{send}(Y, m_2) \in \mathtt{H}^{\mathtt{Z}}.$$

3. From straightforward backchaining, we know that

$$s_n \models \exists x_1 \dots x_4 : t \times t \times t \times t \mid \text{receive}(X, x_1 \dots x_4) \in \mathtt{H}^{\mathtt{Z}}.$$

In what follows, let $x_1 \dots x_4$ be $a_1 \dots a_4$. We want to prove that

$$\text{unmask}(m_2, m_1) = \kappa_x \to m_2 = a_3,$$

or equivalently,

$$m_2 \neq a_3 \to \text{unmask}(m_2, m_1) \neq \kappa_x.$$

4. We first find out the structure of a_3.

   (a) By communication assumption,

   $$s_n \models \text{receive}(X, a_1 \dots a_4) \in \mathtt{H}^{\mathtt{Z}}$$

   implies

   $$s_n \models \text{send}(Z, a_1 \dots a_4) \in \mathtt{H}^{\mathtt{X}},$$

   which, according to rule R_{X_5}, implies that $\exists i < n, g : t$ such that

   $$s_i \models \text{send}(Z, a_1 \dots a_4) \notin \mathtt{H}^{\mathtt{X}} \wedge \text{send}(Y, g) \in \mathtt{H}^{\mathtt{X}},$$

   and

   $$s_{i+1} \models \text{last}(\mathtt{H}^{\mathtt{X}}) = \text{send}(Z, a_1 \dots a_4) \wedge a_3 = \text{mask}(\kappa_x, g).$$

   (b) From 1) we know that $s_n \models \text{receive}(X, m_1) \in \mathtt{H}^{\mathtt{Y}}$, which, by communication assumption, implies that that $s_n \models \text{send}(Y, m_1) \in \mathtt{H}^{\mathtt{X}}$.

   (c) Since only one 'send-to-Y' event could have happened at X, we conclude that $g = m_1$.

   (d) Since $a_3 = \text{mask}(\kappa_x, g)$ and $g = m_1$, we can conclude that

   $$\text{unmask}(a_3, m_1) = \kappa_x.$$

5. Next, we prove that $m_2 \neq a_3 \to \text{unmask}(m_2, m_1) \neq \kappa_x$. If $m_2 \neq a_3$, then $m_2 \neq mask(\kappa_x, g)$. Then m_2 could be:

   (a) a basic message;

   (b) $m_2 = \text{mask}(b_1, b_2)$, where $b_1 \neq \kappa_x$ or $b_2 \neq g$;

   (c) $m_2 = \text{unmask}(b_1, b_2)$;

   (d) $m_2 = \text{hash}(b_1)$; or

   (e) $m_2 = \text{aux\_hash}(b_1, b_2)$.

All the values above are such that $\text{unmask}(m_2, g) \neq \kappa_x$.

□

Lemma 3.29 is also an auxiliary lemma used in the proof of Lemma 3.27. It concerns the values of the second and the fourth components of the message X sends to Z: the second component is the hash of Y's secret, and the fourth component is the hash of the message it receives from Y. This lemma is true because we are considering only executions where X is compliant. It may not be true otherwise.

Lemma 3.29 *Let* $\sigma \in E(\Pi)_X^D$. *Then*

$$\sigma \models^* \Box\,(\exists x_1 \ldots x_4 : t \times t \times t \times t \mid (\text{send}(Z, x_1 \ldots x_4) \in \mathtt{H}^{\mathtt{X}} \rightarrow (x_2 = \text{hash}(\kappa_y) \ \wedge \exists k : t \mid \text{receive}(Y, k) \in \mathtt{H}^{\mathtt{X}} \wedge \text{hash}(k) = x_4))).$$

Proof:

1. Let s_j be state and $a_1 \ldots a_4 : t \times \ldots \times t$ be a message such that

$$s_j \models \text{send}(Z, a_1 \ldots a_4) \in \mathtt{H}^{\mathtt{X}}.$$

   Then, $\exists i < j$ such that s_i is the state at which the event occurred, i.e.,

$$s_i \models \text{send}(Z, a_1 \ldots a_4) \notin \mathtt{H}^{\mathtt{X}} \text{ and } s_{i+1} \models \text{send}(Z, a_1 \ldots a_4) \in \mathtt{H}^{\mathtt{X}}.$$

2. Since R_{X_5} is the only protocol rule that prescribes a 'send-to-Z' event, and X behaves compliantly in σ, we can conclude that

   (a) $a_2 = \rho_y$, and

   (b) $a_4 = \text{hash}(c)$, where c is such that $s_i \models \text{receive}(Y, c) \in \mathtt{H}^{\mathtt{X}}$,

   which is exactly what we want to conclude, given that $\rho_y = \text{hash}(\kappa_y)$ according to the initial conditions.

□

3.6 Summary

In this section, we summarize the findings of our analysis and discuss our contributions towards formalizing semi-trustworthiness.

Table 3.7 summarizes our findings. All-protection is guaranteed in entries with a $\surd$. Some entries come with briefings of relevant facts. Note that the protocol is not all-protective if communication links can fail. To be fair, Franklin and Reiter assume reliable links, even though this assumption does not appear explicit in their paper.

As we mentioned, Franklin and Reiter's protocol is an ideal candidate for our case study because it uses semi-trusted intermediaries. Franklin and Reiter define semi-trusted intermediaries as those that can misbehave on their own, but do not conspire with either of the main parties. Implicit in this definition is the concept of intermediaries that can misbehave in certain ways (misbehavings of

	compliance	abortion	deception
Channel Failures	(1)	(1)	(1)
No Channel Failures	$\surd$	$\surd$ (2)	$\surd$ (3)

(1) All-protection is violated if Z cannot execute the second forward, or if one of the forwarded messages does not get to its destination.

(2) If X terminates her execution before sending the second share of her secret K_x to Z, she will receive nothing from Z and will not be able to obtain K_y. If she terminates her execution after sending the second share of K_x to Z, no harm (except to herself) can be made. The same reasoning applies to Y. Z is trusted not to terminate its execution between forwarding messages to X and Y.

(3) Deceptions by either X or Y are detectable by Z. And Z is trusted not to deceive unfairly.

Figure 3.7: Summary table

their own or "plain" misbehavior), but not in others (misbehavings that are conspiracy). Franklin and Reiter did not precisely characterize either plain misbehaviors or conspiracy misbehaviors, however. In their failure model, only one principal can fail or misbehave in each execution, and there is no restriction on how the intermediary can fail or misbehave.

Even though they do not define conspiracy behaviors, they classify the execution in Fig. 3.5 as exhibiting conspiracy behavior. Instead of ignoring them in their analysis of the protocol (since these behaviors are not exhibited by semi-trusted intermediaries), they took them into account, and questioned whether the protocol should offer protection against conspiracies.

These intuitive notions of conspiracy and semi-trustworthiness are precisely captured in our framework. Plain misbehaviors are deviations that do not violate trust assumptions; conspiracy misbehaviors are those that do; and semi-trusted intermediaries are those that exhibit only plain misbehaviors.

Chapter 4

The NetBill Protocol

In this chapter, we present our second case study, the NetBill Protocol [28] proposed by Cox, Tygar and Sirbu. We specify the protocol in our protocol specification formalism, and analyze it with respect to protection of individuals' interests using our framework. Since we are interested in protection of all parties equally, we analyze it with respect to all-protection.

NetBill differs from Franklin and Reiter's protocol in the type of intermediaries they use. In Franklin and Reiter's protocol, the intermediary is a "stranger" that has no further responsibility once the on-line exchange is finished. In NetBill, the intermediary is a server that has a role to play even after transactions have occurred. NetBill supports off-line dispute resolutions, and the NetBill server is the one to provide critical messages to enable such resolutions.

The NetBill protocol is the second in complexity (among the three we analyze), but its protection properties are the most complex. They are complex because the notion of protection required here is open; and one needs to take into account not only goods and payments exchanged on-line, but also other messages exchanged between the participants. It is critical to consider these messages because they prove what happened on-line; with these messages, participants can claim what they are entitled to, but did not get on-line.

Our analysis does not reveal any surprises. NetBill is both customer-protective and merchant-protective under all three deviation modes, even when communication links can fail. NetBill is not affected by link failures because the NetBill server is assumed to be permanently reachable.

Our analysis does make explicit, however, interesting points about how the protocol guarantees all-protection, and what the NetBill server is trusted to do. Under compliance and abortion modes, all-protection is guaranteed by the server's transaction capabilities alone; certified delivery is needed only under deception mode. More interestingly, certified delivery achieves its goals only if the NetBill server satisfies a small set of trust assumptions. Basically, the server is entrusted to handle accounts and keys honestly; to provide unique and non-repudiable proofs of what happens on-line through transaction slips; and to allow merchants to learn what really happens with a transaction – so that keys are not given to customers without their knowledge. Even though it may seem counter-intuitive at first, NetBill's all-protection does not depend on the server's releasing the keys that merchants send it, or its retaining transaction requests.

The rest of this chapter is structured as follows. In Section 4.1, we introduce the protocol and assumptions made by its designers. In Section 4.2, we present an abstract formulation of the mathematical building blocks used in it. In Section 4.3, we formalize both the protocol and the protection properties. In Section 4.4, we show the main results and their high-level analyses.

Formal analyses of these results are found in Section 4.5. Finally, in Section 4.6, we summarize the findings from Sections 4.4 and 4.5, and discuss insights gained through our exercise.

4.1 Introduction

In this section, we introduce the NetBill protocol. The version presented here is in reality an abstract version of the one given in [84]. We first give some preliminary remarks (Section 4.1.1), then present the protocol (Section 4.1.2), and finally discuss our abstractions (Section 4.1.3).

4.1.1 Preliminaries

NetBill was designed to enable sales of information goods – goods deliverable over the network. It aims to guarantee three levels of atomicity [84]: money atomicity, goods atomicity, and certified delivery. According to Tygar, a protocol is money-atomic if it transfers funds from one party to another without the possibility of creation or destruction of money; a protocol is goods-atomic if it is money-atomic and guarantees exchange of goods for money; finally, a protocol satisfies certified delivery if it is goods-atomic and provides means for both merchants and customers to prove what goods were delivered.

Goods atomicity is different from Franklin and Reiter's fair exchange property. Goods atomicity only guarantees that money is exchanged for something – not necessarily what the customer pays for. To guarantee that customers receive what they pay for, NetBill relies on certified delivery, which allows customers and merchants to prove what happened on-line and settle disputes off-line.

Transactions in NetBill involve three principals: a customer C, a merchant M, and an intermediary N – the NetBill server. C and M may be untrustworthy, but N is assumed to be trusted. For the remainder of this chapter, we refer to C, M, and N respectively as she, he and it. All principals have public key pairs. Public keys are certified before keys are used. Private keys are used to sign selected messages for accountability. C and M have their keys certified by N, whereas N has its key certified by an entity external to the system.

Both C and M must have accounts with N. Accounts are uniquely associated with account holders' ids and public keys. Besides holding accounts for C and M, N mediates transactions and transfers funds between different accounts when parties need to pay each other.

Sale transactions in NetBill consist of four logical phases: ordering, goods delivery, payment, and dispute resolution. And each sale transaction (informal sense) corresponds to a distributed transaction (technical sense) involving C, M, and N. Ordering in NetBill is simple: to order, C sends M a message specifying the goods. Goods delivery happens in two steps. First, M encrypts the goods and delivers it directly to C; then, after the payment, the decryption key is released – either by M itself or through N. Payments happen at N. Upon receiving a transaction request from M, N may commit or abort the transaction. If the transaction commits, N releases a signed slip attesting to the result of the transaction. The result can be successful or unsuccessful. If the result is successful, N transfers funds from C's account to M's account and releases the decryption key in the slip. If the result is unsuccessful, no funds are transferred, and the decryption key is not released. Dispute resolution, if needed, happens off-line, with the help of an outside arbitrator.

1. $C \rightarrow M$: *order*
2. $M \rightarrow C$: $enc(g, k)\ cc(enc(g, k))\ t$
3. $C \rightarrow M$: $epo\ seal(epo, k_c^{-1})$,
   where $epo = c\ cc(enc(g, k))\ t$,
4. $M \rightarrow N$: $tr\ seal(tr, k_m^{-1})$,
   where $tr = m\ signed\_epo\ k$ and
   $signed\_epo = epo\ seal(epo, k_c^{-1})$
5. $N \rightarrow M$: $slip\ seal(slip, k_n^{-1})$
   where $slip = result\ c\ m\ transferred\ k\ t$,
6. $M \rightarrow C$: $slip\ seal(slip, k_n^{-1})$
   where $slip = result\ c\ m\ transferred\ k\ t$.

The various symbols denote:

order :	a predefined message for requesting the goods;
$k_c^{-1}, k_m^{-1}, k_n^{-1}$:	respectively C, M, and N's private keys;
c, m :	respectively C's and M's ids as they appear in their public key certificates;
g :	the goods;
k :	a randomly generated symmetric key;
t :	transaction id;
result, *transferred* :	binary values indicating respectively whether a transaction yielded a successful result and whether any funds have been transferred.

Figure 4.1: The NetBill protocol.

4.1.2 The protocol

In this subsection, we present the protocol itself (Fig. 4.1). In Fig. 4.1, g is the goods for sale, and *order* is a predefined, distinguished message used in goods requests. k_c^{-1}, k_m^{-1}, and k_n^{-1} are respectively C, M, and N's private keys; and c and m are respectively C's and M's ids as they appear in their public key certificates. We assume that principals have cryptographic capabilities: they can produce cryptographic checksums, encrypt and decrypt messages, and sign and verify signatures. We also assume that C's account always has enough funds; thus no transaction is prevented from committing successfully because of C's lack of funds.

To order g, C sends *order* to M (step 1). Upon receiving this message, M starts the goods delivery procedure. To deliver g, M first encrypts it with a randomly generated symmetric key k, and then cryptographically checksums the result of the encryption. M then sends the encrypted goods $enc(g, k)$, its checksum $cc(enc(g, k))$, and a transaction id t to C (step 2). t is a globally fresh id that identifies this transaction. Once C receives the message sent by M, she checks whether the value of the checksum is consistent with the value of the encrypted goods. If so, C prepares an electronic purchase order *epo*, consisting of her own id, the value of the checksum, and the transaction id; signs it; and sends it to M (step 3). At any time before this signed epo is submitted, C may abort the transaction and be in no danger of its being completed against her will. The submission of this signed epo marks the "point of no return" for C.

When M receives this signed epo, he verifies whether both the checksum and the transaction id are the ones he sent in step 2. If so, M prepares a transaction request *tr*, consisting of his own id, the signed epo, and the goods decryption key; signs this whole message; and submits it to N (step 4) for final transaction processing. At any time before this request is submitted to N, M may abort the transaction and be in no danger of its being completed against his will. The submission of this request marks the "point of no return" for M.

When N receives a transaction request, N can commit or abort the transaction. If N can verify both C's and M's signatures on the request, N commits the transaction; otherwise, N aborts it. A transaction commit yields two types of results, depending on whether the request's transaction id is fresh. If the transaction id is fresh, then N transfers funds from C's account to M's account, records the request, and generates a signed slip attesting to a successful result. If it is stale, N only records the request and generates a signed slip attesting to an unsuccessful result. A slip consists of customer id, merchant id, transaction id, and two binary values – *result* and *transferred* – indicating respectively whether the transaction yielded a successful result and whether any funds has been transferred. For transactions with successful results, the decryption key is enclosed. N then sends the slip to M (step 5).

Upon receiving the slip from N, M records it, and forwards it to C (step 6).

Intuitively, having N coordinate transactions is critical to achieving all-protection in this protocol. If the transaction fails as a result of processor or communication failure before N commits the transaction, then no money changes hands, and C never receives the decryption key. On the other hand, if the transaction commits with a successful result, then funds are transferred, and a copy of k is kept at N. Normally, a copy of k is forwarded back to C via M. But if something goes wrong, C can obtain k from N.

In the NetBill model, communication channels between different parties can fail. However, N is assumed to be always reachable: it is guaranteed not to disappear, and, in the worst case, the other parties in the system can reach it physically, off-line. (For other assumptions about N, see page 71 – **Trust Assumptions**.) Because N is always reachable, C and M can always obtain the slip from N, even if it is not received during the normal message exchanges of the protocol.

4.1.3 Our abstractions

In this subsection, we discuss the abstractions we make.

- NetBill can be used for selling both physical goods and information goods. In both cases, the customer expects something in exchange for payment: a receipt (needed to claim the actual physical goods) or the goods itself. Henceforth, we do not distinguish these two cases, and assume that the customer just expects something – in electronic form – in exchange for payment.

- NetBill was designed to deal with several issues secondary to the main exchange. Some examples are price discounts based on membership in different groups, subscriptions, and customer pseudonyms. In this work, we abstract away all its secondary features and focus on the main exchange. These abstractions allow us to use simpler messages in our version of the protocol.

- In NetBill, customers can buy one or more items that merchants have for sale, and price negotiation phase can be very elaborate, taking multiple iterations. Because we want to focus on exchanges, we assume that merchants have only one item for sale, and everyone in the system knows its price. This allows us to skip the price negotiation phase and start the protocol with a goods ordering message. Our simplification does not compromise the adequacy of our modeling or analyses because exchanges start only after prices have been agreed on.

- The original NetBill uses cryptography to implement communication security. Given that communication channels are assumed to be secure in our model, we can abstract away a number of cryptographic operations in our version of the protocol. Kerberos tickets, symmetric key encryptions, and crypto checksums, used for authenticity, confidentiality, and integrity in the original protocol, are all abstracted away. The ones we kept in our version of the protocol serve other purposes such as non-repudiability.

- Accounts in NetBill are credit or debit accounts, and need to be balanced or replenished. Since the NetBill transaction protocol does not deal with these issues, we simply assume that accounts have inexhaustible funds or credits.

- For efficiency, NetBill uses RSA for customer and merchant signatures, and DSA for NetBill signatures. We abstract these details away and model both signature schemes indistinguishly as an abstract public key crypto system that has a signing function and a signature verification function satisfying the property given in Def. 4.1 (8).

- NetBill uses timestamps to expire stale transactions. The notion of time is relevant in defining p-protection because only timely exchanges matter in some cases. For simplicity, however, we abstract the notion of transaction expiration away in our version of the protocol. Our analysis results thus apply to exchanges that are not time-critical.

- We assume that none of the private keys have been compromised or revocated. This assumption allows us to see signatures as expressing their signers' approval, agreement, etc.

4.2 Abstract Formulation of the Building Blocks

4.2.1 Cryptographic Building Blocks

The NetBill protocol, as introduced in Section 4.1.2, uses a number of cryptographic building blocks. In this subsection, we present an abstract formulation of these building blocks. In what follows, $\doteq$ denotes conversion.

Messages can be encrypted in NetBill using symmetric key encryption. If m is a message and k is a symmetric key, then

$$\mathrm{enc}(m, k)$$

denotes the encryption of m by k. enc has a corresponding decryption function dec. Given a message m and a symmetric key k,

$$\mathrm{dec}(m, k)$$

denotes the decryption of m by k. For all encrypted messages $m' = \mathrm{enc}(m, k)$, and keys k', $\mathrm{dec}(m', k') \doteq m$, if $k = k'$. That is,

$$\mathrm{dec}(\mathrm{enc}(m, k), k) \doteq m.$$

Messages can be cryptographically checksummed. Given a message m,

$$\mathrm{cc}(m)$$

denotes m's cryptographic checksum.

In public key crypto-systems, private and public keys come in pairs. If k^{-1} and k are respectively the private key and the public key of a key pair, then they satisfy the predicate keyPair. That is,

$$\mathrm{keyPair}(k^{-1}, k) = \mathrm{true}.$$

Messages can be signed using public key cryptography. Given a message m and a private key k^{-1},

$$\mathrm{seal}(m, k^{-1})$$

denotes the signature of k^{-1} on m. Signatures can be verified using public keys. Given a message m, a signature s, and public key k, the predicate vseal models signature verification:

$$\mathrm{vseal}(m, s, k) = \mathrm{true}$$

if and only if s is a signature of k^{-1} on m, and k^{-1} is the private counterpart of k.

In Def. 4.1, we list the cryptographic building blocks as well as their abstract properties:

Definition 4.1 *Cryptographic building blocks*

1. *Symmetric keys have type t_symk, i.e., if k is a symmetric key, then k: t_symk;*
2. *Private keys have type t_prik, i.e., if k^{-1} is a private key, then k^{-1}: t_prik;*
3. *Public keys have type t_pubk, i.e., if k is a public key, then k: t_pubk;*
4. *Cryptographic checksums have type t_cks, i.e., if $m' = cc(m)$ for a message m, then m': t_cks;*
5. *Signatures have type t_seal, i.e., if $m' = seal(m, k^{-1})$ for a message m and a private key k^{-1}, then m': t_seal;*
6. *t is a "super-type", i.e., for any message m, $m : t$; (t is called $\top$ ("top") in the subtyping literature.)*

   Effectively, we say that a message has type t if we want to look at the message merely as a string of bits, and not as a structured message. For example, in an encryption, it is irrelevant whether the message being encrypted is an integer, a key, or a meaningless string of bits.
7. *Functions and predicates:*

- *enc:* $t \times t\_symk \rightarrow t$;
- *dec:* $t \times t\_symk \rightarrow t$;
- *cc:* $t \rightarrow t\_cks$;
- *keyPair:* $t\_prik \times t\_pubk \rightarrow boolean$;
- *seal:* $t \times t\_prik \rightarrow t\_seal$;
- *vseal:* $t \times t\_seal \times t\_pubk \rightarrow boolean$;

8. *Conversion rules:*

- $\forall$ *m: t, k: t_symk, k′: t_symk, dec(enc(m, k), k′)* $\doteq$ *m, if and only if* $k = k'$;
- $vseal(m, s, k) \; = \; \begin{cases} true, & \text{if } \exists\, k^{-1}: t\_prik \mid s = seal(m, k^{-1}) \wedge keyPair(k^{-1}, k); \\ false, & otherwise. \end{cases}$

4.2.2 Product Types and Projection Functions

The types that appear in Subsection 4.2.1 define messages required and produced by cryptographic functions. Other types of basic messages used in NetBill are *t_pid*, for principal ids; and *t_tid*, for transaction ids.

Some concatenated messages have special semantics in NetBill. For conciseness, we name their corresponding product types, and define projection functions for these types. For example, certificates consist of two components: a principal id and a public key, and have type $t\_pid \times t\_pubk$. For conciseness, we denote the product type $t\_pid \times t\_pubk$ by *t_cert*:

$$t\_cert = t\_pid \times t\_pubk.$$

For extracting the components of a certificate, we define projection functions pid and key. pid returns the principal id, while key returns the public key:

$$\text{If } m = m_1 m_2 \text{ has type } t\_cert\text{, then } \begin{cases} \text{id}(m) = m_1, \text{ and} \\ \text{key}(m) = m_2 \end{cases}$$

We list the product types and their corresponding projection functions below.

Certificates: $t\_cert = t\_pid \times t\_pubk$

$$\text{If } m = m_1 m_2 \text{ has type } t\_cert\text{, then } \begin{cases} \text{id}(m) = m_1, \text{ and} \\ \text{key}(m) = m_2. \end{cases}$$

Epos: $t\_epo = t\_pid \times t\_cks \times t\_tid$

$$\text{If } m = m_1 m_2 m_3 \text{ has type } t\_epo\text{, then } \begin{cases} \text{cid}(m) = m_1, \\ \text{cks}(m) = m_2, \text{ and} \\ \text{tid}(m) = m_3. \end{cases}$$

Signed epos: $t\_sepo = t\_epo \times t\_seal$

$$\text{If } m = m_1 m_2 \text{ has type } t\_sepo\text{, then } \begin{cases} \text{msg}(m) = m_1, \text{ and} \\ \text{sig}(m) = m_2. \end{cases}$$

Transaction requests: $t\_tr = t\_pid \times t\_sepo \times t\_symk$

$$\text{If } m = m_1 m_2 m_3 \text{ has type } t\_tr, \text{ then } \begin{cases} \mathrm{mid}(m) = m_1, \\ \mathrm{sepo}(m) = m_2, \text{ and} \\ \mathrm{key}(m) = m_3. \end{cases}$$

Signed transaction requests: $t\_str = t\_tr \times t\_seal$

$$\text{If } m = m_1 m_2 \text{ has type } t\_str, \text{ then } \begin{cases} \mathrm{msg}(m) = m_1, \text{ and} \\ \mathrm{sig}(m) = m_2. \end{cases}$$

Transaction slips: $t\_slip = boolean \times t\_pid \times t\_pid \times boolean \times t\_symk \times t\_tid$

$$\text{If } m = m_1 m_2 m_3 m_4 m_5 m_6 \text{ has type } t\_slip, \text{ then } \begin{cases} \mathrm{res}(m) = m_1, \\ \mathrm{cid}(m) = m_2, \\ \mathrm{mid}(m) = m_3, \\ \mathrm{trans}(m) = m_4, \\ \mathrm{key}(m) = m_5, \text{ and} \\ \mathrm{tid}(m) = m_6. \end{cases}$$

Signed transaction slips: $t\_sslip = t\_slip \times t\_seal$

$$\text{If } m = m_1 m_2 \text{ has type } t\_sslip, \text{ then } \begin{cases} \mathrm{msg}(m) = m_1, \text{ and} \\ \mathrm{sig}(m) = m_2. \end{cases}$$

Note that some projection functions are overloaded. For example, msg can be applied to messages of types *t_sepo*, *s_str*, and *t_sslip*; and tid can be applied to messages of types *t_epo* and *t_slip*.

For readability, we syntactically distinguish applications of cryptographic functions and projection functions. Applications of cryptographic functions are denoted in a standard way: $f(a_1, \ldots, a_n)$, where f is the function and a_i, $i = 1, \ldots, n$, are the arguments. Applications of projection functions, on the other hand, are denoted as $a.f$, where a is the argument and f is the function. For example, if m is a certificate, then we use $\mathrm{cc}(m)$ to denote m's checksum, and $m.\mathrm{key}$ to denote m's public key component.

4.3 Formalizing the Protocol and the Protection Properties

In this section, we specify the NetBill protocol using our specification formalism (Section 2.3), and give concrete specifications of protection properties as applied to this protocol.

4.3.1 Protocol Specification

Principals of the Protocol

$$\mathcal{P} = \{C, M, N\}$$

Initial Conditions

Let κ_c^{-1}, κ_m^{-1}, and κ_n^{-1} be respectively C, M, and N's private keys; κ_n be N's public key; ι_c and ι_m, and ϕ_c and ϕ_m be respectively C's and M's ids and public key certificates. Also, let γ be the goods for sale, and Θ be the set of stale transaction ids. The initial condition

$$\begin{aligned}\mathcal{I} = \;& \text{shareable}(\kappa_c^{-1}, \{C\}) \wedge \text{shareable}(\kappa_m^{-1}, \{M\}) \wedge \text{shareable}(\kappa_n^{-1}, \{N\}) \;\wedge \text{shareable}(\gamma, \{M\}) \wedge \\ & \text{keyPair}(\kappa_c^{-1}, \phi_c.\text{key}) \wedge \text{keyPair}(\kappa_m^{-1}, \phi_m.\text{key}) \wedge \text{keyPair}(\kappa_n^{-1}, \kappa_n) \;\wedge \\ & \iota_c = \phi_c.\text{id} \wedge \iota_m = \phi_m.\text{id} \;\wedge \\ & \{\kappa_c^{-1}, \iota_c\} \subseteq \mathtt{MS}^{\mathtt{C}} \wedge \{\kappa_m^{-1}, \iota_m, \Theta, \gamma\} \subseteq \mathtt{MS}^{\mathtt{M}} \wedge \{\kappa_n^{-1}, \phi_c, \phi_m, \Theta\} \subseteq \mathtt{MS}^{\mathtt{N}}\end{aligned}$$

says that all principals have their own private keys; N has both C and M's certificates; M has the goods, and M and N have consistent views of which transaction ids are stale. Note that this last condition only holds because our system has only one merchant.

Local Protocols

In what follows, w, x, y, z are variables; identifiers in SMALL CAP are constants; and those in *slanted* fonts are placeholders for more complex messages. The constants that appear below are ORDER, which denotes the predefined message used in goods requests; TRUE and FALSE; and NULL, which indicates the absence of messages.

In the specification below, we need two types of events not listed in Section 2.2.2: *new*(x, y) and *commit*(x, y). We use new(x, y) to model generation of elements y not found in the set x. In the context of NetBill, new(Θ, θ) models the generation of a fresh transaction id θ, different from all those found in Θ, the set of stale transaction ids. And any message (epos, transaction requests, transaction slips, etc) that encloses a fresh transaction id is also fresh.

Commit(x, y) is used to model transaction commits. In our model, two messages are generated with each transaction commit: a transaction slip and a transaction request. They appear respectively as arguments x and y in commit events. We assume that these messages are retained at N, and are promptly made available to arbitrators and parties involved in the transaction upon request.

Next we present the local protocols.

C's local protocol $\mathcal{R}_C$ consists of the following rules:

R_{C_1}: $\text{send}(M, \textsc{Order}) \notin \mathtt{H} \Longrightarrow \{\text{send}(M, \textsc{Order}), \text{exit}\}$

R_{C_2}: $\text{last}(\mathtt{H}) = \text{send}(M, \textsc{Order}) \Longrightarrow \{\text{receive}(M, x : t \times t\_cks \times t\_tid), \text{timeout}, \text{exit}\}$

R_{C_3}: $\exists x_1 : t, x_2 : t\_cks, x_3 : t\_tid \mid \text{last}(\mathtt{H}) = \text{receive}(M, x_1\ x_2\ x_3) \wedge \text{cc}(x_1) = x_2 \Longrightarrow$
$\{\text{send}(M,\ epo\ \text{seal}(epo,\ \kappa_c^{-1})), \text{exit}\}$,

where: $epo = \iota_c\ x_2\ x_3$.

R_{C_4}: $\exists x : t\_sepo \mid \text{last}(\mathtt{H}) = \text{send}(M, x) \Longrightarrow \{\text{receive}(M, y : t\_sslip), \text{timeout}\}$

R_{C_5}: $(\exists x_1 : t, x_2 : t\_cks, x_3 : t\_tid \mid (\text{last}(\mathtt{H}) = \text{receive}(M, x_1\ x_2\ x_3) \wedge \text{cc}(x_1) \neq x_2) \vee$
$\text{last}(\mathtt{H}) = \text{timeout} \vee \exists y : t\_sslip \mid \text{last}(\mathtt{H}) = \text{receive}(M, y) \Longrightarrow \{\text{exit}\}$

M's local protocol $\mathcal{R}_M$ consists of the following rules:

R_{M_1}: $\text{receive}(C, \textsc{Order}) \notin \mathtt{H} \Longrightarrow \{\text{receive}(C, \textsc{Order}), \text{timeout}, \text{exit}\}$

R_{M_2}: $\text{receive}(C, \textsc{Order}) \in \mathtt{H} \wedge (\nexists x : t\_symk \mid \text{random}(x) \in \mathtt{H}) \Longrightarrow \{\text{random}(y : t\_symk), \text{exit}\}$

R_{M_3}: receive(C, ORDER) $\in$ H $\wedge$ ($\nexists x : t\_tid$ | new(Θ, x) $\in$ H) $\Longrightarrow$ {new($\Theta, y : t\_tid$), exit}

R_{M_4}: $\exists x : t\_tid$ | new(Θ, x) $\in$ H $\wedge$ $\exists y : t\_symk$ | random(y) $\in$ H $\wedge$
($\nexists z : t \times t\_cks \times t\_tid$ | send(C, z) $\in$ H) $\wedge$ last(H) $\neq$ timeout $\Longrightarrow$
{send(C, eg cc(eg) x), timeout, exit},

where eg = enc(γ, y)

R_{M_5}: $\exists x : t \times t\_cks \times t\_tid$ | last(H) = send(C, x) $\Longrightarrow$ {receive($C, y : t\_sepo$), timeout, exit}

R_{M_6}: $\exists x$: t_sepo, $y_1\ y_2\ y_3 : t \times t\_cks \times t\_tid$, z: t_symk |
last(H) = receive(C, x) $\wedge$ send($C, y_1\ y_2\ y_3$) $\in$ H $\wedge$ $y_2 = x$.msg.cks $\wedge$ $y_3 = x$.msg.tid $\wedge$
random(z) $\in$ H $\Longrightarrow$ {send(N, tr_req seal(tr_req, κ_m^{-1})), exit}

where $tr\_req = \iota_m\ x\ z$

R_{M_7}: $\exists x : t\_str$ | last(H) = send(N, x) $\Longrightarrow$ {receive($N, y : t\_sslip$), timeout}

R_{M_8}: $\exists x : t\_sslip$ | last(H) = receive(N, x) $\Longrightarrow$ {send(C, x), timeout}

R_{M_9}: last(H) = timeout $\vee$ $\exists z : t\_sslip$ | last(H) = send(C, z) $\vee$
$\exists x : t\_sepo, y_1\ y_2\ y_3 : t \times t\_cks \times t\_tid$ |
(last(H) = receive(C, x) $\wedge$ send($C, y_1\ y_2\ y_3$) $\in$ H $\wedge$ ($y_2 \neq x$.msg.cks $\vee$ $y_3 \neq x$.msg.tid)) $\Longrightarrow$
{exit}

N's local protocol $\mathcal{R}_N$ consists of the following rules:

R_{N_1}: $\nexists x : t\_str$ | receive(M, x) $\in$ H $\Longrightarrow$ {receive($M, y : t\_str$)}

R_{N_2}: $\exists x : t\_str, \exists y : t\_prik$ |
last(H) = receive(M, x) $\wedge$ x.msg.sepo.msg.tid $\notin \Theta$ $\wedge$
x.msg.mid = ϕ_m.id $\wedge$ vseal(x.msg, x.sig, ϕ_m.key) $\wedge$
x.msg.sepo.msg.cid = ϕ_c.id $\wedge$ vseal(x.msg.sepo.msg, x.msg.sepo.sig, ϕ_c.key) $\wedge$
$y = \kappa_n^{-1}$ $\wedge$ $y \in$ MS$^{\mathtt{N}}$ $\Longrightarrow$ {commit($slip$ seal($slip$, y), x)},

where: $slip$ = TRUE x.msg.sepo.msg.cid x.msg.mid TRUE x.msg.key x.msg.sepo.msg.tid

R_{N_3}: $\exists x : t\_str, \exists y : t\_prik$ |
last(H) = receive(M, x) $\wedge$ x.msg.sepo.msg.tid $\in \Theta$ $\wedge$
x.msg.mid = ϕ_m.id $\wedge$ vseal(x.msg, x.sig, ϕ_m.key) $\wedge$
x.msg.sepo.msg.cid = ϕ_c.id $\wedge$ vseal(x.msg.sepo.msg, x.msg.sepo.sig, ϕ_c.key) $\wedge$
$y = \kappa_n^{-1}$ $\wedge$ $y \in$ MS$^{\mathtt{N}}$ $\Longrightarrow$ {commit($slip$ seal($slip$, y), x)},

where: $slip$ = FALSE x.msg.sepo.msg.cid x.msg.mid FALSE NULL x.msg.sepo.msg.tid

R_{N_4}: $\exists x : t\_sslip, y : t\_str$ | last(H) = commit(x, y) $\Longrightarrow$ send(M, x)

R_{N_5}: $\exists x : t\_str$ | (last(H) = receive(M, x) $\wedge$
$\neg$ (x.msg.mid = ϕ_m.id $\wedge$ vseal(x.msg, x.sig, ϕ_m.key) $\wedge$
x.msg.sepo.msg.cid = ϕ_c.id $\wedge$ vseal(x.msg.sepo.msg, x.msg.sepo.sig, ϕ_c.key))) $\vee$
$\exists y : t\_sslip$ | last(H) = send(M, y) $\Longrightarrow$ {exit}

Note that, in our specification, N does not process a transaction request unless it can verify both C and M's signatures (R_{N_5}). If both signatures can be verified, then N commits the transaction, generates a slip (R_{N_2} and R_{N_3}), and sends it to M (R_{N_4}). A transaction can be committed with a successful result (R_{N_2}) or with an unsuccessful one (R_{N_3}). The content of the slip indicates it.

Trust Assumptions

In this subsection, we specify a set of trust assumptions for NetBill. Unlike the intermediary in FR's protocol, the intermediary here – the NetBill server – is assumed to be completely trusted, i.e., is expected to behave exactly as prescribed by the protocol. However, servers can fail and communication links can go down, and trusted servers may in fact exhibit deviant executions. When justifying why the NetBill server keeps cryptographically signed messages, Tygar himself says [84]

> "Many electronic commerce systems depend on some ultimate, trusted authority. For example, NetBill depends on the trustworthiness of a central server. However, even in the case where one uses a trusted server, one can minimize the effects of the security failures of that server. For example, in NetBill, detailed cryptographically-unforgeable records are kept so that if the central server was ever corrupted, it would be possible to unwind all corrupted actions and restore any lost money."

Thus, the trust assumptions we list below were not given explicitly by the authors of the protocol; instead, we inferred them from informal discussions [84] about how NetBill guarantees goods atomicity and certified delivery, and how disputes are settled off-line. By giving this set of trust assumptions, we are implicitly saying that NetBill can be all-protective even if the NetBill server does not behave compliantly; all-protection is guaranteed as long as the following trust assumptions are not violated.

In what follows, we first list informal statements of these trust assumptions; we then specify them formally.

In NetBill, neither C nor M is trusted. Thus, $T_C = \mathsf{true}$ and $T_M = \mathsf{true}$. N is trusted. It is trusted:

- T_{N_1}: To transfer funds only if the transaction commits;
- T_{N_2}: To release the key submitted by M only if the transaction commits;
- T_{N_3}: To sign slips made available by transaction commits.
- T_{N_4}: Not to produce signed slips with corrupted merchant ids; and
- T_{N_5}: To send M only slips made available by transaction commits.

We proceed with formalizations of these trust assumptions.

T_{N_1} and T_{N_2} always hold in our model, and therefore do not need additional specification. T_{N_1} always holds in our model because we do not model funds transfers explicitly. Their occurrences are reflected in transaction slips generated when transactions commit. `True` in a slip's 'transferred' field indicates that the corresponding transaction transferred funds. Given that slips are generated only when transactions commit, we conclude that funds transfers only occur when transactions commit.

This restriction is not serious because even though funds may be transferred independently of slip generations and transaction processings in reality, such transfers are sure to be disputed and voided.

Similarly, T_{N_2} always holds in our model because keys are released only through transaction slips, which are generated when transactions commit.

T_{N_3}: $\forall x$: *t_sslip*, $\Box$ (commit$(x, -^1) \in \texttt{H} \rightarrow$ vseal(x.msg, x.sig, κ_n))

In T_{N_4}, non-corrupted customer ids, merchant ids, and transaction ids mean those originating from transaction requests:

T_{N_4}: $\forall x$: *t_sslip*,
$\Box$ (commit$(x, -) \in \texttt{H} \rightarrow (\exists z : t_str \mid \text{receive}(M, z) \in \texttt{H} \wedge x.\text{msg.mid} = z.\text{msg.mid}))$

T_{N_5}: $\forall x$: *t_sslip*, $\Box$ (send$(M, x) \in \texttt{H} \rightarrow$ commit$(x, -) \in \texttt{H}$)

In Sections 4.4 and 4.5, we show how protection of individuals' interests can be compromised if these trust assumptions are violated in the system.

4.3.2 Specification of Protection Properties

In this subsection, we formally specify both C's and M's protection properties in the context of NetBill. Unlike in FR's protocol, where exchanges take place entirely on-line, here exchanges may start on-line and finish with dispute resolutions off-line. That is, participants in NetBill may not receive what they are supposed to during a protocol execution. The certified delivery mechanisms, however, enables them to take proofs of what really happened on-line to an off-line arbitrator, and claim what they are entitled to. This makes the concept of "receiving an item" (goods or payment) less clear-cut here than in FR's protocol.

Preliminaries

Before plunging into formalizations, we characterize a number of execution outcomes relevant to the specification of protection properties. For the specification of protection properties, relevant outcomes are those where C is guaranteed the goods or M is guaranteed a payment. C is guaranteed the goods if and only if

$C1$: she actually receives the goods during the protocol execution, or

$C2$: she can claim the goods in court.

M is guaranteed a payment if and only if

$M1$: he has proof that C received the goods, or

$M2$: he has proof that N committed the transaction a) with a successful result, and b) and did so in response to a valid request.

Note that for M, it is irrelevant whether funds are actually transferred during the transaction processing. What really matters is whether he is entitled to a payment (Conditions $M1$ and $M2$ make him entitled to a payment). For example, if funds are mistakenly transferred from C's account

[1] We use "-" to stand for messages whose actual values are irrelevant in the context.

to M's account during a transaction processing, C will certainly dispute the debit, and the transfer will certainly be voided. On the other hand, if M is entitled to a payment, but no funds are transferred during the transaction processing, then M can claim it later.

Before detailing each of the conditions above, we clarify the meanings of a few expressions used below: *N makes a message "publicly" available*; *C (M) has access to a message*; *an epo and a transaction slip correspond to each other*; and *a transaction request and a transaction slip correspond to each other.*

As a transaction coordinator and the trusted party of NetBill transactions, N is entrusted to keep a number of messages, and to make them available to selected parties. For example, transaction slips can be made available to the arbitrator and the parties involved in the transaction – C and M, but no one else. For conciseness, we use *N makes a message "publicly" available* to mean N makes a message available to the appropriate group of selected parties.

Related to the concept above is the concept of accessibility of messages. We say that C (M) *has access to a message* if C (M) has the message in his or her local store, or N has made it "publicly" available.

Finally, we say that *an epo and a transaction slip corresponds to each other* if they have the same customer id and transaction id; and *a transaction request and a transaction slip correspond to each other* if they have the same customer id, merchant id, and transaction id.

We now characterize the four conditions listed above. For conciseness, we say that a transaction request is valid if it is signed by both C and M, and its transaction id is fresh.

$C1$: C actually receives the goods g during the protocol execution if and only if:

- C receives a message $m_1\ m_2\ m_3$: $t \times t\_cks \times t\_tid$ with encrypted goods m_1;
- C has access to a transaction slip m_4, whose transaction id is m_3; and
- $\text{dec}(m_1,\ m_4.\text{msg.key}) = g$.

In other words, C actually receives the goods if and only if C receives an encrypted message and a key, and the goods can be obtained by decrypting the encrypted message with the key.

$C2$: C can claim g in court if and only if

- C receives a message $m_1\ m_2\ m_3$: $t \times t\_cks \times t\_tid$ with encrypted goods m_1;
- C has access to a fresh N-signed transaction slip m_4 satisfying the following conditions: m_4's transaction id is m_3, and m_4 attests to a transaction that yielded a successful result;
- N makes a valid m_4-corresponding transaction request m_5 "publicly" available; and
- The value of m_1 is consistent with the checksum enclosed in m_5, but decrypting it with the key enclosed in m_4 yields something other than g.

  That is, $\begin{cases} \text{cc}(m_1) = m_5.\text{msg.sepo.msg.cks} \wedge \\ \text{dec}(m_1,\ m_4.\text{msg.key}) \neq g. \end{cases}$

Intuitively, C can claim g in court if C can show that the transaction was successfully processed by N in response to a valid request, but the goods cannot be retrieved from the decryption key released in the transaction slip and the encrypted message given by M.

$M1$: M has proof that C received the goods g if and only if

- M received a C-signed epo m_1 – with a fresh transaction id – from C;
- M has access to a m_1-corresponding, N-signed transaction slip m_2; and
- encrypting g with the key released in m_2 (m_2.msg.key) yields an encryption whose checksum is consistent with the checksum enclosed in m_1 (m_1.msg.cks). That is

  $$\text{cc}(\text{enc}(g,\, m_2.\text{msg.key})) = m_1.\text{msg.cks}.$$

The three conditions above prove that C received g because 1) the existence of m_1 proves that C received an encrypted message; 2) the existence of m_2 proves that C can access the key released by N; and 3) the checksum test proves that g is retrievable from the encrypted message and the released key.

$M2$: M has proof that N committed the transaction a) with a successful result, and b) and did so in response to a valid fresh request if and only if

- M has access to a fresh N-signed transaction slip m_1 attesting to a transaction that yielded a successful result; and
- N makes a valid m_1-corresponding transaction request m_2 "publicly" available.

Note that C may or may not have received the goods on-line under these conditions. If C received the goods on-line, then M is clearly entitled to a payment. Otherwise, M should still be entitled to it, because C can claim the goods in court.

Formalization

We now use the conditions we just characterized to define protection properties for different parties in NetBill. C's protection property P_C says that

> "If M is entitled to a payment, then C actually receives the goods, or C can claim it in court";

and M's protection property P_M says that

> "If C actually receives the goods, or C can claim it in court, then M is entitled to a payment."

P_C and P_M can be respectively formalized as follows:

$$P_C : \Phi_i \rightarrow \Box((\Phi_{M1} \vee \Phi_{M2}) \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2})),$$

and

$$P_M : \Phi_i \rightarrow \Box((\Phi_{C1} \vee \Phi_{C2}) \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2})),$$

where

$$\Phi_i \quad = \quad \forall x : t\_tid, (\exists y : t \mid (y \in^* \mathtt{MS}^{\mathtt{M}} \vee y \in^* \mathtt{MS}^{\mathtt{N}}) \wedge x \sqsubseteq y) \rightarrow x \in \Theta$$

$$
\begin{aligned}
\Phi_{M1} \;=\;& \exists x : t\_sepo, y : t\_sslip \mid \\
& (\text{receive}(C, x) \in \mathtt{H}^{\mathtt{M}} \wedge x.\text{msg.tid} \notin \Theta \;\wedge \\
& x.\text{msg.cid} = \phi_c.\text{id} \wedge \text{vseal}(x.\text{msg}, x.\text{sig}, \phi_c.\text{key}) \wedge \\
& (\text{receive}(N, y) \in \mathtt{H}^{\mathtt{M}} \vee \text{commit}(y, -) \in \mathtt{H}^{\mathtt{N}}) \wedge \text{vseal}(y.\text{msg}, y.\text{sig}, \kappa_n) \;\wedge \\
& y.\text{msg.mid} = \phi_m.\text{id} \wedge x.\text{msg.cid} = y.\text{msg.cid} \wedge x.\text{msg.tid} = y.\text{msg.tid} \;\wedge \\
& \text{cc}(\text{enc}(\gamma, y.\text{msg.key})) = x.\text{msg.cks}),
\end{aligned}
$$

$$
\begin{aligned}
\Phi_{M2} \;=\;& \exists x : t\_sslip, y : t\_str \mid \\
& (\text{receive}(N, x) \in \mathtt{H}^{\mathtt{M}} \vee \text{commit}(x, y) \in \mathtt{H}^{\mathtt{N}}) \wedge x.\text{msg.mid} = \phi_m.\text{id} \;\wedge \\
& \text{vseal}(x.\text{msg}, x.\text{sig}, \kappa_n) \wedge x.\text{msg.res} \wedge x.\text{msg.tid} \notin \Theta \;\wedge \\
& x.\text{msg.tid} = y.\text{msg.sepo.msg.tid} \;\wedge \\
& x.\text{msg.mid} = y.\text{msg.mid} \wedge x.\text{msg.cid} = y.\text{msg.sepo.msg.cid} \;\wedge \\
& y.\text{msg.mid} = \phi_m.\text{id} \wedge y.\text{msg.sepo.msg.cid} = \phi_c.\text{id} \wedge \\
& \text{vseal}(y.\text{msg}, y.\text{sig}, \phi_m.\text{key}) \wedge \text{vseal}(y.\text{msg.sepo.msg}, y.\text{msg.sepo.sig}, \phi_c.\text{key})
\end{aligned}
$$

$$
\begin{aligned}
\Phi_{C1} \;=\;& \exists x_1\ x_2\ x_3 : t \times t\_cks \times t\_tid, y_1 : t\_sslip \mid \\
& \text{receive}(M, x_1\ x_2\ x_3) \in \mathtt{H}^{\mathtt{C}} \wedge \\
& ((\text{receive}(M, y_1) \in \mathtt{H}^{\mathtt{C}} \wedge y_1.\text{msg.tid} = x_3 \wedge \text{dec}(x_1, y_1.\text{msg.key}) = \gamma) \vee \\
& \;(\text{commit}(y_1, -^2) \in \mathtt{H}^{\mathtt{N}} \wedge y_1.\text{msg.cid} = \phi_c.\text{id} \wedge y_1.\text{msg.tid} = x_3 \wedge \\
& \text{dec}(x_1, y_1.\text{msg.key}) = \gamma)),
\end{aligned}
$$

$$
\begin{aligned}
\Phi_{C2} \;=\;& \exists x_1\ x_2\ x_3 : t \times t\_cks \times t\_tid, y : t\_sslip, z : t\_str \mid \\
& \text{receive}(M, x_1\ x_2\ x_3) \in \mathtt{H}^{\mathtt{C}} \wedge \\
& (\text{receive}(M, y) \in \mathtt{H}^{\mathtt{C}} \vee \text{commit}(y, z) \in \mathtt{H}^{\mathtt{N}}) \;\wedge \\
& \text{vseal}(y.\text{msg}, y.\text{sig}, \kappa_n) \wedge y.\text{msg.res} \wedge y.\text{msg.tid} \notin \Theta \;\wedge \\
& y.\text{msg.cid} = \phi_c.\text{id} \wedge y.\text{msg.tid} = x_3 \wedge \\
& y.\text{msg.tid} = z.\text{msg.sepo.msg.tid} \wedge \\
& y.\text{msg.mid} = z.\text{msg.mid} \wedge y.\text{msg.cid} = z.\text{msg.sepo.msg.cid} \;\wedge \\
& z.\text{msg.mid} = \phi_m.\text{id} \wedge z.\text{msg.sepo.msg.cid} = \phi_c.\text{id} \wedge \\
& \text{vseal}(z.\text{msg}, z.\text{sig}, \phi_m.\text{key}) \wedge \text{vseal}(z.\text{msg.sepo.msg}, z.\text{msg.sepo.sig}, \phi_c.\text{key}) \;\wedge \\
& \text{cc}(x_1) = z.\text{msg.sepo.msg.cks} \wedge \text{dec}(x_1, y.\text{msg.key}) \neq \gamma.
\end{aligned}
$$

A few comments about the formalization are in place here. First, we assume Φ_i for both P_C and P_M. Φ_i specifies that neither M nor N has fresh transaction ids at the beginning of protocol executions. Assuming Φ_i allows us to focus on scenarios where no unresolved transactions are pending.

Φ_{M1}, Φ_{M2}, Φ_{C1}, and Φ_{C2} respectively formalize conditions $M1$, $M2$, $C1$, and $C2$.

Finally, transaction slips retained at N are accessible only to arbitrators and principals involved in the corresponding transactions. Thus, C (M) can access a transaction slip m_1 retained at N only if m_1 indicates C (M) as the customer (merchant) of the transaction. In the formalization above, we model "m_1 is accessible to C" and "m_1 is accessible to M" by $m_1.\text{msg.cid} = \phi_c.\text{id}$ and $m_1.\text{msg.mid} = \phi_m.\text{id}$ respectively. These accessibility conditions do not apply if a slip is received by C or M during a protocol execution: in our model, principals can always access messages they receive. This difference is illustrated in Φ_{C1}, where we require the equality $y_1.\text{msg.cid} = \phi_c.\text{id}$ to hold only if y_1 is the copy retained at N.

4.4 Analysis of the Protocol

In this section, we analyze the NetBill protocol under all three deviation modes defined in Section 2.4.3. Subsections 4.4.2, 4.4.3, and 4.4.4 have respectively analyses of the protocol under deception, compliance, and abortion modes. Subsection 4.4.1 has some preliminaries.

The proof strategy used here differs from the one used for Franklin and Reiter's protocol. To analyze Franklin and Reiter's protocol, we resorted to protocol rules whenever possible; trust assumptions were used only when principals may behave non-compliantly. Thus, we had different proofs for compliant and deviant executions. To analyze NetBill, we mainly resort to trust assumptions. Under this approach, we first prove that trustworthy executions satisfy our target properties; then we argue that maximal compliant executions also do, because they are trustworthy. This proof strategy is interesting because it allows us to use the same proof for both compliant and deviant – but trustworthy – executions.

In this section, we present only the main results and their high-level analysis. For completeness, detailed proofs appear in Section 4.5.

4.4.1 Preliminaries

In this subsection, we present two results needed in the rest of the analysis. Lemma 4.2 says that, in NetBill, each protocol step gets instantiated at most once in executions we consider. Its proof is straightforward, and depends on the fact that the protocol rules either have enabling conditions that prevent same types of events from occurring more than once, or can be enabled only once, by an occurrence of an event that cannot occur more than once.

Lemma 4.2 *Let Π be the NetBill protocol, σ be an execution in $E(\Pi)^C \cup E(\Pi)^A \cup E(\Pi)^D$, e_i be an event in σ, and E be an event-template prescribing a protocol step in Π. If e_i is an instance of E, then there does not exist e_j in σ, $i \neq j$, such that e_j is also an instance of E.*

Theorem 4.3 says that all maximal compliant executions of NetBill are trustworthy.

Theorem 4.3 *Let Π be the NetBill protocol and $\mathcal{T}$ be its trust assumptions. Then*

$$\forall \sigma \in E(\Pi)^C, \ \sigma \models^* \mathcal{T}.$$

Proof: Given that N is the only trusted party, $\mathcal{T} = \mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$, and this theorem follows from Lemmas 4.4, 4.5, and 4.6.

□

Lemma 4.4 *Let Π be the NetBill protocol and $\mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$ be the trust assumptions we make of N (as specified in Section 4.3.1). Then*

$$\forall \sigma \in E(\Pi)^C, \ \sigma \models^* T_{N_3}.$$

Lemma 4.5 *Let* Π *be the NetBill protocol and* $\mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$ *be the trust assumptions we make of* N *(as specified in Section 4.3.1). Then*

$$\forall \sigma \in E(\Pi)^C,\ \ \sigma \models^* T_{N_4}.$$

Lemma 4.6 *Let* Π *be the NetBill protocol and* $\mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$ *be the trust assumptions we make of* N *(as specified in Section 4.3.1). Then*

$$\forall \sigma \in E(\Pi)^C,\ \ \sigma \models^* T_{N_5}.$$

4.4.2 Protection under Deception Mode

In this subsection, we analyze the protocol under deception mode. We first analyze it with respect to C-protection, then with respect to M-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, to analyze the protocol with respect to C-protection under deception mode, we need to analyze all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^D_C$. We do so in Prop. 4.10 and 4.7 respectively.

Proposition 4.7 *Let* Π *be the NetBill protocol,* σ *be an execution in* $E(\Pi)^D_C$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P_C = \Phi_i \rightarrow \Box((\Phi_{M1} \vee \Phi_{M2}) \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2}))$ *be* C*'s protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \ \ \textit{implies} \ \ \sigma \models^* P_C.$$

Proof:

1. Let $\sigma = s_0\ e_1\ s_1\ \ldots$ be a trustworthy execution such that $s_0 \models \Phi_i$. We need to prove that $\sigma \models^* \Box(\Phi_{M1} \vee \Phi_{M2} \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2}))$.
2. According to Lemma 4.8, $\sigma \models^* \Box(\Phi_{M1} \rightarrow \Diamond\Phi_{C1})$; and according to Lemma 4.9, $\sigma \models^* \Box(\Phi_{M2} \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2}))$.

$\Box$

Bearing in mind what Φ_{M1}, Φ_{M2}, Φ_{C1} and Φ_{C2} specify (Section 4.3.2), the proof of Prop. 4.7 shows that if M has proof that C received the goods – and is therefore entitled to a payment, then C actually received the goods (Lemma 4.8); and if M has proof that N committed the transaction with a successful result, and did so in response to a valid request – and is therefore entitled to a payment, then C actually received the goods or C can claim it in court (Lemma 4.9).

Lemma 4.8 *Let* Π *be NetBill protocol,* σ *be an execution in* $E(\Pi)^D_C$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P_C = \Phi_i \rightarrow \Box((\Phi_{M1} \vee \Phi_{M2}) \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2}))$ *be* C*'s protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \ \ \textit{implies} \ \ \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{M1} \rightarrow \Diamond\Phi_{C1}).$$

Lemma 4.9 *Let* Π *be NetBill protocol,* σ *be an execution in* $E(\Pi)^D_C$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P_C = \Phi_i \rightarrow \Box((\Phi_{M1} \vee \Phi_{M2}) \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2}))$ *be C's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* T \quad \textit{implies} \quad \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{M2} \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2})).$$

Prop. 4.10 concerns maximal compliant executions. These executions are trustworthy and have compliant local executions of C. Given that our proof for Prop. 4.7 relies on the fact that σ is trustworthy and has a compliant local execution of C, it can be used as is to prove Prop. 4.10.

Proposition 4.10 *Let* Π *be the NetBill protocol,* σ *be an execution in* $E(\Pi)^C$*, and* P_C *be C's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* P_C.$$

From Prop. 4.7 and 4.10, we can derive the following

Corollary 4.11 *NetBill is C-protective under deception mode.*

Next we address M-protection. Like in the analysis of C-protection, we analyze deceptive and compliant executions in turn.

Proposition 4.12 *Let* Π *be the NetBill protocol,* σ *be an execution in* $E(\Pi)^D_M$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P_M = \Phi_i \rightarrow \Box((\Phi_{C1} \vee \Phi_{C2}) \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2}))$ *be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_M.$$

Proof:

1. Let $\sigma = s_0\ e_1\ s_1\ \ldots$ be a trustworthy execution such that $s_0 \models \Phi_i$. We need to prove that $\sigma \models^* \Box(\Phi_{C1} \vee \Phi_{C2} \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2}))$.
2. According to Lemma 4.13, $\sigma \models^* \Box(\Phi_{C1} \rightarrow \Diamond\Phi_{M1})$; and according to Lemma 4.14, $\sigma \models^* \Box(\Phi_{C2} \rightarrow \Diamond\Phi_{M2})$.

$\Box$

Bearing in mind what Φ_{M1}, Φ_{M2}, Φ_{C1} and Φ_{C2} specify (Section 4.3.2), the proof of Proposition 4.12 shows that if C actually received the goods, then M has proof that C received the goods – and is therefore entitled to a payment (Lemma 4.13); and if C can claim the goods in court, then M has proof that N committed the transaction with a successful result, and did so in response to a valid request – and is therefore entitled to a payment (Lemma 4.14).

Lemma 4.13 *Let* Π *be the NetBill protocol,* σ *be an execution in* $E(\Pi)^D_M$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P_M = \Phi_i \rightarrow \Box((\Phi_{C1} \vee \Phi_{C2}) \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2}))$ *be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{C1} \rightarrow \Diamond\Phi_{M1}).$$

Lemma 4.14 *Let* Π *be the NetBill protocol,* σ *be an execution in* $E(\Pi)^D_M$, $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P_M = \Phi_i \rightarrow \Box((\Phi_{C1} \vee \Phi_{C2}) \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2}))$ *be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{C2} \rightarrow \Diamond\Phi_{M2}).$$

Prop. 4.15 concerns maximal compliant executions and can be proven using the proof for Prop. 4.12. The justification is analogous to that for Prop. 4.10.

Proposition 4.15 *Let* Π *be NetBill protocol,* σ *be an execution in* $E(\Pi)^C$*, and* P_M *be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* P_M.$$

From Prop. 4.12 and 4.15, we can derive the following

Corollary 4.16 *NetBill is M-protective under deception mode.*

Theorem 4.17 summarizes the results in this subsection:

Theorem 4.17 *NetBill is all-protective under deception mode.*

Proof: From Cor. 4.11 and 4.16.

□

Discussion

According to our proofs, C's interests are protected under deception mode. Intuitively, M's misbehavior alone cannot compromise C's interests. This can be explained as follows. M is entitled to a payment if and only if

1. M can prove that C received the goods, or
2. M can prove that N committed the transaction with a successful result, and did so in response to a valid request.

We examine each scenario in turn.

1. To prove that C received the goods, M needs 1) a C-signed epo as a proof that C received an encrypted message; 2) a N-signed transaction slip where the decryption key is released; and 3) a proof that the goods is retrievable from the encrypted message and the decryption key. C must have received an encrypted message, otherwise she would not have signed the epo. C also has access to the decryption key – we are certain of this access because the key is enclosed in the transaction slip produced by N, and this slip is accessible to both C and M. With the encrypted message and the decryption key, C can retrieve the goods.

2. To prove that N committed the transaction with a successful result, and did so in response to a valid request, M needs to show a transaction request and a corresponding slip attesting to such a result. Both these messages are provided by N, and are accessible to C. If the decryption key enclosed in the slip can decrypt the encrypted message C received, then C received the goods. Otherwise, either M or N misbehaved and an invalid key was released. C does not receive the goods during the protocol execution in this case, and needs to claim it in court. To do so, she needs to show that decrypting the message received from M with the key released by N does not yield the expected goods. This can be easily accomplished by C because the checksum of the encrypted message – which attests to what message C actually received – can be found in the transaction request, and the key released by N can be found in the transaction slip.

   Note that even though C does not receive the goods on-line in this case, her interests are still protected. NetBill's certified delivery mechanism allows C to prove that she was not given the goods on-line, and claim the goods – or a refund – off-line. Note also that a problem with the key does not always result from M's misbehavior; N can be the one to blame. For example, M may enclose the right key in the transaction request, but N may release something else other than this key in the slip.

Our proof also shows that protection of C's interests depends on N's satisfying the trust assumption T_{N_5}, which says that N can release at most one transaction slip per transaction. The discussion following the proof of Lemma 4.8 in Section 4.5 gives the intuition behind how the violation of this trust assumption can compromise C-protection.

We focus on M-protection next. According to our proofs, M's interests are protected under deception mode. Intuitively C's misbehavior alone cannot compromise M's interests. In what follows, we briefly explain why this is the case.

There are two ways C can receive the goods:

1. she can receive it on-line, or

2. she can claim it in court.

We show that M will be entitled to a payment in either case. We analyze each scenario in turn.

1. If C received the goods on-line, then she must have received an encrypted message and must have had access to a decryption key released in a transaction slip. M has access to the slip C has access to, and would not have enabled its generation – by sending a transaction request to N – unless he had received a C-signed epo attesting that C had received an encrypted message. Using the slip and the C-signed epo, M can prove that C received the goods, and therefore be entitled to a payment.

   Note that in this case C could not have misbehaved.

2. To claim the goods in court, C needs, among other messages, an N-signed transaction slip – attesting to a fresh and successful transaction – and a corresponding valid transaction request. The existence of these two messages, in their turn, entitles M to a payment.

   Note that C could not have misbehaved in this case either.

Our proof also shows that M's interests are protected only if T_{N_4} and T_{N_4} are assumed. The discussion following the proof of Lemma 4.13 in Section 4.5 gives an intuition of how the violation of these trust assumptions can compromise M-protection.

4.4.3 Protection under Compliance Mode

In this subsection, we analyze the protocol under compliance mode. To verify whether the protocol is all-protective under compliance mode, we need to verify whether both $C's$ and M's interests are protected in maximal compliant executions. But these results have already been verified in Subsection 4.4.2. By Prop 4.10 and 4.15,

Theorem 4.18 *NetBill is all-protective under compliance mode.*

Discussion

A sale transaction in NetBill is a transaction (in the database sense [47]) where N secures the decryption key for C and a funds transfer for M. When all principals behave compliantly, the key secured by N is one that will enable C to retrieve the goods, and a funds transfer to which M is entitled actually occurs. Given that the key retained at N is accessible by C, a transaction commit effectively implements an atomic swap of goods and money between C and M. This atomic swapping protects both C's and M's interests.

Note that, under compliance mode, no dispute will arise, and the certified delivery mechanism is not really needed.

4.4.4 Protection under Abortion Mode

In this subsection, we analyze the protocol under abortion mode. As in the two previous subsections, we need to verify whether the protocol is both C-protective and M-protective. We start with C-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, to verify whether Π protects C's interests under abortion mode, we need to examine all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_C$. Here we focus on abortive executions only, since we have analyzed compliant executions (Prop. 4.10) in Section 4.4.2.

Proposition 4.19 *Let Π be the NetBill protocol, σ be an execution in $E(\Pi)^A_C$, $\mathcal{T}$ be the trust assumptions Π makes, and P_C be X's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_C.$$

Prop. 4.19 can be proven using the proof for Prop. 4.7. That proof is applicable here because it relies only on the fact that executions it considers are trustworthy and have compliant local executions of C, both of which are satisfied by executions we consider under Prop. 4.19. A critical condition satisfied by the proof is that it does not depend on M or N taking further steps, a requirement that could be violated by abortive executions.

Jointly, Prop. 4.10 (Section 4.4.2) and 4.19 address C-protection under abortion mode, and derive the following

Corollary 4.20 *NetBill is C-protective under abortion mode.*

Prop. 4.21 addresses protection of M's interests in abortive executions and can be proven using the proof for Prop. 4.12 (Subsection 4.4.2). The reason why that proof is applicable here is analogous to the one given for Prop. 4.19.

Proposition 4.21 *Let Π be the NetBill protocol, σ be an execution in $E(\Pi)^A_M$, $\mathcal{T}$ be the trust assumptions Π makes, and P_M be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_M.$$

Jointly, Prop. 4.15 (Section 4.4.2) and 4.21 address M-protection under abortion mode. They derive the following

Corollary 4.22 *NetBill is M-protective under abortion mode.*

Theorem 4.23 summarizes the results in this subsection.

Theorem 4.23 *NetBill is all-protective under abortion mode.*

Proof: From Cor. 4.20 and 4.22.

□

Discussion

Our proofs show that NetBill is all-protective even if its principals terminate their local executions prematurely. This is not surprising because the exchange of the decryption key and payment happens atomically in NetBill. If a party stops prematurely its own local execution and prevents N from committing the transaction, then neither the decryption key is released nor are funds transferred.

Since the only possible deviations here are premature terminations, the decryption key released by N is the one that will decrypt the encrypted message and produce the goods. Here too, reliability of communication channels is irrelevant. If C does not receive the decryption key at the end of a protocol execution, she can get it from N off-line.

M is entitled to the funds credited into his account because he can prove that C received the goods.

Under abortion mode, no dispute will arise, and the certified delivery mechanism is not really needed.

4.5 Formal Analysis of the Protocol

4.5.1 Preliminaries

Lemma 4.4 *Let Π be the NetBill protocol and $\mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$ be the trust assumptions we make of Z (as specified in Section 4.3.1). Then*

$$\forall \sigma \in E(\Pi)^C, \ \sigma \models^* T_{N_3}.$$

Proof:

1. Let s_i be a state in σ such that $s_i \models \text{commit}(m_1, -) \in \mathtt{H}^{\mathtt{N}}$, for some signed slip m_1. Then $\exists s_j, j \leq i$, such that

$$s_j \models \text{last}(\mathtt{H}^{\mathtt{N}}) = \text{commit}(m_1, -) \quad \text{and} \quad s_{j-1} \models \text{commit}(m_1, -) \notin \mathtt{H}^{\mathtt{N}}.$$

   Examining N's local protocol, we can conclude that, either R_{N_2} or R_{N_3} was applied at s_{j-1}.

2. Independently of which rule was applied, $m_1 = b\ \text{seal}(b, \kappa_n^{-1})$, for a slip b.

3. Given that keyPair$(\kappa_n^{-1}, \kappa_n)$ holds, we can conclude that vseal$(b$, seal(b, κ_n^{-1}), $\kappa_n) =$ TRUE.

□

Lemma 4.5 *Let Π be the NetBill protocol and $\mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$ be the trust assumptions we make of Z (as specified in Section 4.3.1). Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* T_{N_4}.$$

Proof: Commit events are prescribed by rules R_{N_2} and R_{N_3} only, both of which prescribe using the customer id, the merchant id, and the transaction id from the transaction request in the transaction slip.

□

Lemma 4.6 *Let Π be the NetBill protocol and $\mathcal{T}_N = \{T_{N_3}, T_{N_4}, T_{N_5}\}$ be the trust assumptions we make of Z (as specified in Section 4.3.1). Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* T_{N_5}.$$

Proof: The only rule that prescribes a send(M, x) event is R_{N_4}, which requires x to result from a commit event.

□

4.5.2 Protection Under Deception Mode

Lemma 4.8 *Let Π be NetBill protocol, σ be an execution in $E(\Pi)^D_C$, $\mathcal{T}$ be the trust assumptions Π makes, and $P_C = \Phi_i \rightarrow \Box((\Phi_{M1} \vee \Phi_{M2}) \rightarrow \Diamond(\Phi_{C1} \vee \Phi_{C2}))$ be C's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{M1} \rightarrow \Diamond\Phi_{C1}).$$

Proof: Let $\sigma = s_0\ e_1\ s_1\ \ldots$ be such that $\sigma \models^* \mathcal{T}$ and $\sigma \models^* \Phi_i$. And let s_i be a state in σ such that $s_i \models \Phi_{M1}$. The following shows that $s_i \models \Phi_{C1}$.

1. If $s_i \models \Phi_{M1}$, then there exists a signed epo $m_1 : t\_sepo$ such that

   $s_i \models \text{receive}(C, m_1) \in \mathtt{H}^{\mathtt{M}}$,

   which implies that

$$s_i \models \text{send}(M, m_1) \in \mathtt{H}^{\mathtt{C}}.$$

2. Given that "send(M, m_1)" could have resulted only from applying R_{C_3}, we know that there is a message $m_2\ m_3\ m_4 : t \times t\_cks \times t\_tid$ such that

$$s_i \models \text{receive}(M, m_2\ m_3\ m_4) \in \mathtt{H}^{\mathtt{C}}.$$

3. Next, let $m_5 : t\_sslip$ be the signed slip such that

$$s_i \models \text{commit}(m_5, -) \in \mathtt{H}^{\mathtt{N}}.$$

   (Lemma 4.24 allows us to ignore the occurrence of "receive(N, m_5)" at M, which may or may not have taken place.)

   We now need to prove that m_5.msg.cid $= \phi_c$.id (step 4), m_5.msg.tid $= m_4$ (step 5) and dec$(m_2,\ m_5$.msg.key$) = \gamma$ (step 6).

4. From Φ_{M1}, we know that m_5.msg.cid $= m_1$.msg.cid. Given that

   (a) "send$^C(M, m_1)$" could have resulted only from applying R_{C_3}, and

   (b) m_1.msg.cid $= \iota_c$,

   we can conclude that m_5.msg.cid $= m_1$.msg.cid $= \iota_c$. But $\iota_c = \phi_c$.id, according to the initial condition $\mathcal{I}$. Thus, m_5.msg.cid $= \phi_c$.id.

5. Again, from Φ_{M1}, we know that m_5.msg.tid $= m_1$.msg.tid. And according to Lemma 4.26, m_1.msg.tid $= m_4$. Thus, by transitivity, m_5.msg.tid $= m_4$.

6. (a) From $s_i \models \text{send}(M, m_1) \in \mathtt{H}^{\mathtt{C}}$ and Lemma 4.25, we know that cc$(m_2) = m_3$.

   (b) According to Lemma 4.26, $m_3 = m_1$.msg.cks. And using the result from step (a) and transitivity, we know that cc$(m_2) = m_1$.msg.cks.

   (c) Now, according to Φ_{M1}, cc(enc(γ, m_5.msg.key)) $= m_1$.msg.cks. By transitivity (using the result from step (b)), cc$(m_2) =$ cc(enc(γ, m_5.msg.key)); which allows us to conclude that $m_2 =$ enc(γ, m_5.msg.key), and consequently dec(m_2, m_5.msg.key) $= \gamma$.

□

To prove that C received the goods, M needs, among other things, to have received a fresh C-signed epo attesting that C has received an encrypted message with the enclosed checksum. Our proof of Lemma 4.8 shows that C signs this epo only if she actually receives the encrypted message. As for the decryption key, our proof shows that C does not depend on M's forwarding back the transaction slip to obtain it; she can retrieve it directly from N. Given that N controls access to its data through authentication, C can retrieve the transaction slip from N as long as she can prove that she is in fact C.

Our proof also shows that the trust assumption T_{N_5} is critically important here. In fact, if T_{N_5} does not hold, then it is possible for M to prove that C received the goods without C actually receiving it. We show how this is possible below.

If T_{N_5} does not hold, then it is possible for N to send M a slip m different from the one, m', generated when the transaction committed. Let us now assume that m and m' only differ in the value of the decryption key they enclose: let the key enclosed in m be the right key, and the one enclosed in m' be a wrong key. Let us further assume that C does not receive m from M because of a communication link failure. Then, at the end of the execution, Φ_{M1} will hold if M has also received the signed epo from C. Since C does not receive the transaction slip from M, she will get it from N, who will provide the copy m'. But according to our assumptions m'.msg.key cannot decrypt the encrypted message C received. Effectively, C does not receive the goods at the end of the transaction, and Φ_{C1} does not hold.

Note that, in this case, C will not even be able to claim the goods in court, unless N has kept a copy of the transaction request and makes it available to the arbitrator.

Lemma 4.9 *Let Π be NetBill protocol, σ be an execution in $E(\Pi)^D_C$, $\mathcal{T}$ be the trust assumptions Π makes, and $P_C = \Phi_i \to \Box((\Phi_{M1} \vee \Phi_{M2}) \to \Diamond(\Phi_{C1} \vee \Phi_{C2}))$ be C's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \to \sigma \models^* \Phi_i \to \Box(\Phi_{M2} \to \Diamond(\Phi_{C1} \vee \Phi_{C2})).$$

Proof: Let $\sigma = s_0\ e_1\ s_1\ \ldots$ be such that $\sigma \models^* \mathcal{T}$ and $\sigma \models^* \Phi_i$. And let s_i be a state in σ such that $s_i \models \Phi_{M2}$. The following shows that $s_i \models \Phi_{C1} \vee \Phi_{C2}$.

1. If $s_i \models \Phi_{M2}$, then there exists a signed slip m_1 and signed transaction request m_8 such that

   $$s_i \models \text{commit}(m_1, m_8) \in \mathtt{H}^{\mathtt{N}}.$$

   Note that Lemma 4.24 allows us to ignore the occurrence of "receive(N, m_1)" at M, which may or may not have occurred.

2. To be useful to C, m_1 must be accessible by C (both Φ_{C1} and Φ_{C2} specify this condition). That is,

   $$s_i \models m_1.\text{msg.cid} = \phi_c.\text{id}.$$

   But we know, from Φ_{M2} that

   $$s_i \models m_1.\text{msg.cid} = m_8.\text{msg.sepo.msg.cid} \wedge m_8.\text{msg.sepo.msg.cid} = \phi_c.\text{id},$$

   which implies that

   $$s_i \models m_1.\text{msg.cid} = \phi_c.\text{id}.$$

3. Also, according to Φ_{M2}, m_1.msg.tid $\notin \Theta$. Applying contrapositive to the implication regarding transaction ids in Φ_i, we conclude that m_1.msg.tid is not a submessage of messages retrievable from $s_0(\mathtt{MS}^{\mathtt{N}})$; hence, it must have received it from M. That is, there must be a transaction request $m_2 : t\_str$ such that

   $$s_i \models \text{receive}(M, m_2) \in \mathtt{H}^{\mathtt{N}} \quad \wedge \quad m_1.\text{msg.tid} \sqsubseteq m_2.$$

4. Examining the structure of m_2, we conclude that

   $$m_1.\text{msg.tid} = m_2.\text{msg.sepo.msg.tid}.$$

5. Still according to Φ_{M2}, m_8.msg.sepo.msg.tid $= m_1$.msg.tid.

6. Also according to Φ_{M2},

$$s_i \models \text{vseal}(m_8.\text{msg.sepo.msg}, m_8.\text{msg.sepo.sig}, \phi_c.\text{key}), \tag{4.1}$$

which implies that there exists a private key $k : t\_prik$ such that

$$s_i \models m_8.\text{msg.sepo.sig} = \text{seal}(m_8.\text{msg.sepo.msg}, k) \wedge \text{keyPair}(k, \phi_c.\text{key}).$$

7. Now, according to the initial condition $\mathcal{I}$, κ_c^{-1} is such that $\text{keyPair}(\kappa_c^{-1}, \phi_c.\text{key})$; and κ_c^{-1} is kept private to C throughout the execution. Thus, N could not have produced m_8.msg.sepo.sig locally, and must have received it as a submessage of m_2 from M.

8. Applying the reasoning steps we just used to conclude that m_8.msg.sepo.sig could not have been built locally by N, we also conclude that m_8.msg.sepo.sig could not have been built locally by M, and therefore there must exist a signed epo $m_4 : t\_sepo$ such that

$$\begin{cases} s_i \models \text{receive}(C, m_4) \in \mathtt{H}^{\mathtt{M}} \wedge \\ m_8.\text{msg.sepo.sig} \sqsubseteq m_4. \end{cases} \tag{4.2}$$

Now, given that messages must be sent before they are received, we have

$$s_i \models \text{send}(M, m_4) \in \mathtt{H}^{\mathtt{C}}. \tag{4.3}$$

By Lemma 4.25, we conclude that there exists $m_5\ m_6\ m_7 : t \times t\_cks \times t\_tid$ such that

$$s_i \models \text{receive}(M, m_5\ m_6\ m_7) \in \mathtt{H}^{\mathtt{C}}, \tag{4.4}$$

(which is one of the conjuncts in both Φ_{C1} and Φ_{C2}).

9. Also, unless m_1 has the transaction id C is focused on in this transaction (m_7), it may be ignored by C as useless data. This justifies why we require that

$$m_1.\text{msg.tid} = m_7,$$

as is specified in both Φ_{C1} and Φ_{C2}. We prove this last equality below:

   (a) From expression 4.2 (step 8) and the structure of m_4, we know that

   $$m_8.\text{msg.sepo.sig} = m_4.\text{sig}.$$

   (b) Using straightforward steps omitted here, we can prove that

   $$s_i \models \text{vseal}(m_4.\text{msg}, m_4.\text{sig}, \phi_c.\text{key}),$$

   which, in conjunction with expression 4.1 (step 6), allows us to prove that

   $$m_8.\text{msg.sepo} = m_4. \tag{4.5}$$

   (c) Applying Lemma 4.26 to expressions 4.3 and 4.4 (step 8), we know that

   $$m_4.\text{msg.tid} = m_7. \tag{4.6}$$

(d) From Φ_{M2}, we know that

$$m_1.\text{msg.tid} = m_8.\text{msg.sepo.msg.tid}. \tag{4.7}$$

Now, applying transitivity to expressions 4.7, 4.5, and 4.6, we conclude that
$m_1.\text{msg.tid} = m_7$.

10. If $\text{dec}(m_5,\ m_1.\text{msg.key}) = \gamma$, then $s_i \models \Phi_{C1}$. Otherwise, we need to prove that

    $\text{cc}(m_5) = m_8.\text{msg.sepo.msg.cks}$,

    which is what we do in the remainder of this proof.

11. Proving that $\text{cc}(m_5) = m_8.\text{msg.sepo.msg.cks}$:

    (a) According to Lemma 4.25, $\text{cc}(m_5) = m_6$.

    (b) According to Lemma 4.26, $m_6 = m_4.\text{msg.cks}$. And, by transitivity, $\text{cc}(m_5) = m_4.\text{msg.cks}$.

    (c) We know, from step 8, that
    $m_8.\text{msg.sepo.sig} \sqsubseteq m_4$.
    Examining the structure of m_4, we can conclude that $m_8.\text{msg.sepo.sig} = m_4.\text{sig}$.

    (d) And, by straightforward steps omitted here, we can prove that
    $\text{vseal}(m_4.\text{msg},\ m_4.\text{sig},\ \phi_c.\text{key})$.

    (e) But we also know (from step 6) that
    $\text{vseal}(m_8.\text{msg.sepo.msg},\ m_8.\text{msg.sepo.sig},\ \phi_c.\text{key})$.

    (f) Using (c)-(e), we can then conclude that
    $m_4.\text{msg} = m_8.\text{msg.sepo.msg}$.

    (g) Finally, combining (b) and (f), we obtain
    $\text{cc}(m_5) = m_8.\text{msg.sepo.msg.cks}$.

12. All the other conjuncts in Φ_{C2} can be obtained as they are from Φ_{M2}.

□

Recall from Subsection 4.3.2 that M needs two messages to prove that N committed a new transaction with a successful result, and did so in response to a valid request: a fresh N-signed slip and a corresponding transaction request. In her turn, C needs an encrypted message and a decryption key enclosed in a transaction slip to obtain the goods. Our proof of Lemma 4.9 shows that: 1) both the slip and the transaction request that service M also service C; and 2) C must have received an encrypted message if a fresh slip was generated by N. That is, if M can claim his entitlement to a payment by showing that a transaction – that C and himself agreed on – was carried through, then C will obtain the goods.

We focus in turn on smaller steps below.

The slip and the transaction request that service M also service C because N makes them available to both parties involved in the transaction.

C must have received an encrypted message because 1) N only generates a signed slip attesting to a successful result if N receives a fresh transaction request correctly signed by C and M; 2) fresh transaction requests could not be generated unless C signs an epo in the current execution; and 3) C only signs an epo if she receives an incorrupt encrypted message. Note that the assumption (Φ_i) that there is no pending transactions at the beginning of the protocol execution is important here.

With the encrypted message and the decryption key enclosed in the transaction slip, C may or may not be able to retrieve the goods. If the encrypted message can be decrypted by the decryption key, then C obtains the goods at the end of the protocol execution, and no dispute is needed. Otherwise, C will need to claim the goods in court. C can do so by showing to the arbitrator that the encrypted message she received is consistent with the checksum sanctioned by both C and M, but cannot be decrypted using the decryption key provided by N. Our proof shows that the checksum enclosed in the transaction request is in fact the checksum against which C checked the integrity of the encrypted message. (The checksum is signed by C, and because the signed message includes a fresh transaction id, the signature cannot be a replay.)

Note that even though we use Lemma 4.24, and ultimately T_{N_5}, in this proof, they are not strictly necessary here. That is, Lemma 4.9 can be proven even if T_{N_5} does not hold in σ. Its assumption, however, makes the proof simpler. Intuitively, T_{N_5} is not needed here because we are under scenarios where transaction requests are kept by N and made available to the arbitrator. The availability of transaction requests makes the role of transaction slips, and their uniqueness, less critical. Effectively, no matter what key N encloses in the slip, we can always resort to the transaction request, and either get the key directly from there, if the key enclosed in the request is the right decryption key, or demand it from M otherwise.

The following three lemmas appear in our proof of Lemma 4.9. Lemma 4.24 says that the slip M receives from N is the one generated with the transaction commit and retained at N. Its proof consists of straightforward backchaining steps and relies on the fact that 1) N sends the slip only after it commits the transaction, and 2) N is trusted to send M the slip generated with the transaction commit (T_{N_5}).

Lemma 4.24 *Let* Π *be NetBill protocol,* σ *be an execution in* $E(\Pi)_C^D$, $\mathcal{T}$ *be the trust assumptions* Π *makes. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \forall x : t\_sslip, \Box(\text{receive}(N, x) \in \mathtt{H}^{\mathtt{M}} \rightarrow \text{commit}(x, -) \in \mathtt{H}^{\mathtt{N}}).$$

Proof: Let σ be a trustworthy execution, s_i be a state in σ, and $m_1 : t\_sslip$ be a signed slip such that

$s_i \models \text{receive}(N, m_1) \in \mathtt{H}^{\mathtt{M}}$.

Given that messages must be sent before they are received, we know that

$s_i \models \text{send}(M, m_1) \in \mathtt{H}^{\mathtt{N}}$.

By T_{N_5}, we can conclude that $s_i \models \text{commit}(m_1, -) \in \mathtt{H}^{\mathtt{N}}$.

□

Lemma 4.25 says that if C sends a signed epo to M, then C received an encrypted message, and the message is consistent with the accompanying checksum. Our proof relies on the fact that C behaves compliantly.

Lemma 4.25 *Let Π be NetBill protocol and σ be an execution in $E(\Pi)^D_C$. Then*

$$\begin{aligned}\sigma \quad \models^* \quad & \Box(\exists x_1 : t\_sepo \mid \text{send}(M, x_1) \in \mathtt{H}^{\mathtt{C}} \rightarrow \\ & \quad \exists x_2\ x_3\ x_4 : t \times t\_cks \times t\_tid \mid (\text{receive}(M, x_2\ x_3\ x_4) \in \mathtt{H}^{\mathtt{C}} \wedge cc(x_2) = x_3)).\end{aligned}$$

Proof: Let s_i be a state in σ and m_1: t_sepo be a message such that

$$s_i \models \text{send}(M, m_1) \in \mathtt{H}^{\mathtt{C}}.$$

"send(M, m_1)" could have resulted only from an application of R_{C_3}. Thus, $\exists s_j, j \leq i$, such that

$$s_j \models \text{last}(\mathtt{H}^{\mathtt{C}}) = \text{send}(M, m_1)$$

and

$$\begin{aligned}s_{j-1} \quad \models \quad & \text{send}(M, m_1) \notin \mathtt{H}^{\mathtt{C}} \wedge \\ & \exists x_2\ x_3\ x_4 : t \times t\_cks \times t\_tid \mid \text{last}(\mathtt{H}^{\mathtt{C}}) = \text{receive}(M, x_2\ x_3\ x_4) \wedge \text{cc}(x_2) = x_3.\end{aligned}$$

□

Lemma 4.26 says that the checksum and the transaction id enclosed in the signed epo C sends to M are the ones C received from M. Its proof also relies on the fact that C behaves compliantly.

Lemma 4.26 *Let Π be NetBill protocol and σ be an execution in $E(\Pi)^D_C$. Then*

$$\begin{aligned}\sigma \quad \models^* \quad & \forall x_1\ x_2\ x_3 : t \times t\_cks \times t\_tid, x_4 : t\_sepo, \\ & \Box(\text{receive}(M, x_1\ x_2\ x_3) \in \mathtt{H}^{\mathtt{C}} \wedge \text{send}(M, x_4) \in \mathtt{H}^{\mathtt{C}} \rightarrow \\ & \quad x_2 = x_4.\text{msg.cks} \wedge x_3 = x_4.\text{msg.tid}).\end{aligned}$$

Proof:

1. Let s_i be a state in σ and $m_1, \ldots, m_4$ be messages such that

$$s_i \quad \models \quad \text{receive}(M, m_1\ m_2\ m_3) \in \mathtt{H}^{\mathtt{C}} \wedge \text{send}(M, m_4) \in \mathtt{H}^{\mathtt{C}}.$$

2. Examining C's local protocol, we conclude that "send(M, m_4)" could have resulted only from an application of R_{C_3}. This implies that, if $s_j, j < i$, is the state at which R_{C_3} is applied, i.e.,

$$s_j \models \text{send}(M, m_4) \notin \mathtt{H}^{\mathtt{C}} \quad \wedge \quad s_{j+1} \models \text{send}(M, m_4) \in \mathtt{H}^{\mathtt{C}},$$

then

$$\begin{aligned}& s_j \models \text{receive}(m_1, m_2, m_3) \in \mathtt{H}^{\mathtt{C}}, \text{ and} \\ & m_4.\text{msg.cks} = m_2 \wedge m_4.\text{msg.tid} = m_3.\end{aligned}$$

□

Lemma 4.13 *Let Π be the NetBill protocol, σ be an execution in $E(\Pi)^D_M$, $\mathcal{T}$ be the trust assumptions Π makes, and $P_M = \Phi_i \rightarrow \Box((\Phi_{C1} \vee \Phi_{C2}) \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2}))$ be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{C1} \rightarrow \Diamond\Phi_{M1}).$$

Proof: Let $\sigma = s_0\ e_1\ s_1\ \dots$ be such that $\sigma \models^* \mathcal{T}$ and $\sigma \models^* \Phi_i$, and let s_i be a state in σ such that $s_i \models \Phi_{C1}$. The following shows that $s_i \models \Phi_{M1}$.

1. If $s_i \models \Phi_{C1}$, then there exists messages $m_1\ m_2\ m_3 : t \times t\_cks \times t\_tid$ and $m_4 : t\_sslip$ such that

$$\begin{aligned} s_i \quad \models \quad & \text{receive}(M, m_1\ m_2\ m_3) \in \mathtt{H}^{\mathtt{C}} \wedge \\ & \text{commit}(m_4, -) \in \mathtt{H}^{\mathtt{N}} \wedge \\ & \text{dec}(m_1, m_4.\text{msg.key}) = \gamma. \end{aligned}$$

   For simplicity of the proof, we have applied Lemma 4.27 here.

To prove that $s_i \models \Phi_{M1}$, we first need to show that M received a signed epo from C (steps 2, 3, and 4):

2. Given that

$$s_i \models \text{commit}(m_4, -) \in \mathtt{H}^{\mathtt{N}} \text{ (from step 1)}$$

   we can conclude that either R_{N_2} or R_{N_3} has been applied. In either case, $\exists j, j < i$, and a message m_5: *t_str* such that

$$s_j \models \text{last}(\mathtt{H}^{\mathtt{N}}) = \text{receive}(M, m_5).$$

3. Since messages must be sent before they are received, we have

$$s_j \models \text{send}(N, m_5) \in \mathtt{H}^{\mathtt{M}}$$

4. "send(N, m_5)" could have only resulted from an application of R_{M_6}, which implies that $\exists k, k \leq j$, and a signed epo m_6: *t_sepo*, such that

$$\begin{aligned} & s_k \models \text{last}(\mathtt{H}^{\mathtt{M}}) = \text{send}(N, m_5), \text{ and} \\ & s_{k-1} \models \text{last}(\mathtt{H}^{\mathtt{M}}) = \text{receive}(C, m_6). \end{aligned}$$

Next we show that m_6 is C-signed (steps 5, 6, and 7):

5. Going back to step 2, an application of R_{N_2} or R_{N_3} actually allows us to draw more conclusions. For instance, the signed epo included in the transaction request must be signed by C:

$$s_j \models m_5.\text{msg.sepo.msg.cid} = \phi_c.\text{id} \wedge \text{vseal}(m_5.\text{msg.sepo.msg}, m_5.\text{msg.sepo.sig}, \phi_c.\text{key})$$

6. In step 4, an application of R_{M_6} also leads to more conclusions. For example, the signed epo enclosed in the transaction request m_5 is the one received from C:

$$m_5.\text{msg.sepo} = m_6.$$

7. From steps 5 and 6, we can conclude that

$$\text{vseal}(m_6.\text{msg}, m_6.\text{sig}, \phi_c.\text{key})$$

   holds, that is, m_6 is C-signed.

We also need to prove that m_6's transaction id is fresh, i.e., m_6.msg.tid $\notin \Theta$ (steps 8 and 9):

8. Going back again to step 4, we list other conditions that s_{k-1} satisfies. For example, there are messages $m'_1 : t, m'_2 : t\_cks$ and $m'_3 : t\_tid$, such that

$$s_{k-1} \models \text{send}(C, m'_1\ m'_2\ m'_3) \in \mathtt{H}^{\mathtt{M}} \wedge m'_2 = m_6.\text{msg.cks} \wedge m'_3 = m_6.\text{msg.tid}.$$

9. "send$(C,\ m'_1\ m'_2\ m'_3)$" could have resulted only from an application of R_{M_4}, which implies that m'_3 was generated by a "new(Θ, m'_3)" event. Thus, $m'_3 \notin \Theta$.

   Given that $m'_3 = m_6$.msg.tid (step 8), we can conclude that m_6.msg.tid $\notin \Theta$.

Next, we show that the signed slip generated during the transaction processing is signed by N (step x) and has the required values for its customer id, merchant id and transaction id (steps 11, 12, and 13), i.e.,

$$m_4.\text{msg.mid} = \phi_m.\text{id} \wedge m_4.\text{msg.cid} = m_6.\text{msg.cid} \wedge m_4.\text{msg.tid} = m_6.\text{msg.tid}$$

10. To show that the slip is N-signed is to show that

$$\text{vseal}(m_4.\text{msg},\ m_4.\text{sig},\ \kappa_n).$$

   But from step 1, we know that $s_i \models \text{commit}(m_4, -) \in \mathtt{H}^{\mathtt{N}}$. And according to T_{N_3}, vseal$(m_4$.msg, m_4.sig, $\kappa_n)$.

11. Going back to step 2, an application of R_{N_2} or R_{N_3} allows us to conclude that the transaction request m_5 is signed by M:

$$s_j \models m_5.\text{msg.mid} = \phi_m.\text{id} \wedge \text{vseal}(m_5.\text{msg}, m_5.\text{sig}, \phi_m.\text{key}).$$

   And T_{N_4} says that the transaction slip generated during the transaction processing uses the value of mid from m_5 in its mid field. That is:

$$m_4.\text{msg.mid} = m_5.\text{msg.mid}.$$

   By transitivity, we conclude that m_4.msg.mid $= \phi_m$.id.

12. From Φ_{C1}, we know that m_4.msg.cid $= \phi_c$.id; from step 5, we know that m_5.msg.sepo.msg.cid $= \phi_c$.id; and from step 6 we know that m_5.msg.sepo $= m_6$. By transitivity, m_4.msg.cid $= m_6$.msg.cid.

13. From Φ_{C1}, we know that m_4.msg.tid $= m_3$. From step 8, $m'_3 = m_6$.msg.tid. Given that "send$(C, m'_1\ m'_2\ m'_3)$" takes place only once in M's execution, we can conclude that $m_1 = m'_1, m_1 = m'_1, m_1 = m'_1$. By transitivity, m_4.msg.tid $= m_6$.msg.tid.

Finally (step 13), we prove that cc(enc$(\gamma,\ m_4$.msg.key)) $= m_6$.msg.cks.

13. From Φ_{C1}, we know that dec$(m_1, m_4$.msg.key$) = \gamma$, which implies that $m_1 =$ enc$(\gamma, m_4$.msg.key).

   Applying cryptographic checksum on both sides of the equality, we get

$$\text{cc}(m_1) = \text{cc}(\text{enc}(\gamma, m_4.\text{msg.key})).$$

   Going back to step 9, the fact that "send(C, $m'_1\ m'_2\ m'_3$)" could have resulted only from an application of R_{M_4} also implies that

$$cc(m_1') = m_2'.$$

Given that $m_1 = m_1'$, we can conclude by transitivity that $m_2' = \mathrm{cc}(\mathrm{enc}(\gamma, m_4.\mathrm{msg.key}))$. But from step 8, we know that $m_2' = m_6.\mathrm{msg.cks}$, which allows us to finally conclude that

$$\mathrm{cc}(\mathrm{enc}(\gamma, m_4.\mathrm{msg.key})) = m_6.\mathrm{msg.cks}.$$

□

Our proof of Lemma 4.13 shows that if C can obtain the goods from the encrypted message she received from M and the key released in the transaction slip she has access to, then M can prove that C received an encrypted message and a key that would allow her to retrieve the goods.

We focus in turn on smaller steps below.

If C has access to a transaction slip, then N must have committed a transaction, which must have been requested by M. M must have received a signed epo from C, otherwise he would not have submitted the transaction request.

This epo must be fresh and must have been signed by C. It must be fresh because M would not have proceeded with the protocol otherwise. And it must have been signed by C, because N would not have committed the transaction otherwise. With this fresh, C-signed epo, M can prove that C received an encrypted message during the protocol execution.

Next, we show that the slip from which C extracts the key is accessible to M. This is so because the transaction request submitted by M identifies him as the merchant (remember that M behaves compliantly here), and N faithfully transcribes this information to the only slip it generates per transaction. Note that, to be useful in court, this slip needs to be N-signed.

Finally, through the value of the checksum enclosed in the C-signed epo, M can show that C can retrieve the goods from the encrypted message C received and the key released in the slip. He does so by showing that the encrypted message and the result of encrypting the goods with the key have the same checksum.

Our proof relies on three trust assumptions: T_{N_4}, T_{N_4} and T_{N_5}. T_{N_4} is critical because it makes N's intermediation actions non-repudiable. In the context of this proof, non-repudiation allows M to use the slip N released to prove that the decryption key was in fact released by N and is accessible to C. If T_{N_4} did not hold, then N might release a slip that is not signed, which would not help M in court.

T_{N_4} is critical because it guarantees that the values of customer id and merchant id enclosed in the transaction slip are exactly those from the transaction request. These values are important here because they determine whether the slip C has access to is also accessible to M. Remember that M needs a N-signed slip and a corresponding C-signed epo to make his case. If T_{N_4} does not hold, then N could generate a slip that is accessible to C, but not to M. If this slip also encloses the decryption key, C would effectively get the goods, without M's being able to prove it. At a higher level of abstraction, this depicts a scenario where N cheats by giving the decryption key to C behind M's back.

T_{N_5} is not critical for Lemma 4.13 to hold. It only simplifies the proof. If T_{N_5} does not hold, then N might send M a copy of the slip that is different from the one retained at N. In the worst case, the copy sent to M will not enclose the key, while the one retained at N will. Under this scenario, C can get the goods, and M may not even know about it. But M can always uncover these cases by checking whether the slip he received is a copy of the one retained at N.

Lemma 4.27 was used in our proof of Lemma 4.13. It says that in trustworthy executions where M behaves compliantly, the slip received by C is the one retained at N. Its proof relies on 1) the fact that M behaves compliantly – and thus forwards the message he receives from N to C, and 2) the assumption (T_{N_5}) that N is trusted to send M only the slip produced with the transaction commit and retained at N.

Lemma 4.27 *Let Π be the NetBill protocol, σ be an execution in $E(\Pi)^D_M$, and $\mathcal{T}$ be the trust assumptions Π makes. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \forall x : t\_sslip, \Box(\textit{receive}(M, x) \in \mathtt{H}^{\mathtt{C}} \rightarrow \text{commit}(x, -) \in \mathtt{H}^{\mathtt{N}}).$$

Proof:

1. Let σ be a trustworthy execution, s_i be a state in σ, and $m_1 : t\_sslip$ be a signed slip such that

   $$s_i \models \text{receive}(M, m_1) \in \mathtt{H}^{\mathtt{C}}.$$

   Since messages must be sent before they are received, we know that

   $$s_i \models \text{send}(C, m_1) \in \mathtt{H}^{\mathtt{M}}.$$

2. An occurrence of "send(C, m_1)" could have resulted only from an application of R_{M_8}; thus

   $$s_i \models \text{receive}(N, m_1) \in \mathtt{H}^{\mathtt{M}}.$$

3. From $s_i \models \text{receive}(N, m_1) \in \mathtt{H}^{\mathtt{M}}$, we conclude that $s_i \models \text{send}(M, m_1) \in \mathtt{H}^{\mathtt{N}}$. And according to T_{N_5}, $s_i \models \text{commit}(m_1, -) \in \mathtt{H}^{\mathtt{N}}$.

□

The proof of Lemma 4.14 is straightforward. It relies on the fact that both the transaction request and the transaction slip needed by M to claim a payment are needed by C to claim the goods in court. Note that even though we assume trustworthy executions, the proof does not need this assumption.

Lemma 4.14 *Let Π be the NetBill protocol, σ be an execution in $E(\Pi)^D_M$, $\mathcal{T}$ be the trust assumptions Π makes, and $P_M = \Phi_i \rightarrow \Box((\Phi_{C1} \vee \Phi_{C2}) \rightarrow \Diamond(\Phi_{M1} \vee \Phi_{M2}))$ be M's protection property as specified in Section 4.3.2. Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* \Phi_i \rightarrow \Box(\Phi_{C2} \rightarrow \Diamond\Phi_{M2}).$$

Proof: Let $\sigma = s_0\ e_1\ s_1\ \ldots$ be such that $\sigma \models^* \mathcal{T}$ and $\sigma \models^* \Phi_i$, and let s_i be a state in σ such that $s_i \models \Phi_{C2}$. The following shows that $s_i \models \Phi_{M2}$.

If $s_i \models \Phi_{C2}$, then there exists a message $m_1 : t\_sslip$ and $m_2 : t\_str$ such that

$$\begin{array}{lll} s_i & \models & \text{commit}(m_1, m_2) \in \mathtt{H}^{\mathtt{N}} \wedge m_1.\text{msg.mid} = \phi_m.\text{id} \wedge \\ & & \text{vseal}(m_1.\text{msg}, m_1.\text{sig}, \kappa_n) \wedge m_1.\text{msg.res} \wedge m_1.\text{msg.tid} \notin \Theta \wedge \\ & & m_1.\text{msg.tid} = m_2.\text{msg.sepo.msg.tid} \wedge \\ & & m_1.\text{msg.mid} = m_2.\text{msg.mid} \wedge m_1.\text{msg.cid} = m_2.\text{msg.sepo.msg.cid} \wedge \\ & & m_2.\text{msg.mid} = \phi_m.\text{id} \wedge m_2.\text{msg.sepo.msg.cid} = \phi_c.\text{id} \wedge \\ & & \text{vseal}(m_2.\text{msg}, m_2.\text{sig}, \phi_m.\text{key}) \wedge \text{vseal}(m_2.\text{msg.sepo.msg}, m_2.\text{msg.sepo.sig}, \phi_c.\text{key}) \end{array} \tag{4.8}$$

But this is exactly what Φ_{M2} specifies.

$\square$

4.6 Summary and Conclusion

In this section, we summarize the findings of our analysis and conclude with some insights.

4.6.1 Summary

We start with the summary in Table 4.1. All-protection is guaranteed in entries with a $\surd$; certified delivery is needed in entries with a $*$. The table shows that NetBill is all-protective under all deviation modes, and the protection does not rely on reliable communication channels. It also shows that the certified delivery mechanism is needed only under deception mode.

	compliance	abortion	deception
Channel Failures	$\surd$	$\surd$	$\surd$ $*$
No Channel Failures	$\surd$	$\surd$	$\surd$ $*$

Table 4.1: Summary table.

In what follows, we focus on each of the entries in the table.

First, it is not surprising that all-protection can be guaranteed in NetBill without reliable communication channels. This is so because neither payment nor decryption key delivery, which concludes goods delivery, requires reliable communication among the parties. Payments are not communication-dependent in NetBill because they are funds transfers that occur at the NetBill server. Key deliveries are not communication dependent because, once released, keys are retained at N, which makes them always accessible, even when communication channels are down.

NetBill is all-protective under compliance mode because payments and releases of decryption keys happen atomically when transactions commit. Since everyone behaves compliantly, funds are actually transferred during the transaction processing, and the key is in fact the right one for retrieving the goods. Under compliance mode, no dispute will arise.

For the same reasons, NetBill is all-protective under abortion mode. This is not surprising. Premature terminations of local executions can happen either before a transaction commits, and prevent it from committing; or after it has committed. In the first case, no money changes hands and no key is released; in the second case, the exchange has taken place. All-protection holds in either case. Here too, no dispute will arise.

Under deception mode, all-protection can no longer be guaranteed by transaction processing alone. Certified delivery and a few trust assumptions are needed. For example, if M deceives, and encloses a bogus key in the transaction request, then the mere atomic swapping of the key with the money does not protect C's interests. In such cases, goods cannot be retrieved using the released key, and C will need to claim them in court. Unless C can non-repudiably show that she was not delivered the goods, the payment will not bring her anything in return. NetBill's certified delivery mechanism plays a critical role in protecting C's interests here because it enables C to prove what exactly she was delivered. Similarly, N can deceive. For example, it can release the key enclosed in the transaction request without transferring funds from C to M. In such cases, M will need to complain, and claim the payment off-line. Here again, NetBill's certified delivery mechanism can protect M's interests. It enables M to prove that he actually delivered the goods, and is therefore entitled to payment. Unless M can non-repudiably prove that he actually delivered the goods, he may risk releasing the goods for nothing.

These examples show that much can happen after a protocol execution has finished, and off-line arbitrations can remedy anomalies that happen on-line.

Not all deceptions can be remedied by certified delivery however; some cannot be easily resolved and can compromise C-protection or M-protection. They are deceptions that violate the trust assumptions.

During our modeling and analysis exercise, we identified a number of trust assumptions for N, the NetBill server. These assumptions specify what N must not do in order to guarantee NetBill's all-protection.

First, it must not generate unsigned transaction slips (T_{N_3}). Unsigned slips are repudiable, and therefore not useful as proofs in court. When releasing unsigned slips, N is effectively providing C with means of retrieving the goods, if the right decryption key is enclosed, without assuring M means for proving this in court.

Second, N must not generate slips with corrupted merchant ids (T_{N_4}). Transaction slips are vehicles through which decryption keys are released. And slips with corrupted merchant ids are typically unavailable to M. Thus, by releasing slips with corrupted merchant ids, N can be effectively giving keys to C behind M's back.

Finally, N must not generate more than one slip per transaction (T_{N_5}). A transaction slip provides part of the record of what happens with a transaction. When a transaction has conflicting records, disputes are hard to resolve. In practice, the existence of conflicting records may favor those that do not need much else to make their cases. For example, giving M a slip with the correct decryption key can entitle him to a payment, while giving C a slip with a wrong decryption key alone will not be enough for her to claim the goods in court.

Our analysis shows that NetBill is C-protective only if N satisfies T_{N_3} and T_{N_4}; and is M-protective only if N satisfies T_{N_5}.

4.6.2 Conclusion and Insights

As a protocol that supports dispute resolution, NetBill brings extra challenge to the specification of its protection properties. Since the protocol offers its participants ways of correcting unfair outcomes of an execution after the fact (this is the notion of open protection), the specifications need to take into account not only on-line exchanges of money and goods between the customer and the merchant, but also how disputes are resolved. In the context of a protocol where non-repudiable messages collected by the intermediary are used to resolve disputes, it does not suffice to know what messages each of the main parties has access to; we also need to know what the intermediary retained and is willing to make available. This explains the complexity of both C's and M's protection properties.

Even though NetBill is all-protective, it only guarantees that C receives what M set out to release, which may not be what C expects to receive. Unlike Franklin and Reiter's protocol, NetBill does not provide means for C to check whether what M is willing to release is indeed what she is after. Thus, false advertisements would need to be dealt with outside the protocol.

NetBill treats customers and merchants differently. It is, in some sense, stricter with C than with M. For a transaction to commit, C could not have misbehaved (assuming that N does not violate the trust assumptions and M behaves compliantly). M however can misbehave: he may not forward the slip back to C or he may release a bogus key. His misbehaviors are dealt with after the fact.

Lastly, the set of trust assumptions we identified seems to have captured the essence of being an intermediary in NetBill, even though it is unexpectedly small. Basically, the intermediary is entrusted to handle accounts and keys honestly (T_{N_1} and T_{N_2}); to provide unique (T_{N_5}) and non-repudiable (T_{N_3}) proofs of what happens on-line; and to allow M to learn what really happens with the transaction (T_{N_4}) so that keys are not given to C without his knowledge.

Even though it may seem counter-intuitive at first, NetBill's all-protection does not depend on N's releasing keys that M sends it, or N's retaining transaction requests.

Chapter 5

Brands's Off-line Cash with Observers Protocol

Our last case study is Brands's untraceable, off-line cash with observers protocol [14], henceforth referred to as Brands's protocol. This case study is the most interesting for three reasons. First, we have to derive an abstract version of the protocol from its purely mathematical formulation [14]. This exercise is challenging because the protocol is very complex (not only compared to the other two protocols, but in absolute terms), and it is not immediately clear how abstract we should model the protocol.

Second, there has not been, to our knowledge, analysis of electronic cash protocols with respect to protection of individuals' interests. This means that there are no standard protection properties for these protocols, and we have to propose them afresh.

Finally, our analysis reveals a number of weaknesses in the protocol. Some of the weaknesses are well-known and far from subtle; for example, that ecoins can get lost during transmission. Others, however, are quite subtle, and have not been found before. For example, a payee can either abort or deceive, and effectively make the payer's money disappear, even if communication links are reliable. Also, withdrawers can deceive, and withdraw perfectly valid coins that they can spend, but will be neither accepted for deposit nor usable for tracing the identity of the withdrawer. Note that our analysis results only apply to the protocol as it appears in [14]. In fact, none of the weaknesses we point out in this chapter can be found in Brands' later designs [15, 16]. (Readers interested in Brands's reaction to our analysis and his later versions of the protocol should go to Section 5.10.)

In this chapter, we specify the protocol in our specification formalism (Section 2.3), and analyze it with respect to protection of individuals' interests using our framework. Since we are interested in protection of all parties equally, we analyze it with respect to all-protection.

The rest of this chapter is structured as follows. In Section 5.1, we introduce basic concepts associated with electronic cash systems. In Section 5.2, we introduce the protocol and assumptions made by Brands. In Section 5.3, we present an abstract formulation of the mathematical building blocks used in it. In Section 5.4, we formalize the protocol and its protection properties. In Section 5.5, we prove some preliminary results needed in the analysis of the protocol. In Sections 5.6 - 5.8, we analyze the withdrawal, the payment, and the deposit subprotocols respectively. In Section 5.9, we conclude and discuss insights gained through our exercise. Finally, in Section 5.10, we include Brands's comments on our analysis.

5.1 Electronic Cash Systems – Preliminaries

Electronic cash [25, 22, 23, 50, 62, 21, 24, 29, 14, 41, 40, 73, 72, 37], henceforth called *ecash*, was first proposed by D. Chaum [21] as an alternative to conventional, account-based electronic payment schemes. Unlike in account-based schemes, where money exists solely as account balances, in ecash systems money can exist in the form of digital tokens as well. This feature is a plus from the point of view of privacy: while payments are traceable in account-based schemes, they are untraceable in ecash systems. Payments are traceable in account-based schemes because they are direct fund transfers, and account managers know the origin (payer) and the destination (payee) of a transfer. Payments in ecash systems are untraceable because they are token transfers from one party to another, and the payer's identity cannot be inferred from the token used in the payment.

In what follows, we use 'ecash' to refer to untraceable, token-based payment schemes and the notion of 'electronic money' in general. We reserve 'ecoin' (electronic coin) to refer to the token itself. When there is no ambiguity, we also use 'coin' instead of 'ecoin'.

Ecash is not a perfect counterpart of physical cash. Due to intrinsic properties of the electronic medium, ecash has characteristics of its own, and needs to rely on auxiliary concepts and mechanisms to achieve a comparable level of security. For example, copying ecoins is trivial, and unless additional measures are taken, one can double-spend a coin, by keeping a copy of a coin one has already spent and spending it again later.

Double-spending is a problem because only one of all copies of a coin (the one accepted by the bank) can be converted back to physical cash. Thus, effectively, among all payees of a double-spent coin, only one will get paid, even though all of them received the coin. Moreover, since ecoins are untraceable, the remaining payees may be left without any recourse.

To counteract double-spending, some ecash systems [25] take an *on-line* approach, in which the bank keeps the status (spent or not-yet-spent) of each of the coins in the system, and the payees consult the bank on-line before accepting a coin. Double-spendings are impossible in on-line systems [30].

Brands's protocol is *off-line*. Here the bank is not consulted before coins are accepted. To deal with double-spending, Brands uses a two-step approach. The first step relies on *observers* and tries to restrain double-spending. Observers are tamper-resistant smartcards attached to the payers' systems, and are entrusted with authorizing coin spendings. They restrain double-spendings because they are made to give only one spending authorization per coin. Since smartcards are not tamper-proof, they can be compromised, and authorize multiple spendings, effectively allowing a coin to be double-spent.

The second step works remedially: once the preventive restraining step fails and a double-spending occurs, the protocol tries to identify the double-spender. The intent is that the identification will enable those that could not deposit the coin to go after the double-spender, and claim a real payment. (Note, however, that malfunctioning observers do not always authorize multiple spendings; they may not authorize any spending at all, and effectively lead to losses of coins.)

Ecoins themselves are untraceable; thus the identification mechanism must rely on something else. In Brands's protocol, it relies on rights certificates, which we introduce in Section 5.1.2.

In the rest of this section, we introduce other concepts used in Brands's protocol.

5.1.1 Ecoins and Blind Signature Protocols

Typically implemented as digitally-signed messages, ecoins can be abstractly represented as pairs (u, s) of *substrate* u and *signature* s. Substrates are arbitrary messages; they differ from one coin to another. Signatures are digital signatures of coin issuer on the substrates.

To achieve untraceability, substrates are typically signed using blind signature schemes. In blind signature schemes, a signature s signed on a message u can be used to derive a signature s' for a different message u', derived from u. Using blind signature schemes, one can effectively obtain signatures for messages that the signer did not directly sign.

Fig. 5.1 depicts an abstract blind signature protocol, where a requester R obtains from a signer S a blind signature for message m. In the protocol, blind is a function whose inverse is unblind;

$$\begin{array}{lll} 1. & R \rightarrow S: & bm = blind(m, b) \\ 2. & S \rightarrow R: & s = seal(bm, k^{-1}) \end{array}$$

The various symbols denote:

$$\begin{array}{rl} m: & \text{the message to be signed;} \\ b: & \text{a randomly generated blinding factor;} \\ k^{-1}: & S\text{'s signing key.} \end{array}$$

Figure 5.1: An abstract blind signature protocol.

b is a randomly generated *blinding factor*; seal is a signature function; k^{-1} is S's signing key; and the functions are such that unblind(s, b) equals seal(m, k^{-1}). The protocol shows that, to obtain a blind signature for m, R needs to send S a blinded message (bm) derived from m. The signature for m can then be obtained by unblinding the signature returned by S. Effectively, m gets signed, even though S does not concretely sign it.

Blind signatures guarantee untraceability because S cannot link the signature seal(m, k^{-1}) with R even if it keeps a record of all signatures it issued with their respective requesters.

5.1.2 Ecoins and Rights Certificates

Ecash systems that rely solely on ecoins have a number of problems. For example, they allow double-spending and theft (through message hijacking or eavesdropping).

One possible solution for these problems is to make ecoins themselves worthless unless accompanied by *rights certificates*. Rights certificates are coin-specific and payee-specific, and are used by a payer to give the rights of a coin to a payee. If the withdrawer of a coin (a.k.a. payer) were the only one able to provide rights certificates for it, then no one other than the withdrawer would be able to spend the coin, and potential hijackers that seize the coin, while it is in transit between the bank and the withdrawer, would have no way of profiting from the hijacking. If, in addition, the withdrawer were able to provide at most one rights certificate per coin, then double-spending would not be possible. Finally, if the bank deposits a coin only when it is accompanied by a rights certificate indicating the depositor (a.k.a. payee) as a lawful holder of the coin, then coin hijackings between a payer and a payee, or between a depositor and the bank would also be non-profitable.

Brands takes exactly this approach; his implementation of rights certificates appears in Section 5.2.3.

In addition to transferring rights, counteracting theft, and restraining double-spendings, rights certificates play a pivotal role in tracing the identity of double-spenders in Brands's protocol. In Brands's protocol, the number of rights certificates a withdrawer is able to issue per coin is controlled by the observer. If the observer is compromised, double-spending becomes a problem. Rights certificates, however, embed identification information about the withdrawer of a coin. They are designed in such a way that a single certificate would not yield any information about the identity of the withdrawer, but two different certificates of a coin would allow the bank to reveal this information. When a coin is double-spent, each of the payees receives a different certificate. When more than one of them submits the coin for deposit, the bank can reveal the identity of the double-spender using the accompanying certificates.

5.2 Brands's Protocol – An Introduction

Brands's protocol requires four principals: a payer P, a payee E, a bank B, and an observer O. In the rest of this chapter, P will be referred to as she, E as he, and B and O as it. B keeps accounts for both P and E, can issue coins, and can receive them back in deposits. In this protocol, B issues coins of only one denomination. P can withdraw coins from her account and use them in payments. E can receive coins from P and deposit them into his account. O, in its turn, participates in the generation of substrates and rights certificates, and plays a pivotal role in restraining P from double-spending.

In this section, we introduce Brands's protocol, which consists of three subprotocols: withdrawal, payment, and deposit. The version presented here is an abstraction of the one given in [14], and will be used in our analysis. Our presentation here is informal. See Section 5.4 for a formal specification of the protocol, and Section 5.3 for an abstract, but precise, definition of the cryptographic functions and predicates we use. We do not transcribe the concrete protocol here; interested readers should refer to [14] directly. In what follows, we first present the setup of the system, then describe each of the subprotocols.

Notation

In Brands's protocol, principals hold secrets and show knowledge of their secrets throughout the protocol. Since secrets are not to be revealed, Brands has secret holders show their knowledge of a secret by showing the value of the application of a one-way function to the secret. The protocol thus uses a number of different publicly known and agreed upon one-way functions whose sole purpose is to mask secrets.

We use lowercase Greek letters to denote secrets, and $\bar{\cdot}$ (overlining) to denote their one-way functions. Note that we do not distinguish different one-way functions; we represent them all by $\bar{\cdot}$. Secrets have subscripts indicating to whom they belong. Sometimes, integer subscripts are used to number them. For example, θ_{2_o} denotes a secret belonged to the observer, while $\overline{\theta_p}$ denotes a one-way function of a secret belonged to the payer.

Some one-way functions have standard, distinguished names. 'blind' (without the quote marks) is an example.

5.2.1 The Setup of the System

B has two databases: an account database for information about account holders; and a deposit database for deposit transcripts. B also has a public key pair, k^{-1} (private key) and k (public key), which it uses for signature generation and verification. Both P and E also have k. Finally B has a collection of smartcards.

To open an account, P generates a random number θ_p and its corresponding one-way function $\overline{\theta_p}$. θ_p is kept private at P, while $\overline{\theta_p}$ is sent to B. B checks $\overline{\theta_p}$ for uniqueness before accepting it as P's account number.

Once B accepts $\overline{\theta_p}$ as P's account number, it chooses a smartcard O, and initializes it by writing a randomly generated number θ_o to its ROM. B then creates an entry $(i_p, \overline{\theta_p}, \theta_o)$, where i_p is P's id, in the account database; computes $\overline{\theta_o}$; and gives both O and $\overline{\theta_o}$ to P. O is from now on P's observer. Fig. 5.2 shows what each of the principals has once the system is set up. Note that E's entry in B's account database has less information than P's entry has. This difference reflects the fact that the system can accommodate two types of accounts: those from which coins can be withdrawn (P's account) and those that cannot (E's account).

$$B : \begin{cases} \bullet \text{ an account database including the two entries below:} \\ \quad * \; i_e, \\ \quad * \; i_p,\ \overline{\theta_p},\ \theta_o; \\ \bullet \text{ a deposit database;} \\ \bullet \text{ a public key pair } (k^{-1}, k). \end{cases}$$

$$P : \begin{cases} \bullet \; B\text{'s public key } k; \\ \bullet \; \theta_p; \\ \bullet \; \overline{\theta_o} \; ; \\ \bullet \; i_p. \end{cases}$$

$$E : \begin{cases} \bullet \; B\text{'s public key } k; \\ \bullet \; i_e. \end{cases}$$

$$O : \begin{cases} \bullet \; \theta_o. \end{cases}$$

The various symbols denote:

$i_p,\ i_e$: P's id and E's id, respectively;
θ_p : P's account secret;
θ_o : O's secret.

Figure 5.2: What the principals have once the system is set up.

Brands makes a few assumptions for the setup of the system. He assumes that smartcards' ROMs cannot be read directly, except by O, and can be written only by B. He also assumes that initial exchanges described above can be securely and correctly carried out between P and B.

In the protocol, θ_p is P's account secret. It appears in all the coins withdrawn from P's account, and is the information revealed by B if P double-spends. θ_o is the observer secret. Like θ_p, it appears in all the coins withdrawn from P's account. Note that θ_o is known only to B and O. P cannot learn the value of θ_o because she cannot read O's ROM and cannot retrieve θ_o from $\overline{\theta_o}$.

5.2.2 The Withdrawal Protocol

Brands's withdrawal protocol (Fig. 5.3) is in its core (steps 3 and 4) a blind signature protocol (Fig. 5.1.1) where B blindly signs the substrate (u_1, u_2) of the coin being withdrawn. Since (u_1, u_2) is being blindly signed, it is blinded by a randomly generated blinding factor β_p before it is sent for signature (step 3), and the signature itself can be obtained by unblinding the value *sl* returned by B (step 4). At the end of the protocol, $((u_1, u_2)$, unblind(*sl*, β_p)) is the coin withdrawn by P.

$$
\begin{array}{lll}
1. & P \rightarrow O: & \textit{secret-request} \\
2. & O \rightarrow P: & \overline{\nu_o} \\
3. & P \rightarrow B: & bu = blind((u_1, u_2), \beta_p) \\
 & & \text{where } \left\{ \begin{array}{l} u_1 = subs_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{ and} \\ u_2 = subs_2(\nu_{1p}, \nu_{2p}, \overline{\theta_o}, \overline{\nu_o}), \end{array} \right. \\
4. & B \rightarrow P: & sl = seal(bu, \overline{\theta_p}, \overline{\theta_o}, k^{-1})
\end{array}
$$

The various symbols denote:

secret-request:	a pre-defined message for requesting a secret from O;
$\nu_o, \nu_{1p}, \nu_{2p}$:	randomly generated secrets;
θ_p, θ_o :	P's account secret and O's secret, respectively;
β_p :	a randomly generated blinding factor;
k^{-1} :	B's private key.

Figure 5.3: The withdrawal protocol.

Besides this core structure, two other features deserve attention in this protocol. First, the substrate (u_1, u_2), where

$$
\left\{ \begin{array}{l} u_1 = subs_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{ and} \\ u_2 = subs_2(\nu_{1p}, \nu_{2p}, \overline{\theta_o}, \overline{\nu_o}), \end{array} \right.
$$

is a pair whose elements are functions (subs_1 and subs_2) of two sets of values. The first set is account specific and consists of $\overline{\theta_p}$ and $\overline{\theta_o}$, which are available to P once the system is set up. They encode information about respectively which account the coin is withdrawn from and who is the observer. The second set is coin-specific and consists of ν_{1p}, ν_{2p}, and $\overline{\nu_o}$, which are all randomly generated during the withdrawal protocol. While ν_{1p} and ν_{2p} are locally generated by P, $\overline{\nu_o}$ is provided by O under P's request (steps 1 and 2). O's participation towards building the substrate enables it to control the spending of the coin later in the payment protocol. We explain the control mechanism in Section 5.2.3. subs_1 and subs_2 are two one-way functions named to suggest that they produce respectively the first and the second components of a substrate.

Second, the signature function (seal) used by B differs from the basic version shown in Fig. 5.1: sl is a function not only of blinded message bu and B's private key k^{-1}, but also of two other values $\overline{\theta_p}$ and $\overline{\theta_o}$, that B retrieves from P's entry in the account database. This signature function is such that a signature is valid only if it embeds the right private key, and $\overline{\theta_p}$ and $\overline{\theta_o}$ are exactly those embedded in bu. Thus, unless P embeds the correct account and observer information into the substrate, the signature returned by B will not be valid, and P will not have obtained a coin. This effectively protects the system from P's misbehaving. $\overline{\theta_p}$ and $\overline{\theta_o}$, now embedded in the substrate, will later appear in the coin's rights certificates, and be used to trace a double-spent coin back to P. By ensuring that only correct "origin" information is embedded in valid coins, Brands seeks to ensure that double-spenders will always be unmistakenly identified.

Finally, B debits P's account before sending sl to P. This is not shown in Fig 5.3 because we omit internal actions in this informal presentation.

5.2.3 The Payment Protocol

Brands's payment protocol (Fig. 5.4) consists of two nested protocols. In the outer one (steps 1, 2 and 5), P gives E the coin c (whose substrate is as specified in Section 5.2.2) and an accompanying rights certificate rc_p; in the inner one (steps 3 and 4), P obtains the help she needs from O to generate the certificate.

$$\begin{array}{lll}
1. & P \rightarrow E: & c = ((u_1, u_2), us) \\
2. & E \rightarrow P: & ch = chal(i_e, t) \\
3. & P \rightarrow O: & ch \\
4. & O \rightarrow P: & rc_o = pcert(ch, \theta_o, \nu_o) \\
5. & P \rightarrow E: & rc_p = rcert(rc_o, \theta_p, \nu_{1p}, \nu_{2p})
\end{array}$$

The various symbols denote:

$$\begin{array}{rl}
c: & \text{a coin whose substrate } (u_1, u_2) \text{ is as specified in Section 5.2.2;} \\
i_e: & E\text{'s id;} \\
t: & \text{a challenge seed;} \\
ch: & \text{a challenge;} \\
\nu_o, \nu_{1p}, \nu_{2p}: & \text{secrets found in } (u_1, u_2)\text{;} \\
\theta_p, \theta_o: & P\text{'s account secret and } O\text{'s secret, respectively;} \\
rc_o: & \text{a protocertificate;} \\
rc_p: & \text{a rights certificate.}
\end{array}$$

Figure 5.4: The payment protocol.

As shown in Fig. 5.4, rc_p is a response to the challenges ch presented by E. It embeds not only the challenge itself, but also the secrets embedded in the coin's substrate (note that rc_p is a function of θ_p, ν_{1p}, ν_{2p}, and rc_o, which, in its turn, is a function of ch, θ_o and ν_o). rc_p is payee-specific because it is a function of ch, which is a function of the payee's id i_e. rc_p is coin-specific because it embeds coin-specific secrets – ν_o, ν_{1p}, and ν_{2p} – found in the coins' substrate.

Note that P is unable to provide the certificate herself: the substrate embeds two secrets – θ_o and ν_o – whose values are known only to O. The inner protocol now comes into play: it allows P to request and receive O's help with these values. P never learns the values of these secrets, however, because O embeds them in rc_o – henceforth called a *protocertificate* – rather than providing them in clear. Control of these secrets gives O control of generation of certificates, and ultimately of coin spending. Because O is designed to erase ν_o once it has been used towards generating a protocertificate, P will not be able to provide a second rights certificate for the coin. The coin therefore cannot be double-spent. If O, however, is tampered with and does not erase ν_o, it will successfully provide P with another protocertificate the next time it is presented a challenge. This would enable P to provide a second rights certificate for the coin and double-spend it. Finally, no one other than P and O (jointly) can provide a certificate: P is the only one that knows the value of the remaining secrets (θ_p, ν_{1p}, and ν_{2p}).

In Fig. 5.4, function *chal*'s second argument t is a locally unique value that differs from one transaction to another. This transaction-specificity serves two purposes. First, it prevents replays of both rc_o and rc_p. Second, it enables traceability of double-spenders: the deposit protocol (Section 5.2.4) hinges on having different rights certificates for different spendings of a coin to reveal the identity of a double-spender. Note that we use t, instead of a lowercase Greek letter (reserved for values that are to be kept secret), to denote this value. This indicates that t does not need to be randomly generated or kept secret. In fact, the date and the time of a payment are a perfect candidate for t. Since these t's are used to generate challenges, we call them *challenge seeds*.

Before accepting the payment, E carries out two verifications. First, he verifies whether the coin is valid, i.e., whether it has a valid signature from B. Then he verifies whether the rights certificate is valid, i.e., whether rc_p embeds the challenge he sent and the secrets embedded in the coin's substrate (u_1, u_2). Note that E can verify the consistency between the secrets embedded in (u_1, u_2) with those embedded in rc_p only *indirectly*, because subs_1, subs_2, and cert_p are all one-way functions, and E learns neither the actual secrets embedded in (u_1, u_2) nor those embedded in rc_p.

The same observation applies to P's verification of rc_o's validity, where P checks the value of rc_o against those of $\overline{\theta_o}$ and $\overline{\nu_o}$.

Indirect verifications are customary in cryptography. Public key signature verification is an example where the verifier can check the validity of a signature indirectly, without learning the value of the signing private key.

5.2.4 The Deposit Protocol

To deposit a coin c (Fig. 5.5), E sends B the coin itself, an accompanying rights certificate rc_p, and the challenge seed t he used to generate the challenge embedded in rc_p.

Before accepting c for deposit, B carries out three verifications. First, it verifies whether c has B's signature on it. Second, it verifies whether rc_p is a valid rights certificate giving E the rights to c. To do this verification, B first generates, using E's id and t, the challenge that was supposedly used by E in the payment protocol. Using this challenge and c's substrate, B can now verify the validity of rc_p. Like E in the payment protocol, B can do this verification only indirectly, because it knows neither the secrets embedded in the substrate nor those embedded in rc_p. Lastly, B verifies whether c has been deposited before. For this verification, B searches its deposit database. If c has been deposited before, the first component u_1 of its substrate will be found in the database.

$$1.\quad E \rightarrow B: \quad c,\ t,\ rc_p$$

The various symbols denote:

c : a coin;
rc_p : c's accompanying rights certificate;
t : challenge seed that appear in rc_p.

Figure 5.5: The deposit protocol.

If either of the first two verifications fail (c is not a valid coin or rc_p is not a valid rights certificate giving E the rights to c), B simply ignores this deposit attempt.

If the first two verifications succeed, but the third one fails (c is a valid coin, has not been deposited before, and E has the rights to it), B accepts the coin for deposit. In a deposit, B credits E's account and creates a new entry in the deposit database. The new entry consists of u_1, the challenge $chal(i_e, t)$, and rc_p. Note that the second component u_2 of the substrate is left out of the database for efficiency reasons. The assumption here is that two substrates that share their first components also share their second.

If all three verifications succeed (E was given the rights to a valid coin that has been deposited before), then there is an entry in B's deposit database for c, and a fraud is flagged. Either P or E can be the fraudulent party. If the challenge in the database entry equals the one from the current deposit attempt, E is trying to double-deposit the coin; otherwise, P has responded to two different challenges, and has double-spent the coin. In the first case, B does not take further action, because E's attempted fraud has been detected and no one was hurt. In the second case, E is left with a coin that cannot be deposited; effectively he was not paid.

Here is where traceability of double-spenders comes to the rescue. Using the rights certificate found in the database entry submitted when c was first deposited and the one submitted with the current deposit attempt, B can retrieve θ_p, derive $\overline{\theta_p}$, and obtain P's identity i_p. These two rights certificates are necessarily different because challenges they embed are different.

5.2.5 Assumptions

Brands's paper is specific in how it addresses various assumptions needed by the protocol. While it includes a good amount of discussion about cryptographic assumptions that the protocol relies on, it fails to provide a satisfactory account of system-related assumptions. In fact, there is no mention as to whether communication links can fail, or how exactly different participants can fail or misbehave.

One can infer that he implicitly assumes reliable communication links; however, there does not seem to be a consistent model of the protocol participants themselves. For example, P's attacks to the protocol seem restricted to double-spending and analyzing messages from B and O in order to forge coins. E's attacks seem restricted to double-depositing. Conversely, being able to spend a coin she withdraws and assuring her privacy (that her payments will not be traceable unless she double-spends) seem to be P's main concerns. Whether or not she could lose a coin while trying to spend it seems irrelevant. Except for double-spending and double-depositing, all the attacks he considers are cryptographic in nature.

We attribute this inconsistency to the nature of the work. Brands is focused on using novel cryptographic primitives to formulate a novel protocol. It is understandable that he is primarily concerned with cryptography. However, his neglect of system- and protocol-level issues has impact on the protocol's overall security, as it will become apparent from our analysis.

5.3 Abstract Formulation of the Building Blocks

5.3.1 Cryptographic Building Blocks

Brands's protocol [14] uses a number of cryptographic building blocks. They are abstractly represented in our introduction to the protocol in Section 5.2. In this subsection, we present an abstract formulation of these building blocks. We start with an informal introduction; the formalization appears in Def. 5.1. In what follows, $\doteq$ denotes conversion.

All arithmetic in Brands's protocol is performed in groups of prime order for which polynomial-time algorithms are known for multiplication, inversion, equality test, membership test, and random selection of elements. We use type *t_num* to represent such groups in our abstraction, and refer to their elements simply as numbers.

One-way functions are widely used in Brands's protocol to generate secret-embedding messages. Some of these messages have special meanings, while others do not. One-way functions that generate the former appear below; those that generate the latter are represented simply as $\bar{\cdot}$.

The first two functions, subs_1 and subs_2, are used to generate respectively the first and the second components of coin substrates. subs_1 takes three arguments, while subs_2 takes four. The arguments need to satisfy the following relation: Let $n_1, \ldots, n_7$ be seven numbers,

$$(\mathrm{subs}_1(n_1, \ldots, n_3), \mathrm{subs}_2(n_4, \ldots, n_7)) \text{ is a substrate if and only if } \left\{ \begin{array}{l} n_2 = n_4, \text{and} \\ n_3 = n_6. \end{array} \right.$$

The other numbers do not need to relate to each other in any specific way. Some of them need to be drawn from specific sources (Section 5.2.2), however.

To generate a coin, substrates are blindly signed using public key cryptography. In public key crypto-systems, private and public keys come in pairs. If k^{-1} and k are respectively the private key and the public key of a key pair, then they satisfy the predicate keyPair. That is

$$\mathrm{keyPair}(k^{-1}, k) = \mathrm{true}.$$

Given a substrate $u = (\mathrm{subs}_1(n_1, -, n_2),\ \mathrm{subs}_2(-, -, n_2, -))$, where "-" stands for arguments whose actual values are irrelevant in the context, and a private key k^{-1},

$$\mathrm{seal}(u, n_1, n_2, k^{-1})$$

denotes the signature of k^{-1} on u. Note that seal is an non-standard signing function here; the signature is a function not only of the substrate and the private key, but also of two numbers embedded in the substrate. Signatures can be verified using public keys; predicate vseal models signature verification. Given a substrate u, a signature s, and public key k,

$$\mathrm{vseal}(u, s, k) = \mathrm{true}$$

if and only if s is a signature of k^{-1} on u, and k^{-1} is the private counterpart of k.

Messages can be blinded. Given a message m and a number b,

$$\text{blind}(m, b)$$

denotes the result of blinding m with b. The corresponding unblinding function,

$$\text{unblind}(m, b)$$

denotes the result of unblinding m with b. For all blinded messages $m' = \text{blind}(m, b)$ and numbers b', $\text{unblind}(\text{blind}(m, b),\ b') \doteq m$, if $b = b'$. That is,

$$\text{unblind}(\text{blind}(m, b), b) \doteq m.$$

Unblinding commutes with signing. Given a message $\text{seal}(u, n_1, n_2, k^{-1})$ and a number b,

$$\text{unblind}(\text{seal}(u, n_1, n_2, k^{-1}), b) \doteq \text{seal}(\text{unblind}(u, b), n_1, n_2, k^{-1}).$$

Note that both blind and unblind are polymorphic functions.

To generate challenges for rights certificates, we need a type that can provide different values for different transactions. Let *t_cs* denote such a type. Then, given a principal id *id* and an element c from *t_cs*,

$$\text{chal}(id, c)$$

denotes a challenge.

To generate a rights certificate, one needs a protocertificate. Given a challenge *ch* and two numbers n_1 and n_2,

$$\text{pcert}(ch, n_1, n_2)$$

is a protocertificate. One can indirectly check the values embedded in a protocertificate; predicate vcert_o models this check. Given a protocertificate p, a challenge *ch*, and two numbers n_1 and n_2,

$$\text{vpcert}(p, ch, n_1, n_2) = \text{true}$$

if and only if p is consistent with *ch*, n_1, and n_2. The consistency condition is given in Def. 5.1.

Given a protocertificate p and numbers n_1, n_2, and n_3,

$$\text{rcert}(p, n_1, n_2, n_3)$$

denotes a rights certificate. Rights certificates can be checked against a challenge and a substrate. Predicate vcert_p models this check: given a rights certificate *rc*, a challenge *ch*, and a substrate u,

$$\text{vrcert}(rc, ch, u) = \text{true}$$

if and only if the values embedded in *rc* are consistent with *ch* and those embedded in u.

Finally, two different rights certificates of a coin can be used to reveal the account secret embedded in the coin. We use reveal_sec to denote the secret revelation function. Let *rc* and *rc'* be two different rights certificates of a coin c, and n be a number,

$$\text{reveal\_sec}(rc, rc', n)$$

corresponds to the account secret embedded in c, if n is the observer secret embedded in c.

In Def. 5.1, we list the cryptographic building blocks as well as their abstract properties:

Definition 5.1 *Cryptographic building blocks*

1. *Numbers have type t_num, i.e., if m is a number, then m: t_num;*
2. *Component one of substrates have type* t_subs_1*, i.e., if m is a value of function* $subs_1$*, then m:* t_subs_1*;*
3. *Component two of substrates have type* t_subs_2*, i.e., if m is a value of function* $subs_2$*, then m:* t_subs_2*;*
4. *Private keys have type t_prik, i.e., if m is a private key, then m: t_prik;*
5. *Public keys have type t_pubk, i.e., if m is a public key, then m: t_pubk;*
6. *Signatures have type t_seal, i.e., if m is a value of function seal, then m: t_seal;*
7. *Principal ids have type t_id; i.e., if m is a principal id, then m: t_id;*
8. *Challenge seeds have type t_cs, i.e., if m is a challenge seed, then m: t_cs;*
9. *Challenges have type t_chal, i.e., if m is a value of function chal, then m: t_chal;*
10. *Protocertificates have type t_pcert, i.e., if m is a value of function pcert, then m: t_pcert;*
11. *Rights certificates have type t_rcert, i.e., if m is a value of function rcert, then m: t_rcert;*
12. *Functions and predicates:*
    - $\bar{\cdot}$: $t\_num \rightarrow t\_num$;
    - $subs_1$: $t\_num \times t\_num \times t\_num \rightarrow t\_subs_1$;
    - $subs_2$: $t\_num \times t\_num \times t\_num \times t\_num \rightarrow t\_subs_2$;
    - *keyPair*: $t\_prik \times t\_pubk \rightarrow$ *boolean;*
    - *seal*: $(t\_subs_1 \times t\_subs_2) \times t\_num \times t\_num \times t\_prik \rightarrow t\_seal$;
    - *vseal*: $(t\_subs_1 \times t\_subs_2) \times t\_seal \times t\_pubk \rightarrow$ *boolean;*
    - *blind*: $x \times t\_num \rightarrow x$, *where x is a type variable;*
    - *unblind*: $x \times t\_num \rightarrow x$, *where x is a type variable;*
    - *chal*: $t\_id \times t\_cs \rightarrow t\_chal$;
    - *pcert*: $t\_chal \times t\_num \times t\_num \rightarrow t\_pcert$;
    - *rcert*: $t\_pcert \times t\_num \times t\_num \times t\_num \rightarrow t\_rcert$;
    - *vpcert*: $t\_pcert \times t\_chal \times t\_num \times t\_num \rightarrow$ *boolean;*
    - *vrcert*: $t\_rcert \times t\_chal \times (t\_subs_1 \times t\_subs_2) \rightarrow$ *boolean;*
    - *reveal_sec*: $t\_rcert \times t\_rcert \times t\_num \rightarrow t\_num$;
13. *Conversion rules:*
    - $unblind(blind(m,\ b),\ b') = m$, *if and only if* $b = b'$;

- $unblind(seal(u,\ n_1,\ n_2,\ k^{-1}),\ b)\ =\ seal(unblind(u,\ b),\ n_1,\ n_2,\ k^{-1});$

- $vseal(u,\ s,\ k)\ = \begin{cases} true,\ if\ \exists n_1, n_2 : t\_num\ and\ k^{-1}\text{: } t\_prik\ such\ that \\ \qquad u = (subs_1(n_1, -, n_2), subs_2(-, -, n_2, -)) \wedge \\ \qquad s = seal(u, n_1, n_2, k^{-1})\ \wedge\ keyPair(k^{-1}, k); \\ false,\ otherwise. \end{cases}$

  $vseal(u,\ s,\ k)$ is true if and only if numbers n_1 and n_2 embedded in the signature are also embedded in the substrate, and the signing key is the private counterpart of the verification key.

- $vpcert(p, ch, n_1, n_2)\ = \begin{cases} true,\ if\ \exists n'_1, n'_2 : t\_num\ such\ that \\ \qquad \overline{n'_1} = n_1\ \wedge\ \overline{n'_2} = n_2 \wedge\ p = pcert(ch, n'_1, n'_2); \\ false,\ otherwise. \end{cases}$

  $vpcert(p,\ ch,\ n_1,\ n_2)$ is true if and only if p embeds ch and the secrets embedded in n_1 and n_2.

- $vrcert(r, ch, u)\ = \begin{cases} true,\ if\ \exists n_1, \ldots, n_5 : t\_num\ such\ that \\ \qquad r = rcert(pcert(ch, n_1, n_2), n_3, n_4, n_5)\ \wedge \\ \qquad u = (subs_1(\overline{n_3}, n_4, \overline{n_1}), subs_2(n_4, n_5, \overline{n_1}, \overline{n_2})); \\ false,\ otherwise. \end{cases}$

  $vrcert(r,\ ch,\ u)$ is true if and only if r embeds ch and all the secrets embedded in u.

- *Let*

  $$r = rcert(pcert(ch, n_1, n_2), n_3, n_4, n_5)\ \ and\ \ r' = rcert(pcert(ch', n_1, n_2), n_3, n_4, n_5)$$

  be two rights certificates.

  $$reveal\_sec(r, r', n_1) = n_3,\ if\ and\ only\ if\ ch \neq ch'.$$

  One can reveal the coin's account secret (n_3) if and only if one has two different rights certificates of a coin (note that both r and r' refer to the same substrate, but are responses to different challenges) and the coin's observer secret (n_1).

5.3.2 Product Types and Projection Functions

Some composite messages have special semantics in Brands's protocol. For example, substrates are ordered pairs, and each entry in the bank's deposit database consists of three components: component one of a substrate, a challenge, and a rights certificate. For conciseness, we define product types for these composites and projection functions for these types.

Substrates: $t\_subs\ =\ t\_subs_1\ \times\ t\_subs_2$

$$\text{If } m = (m_1, m_2) \text{ has type } t\_subs\text{, then } \begin{cases} \text{comp}_1(m) = m_1,\ \text{and} \\ \text{comp}_2(m) = m_2. \end{cases}$$

Account database entries: $t\_acc\ =\ t\_id\ \times\ t\_num\ \times\ t\_num$

$$\text{If } m = (m_1, m_2, m_3) \text{ has type } t\_acc\text{, then } \begin{cases} \text{id}(m) = m_1, \\ \text{accn}(m) = m_2,\ \text{and} \\ \text{osecret}(m) = m_3. \end{cases}$$

Deposit database entries: $t\_deposit = t\_subs_1 \times t\_cs \times t\_rcert$

$$\text{If } m = (m_1, m_2, m_3) \text{ has type } t\_deposit\text{, then } \begin{cases} \text{subsComp}(m) = m_1, \\ \text{cs}(m) = m_2, \text{ and} \\ \text{rc}(m) = m_3. \end{cases}$$

For readability, we syntactically distinguish applications of cryptographic functions and projection functions. Applications of cryptographic functions are denoted in a standard way: $f(a_1, \ldots, a_n)$, where f is the function and a_i, $i = 1, \ldots, n$, are the arguments. Applications of projection functions, on the other hand, are denoted as $a.f$, where a is the argument and f is the function. For example, if m is a substrate, then we use blind(m, b) to denote the result of blinding m with b, and $m.\text{subs}_1$ to denote m's first component.

5.4 Formalizing the Protocol and the Protection Properties

In this section, we specify Brands's protocol using our specification formalism (Section 2.3), and give concrete specifications of protection properties as applied to this protocol.

Different participants have different interests to preserve in different ecash transactions. As a result, the notion of protection should be applied to withdrawal, payment, and deposit transactions separately. This enables us to tackle each subprotocol in turn. In each of the following subsections, we first specify the subprotocol, then the protection properties for that subprotocol.

In our framework, part of specifying a protocol is making its trust assumptions explicit. For Brands's protocol to work, both B and O need to be trusted in different ways. Brands does not address this issue uniformly. In [14], he explicitly assumes that O generates and erases secrets as prescribed; however, he does not make trust assumptions about B explicit. Thus, the trust assumptions that appear in our specification of the protocol are results of our inference, rather than transcriptions of what appear in [14].

To specify the protection properties, we first formulate them abstractly; we then make them concrete for Brands's protocol; finally, we formalize the concrete refinements. Identifying protection properties for each of ecash subprotocols has turned out to be challenging. While there have been a number of studies [3, 42, 89] on fairness (and protection properties indirectly) for exchange protocols, no similar studies have been done with ecash protocols.

In what follows, w, x, y, z are variables; identifiers in SMALL CAP are constants; and those in *slanted* fonts are placeholders for more complex messages.

5.4.1 The Withdrawal Protocol

Principals of the Protocol

$\mathcal{P} = \{B, P, O\}$

Initial Conditions

Let k_b^{-1} and k_b be respectively a private and a public key; i_p be a principal id; θ_p and θ_o be two numbers; and ac_p be an account database entry. The initial condition

$$\begin{aligned}\mathcal{I}^w = \ & \{k_b^{-1}, ac_p\} \subseteq \texttt{MS}^{\texttt{B}} \wedge \{i_p, \theta_p, \overline{\theta_o}, k_b\} \subseteq \texttt{MS}^{\texttt{P}} \wedge \{\theta_o\} \subseteq \texttt{MS}^{\texttt{O}} \wedge \\ & \text{shareable}(k_b^{-1}, \{B\}) \wedge \text{keyPair}(k_b^{-1}, k_b) \wedge \\ & \text{shareable}(ac_p, \{B\}) \wedge ac_p.\text{id} = i_p \wedge ac_p.\text{accn} = \overline{\theta_p} \wedge ac_p.\text{osecret} = \theta_o \wedge \\ & \text{shareable}(\theta_p, \{P\}) \wedge \text{shareable}(\theta_o, \{B, O\})\end{aligned}$$

says that 1) k_b^{-1} and k_b are respectively B's private and public keys, θ_p is P's account secret, θ_o is O's observer secret, i_p is P's id, and ac_p is P's account; 2) these messages are held by the appropriate principals; and 3) O is P's observer.

Local Protocols

In what follows, Secret-Request denotes a predefined message used by P to request O's contribution to a substrate.

B's local protocol $\mathcal{RW}_B$ consists of the following rules:

RW_{B_1}: $\nexists x : t\_subs \mid \text{receive}(P, x) \in \texttt{H} \Longrightarrow \{\text{receive}(P,\ y\text{: } t\_subs)\}$

RW_{B_2}: $\exists x\text{: } t\_subs \mid \text{last}(\texttt{H}) = \text{receive}(P, x) \Longrightarrow \{\text{send}(P, \text{seal}(x,\ ac_p.\text{accn},\ \overline{ac_p.\text{osecret}},\ k_b^{-1}))\}$

RW_{B_3}: $\exists x : t\_seal \mid \text{last}(\texttt{H}) = \text{send}(P, x) \Longrightarrow \{\text{exit}\}$

P's local protocol $\mathcal{RW}_P$ consists of the following rules:

RW_{P_1}: $\text{send}(O, \text{Secret-Request}) \notin \texttt{H} \Longrightarrow \{\text{send}(O, \text{Secret-Request})\}$

RW_{P_2}: $\nexists x_1, x_2, x_3 : t\_num \mid x_1 \neq x_2 \ \wedge\ x_1 \neq x_3 \ \wedge\ x_2 \neq x_3 \wedge$
$\text{random}(x_1) \in \texttt{H} \ \wedge\ \text{random}(x_2) \in \texttt{H} \ \wedge\ \text{random}(x_3) \in \texttt{H} \Longrightarrow \{\text{random}(y\text{: } t\_num)\}$

RW_{P_5}: $\text{send}(O, \text{Secret-Request}) \in \texttt{H} \wedge \nexists x : t\_num \mid \text{receive}(O,\ x) \in \texttt{H} \Longrightarrow$
$\{\text{receive}(O,\ y\text{: } t\_num)\}$

RW_{P_6}: $\exists y_1, y_2, y_3, x : t\_num \mid y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3 \wedge$
$\text{random}(y_1) \in \texttt{H} \wedge \text{random}(y_2) \in \texttt{H} \wedge \text{random}(y_3) \in \texttt{H} \wedge \text{receive}(O,\ x) \in \texttt{H} \wedge$
$\nexists z\text{: } t\_subs \mid \text{send}(B, z) \in \texttt{H} \Longrightarrow \{\text{send}(B,\ \text{blind}(comp1\ comp2,\ y_3))\}$,
where $comp1 = \text{subs}_1(\overline{\theta_p},\ y_1,\ \overline{\theta_o})$ and $comp2 = \text{subs}_2(y_1, y_2, \overline{\theta_o}, x)$

RW_{P_7}: $\exists x\text{: } t\_subs \mid \text{last}(\texttt{H}) = \text{send}(B, x) \Longrightarrow \{\text{receive}(B,\ y\text{: } t\_seal)\}$

RW_{P_8}: $\exists x\text{: } t\_seal \mid \text{last}(\texttt{H}) = \text{receive}(B, x) \Longrightarrow \{\text{exit}\}$

O's local protocol $\mathcal{RW}_O$ consists of the following rules:

RW_{O_1}: $\text{receive}(P, \text{Secret-Request}) \notin \texttt{H} \Longrightarrow \{\text{receive}(P, \text{Secret-Request})\}$

RW_{O_2}: $\text{last}(\mathtt{H}) = \text{receive}(P, \text{SECRET-REQUEST}) \Longrightarrow \{\text{random}(y\text{: } t\_num)\}$

RW_{O_3}: $\exists x\text{: } t\_num \mid \text{last}(\mathtt{H}) = \text{random}(x) \Longrightarrow \{\text{send}(P, \overline{x})\}$

RW_{O_4}: $\exists x\text{: } t\_num \mid \text{last}(\mathtt{H}) = \text{send}(P, x) \Longrightarrow \{\text{exit}\}$

Trust Assumptions

P is not a trusted party; thus, $T_{P_w} = \texttt{true}$. B and O are both trusted. B is trusted:

T_{B_w} : To generate only valid signatures;

and O is trusted:

T_{O_w} : To provide P, as its contribution towards building a substrate, with a number for which it can produce protocertificates.

These trust assumptions can be formalized as follows:

$$T_{B_w}\text{: } \forall x : t\_seal, \Box(\text{send}(P, x) \in \mathtt{H}^{\mathtt{B}} \rightarrow \exists y : t\_subs \mid \text{receive}(P, y) \in \mathtt{H}^{\mathtt{B}} \ \wedge \ x = \text{seal}(y, ac_p.\text{accn}, \overline{ac_p.\text{osecret}}, k^{-1})).$$

$$T_{O_w}\text{: } \forall x : t\_num, \Box(\text{send}(P, x) \in \mathtt{H}^{\mathtt{O}} \rightarrow \exists y : t\_num \mid y \in^* \mathtt{MS}^{\mathtt{O}} \wedge \overline{y} = x).$$

T_{B_w} says that, at any state of an execution, if B sends P a signature, then the signature is correctly generated. The correctness condition, expressed by $x = \text{seal}(y, ac_p.\text{accn}, \overline{ac_p.\text{osecret}}, k^{-1})$ in our formalization, says that a signature is correctly generated if it embeds the right substrate, the right private key, and the right information from the withdrawing account. T_{O_w} says that, at any state of an execution, if O sends P a number x, then it must be a one-way function of a number y retrievable from O's local state. Intuitively, unless this condition is satisfied, O will not be able to provide protocertificates for the coin being withdrawn, because to do so, O needs a number whose one-way function equals the number it gave to P during the withdrawal protocol.

Protection Properties

Abstractly, withdrawal transactions are transactions where account holders withdraw cash from their accounts. A withdrawal transaction is correct if and only if the amount of cash the account holder receives equals the amount deducted from his or her account. Two protection properties derive from this correctness condition. For the account holder, his or her total amount of money should not decrease; that is, the amount deducted from the account should never be bigger than the amount of cash he or she receives. For the bank, the account holder should not be able to get cash from nowhere; that is, the amount of cash the account holder receives should never be bigger than the amount deducted from the account.

In the abstract properties above, the word cash implies something that is *spendable* and that will be *accepted for deposit* in the bank. In Brands's protocol, spendability of a coin is a principal-dependent notion: a principal can spend a coin if and only if he or she can provide rights certificates for it. For deposit, coins are accepted only once: a coin is accepted for deposit only if it has not been deposited before. To ensure that all rightfully withdrawn coins can be deposited, Brands's protocol generates only coins that do not exist anywhere in the system before. We call them *newly-created* coins.

Taking into account the mechanisms just described, and the facts that only one coin is issued per transaction and that there are coins of only one denomination (one unit) in Brands's withdrawal protocol, P's and B's protection properties, P_P^w and P_B^w, can be respectively refined into

> "If B deducts one unit from P's account, then P will acquire a newly-created coin for which she can provide rights certificates,"

and

> "If P acquires a newly-created coin for which she can provide rights certificates, then B deducted one unit from P's account."

We do not model account deduction explicitly in Brands's protocol, but assume that B has capabilities for transaction processing, and will deduct one unit from P's account if and only if it issues P a coin. With this assumption, P_P^w and P_B^w can now be formalized as

$$P_P^w : \Phi_i^w \rightarrow \Box(\Phi_B^w \rightarrow \Diamond \Phi_P^w),$$

and

$$P_B^w : \Phi_i^w \rightarrow \Box(\Phi_P^w \rightarrow \Diamond \Phi_B^w),$$

where

$$\begin{aligned}
\Phi_i^w &= \forall x : t_subs,\ (\exists y : t_seal \mid \mathrm{vseal}(x, y, k_b) \wedge (y \in^* \mathtt{MS}^{\mathtt{P}} \vee y \in^* \mathtt{MS}^{\mathtt{B}} \vee y \in^* \mathtt{MS}^{\mathtt{E}})) \leftrightarrow x \in^* \Omega,\\
\Phi_B^w &= \exists x : t_seal \mid \mathrm{send}(P, x) \in \mathtt{H}^{\mathtt{B}},\\
\Phi_P^w &= \exists x_1 : t_subs_1; x_2 : t_subs_2; x_3 : t_seal; y_1, y_2, y_3, z : t_num \mid \phi_1 \ \wedge\ \phi_2 \ \wedge\ \phi_3,
\end{aligned}$$

and

$$\begin{aligned}
\phi_1 &= \{x_1, x_2, x_3\} \subset^* \mathtt{MS}^{\mathtt{P}} \ \wedge\ \mathrm{vseal}(x_1 x_2, x_3, k_b),\\
\phi_2 &= x_1 x_2 \not\in^* \Omega,\\
\phi_3 &= x_1 = \mathrm{subs}_1(\overline{\theta_p}, y_1, \overline{\theta_o}) \ \wedge\ x_2 = \mathrm{subs}_2(y_1, y_2, \overline{\theta_o}, z) \ \wedge\ z = \overline{y_3} \ \wedge\\
&\quad\ \{\theta_p, \overline{\theta_o}, y_1, y_2, z, \} \subset^* \mathtt{MS}^{\mathtt{P}} \ \wedge\ \{\theta_o, y_3\} \subset^* \mathtt{MS}^{\mathtt{P}} \cup \mathtt{MS}^{\mathtt{O}}.
\end{aligned}$$

Φ_i^w defines Ω as the set of substrates for which B's signatures exist. We use Ω to distinguish preexistent coins from newly-created ones: if a coin's substrate is in Ω, then the coin is preexistent; otherwise, it is newly-created. Φ_B^w formalizes "B issued P a coin". And ϕ_1, ϕ_2, and ϕ_3 respectively formalize "P owns a coin", "the coin is newly-created", and "P jointly with O can provide rights certificates for it".

5.4.2 The Payment Protocol

Principals of the Protocol

$\mathcal{P} = \{P, O, E\}$

Initial Conditions

Let θ_p, θ_o, ν_{1p}, ν_{2p}, and ν_o be numbers; u be a substrate; g be a signature; k_b^{-1} and k_b be respectively a private and a public key; i_e be E's id; Γ be the set of challenge seeds E has used before; B be the bank; and Δ be B's deposit database. The initial condition

$$\begin{aligned}\mathcal{I}^p = \;& \{u, g, \theta_p, \nu_{1p}, \nu_{2p}, \overline{\theta_o}, \overline{\nu_o}\} \subseteq^* \mathtt{MS}^{\mathtt{P}} \;\wedge\; \{\theta_o, \nu_o\} \subseteq^* \mathtt{MS}^{\mathtt{O}} \;\wedge\; \{k_b, i_e, \Gamma\} \subseteq^* \mathtt{MS}^{\mathtt{E}} \;\wedge\; \Delta \in^* \mathtt{MS}^{\mathtt{B}} \wedge \\ & \text{shareable}(k_b^{-1}, \{B\}) \;\wedge\; \text{keyPair}(k_b^{-1}, k_b) \wedge \\ & u = (\text{subs}_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{subs}_2(\nu_{1p}, \nu_{2p}, \overline{\theta_o}, \overline{\nu_o})) \;\wedge\; \text{vseal}(u, g, k_b) \wedge \\ & \text{shareable}(\theta_p, \{P\}) \;\wedge\; \text{shareable}(\nu_{1p}, \{P\}) \;\wedge\; \text{shareable}(\nu_{2p}, \{P\}) \wedge \\ & \text{shareable}(\theta_o, \{B, O\}) \;\wedge\; \text{shareable}(\nu_o, \{O\}) \wedge \\ & \forall x : t\_cs, \text{chal}(i_e, x) \in^* \Delta \rightarrow x \in \Gamma\end{aligned}$$

says that k_b^{-1} and k_b are respectively B's private and public keys; P has a valid coin ug only she jointly with O can spend; and Γ actually contains all stale challenge seeds. Note that B appears in $\mathcal{I}_p$, even though it does not actively participate in the payment protocol. B is needed in $\mathcal{I}_p$ because the validity of a coin's signature is defined in terms of B's private key.

Local Protocols

In the specification below, we need two types of events not listed in Def. 2.4 (Section 2.2.2): $new(x, y)$ and $delete(x)$. We use new(x, y) to model the generation of elements y not found in the set x. In the context of Brands's payment protocol, new(Γ, a) models the generation of a challenge seed a, different from all those that E has used before. delete(x) is used to model the deletion of element x from a principal's message set MS. This type of event is needed to specify O's local protocol, where O erases coin-specific secrets ν_o after they have been used towards generating a protocertificate.

P's local protocol $\mathcal{RP}_P$ consists of the following rules:

RP_{P_1}: $\nexists x : t\_subs, y : t\_seal \mid \text{send}(E, xy) \in \mathtt{H} \Longrightarrow \{\text{send}(E, ug)\}$

RP_{P_2}: $\exists x : t\_subs, y : t\_seal \mid \text{last}(\mathtt{H}) = \text{send}(E, xy) \Longrightarrow \{\text{receive}(E, z : t\_chal)\}$

RP_{P_3}: $\exists x : t\_chal \mid \text{last}(\mathtt{H}) = \text{receive}(E, x) \Longrightarrow \{\text{send}(O, x)\}$

RP_{P_4}: $\exists x : t\_chal \mid \text{last}(\mathtt{H}) = \text{send}(O, x) \Longrightarrow \{\text{receive}(O, y : t\_pcert)\}$

RP_{P_5}: $\exists x : t\_pcert, y : t\_chal \mid \text{last}(\mathtt{H}) = \text{receive}(O, x) \;\wedge\; \text{receive}(E, y) \in \mathtt{H} \;\wedge$
$\quad\text{vpcert}(x, y, \overline{\theta_o}, \overline{\nu_o}) \Longrightarrow \{\text{send}(E, \text{rcert}(x, \theta_p, \nu_{1p}, \nu_{2p}))\}$

RP_{P_6}: $\exists x : t\_pcert, y : t\_chal, z : t\_rcert \mid \text{last}(\mathtt{H}) = \text{send}(E, z) \;\vee$
$\quad(\text{last}(\mathtt{H}) = \text{receive}(O, x) \;\wedge\; \text{receive}(E, y) \in \mathtt{H} \;\wedge\; \neg\text{vpcert}(x, y, \overline{\theta_o}, \overline{\nu_o})) \Longrightarrow \{\text{exit}\}$

O's local protocol $\mathcal{RP}_O$ consists of the following rules:

RP_{O_1}: $\nexists x : t\_chal \mid receive(P, x) \in \mathtt{H} \Longrightarrow \{\text{receive}(P, y : t\_chal)\}$

RP_{O_2}: $\exists x : t\_chal \mid \text{last}(\mathtt{H}) = \text{receive}(P, x) \;\wedge\; \nu_o \in^* \mathtt{MS}^{\mathtt{O}} \Longrightarrow \{\text{send}(P, \text{pcert}(x, \theta_o, \nu_o))\}$

RP_{O_3}: $\exists x_1 : t\_pcert, x_2 : t\_num \mid \text{last}(\mathtt{H}) = \text{send}(P, x_1) \;\wedge\; x_1 = \text{pcert}(-, -, x_2) \Longrightarrow \{\text{delete}(x_2)\}$

RP_{O_4}: $(\exists x : t\_num \mid \text{last}(\mathtt{H}) = \text{delete}(x)) \;\vee$
$(\exists y : t\_chal \mid \text{last}(H) = \text{receive}(P, y) \;\wedge\; \nu_o \notin \mathtt{MS}^{\mathtt{O}}) \Longrightarrow \{\text{exit}\}$

E's local protocol $\mathcal{RP}_E$ consists of the following rules:

RP_{E_1}: $\nexists x : t\_subs \times t\_seal \mid \text{receive}(P, x) \in \mathtt{H} \Longrightarrow \{\text{receive}(P, y : t\_subs \times t\_seal)\}$

RP_{E_2}: $\exists x : t\_subs \times t\_seal \mid \text{last}(\mathtt{H}) = \text{receive}(P, x) \Longrightarrow \{\text{new}(\Gamma, y : t\_cs)\}$

RP_{E_3}: $\forall x : t\_id, (x = i_e \wedge \exists y : t\_cs \mid \text{last}(\mathtt{H}) = \text{new}(\Gamma, y)) \Longrightarrow \{\text{send}(P, \text{chal}(x, y))\}$

RP_{E_4}: $\exists x : t\_chal \mid \text{last}(\mathtt{H}) = \text{send}(P, x) \Longrightarrow \{\text{receive}(P, y : t\_rcert)\}$

RP_{E_5}: $\exists x : t\_rcert \mid \text{last}(\mathtt{H}) = \text{receive}(P, x) \Longrightarrow \{\text{exit}\}$

Trust Assumptions

Neither P nor E is trusted. Thus, $T_P = \mathsf{true}$ and $T_E = \mathsf{true}$. O is trusted. It is trusted:

- T_{O_p}: To delete ν_o only after it has provided P with a protocertificate.

T_{O_p} can be formalized as follows:

T_{O_p}: $\Box(\text{delete}(\nu_o) \in \mathtt{H}^{\mathtt{O}} \rightarrow (\exists x_1 : t\_chal, x_2 : t\_pcert \mid$
$\text{receive}(P, x_1) \in \mathtt{H}^{\mathtt{O}} \wedge \text{send}(P, x_2) \in \mathtt{H}^{\mathtt{O}} \wedge x_2 = \text{pcert}(x_1, \theta_o, \nu_o)))$.

The formalization says that, at any state of an execution, if O deletes ν_o, then it must have sent P a correctly generated protocertificate embedding ν_o.

Protection Properties

Payment transactions are transactions where money is given from one party to another. Abstractly, a payer's interests are protected if his or her money does not disappear, that is, if what he or she gives up is actually received by the payee. A payee's interests are protected, on the other hand, if what he or she receives is or will eventually be worth money.

In Brands's protocol, payments are made with coins. However, when P pays E, she does not give up a coin; she can always keep a copy of the coin around. Instead, she gives up her rights to the coin. Concretely, P makes payments by providing rights certificates for the coins being paid; and the generation of a rights certificate for a coin makes her lose her ability to generate other rights certificates for the same coin.

A coin and its accompanying rights certificate is or will eventually be worth money if the coin can be deposited in the bank or if the identity of the payer can be revealed – thus enabling E to go after P and claim the amount due off-line. A coin can be deposited if it has not been before, and the rights certificate gives rights to the depositor. The identity of the payer can be revealed if the coin has been deposited, but the rights certificate recorded in the deposit database (presented when the coin was accepted for deposit) differs from the one being presented.

Tailored to Brands's protocol, P's and E's protection properties, P_P^p and P_E^p, can now be informally stated as

"If P loses her ability to provide rights certificates for a coin, E will receive the rights to this coin",

and

"If E receives a coin and an accompanying rights certificate provided in response to his challenge, then E can deposit the coin or can have P's identity revealed."

P^p_P and P^p_E can now be formalized as:

$$P^p_P : \forall x_1 : t\_subs_1; x_2 : t\_subs_2; z : t\_seal; y_1, \ldots, y_5, w_1, w_2 : t\_num,$$
$$(\Phi^p_i \to \Box(\Phi^p_P \to \Diamond\Phi^p_{E1}))$$

and

$$P^p_E : \forall x_1 : t\_subs; x_2 : t\_seal; x_3 : t\_rcert; x_4 : t\_chal, \Box(\Phi^p_{E2} \to \Phi^p_B),$$

where

$$\begin{aligned}
\Phi^p_i &= \{y_1, \ldots, y_5, z\} \subset^* \mathtt{MS^P} \ \wedge \ \{w_1, w_2\} \subset^* \mathtt{MS^P} \cup \mathtt{MS^O} \ \wedge \ \overline{w_1} = y_4 \ \wedge \ \overline{w_2} = y_5 \ \wedge \\
&\qquad x_1 = \mathrm{subs}_1(\overline{y_1}, y_2, y_4) \ \wedge \ x_2 = \mathrm{subs}_2(y_2, y_3, y_4, y_5) \ \wedge \ \mathrm{vseal}(x_1 x_2, z, k_b),\\
\Phi^p_P &= w_2 \not\in^* \mathtt{MS^P} \cup \mathtt{MS^O},\\
\Phi^p_{E1} &= \exists z_1 : t_rcert, z_2 : t_cs \mid \mathrm{receive}(P, x_1 x_2 z) \in \mathtt{H^E} \ \wedge \ \mathrm{receive}(P, z_1) \in \mathtt{H^E} \ \wedge \\
&\qquad z_2 \in^* \mathtt{MS^E} \ \wedge \ \mathrm{vrcert}(z_1, \mathrm{chal}(i_e, z_2), x_1 x_2),\\
\Phi^p_{E2} &= \mathrm{receive}(P, x_1 x_2) \in \mathtt{H^E} \ \wedge \ \mathrm{vseal}(x_1, x_2, k_b) \ \wedge \ \mathrm{send}(P, x_4) \in \mathtt{H^E} \ \wedge \\
&\qquad \mathrm{receive}(P, x_3) \in \mathtt{H^E} \ \wedge \ \mathrm{vrcert}(x_3, x_4, x_1),\\
\Phi^p_B &= x_4 = \mathrm{chal}(i_e, -) \ \wedge \ (x_1.\mathrm{comp}_1 \not\in^* \Delta \ \vee \ x_3 \not\in^* \Delta).
\end{aligned}$$

Φ^p_i formalizes "P jointly with O can spend a coin"; Φ^p_P formalizes "P loses the ability to spend the coin"; Φ^p_{E1} formalizes "E receives the rights to the coin"; Φ^p_{E2} formalizes E receives a coin and an accompanying rights certificate provided in response to his challenge; finally, Φ^p_B formalizes "E was given the rights to the coin" and "not both the coin and the rights certificate can be found in the deposit database".

Note that P^p_P and P^p_E differ from all protection properties we have seen so far in this dissertation. While protection is about getting a fair return for what one gives up or is taken away from in all previous protection properties, it is about preservation of a value in P^p_P, and validity of what one receives in P^p_E.

5.4.3 The Deposit Protocol

Principals of the Protocol

$$\mathcal{P} = \{E, B\}$$

Initial Conditions

Let u, g, and r be respectively a substrate, a signature, and a rights certificate; i_e be E's id; t be a challenge seed; k_b^{-1} and k_b be respectively a private and a public key; ac_e and ac_p be two entries in B's account database; Δ be B's deposit database; θ_p and θ_o be numbers; and P be the payer. The initial condition

$$\begin{aligned}\mathcal{I}^d = \ & \{u, g, r, i_e, t\} \subseteq^* \mathtt{MS^E} \ \wedge \ \{k_b^{-1}, k_b, ac_e, ac_p, \Delta\} \subseteq^* \mathtt{MS^B} \ \wedge \\ & \text{shareable}(k_b^{-1}, \{B\}) \ \wedge \ \text{keyPair}(k_b^{-1}, k_b) \ \wedge \\ & \text{shareable}(\theta_p, \{P\}) \ \wedge \ ac_p.\text{accn} = \overline{\theta_p} \ \wedge \ ac_p.\text{osecret} = \theta_o \ \wedge \\ & u = (\text{subs}_1(\overline{\theta_p}, -, \overline{\theta_o}), \text{subs}_2(-, -, \overline{\theta_o}, -)) \ \wedge \\ & \text{vseal}(u, g, k_b) \ \wedge \ ac_e.\text{id} = i_e \ \wedge \ \text{vrcert}(r, chal(i_e, t), u)\end{aligned}$$

says that k_b^{-1} and k_b are respectively B's private and public keys, ac_e and ac_p are respectively E and P's accounts; P's account secret is known to no one other than P; and E has a valid coin (ug) withdrawn from P's account and a rights certificate (r) giving him rights to the coin. Note that P appears in $\mathcal{I}^d$, even though she does not actively participate in the deposit protocol. P is needed in $\mathcal{I}^d$ to specify the value of the account secret θ_p embedded in the coin.

Local Protocols

In the specification below, we need one type of event not listed in Def. 2.4 (Section 2.2.2): *credit*(x). Credit(x) models B's crediting account x.

E's local protocol $\mathcal{RD}_E$ consists of the following rules:

RD_{E_1}: $\nexists x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert \mid \text{send}(B, x_1x_2x_3x_4) \in \mathtt{H} \Longrightarrow \{\text{send}(B, ugtr)\}$

RD_{E_2}: $\exists x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert \mid \text{last}(\mathtt{H}) = \text{send}(B, x_1x_2x_3x_4) \Longrightarrow \{\text{exit}\}$

B's local protocol $\mathcal{RD}_B$ consists of the following rules:

RD_{B_1}: $\nexists x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert \mid \text{receive}(E, x_1x_2x_3x_4) \in \mathtt{H} \Longrightarrow$
$\{\text{receive}(E, x_1'x_2'x_3'x_4')\}$,
where x_1': *t_subs*, x_2': *t_seal*, x_3': *t_cs*, and x_4': *t_rcert*

RD_{B_2}: $\forall x : t\_acc, x = i_e \wedge$
$(\exists x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert \mid \text{last}(\mathtt{H}) = \text{receive}(E, x_1x_2x_3x_4) \ \wedge$
$\text{vseal}(x_1, x_2, k_b) \ \wedge \ \text{vrcert}(x_4, \text{chal}(i_e, x_3), x_1) \ \wedge$
$\nexists x_5 : t\_deposit \in \Delta \mid x_5.\text{subsComp} = x_1.\text{comp}_1) \Longrightarrow \{\text{credit}(x)\}$

RD_{B_3}: $\exists x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert \mid (\text{last}(\mathtt{H}) = \text{receive}(E, x_1x_2x_3x_4) \ \wedge$
$(\neg\text{vseal}(x_1, x_2, k_b) \ \vee \ \neg\text{vrcert}(x_4, \text{chal}(ac_e.\text{id}, x_3), x_1) \ \vee$
$\exists x_5 : t\_deposit \in \Delta \mid x_5.\text{subsComp} = x_1.\text{comp}_1)) \ \vee$
$\text{last}(\mathtt{H}) = \text{credit}(ac_e) \Longrightarrow \{\text{exit}\}$

Trust Assumptions

E is not trusted. Thus, $T_E = \textsf{true}$. B is trusted. It is trusted:

- T_{B_d}: To deposit not-yet-deposited valid coins into the accounts of depositors who have rights to them.

T_{B_d} can be formalized as follows:

T_{B_d}: $\forall x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert,$
$\Box\ (\text{last}(\mathtt{H^B}) = \text{receive}(E, x_1 \ldots x_4)\ \wedge\ \text{vseal}(x_1, x_2, k_b)\ \wedge\ \text{vrcert}(x_4, \text{chal}(ac_e.\text{id}, x_3), x_1)\ \wedge$
$\nexists x_5 : t\_deposit \in \Delta \mid x_5.\text{subsComp} = x_1.\text{comp}_1 \rightarrow \Diamond\ \text{credit}(ac_e)).$

The formalization says that, at any state of an execution, if B receives a not-yet-deposited valid coin and a rights certificate giving the depositor the rights to the coin, then B will credit the depositor's account.

Protection Properties

Deposit transactions are transactions where cash is put in a banking account. Physical world deposit transactions are correct if and only if the amount of cash given up by the depositor equals the increase in the depositor's account balance.

To emulate physical deposit transactions in the electronic realm, Brands's protocol uses auxiliary measures. For example, to avoid accepting a coin for deposit a second time, banks keep a record of coins that have been deposited. To prevent thieves from depositing stolen coins, rights certificates are required in deposit transactions.

Taking into account these auxiliary measures, B's and E's protection properties, P^d_B and P^d_E, can be informally stated as:

> "If B accepts a coin for deposit, then the coin is valid and has not been deposited before,"

and

> "When E submits for deposit a valid coin c and a rights certificate giving him the rights to c – for the first time, either c will be deposited into E's account or P's identity will be revealed,"

respectively.

P^d_B and P^d_E can now be formalized as:

$P^d_B : \forall x_1 : t\_subs, x_2 : t\_seal, \Box(\Phi^d_{B1} \rightarrow \Phi^d_{B2}),$

$P^d_E : \forall x_1 : t\_subs, x_2 : t\_seal, x_3 : t\_cs, x_4 : t\_rcert, \Box(\Phi^d_{B3} \rightarrow \Diamond\Phi^d_E),$

where

$$\Phi^d_{B1} = \text{receive}(E, x_1 x_2 - -) \in \mathtt{H^B}\ \wedge\ \text{credit}(-) \in \mathtt{H^B},$$

$$\Phi^d_{B2} = \text{vseal}(x_1, x_2, k_b)\ \wedge\ x_1.\text{comp}_1 \not\in^* \Delta,$$

$$\Phi^d_{B3} = \text{receive}(E, x_1x_2x_3x_4) \in \mathtt{H}^{\mathtt{B}} \ \wedge \ \text{vseal}(x_1, x_2, k_b) \ \wedge \ \text{vrcert}(x_4, \text{chal}(ac_e.\text{id}, x_3), x_1) \ \wedge \ x_4 \notin^* \Delta,$$

$$\Phi^d_E = \text{credit}(ac_e) \in \mathtt{H}^{\mathtt{B}} \vee \theta_p \in^* \mathtt{MS}^{\mathtt{B}}.$$

Φ^d_{B1} formalizes "B accepts the coin x_1x_2 for deposit"; Φ^d_{B2} formalizes "x_1x_2 is a valid coin that has not been deposited"; Φ^d_{B3} formalizes "B receives a valid coin and a not-yet-submitted rights certificate giving E the rights to the coin"; finally, Φ^d_E formalizes "E's account is credited or P's identity is revealed".

Note that our formalization captures the fact that all B cares about is depositing valid coins that have not been deposited before. Depositing stolen coins does not really hurt B's interests; it hurts E's interests instead. Also if B does not accept the coin for deposit, then P's identity should be revealed, so that E can go after P and claim the amount due off-line.

5.5 Analysis of the Protocol: Preliminaries

In the rest of this chapter, we analyze Brands's protocol with respect to protection of individuals' interests. As our specification (Section 5.4) of transaction-specific protection properties may have forecasted, we do not analyze the protocol as a whole. Instead, we see it as consisting of three subprotocols, and analyze each of them separately. Each subprotocol is analyzed under all three deviation modes defined in Chapter 2. In Section 5.6, Subsections 5.6.1, 5.6.2, and 5.6.3 have analyses of the withdrawal protocol under deception, compliance, and abortion modes, respectively; Subsection 5.6.4 has summary and conclusions. Sections 5.7 and 5.8, with parallel structures to Section 5.6, concern the payment protocol and the deposit protocol respectively.

To analyze Brands's protocol, we repeat the proof strategy we used for NetBill. That is, we first prove that trustworthy executions satisfy our target properties; and then argue that maximal compliant executions also do, because they are trustworthy.

In what follows, we present results needed in the following three sections. Lemmas 3.2 says that, in Brands's protocol, each protocol step gets instantiated at most a finite number of times in executions we consider. Its proof is straightforward, and depends on the fact that the protocol rules either have enabling conditions that prevent the same types of events from occurring more than n (n finite) times, or can be enabled only once, by an occurrence of an event that cannot occur more than once.

Lemma 5.2 *Let Π be withdrawal (payment, or deposit) protocol, σ be an execution in $E(\Pi)^C \cup E(\Pi)^A \cup E(\Pi)^D$, e_i be an event in σ, and E be an event-template prescribing a protocol step in Π. If e_i is an instance of E, then there exist only finite number of different $e_{j_1}, e_{j_2}, \ldots, e_{j_n}$ in σ, $i \neq j_k$, $k = 1, \ldots, n$ such that e_{j_k} is also an instance of E.*

Theorems 5.3, 5.4, and 5.5 say respectively that all maximal compliant executions of Brands's withdrawal, payment and deposit protocols are trustworthy.

Theorem 5.3 *Let Π be Brands's withdrawal protocol and $\mathcal{T} = T_{B_w} \wedge T_{O_w}$ be its trust assumptions. Then*

$$\forall \sigma \in E(\Pi)^C, \ \sigma \models^* \mathcal{T}.$$

Proof: Straightforwardly, from rules RW_{B_2} and RW_{O_3}.

□

Theorem 5.4 *Let* Π *be Brands's payment protocol and* $\mathcal{T} = T_{O_p}$ *be its trust assumptions. Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* \mathcal{T}.$$

Proof:

1. Let s_i be a state in σ such that $s_i \models \text{delete}(\nu_o) \in \mathtt{H}^{\mathtt{O}}$.

   $\text{delete}(\nu_o)$ must have resulted from an application of rule RP_{O_3}. Therefore, there must be a state s_j, $j < i$, and a protocertificate m such that $s_j \models \text{last}(\mathtt{H}^{\mathtt{O}}) = \text{send}(m)$.

2. Send(m) must have resulted from an application of RP_{O_2}. Therefore, there must be a state s_k, $k < j$, and a challenge m' such that $s_k \models \text{last}(\mathtt{H}^{\mathtt{O}}) = \text{receive}(P, m')$ and $m = \text{pcert}(m', \theta_o, \nu_o)$.

□

Theorem 5.5 *Let* Π *be Brands's deposit protocol and* $\mathcal{T} = T_{B_d}$ *be its trust assumptions. Then*

$$\forall \sigma \in E(\Pi)^C,\ \sigma \models^* \mathcal{T}.$$

Proof: Straightforwardly, from rule RD_{B_2}.

□

In the rest of this analysis, we assume that the communication channels between P and O are reliable, even though communication channels are generally unreliable in our model. We make this assumption because O is likely to stay physically close to P, and the link between them is not as likely to go down.

5.6 Analysis of the Withdrawal Protocol

5.6.1 Protection under Deception Mode

In this subsection, we analyze the protocol under deception mode. We first analyze it with respect to P-protection, then with respect to B-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, if Π is Brands's withdrawal protocol, then to analyze it with respect to P-protection under deception mode, we need to analyze all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^D_P$. We do so in Prop. 5.7 and 5.6 respectively.

Proposition 5.6 *Let* Π *be Brands's withdrawal protocol,* σ *be an execution in* $E(\Pi)^D_P$*,* $\mathcal{T}$ *be the trust assumptions* Π *makes, and* $P^w_P = \Phi^w_i \rightarrow \Box(\Phi^w_B \rightarrow \Diamond\Phi^w_P)$ *be* P*'s protection property as specified in Section 5.4.1 (p. 112). Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P^w_P.$$

Analysis: Let $\sigma = s_0\ e_1\ s_1 \ldots$ be a trustworthy execution, s_0 be such that $s_0 \models \Phi_i^w$, and s_i be a state in σ such that $s_i \models \Phi_B^w$. The following shows that there exists j, $j > i$, such that $s_j \models \Phi_P^w$, *if we assume that communication channels are reliable in the system.*

1. If $s_i \models \Phi_B^w$, then there exists a signature m such that
$$s_i \models \text{send}(P, m) \in \mathtt{H}^{\mathsf{B}}. \tag{5.1}$$
From expression 5.1, we can reach two conclusions. First, assuming reliability of communication channels, we conclude that there exists j, $j > i$, such that
$$s_j \models \text{receive}(B, m) \in \mathtt{H}^{\mathsf{P}}. \tag{5.2}$$
Second, according to the trust assumption T_{B_w}, there exists a substrate m' such that
$$s_i \models \text{receive}(P, m') \in \mathtt{H}^{\mathsf{B}} \wedge m = \text{seal}(m', ac_p.\text{accn}, \overline{ac_p.\text{osecret}}, k_b^{-1}). \tag{5.3}$$

2. A message needs to be sent before it can be received. Thus, from expression 5.3 (step 1), we can conclude that
$$s_i \models \text{send}(B, m') \in \mathtt{H}^{\mathsf{P}},$$
where $\text{send}^{\mathsf{P}}(B, m')$ must have resulted from an application of RW_{P_6}.

3. Next, let $s_{i'}$, $i' < i$, be the state at which RW_{P_6} was applied. Then there exist three different numbers n_1, n_2, and n such that
$$s_{i'} \models \text{random}(n_1) \in \mathtt{H}^{\mathsf{P}} \wedge \text{random}(n_2) \in \mathtt{H}^{\mathsf{P}} \wedge \text{random}(n) \in \mathtt{H}^{\mathsf{P}},$$
and a fourth number, n', such that
$$s_{i'} \models \text{receive}(O, n') \in \mathtt{H}^{\mathsf{P}}, \tag{5.4}$$
and m' is such that
$$m' = \text{blind}(m_1 m_2, n),$$
where $m_1 = \text{subs}_1(\overline{\theta_p}, n_1, \overline{\theta_o})$ and $m_2 = \text{subs}_2(n_1, n_2, \overline{\theta_o}, n')$.

4. From expression 5.4 (step 3), we can conclude that
$$s_{i'} \models \text{send}(P, n') \in \mathtt{H}^{\mathsf{O}}.$$
And by trust assumption T_{O_w}, we know that there must be a number n_3 such that
$$s_{i'} \models n_3 \in^* \mathtt{MS}^{\mathsf{O}} \wedge n' = \overline{n_3}.$$

5. We can now prove that
$$s_j \models \phi_1[m_1/x_1, m_2/x_2, \text{unblind}(m, n)/x_3].$$
   (a) From expression 5.2 (step 1), and the fact that $s_0 \models \{\theta_p, \overline{\theta_o}\} \subset^* \mathtt{MS}^{\mathsf{P}}$, we know that
$$s_j \models \{m_1, m_2, \text{unblind}(m, n)\} \subset^* \mathtt{MS}^{\mathsf{P}}.$$

(b) And vseal(m_1m_2, unblind(m, n), k_b) can be proven straightforwardly using the conversion rules in Def. 5.1 (Section 5.3).

6. We can also prove that
$$s_j \models \phi_2[m_1/x_1, m_1/x_2],$$
since m_1 and m_2 have submessages generated at random.

7. Finally, we can prove that
$$s_j \models \phi_3\alpha,$$
where $\alpha = [m_1/x_1, m_2/x_2, \text{unblind}(m,n)/x_3, n_1/y_1, n_2/y_2, n_3/y_3, n'/z]$.

   (a) From the initial condition $\mathcal{I}$ and step 3, we have
   $$s_j \models \{\theta_p, \overline{\theta_o}, n_1, n_2, n'\} \subset^* \mathtt{MS}^{\mathtt{P}};$$

   (b) Also from the initial condition $\mathcal{I}$ and step 4, we have
   $$s_j \models \{\theta_o, n_3\} \subset^* \mathtt{MS}^{\mathtt{O}}.$$

Note, however, that communication channels are not reliable in our model. Thus, ϕ_1 may never be satisfied because P may never receive the signature from B.

□

Bearing in mind what Φ_i^w, Φ_B^w, and Φ_P^w specify (Section 5.4.1), the analysis of Prop. 5.6 shows that if B issues P a signature, consequently deducting one unit from P's account, and the communication channel between B and P does not fail, then P will acquire a newly-created coin for which she can provide rights certificates.

For P to acquire such a coin, both B and O need to be trustworthy. If B does not behave as specified by the trust assumption T_{B_w}, and generates a signature that embeds numbers other than those specified, the signature it returns to P will not be valid, and P will not have received a coin. If O does not behave as specified by T_{O_w} and returns a number from which it cannot generate an appropriate protocertificate later in the payment protocol, then P will not be able to generate rights certificates for the coin. Any number that is not one-way function of a second number O possesses will lead to such an outcome.

Finally, reliability of the communication channel between B and P is critical because P will have a coin only if she receives the signature from B. If the link between B and P goes down and the signature gets lost in the transmission, P will have her account deducted without receiving a coin in return.

Prop. 5.7 concerns maximal compliant executions. These executions are trustworthy and have compliant local executions of P. Given that our analysis of Prop. 5.6 relies on the fact that σ is trustworthy and has a compliant local execution of P, it can be used as is to analyze Prop. 5.7.

Proposition 5.7 *Let Π be Brands's withdrawal protocol, σ be an execution in $E(\Pi)^C$, and P_P^w be P's protection property as specified in Section 5.4.1 (p. 112). Then*
$$\sigma \models^* P_P^w.$$

From Prop. 5.6 and 5.7, we can derive the following

Corollary 5.8 *Brands's withdrawal protocol is P-protective under deception mode, if the communication channel between B and P is reliable.*

Next we address B-protection. Like in the analysis of P-protection, we analyze deceptive and compliant executions in turn.

Proposition 5.9 *Let Π be Brands's withdrawal protocol, σ be an execution in $E(\Pi)^D_B$, and $P^w_B = \Phi^w_i \rightarrow \Box(\Phi^w_P \rightarrow \Diamond\Phi^w_B)$ be B's protection property as specified in Section 5.4.1 (p. 112). Then*

$$\sigma \models^* P^w_B.$$

Proof: Let s_0 be such that $s_0 \models \Phi^w_i$, and s_i be a state in σ such that $s_i \models \Phi^w_P$. The following shows that there exists j, $j > i$, such that $s_j \models \Phi^w_B$.

1. If $s_i \models \Phi^w_P$, then there exists a substrate m_1m_2 and a signature m_3 such that

$$s_i \models m_1m_2 \not\in^* \Omega \wedge \{m_1m_2, m_3\} \subset^* \mathtt{MS}^\mathtt{P} \wedge \mathrm{vseal}(m_1m_2, m_3, k_b) \tag{5.5}$$

2. From the first conjunct in expression 5.5 (step 1), we know that

$$s_0 \models \nexists x : t\_seal \mid x \in^* \mathtt{MS}^\mathtt{P} \wedge \mathrm{vseal}(m_1m_2, x, k_b),$$

   which means that m_3 must have been generated or received by P during the current protocol execution.

3. From the last conjunct in expression 5.5 (step 1), we know that

$$m_3 = \mathrm{seal}(m_1m_2, -, -, k_b^{-1}).$$

   Since k_b^{-1} is kept private to B at all times, we can conclude that m_3 could not have been generated by P, but must have been received from B. And according to P's protocol, this message is a signature. Thus, we conclude that there is a signature m, such that

$$s_i \models \mathrm{receive}(B, m) \in \mathtt{H}^\mathtt{P}. \tag{5.6}$$

4. Finally, from expression 5.6 (step 3), we can conclude that

$$s_i \models \mathrm{send}(P, m) \in \mathtt{H}^\mathtt{B}.$$

□

Bearing in mind what Φ^w_i, Φ^w_B, and Φ^w_P specify (Section 5.4.1), the proof of Prop. 5.9 shows that if P acquires a newly-created coin, B must have issued it. This is so because, to acquire a newly-created coin, P needs a newly-generated signature, which can be provided only by B. Note that B-protection depends on neither trust assumptions nor reliability of communication channels.

Prop. 5.10 concerns maximal compliant executions. Our proof of Prop. 5.9 is readily applicable here because it only relies on the fact that σ has compliant local executions of B, a condition satisfied by maximal compliant executions.

Proposition 5.10 *Let Π be Brands's withdrawal protocol, σ be an execution in $E(\Pi)^C$, and P_B^w be B's protection property as specified in Section 5.4.1 (p. 112). Then*

$$\sigma \models^* P_B^w.$$

From Prop. 5.9 and 5.10, we can derive the following

Corollary 5.11 *Brands's withdrawal protocol is B-protective under deception mode.*

Theorem 5.12 summarizes the results in this subsection:

Theorem 5.12 *Brands's withdrawal protocol is all-protective under deception mode, if the communication channel between B and P is reliable.*

Proof: From Cor. 5.8 and 5.11.

□

Discussion

According to our analysis, P's interests are protected under deception mode only if the communication channel between B and P is reliable. Clearly, if the channel is unreliable, and the link goes down after B sends P the signature, but before P gets it, then B will have deducted from P's account, but P will not have received a coin.

The dependency on reliable communication channels to guarantee P-protection is a weakness of this protocol. There are ways of bypassing this weakness however. For example, if B keeps a database of blinded substrates for which it has issued a signature, then P can re-submit a substrate if she does not receive the signature from B. B issues signatures for all requests, but only debiting the requester's account when the substrate cannot be found in the database. Note that B does not need to keep these substrates forever. P can send B an acknowledgment when she gets the signature, and B can delete the corresponding entry in the database upon receiving the acknowledgment.

Protection of P's interests also depends on B and O satisfying T_{B_w} and T_{O_w}. The discussion following the analysis of Prop. 5.6 gives an intuition of how the violation of these trust assumptions can compromise P-protection.

It is reasonable to assume that B will behave as trusted; after all, it is B's interest to preserve long term relationships with its clients. There is an alternative, however, if B cannot be trusted for some reason: The alternative consists of making receipt of messages non-repudiable [89]. With non-repudiation and P's ability to verify a signature's validity, P can show to a third party that B did not issue a valid signature for the substrate she had sent and can therefore demand a refund.

T_{O_w} is a more critical assumption, and relies on correct implementations of smartcards. Unlike with B, it is impossible for P to tell whether O has behaved as trusted during a withdrawal transaction. O's misbehaving can be detected only later, when P tries to spend the coin, and finds out that O is unable to provide appropriate protocertificates. One possible safeguard against O's misbehavior is to have B keep a record of the signatures it issued with their respective requesters. When P finds out that she is unable to spend a coin because of O, she can then go to B to revoke the corresponding coin and get a refund.

Next, we focus on B-protection. According to our analysis, B's interests are protected under deception mode, independent of link reliability and trust assumptions. This is not surprising because the signature of a newly-created coin can be generated only in the current run of the protocol, and B is the only one able to generate it. Thus, if P acquires a newly-created coin, B must have provided the signature, and consequently deducted from P's account.

Note that both P and O can deceive. But their deceptions do not violate B's interests. P's deceptions can compromise P's own interests however. For example, if P embeds something other than the number returned by O in the substrate, P compromises O's ability to provide appropriate protocertificates later in payment transactions, and makes the coin non-spendable. Most surprisingly, however, P's deceptions can compromise E's interests in deposit transactions. We will show how later in Section 5.8.

All in all, P is much more vulnerable than B in withdrawal transactions. While P needs to rely on B and O's trusted behaviors and reliability of communication channels, B needs to rely on only the cryptographic strength of the signature scheme.

5.6.2 Protection under Compliance Mode

In this subsection, we analyze the protocol under compliance mode. To verify whether the protocol is all-protective under compliance mode, we need to verify whether both P's and B's interests are protected in maximal compliant executions. But these results have already been verified in Subsection 5.6.1. According to Prop 5.7 and 5.10,

Theorem 5.13 *Brands's withdrawal protocol is all-protective under compliance mode, if the communication channels between P and B are reliable.*

Discussion

Brands's withdrawal protocol is not all-protective under compliance mode. Like under deception mode, it is B-protective, but not P-protective. It is B-protective because B depends on nothing other than the strength of the signature scheme to protect its interests. It is not P-protective because P still risks not receiving the signature from B due to unreliable communication channels.

5.6.3 Protection under Abortion Mode

In this subsection, we analyze the protocol under abortion mode. Like in the two previous subsections, we need to verify whether the protocol is both P-protective and B-protective. We start with P-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, to verify whether Π protects P's interests under abortion mode, we need to examine all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_P$. Here we focus on abortive executions only, since we have analyzed compliant executions (Prop. 5.7) in Section 5.6.1.

Proposition 5.14 *Let Π be Brands's withdrawal protocol, σ be an execution in $E(\Pi)^A_P$, $\mathcal{T}$ be the trust assumptions Π makes, and P^w_P be P's protection property as specified in Section 5.4.1 (p. 112). Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P^w_P.$$

Prop. 5.14 can be analyzed using the proof for Prop. 5.6. That proof is applicable here because it relies only on the fact that executions it considers are trustworthy and have compliant local executions of P, both of which are satisfied by executions we consider under Prop. 5.14. A critical condition satisfied by the proof is that it does not depend on B or O taking further steps, a requirement that could be violated by abortive executions.

From Prop. 5.7 (Section 5.6.1) and 5.14, which jointly address P-protection under abortion mode, we derive the following

Corollary 5.15 *Brands's withdrawal protocol is P-protective under abortion mode, if the communication channels between P and B are reliable.*

Prop. 5.16 addresses protection of B's interests in abortive executions and can be proven using the proof for Prop. 5.9 (Subsection 5.6.1). That proof is applicable here for a reason analogous to the one given for Prop. 5.14.

Proposition 5.16 *Let Π be Brands's withdrawal protocol, σ be an execution in $E(\Pi)^A_B$, and P^w_B be B's protection property as specified in Section 5.4.1 (p. 112). Then*

$$\sigma \models^* P^w_B.$$

From Prop. 5.10 (Section 5.6.1) and 5.16, which jointly address B-protection under abortion mode, we derive the following

Corollary 5.17 *Brands's withdrawal protocol is B-protective under abortion mode.*

Theorem 5.18 summarizes the results in this subsection.

Theorem 5.18 *Brands's withdrawal protocol is all-protective under abortion mode, if the communication channels between P and B are reliable.*

Proof: From Cor. 5.15 and 5.17.

□

Discussion

Brands's withdrawal protocol is not all-protective under abortion mode. The violation of protection still results from unreliable communication channels, and not premature terminations of local executions.

Surely, if O terminates its local execution before sending P a message, P will not generate a substrate and will not even contact B. If B terminates its local execution before sending P the signature, P will not acquire a new coin, but neither will B deduct from P's account. If P terminates its local execution before sending the substrate, B again does not get contacted. In all these cases, no one's interest is hurt.

If P terminates its local execution after sending the substrate, but before receiving the signature, B may deduct from P's account without P receiving a signature. We say "may" instead of "will" because the substrate may get lost in transmission and never trigger a local withdrawal processing at B. In any case, B's interests are not hurt.

5.6.4 Summary

We summarize our findings in Table 5.1. All-protection is guaranteed in entries with a $\surd$. Some entries come with briefings of relevant facts. The table shows that Brands's withdrawal protocol relies critically on reliable communication channels.

	compliance	abortion	deception
Channel Failures	(1)	(1)	(1)
No Channel Failures	$\surd$	$\surd$ (2)	$\surd$ (3)

1. P-protection is violated if the signature sent by B does not get to P.
2. B executes signature sending and account deduction atomically. Its premature terminations are inconsequential.

   If O terminates prematurely, P will not proceed with the protocol, and no substrate will be sent to B.

   If P terminates her execution before sending B the substrate, no signature is issued, and no account is deducted. If P terminates her execution after sending the substrate, but before receiving the signature, no harm (except to herself) can be made.
3. According to T_{B_w} and T_{O_w}, neither B nor O deceives. P's deceptions do not hurt B's interests; they hurt her own interests in withdrawal transactions and E's interests in deposit transactions.

Table 5.1: Summary table for the withdrawal protocol

5.7 Analysis of The Payment Protocol

5.7.1 Protection under Deception Mode

In this subsection, we analyze Brands's payment protocol under deception mode. We first analyze it with respect to P-protection, then with respect to E-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, if Π is Brands's payment protocol, then to analyze it with respect to P-protection under deception mode, we need to analyze all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^D_P$. We do so in Prop. 5.22 and 5.19 respectively.

Proposition 5.19 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)^D_P$, $\mathcal{T}$ be the trust assumptions Π makes, and P^p_P be P's protection property as specified in Section 5.4.2 (p. 115). Then*

$$\sigma \models^* \mathcal{T} \quad implies \quad \sigma \models^* P^p_P.$$

Analysis: In what follows, Φ^p_i, Φ^p_P, and Φ^p_{E1} are subformulas of P^p_P as specified in Section 5.4.2, and $\sigma = s_0\ e_1\ s_1 \ldots$ is a trustworthy execution. Because σ is an execution of Π, s_0 satisfies Π's initial condition $\mathcal{I}^p$. Examining $\mathcal{I}^p$, we see that

$$\alpha = [\theta_p/y_1, \nu_{1p}/y_2, \nu_{2p}/y_3, \overline{\theta_o}/y_4, \overline{\nu_o}/y_5, g/z, \theta_o/w_1, \nu_o/w_2, u.\mathrm{comp}_1/x_1, u.\mathrm{comp}_2/x_2]$$

is a substitution such that $s_0 \models \Phi^p_i \alpha$.

Next, let s_i be a state in σ such that $s_i \models \Phi^p_P \alpha$. That is, $s_i \models \nu_o \not\in^* \mathtt{MS}^{\mathtt{P}} \cup \mathtt{MS}^{\mathtt{O}}$. We would like to show that there exist a state s_j, $j > i$, such that

$$s_j \models \Phi^p_{E1} \alpha,$$

if the communication channels are reliable in the system.

1. From $s_0 \models \mathcal{I}^p$, we know that
$$s_0 \models \nu_o \in^* \mathtt{MS}^{\mathtt{O}}.$$
Thus, if $s_i \models \nu_o \not\in^* \mathtt{MS}^{\mathtt{P}} \cup \mathtt{MS}^{\mathtt{O}}$, it must be the case that ν_o was deleted from $\mathtt{MS}^{\mathtt{O}}$. That is,
$$s_i \models \text{delete}(\nu_o) \in \mathtt{H}^{\mathtt{O}}. \tag{5.7}$$
From expression 5.7 and trust assumption T_{O_p}, we can conclude that there exist messages m_1: *t_chal* and m_2 : *t_pcert*, such that
$$s_i \models \text{receive}(P, m_1) \in \mathtt{H}^{\mathtt{O}} \ \wedge \ \text{send}(P, m_2) \in \mathtt{H}^{\mathtt{O}} \ \wedge \ m_2 = \text{pcert}(m_1, \theta_o, \nu_o). \tag{5.8}$$

2. From Lemma 5.20, the fact that $s_i \models \text{receive}(P, m_1) \in \mathtt{H}^{\mathtt{O}}$ (expression 5.8, step 1), and the fact that P behaves compliantly in σ, we can conclude that
$$s_i \models \text{receive}(P, ug) \in \mathtt{H}^{\mathtt{E}}. \tag{5.9}$$

3. From expression 5.8 (step 1) and Lemma 5.21, we conclude that there exists a state $s_{i'}$, $i' \geq i$, and a rights certificate m_3, such that
$$s_{i'} \models \text{receive}(P, m_3) \in \mathtt{H}^{\mathtt{E}}, \tag{5.10}$$
and
$$m_3 = \text{rcert}(\text{pcert}(m_1, \theta_o, \nu_o), \theta_p, \nu_{1p}, \nu_{2p}).$$
And it is straightforward to see that
$$\text{vrcert}(m_3, m_1, u). \tag{5.11}$$
Since P behaves compliantly in σ, we know that m_1 is the challenge P received from E. Thus, we know that
$$s_{i'} \models \text{send}(P, m_1) \in \mathtt{H}^{\mathtt{E}}.$$

4. Examining E's local protocol, we see that $\text{send}^E(P, m_1)$ must have resulted from an application of RP_{E_3}, which prescribes sending a challenge derived from E's id i_e and a challenge seed newly generated.

   (a) If E behaves compliantly, we would have
   $$s_{i'} \models \exists x : t\_cs \mid m_1 = \text{chal}(i_e, x) \wedge x \in^* \mathtt{MS}^{\mathtt{E}}. \tag{5.12}$$
   And from expressions 5.9, 5.10, and 5.12, we would obtain
   $$s_{i'} \models \Phi^p_{E1} \alpha,$$
   as we would like.

(b) But σ is a deceptive execution, and E may not have derived m_1 from its own id. Thus, it is possible that

$$s_{i'} \models \nexists x : t\_cs \mid m_1 = \text{chal}(i_e, x). \tag{5.13}$$

From expressions 5.11 and 5.13, we can now conclude that

$$\nexists x : t\_cs \mid \text{vrcert}_{\text{p}}(m_3, \text{chal}(i_e, x), u).$$

$\square$

Bearing in mind what Φ_i^p, Φ_P^p, and Φ_{E1}^p specify (Section 5.4.2), the analysis of Prop. 5.19 shows that P's losing the ability to provide rights certificates for a coin during a payment transaction does not always entail E's acquiring the rights to the coin, even if communication channels are reliable in the system. Effectively, this means that the value P relinquishes is not transferred to E, but vanishes instead.

Clearly, if the communication channel between P and E can go down, the rights certificate P sends to E (presumably transferring the rights to the coin from P to E) can get lost in transmission. But even if the communication channels are reliable and E receives the rights certificate, E may still not acquire the rights to the coin. This is so because E may deceive and use something other than its own id to generate the challenge. Rights certificates generated in response to challenges that do not embed E's id do not give E the rights to the coin. Note that these problems exist even if O behaves as trusted. If O does not behave as specified in T_{O_p}, and consumes ν_o to produce a protocertificate different from that prescribed, P has a different problem: the protocertificate she receives will not pass the validity check, and no rights certificate will be generated.

The following two lemmas appear in our analysis of Prop. 5.19. Lemma 5.20 says that, in compliant executions and deceptive executions where P behaves compliantly, if O receives a challenge from P, then E must have received a coin from P.

Lemma 5.20 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)_P^D \cup E(\Pi)^C$, and s_i be a state in σ such that*

$$s_i \models \exists x : t\_chal \mid \text{receive}(P, x) \in \mathtt{H}^{\mathtt{O}}.$$

Then,

$$s_i \models \exists x : t\_subs \times t\_seal \mid \text{receive}(P, x) \in \mathtt{H}^{\mathtt{E}}.$$

Proof: By straightforward backchaining, using the protocol rules and the fact that messages need to be sent before they can be received.

$\square$

Lemma 5.21 says that, in compliant executions and deceptive executions where P behaves compliantly, if the communication channels are reliable, then once O sends P the expected protocertificate, E will receive a rights certificate.

Lemma 5.21 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)_P^D \cup E(\Pi)^C$, θ_o and ν_o be numbers as specified in $\mathcal{I}^p$, and s_i be a state in σ such that*

$$s_i \models \exists x_1 : t\_chal, x_2 : t\_pcert \mid \text{receive}(P, x_1) \in \mathtt{H}^{\mathtt{O}} \ \wedge \ \text{send}(P, x_2) \in \mathtt{H}^{\mathtt{O}} \ \wedge \ x_2 = \text{pcert}(x_1, \theta_o, \nu_o).$$

Then, there exists a state $s_j, j \geq i$, such that

$$s_j \models \exists x : t\_rcert \mid \text{receive}(P, x) \in \mathtt{H}^{\mathtt{E}},$$

if the communication channels are reliable.

Proof:

1. Let m_1: *t_chal*, m_2 : *t_pcert* be messages such that

$$s_i \models \text{receive}(P, m_1) \in \mathtt{H}^{\mathtt{O}} \;\wedge\; \text{send}(P, m_2) \in \mathtt{H}^{\mathtt{O}} \;\wedge\; m_2 = \text{pcert}(m_1, \theta_o, \nu_o). \tag{5.14}$$

2. From $s_i \models \text{receive}(P, m_1) \in \mathtt{H}^{\mathtt{O}}$ (expression 5.14), we conclude, by straightforward backchaining, that

$$s_i \models \text{receive}(E, m_1) \in \mathtt{H}^{\mathtt{P}}. \tag{5.15}$$

3. *If communication channels are reliable*, we can conclude, from $s_i \models \text{send}(P, m_2) \in \mathtt{H}^{\mathtt{O}}$ (expression 5.14), that there exists a state $s_{i'}, i' \geq i$, such that

$$s_{i'} \models \text{receive}(O, m_2) \in \mathtt{H}^{\mathtt{P}}. \tag{5.16}$$

4. From expressions 5.15 and 5.16, and the fact that $m_2 = \text{pcert}(m_1, \theta_o, \nu_o)$, we can conclude that there exists a state $s_{i''}, i'' \geq i'$, such that

$$s_{i''} \models \text{send}(E, \text{rcert}(m_2, \theta_p, \nu_{1p}, \nu_{2p})) \in \mathtt{H}^{\mathtt{P}}. \tag{5.17}$$

5. *If communication channels are reliable*, we can conclude, from expression 5.17, that there exists a state $s_{i'''}, i''' > i''$, such that

$$s_{i'''} \models \text{receive}(P, \text{rcert}(m_2, \theta_p, \nu_{1p}, \nu_{2p})) \in \mathtt{H}^{\mathtt{E}}.$$

□

Prop. 5.22 concerns maximal compliant executions. These executions are trustworthy and have compliant local executions of P and E. Its analysis is identical to that of Prop. 5.19, except for the last step.

Proposition 5.22 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)^C$, and P_{P}^{p} be P's protection property as specified in Section 5.4.2 (p. 115). Then*

$$\sigma \models^* P_{\mathrm{P}}^{p}.$$

Analysis: The first part (steps 1 - 3) of this analysis is identical to that of Prop 5.19. Under compliance mode, we reach a different conclusion:

4. Since E behaves compliantly, we conclude that $m_1 = \text{chal}(i_e, m_6)$, where m_6 is a challenge seed newly generated by E. Finally, we have

$$s_{i'} \models m_6 \in^* \mathtt{MS}^{\mathtt{E}} \;\wedge\; \text{vrcert}(m_1, \text{chal}(i_e, m_6), u).$$

□

The analysis above shows that the only problem P faces in maximal compliant executions is unreliable communication channels, which can prevent E from receiving the rights certificate sent by P.

From Prop. 5.19 and 5.22, we can derive the following

Corollary 5.23 *Brands's payment protocol is not P-protective under deception mode, even if the communication channels are reliable.*

Next we address E-protection. Like in the analysis of P-protection, we analyze deceptive and compliant executions in turn.

Proposition 5.24 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)^D_E$, and P^p_E be E's protection property as specified in Section 5.4.2 (p. 115). Then*

$$\sigma \models^* P^p_E.$$

Proof: In what follows, Φ^p_{E2} and Φ^p_B are subformulas of P^p_E, as specified in Section 5.4.2.

1. Let s_i be a state in σ, and $m_1 : t\_subs$, m_2: t_seal, $m_3 : t\_rcert$, and m_4: t_chal be messages such that
$$s_i \models \Phi^p_{E2}[m_1/x_1, \ldots, m_4/x_4].$$
Then,
$$s_i \models \text{send}(P, m_4) \in \mathtt{H}^{\mathtt{E}} \ \wedge \ \text{receive}(P, m_3) \in \mathtt{H}^{\mathtt{E}} \ \wedge \ \text{vrcert}(m_3, m_4, m_1). \tag{5.18}$$

2. $\text{Send}^E(P, m_4)$ (expression 5.18) could have resulted only from an application of RP_{E_3}. And since E behaves compliantly in σ, we know that
$$m_4 = \text{chal}(i_e, m_5),$$
where m_5 is a newly generated challenge seed.

3. From $\text{vrcert}(m_3, m_4, m_1)$, we know that
$$m_3 = \text{rcert}(\text{pcert}(m_4, -, -), -, -, -).$$
And since $m_5 \notin \Gamma$, we know that $m_4 \notin^* \Delta$, which implies that $m_3 \notin^* \Delta$.

□

Bearing in mind what Φ^p_{E2} and Φ^p_B specify (Section 5.4.2), the proof of Prop. 5.24 shows that if E accepts a coin and an accompanying rights certificate, then E must have been given the rights to the coin, and the rights certificate will not be found in B's deposit database. E is given the rights to the coin because the rights certificate he accepts can be checked against his challenge (the one he presented to P), and this challenge embeds E's id. The rights certificate will not be found in B's deposit database because it embeds the challenge seed embedded in E's challenge, and this challenge seed is different from all those E has used before. Note that Prop. 5.24 holds independent of trust assumptions or reliability of communication channels.

Of course, P may deceive in executions in $E(\Pi)_E^D$, and send E invalid coins and rights certificates. E will not be fooled however, because he checks their validity before accepting them. Validity conditions checked by E (expressed by predicates vseal and vrcert) are included in Φ_{E2}^p.

Prop. 5.25 concerns maximal compliant executions. Our proof of Prop. 5.24 is readily applicable here because it only relies on the fact that σ has compliant local executions of E, a condition satisfied by maximal compliant executions.

Proposition 5.25 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)^C$, and P_E^p be E's protection property as specified in Section 5.4.2 (p. 115). Then*

$$\sigma \models^* P_E^p.$$

From Prop. 5.24 and 5.25, we can derive the following

Corollary 5.26 *Brands's payment protocol is E-protective under deception mode.*

Theorem 5.27 summarizes the results in this subsection:

Theorem 5.27 *Brands's payment protocol is not all-protective under deception mode, even if the communication channels are reliable.*

Proof: From Cor. 5.23 and 5.26.

□

Discussion

According to our analysis, P's interests are not protected under deception mode even if communication channels in the system are reliable. In fact, E can deceive and send P a challenge that embeds a bogus principal id. This will cause P to generate a bogus rights certificate, which will consume P's one-time capability of generating rights certificates for a coin, but will not give E, or any other principal, the rights to the coin. Effectively, the corresponding coin is rendered worthless because P can no longer spend it, and no one acquired the rights to deposit it.

There are countermeasures against this threat, however. For example, if P can generate challenges herself, then she will not risk generating rights certificates for bogus challenges. To implement this approach, E can send P only a challenge seed (t), and allow P to look up E's id in some trusted public directory. Using t and E's id, P can then generate a challenge herself, and be sure that it embeds a valid E id. This approach does not affect E-protection, since E can always reconstitute challenges himself to check the validity of corresponding rights certificates.

Alternatively, if message receipts are non-repudiable, P can show to a third party the challenge she received from E, and prove that E is responsible for the loss and therefore should be the only one held accountable for it. Under this second approach, E's deception is no longer an act of sabotage inconsequential to E. Instead, it will be an assault to E's own interests.

Aside from E's deceptions, unreliable communication channels and O's violation of trust assumptions can also compromise P-protection. There are countermeasures for these threats too. To safeguard against loss of rights certificates due to unreliable communication channels, P can keep a copy of the rights certificate and send it later to E, assuming that E is willing to receive it. To safeguard against P's inability to generate an appropriate rights certificate due to O's violation of T_{O_p}, we can use the same measure suggested to counteract O's violation of T_{O_w} in the withdrawal protocol. It consists of having B keep a record of the signatures it issued with their respective requesters during the withdrawal protocol. When P finds out that she is unable to spend a coin because of O, she can then go to B to revoke the corresponding coin and get a refund.

Next, we address E-protection. According to our analysis, it is independent of link reliability or trust assumptions, and solely relies on E's behaving compliantly. In Brands's ecash system, E is, thus, much less vulnerable than P.

5.7.2 Protection under Compliance Mode

In this subsection, we analyze the protocol under compliance mode. To verify whether the protocol is all-protective under compliance mode, we need to verify whether both P's and E's interests are protected in maximal compliant executions. These results have already been verified in Subsection 5.7.1, however. According to Prop 5.22 and 5.25,

Theorem 5.28 *Brands's payment protocol is all-protective under compliance mode, if the communication channel between P and E is reliable.*

Discussion

Brands's payment is not all-protective under compliance mode. Like under deception mode, it is E-protective, but not P-protective. It is E-protective because E depends on nothing other than his own compliance to the protocol to protect his interests. It is not P-protective because P still risks losing the only rights certificate she can generate due to unreliable communication channels.

5.7.3 Protection under Abortion Mode

In this subsection, we analyze the protocol under abortion mode. Like in the two previous subsections, we need to verify whether the protocol is both P-protective and E-protective. We start with P-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, to verify whether Π protects P's interests under abortion mode, we need to examine all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_P$. Here we focus on abortive executions only, since we have analyzed compliant executions (Prop. 5.22) in Section 5.7.1.

Proposition 5.29 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)^A_P$, $\mathcal{T}$ be the trust assumptions Π makes, and P^p_P be P's protection property as specified in Section 5.4.2 (p. 115). Then*

$$\sigma \models^* \mathcal{T} \ \ \textit{implies} \ \ \sigma \models^* P^p_P.$$

Analysis: A quick glance over the property P^p_P allows us to conclude that this proposition does not hold: E can terminate his execution right before receiving the rights certificate from P, preventing Φ^p_{E1} from ever becoming true.

□

From Prop. 5.22 (Section 5.7.1) and 5.29, which jointly address P-protection under abortion mode, we derive the following

Corollary 5.30 *Brands's payment protocol is not P-protective under abortion mode, even if the communication channel between P and E is reliable.*

Prop. 5.31 addresses protection of E's interests in abortive executions and can be proven using the proof for Prop. 5.24 (Subsection 5.7.1). That proof is applicable here for a reason analogous to the one given for Prop. 5.25.

Proposition 5.31 *Let Π be Brands's payment protocol, σ be an execution in $E(\Pi)^A_E$, and P^p_E be E's protection property as specified in Section 5.4.2 (p. 115). Then*

$$\sigma \models^* P^p_E.$$

From Prop. 5.25 (Section 5.7.1) and 5.31, which jointly address E-protection under abortion mode, we derive the following

Corollary 5.32 *Brands's payment protocol is E-protective under abortion mode.*

Theorem 5.33 summarizes the results in this subsection.

Theorem 5.33 *Brands's payment protocol is not all-protective under abortion mode, even if the communication channel between P and E is reliable.*

Proof: From Cor. 5.30 and 5.32.

□

Discussion

Brands's payment protocol is not all-protective under abortion mode. Like in compliance and deception modes, it is E-protective, but not P-protective. It is E-protective because E's protection property is only concerned with the validity of what E receives, which is not affected by P's or O's premature terminations.

Brands's payment protocol is not P-protective, however; P's interests can be compromised not only by unreliable communication channels, but also by E's premature termination. Premature termination models, among other things, E's refusal in receiving a rights certificate for reasonable (e.g., transaction timeout) or unreasonable reasons. To safeguard her interests against E's premature termination, P should get a commitment from E with respect to accepting the rights certificate before she asks O to provide the protocertificate. A proof of this commitment will force E to receive the rights certificate eventually, or hold him accountable for P's coin loss.

Note that the problem only arises if O deletes ν_o, but E does not receive the rights certificate sent by P. O may not delete ν_o, however, if O is compromised. Not having ν_o deleted means that P is able to provide another rights certificate for the coin, which makes the loss of the current one inconsequential. This interesting relationship between ν_o's deletion and protection of P's interests is captured in our specification of P^p_P (Section 5.4.2). In an execution where ν_o is not deleted, P^p_P is vacuously true because Φ^p_P is never true.

5.7.4 Summary

We summarize our findings in Table 5.2. All-protection is guaranteed in entries with a $\surd$. Some entries come with briefings of relevant facts.

	compliance	abortion	deception
Channel Failures	(1)	(1)	(1)
No Channel Failures	$\surd$	(2)	(3)

1. Rights certificates sent by P may get lost in transmission and not reach E.

2. E may terminate his execution prematurely, and fail to receive the rights certificate sent by P. P-protection is violated only in executions where O deletes ν_o, however.

3. P will generate a bogus rights certificate if E's challenge is bogus. According to T_{O_p}, O does not deceive.

Table 5.2: Summary table for the payment protocol.

As is shown in Table 5.2, Brands's payment protocol is quite vulnerable: it is all-protective only if all parties behave compliantly and the communication channels are reliable. P and E are not equally vulnerable, however. In fact, only P-protection is susceptible to attacks; the protocol is E-protective in all the cases.

P's vulnerability results from her ability to generate only one rights certificate per coin. Unless this one-time capability is used to generate a valid certificate that is eventually received by E, the value borne by the corresponding coin would not be preserved and passed to E. The problem is that neither the generation nor the delivery of this certificate is entirely under P's control. E can make P generate a bogus certificate by sending her a bogus challenge (deception mode); or E can make P generate a certificate, and disappear before receiving it (abortion mode); finally, a valid certificate may simply not reach a willing E because of unreliable communication links.

Note that P would be less vulnerable if she could generate multiple rights certificates per coin. Unsuccessful spendings of a coin – due to bogus or missing certificates – would not matter in this case since she can try again. Of course, this also means that a dishonest P could spend a coin a second time even if the first attempt were successful.

While making P less vulnerable, P's ability to generate multiple rights certificates per coin does not (arguably) bring unrecoverable losses to E. If E receives a coin that has been double-spent, the double-spender's identity can presumably be revealed, allowing E to collect the due amount off-line. Of course P may simply disappear and make these off-line collections impossible; or these collections may be an overhead that E is unable or unwilling to incur.

Thus, whether or not to allow multiple generations of rights certificates may be best determined by the characteristics of a system (type of participants, reliability of system components, etc). For example, if the payees of a system are well-established merchants who will not misbehave, then the one-time restriction is not as harmful. On the other hand, if the communication channels are unreliable in a system, but all participants are honest, then allowing multiple generations of rights certificates seems more reasonable.

5.8 Analysis of The Deposit Protocol

5.8.1 Protection under Deception Mode

In this subsection, we analyze Brands's deposit protocol under deception mode. We first analyze it with respect to B-protection, then with respect to E-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, if Π is Brands's deposit protocol, then to analyze it with respect to B-protection under deception mode, we need to analyze all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^D_B$. We do so in Prop. 5.35 and 5.34 respectively.

Proposition 5.34 *Let Π be Brands's deposit protocol, σ be an execution in $E(\Pi)^D_B$, and P^d_B be B's protection property as specified in Section 5.4.3 (p. 118). Then*

$$\sigma \models^* P^d_B.$$

Proof: In what follows, Φ^d_{B1} and Φ^d_{B2} are subformulas of P^d_B as specified in Section 5.4.3; s_i is a state in σ; and m_1: *t_subs* and m_2: *t_seal* are messages such that

$$s_i \models \Phi^d_{B1}[m_1/x_1, m_2/x_2]. \tag{5.19}$$

We would like to prove that

$$s_i \models \text{vseal}(m_1, m_2, k_b) \ \wedge \ m_1.\text{comp}_1 \not\in^* \Delta.$$

1. From expression 5.19, we have

   $$s_i \models \text{receive}(E, m_1 m_2 - --) \in \mathtt{H}^\mathtt{B} \ \wedge \ \text{credit}(-) \in \mathtt{H}^\mathtt{B},$$

   where credit(-) must have resulted from an application of RD_{B_2}.

2. If RD_{B_2} was applied, then its antecedent must be true at a state s_j, $j < i$. And given that there can be only one event of type receive(E, x), x : *t_subs* $\times$ *t_seal* $\times$ *t_cs* $\times$ *t_rcert*, in σ (Lemma 5.2), it must be the case that

   $$s_i \models \text{vseal}(m_1, m_2, k_b) \wedge \ \nexists x_5 : \textit{t\_deposit} \in \Delta \mid x_5.\text{subsComp} = m_1.\text{comp}_1. \tag{5.20}$$

3. Since each entry in the deposit database only records one message of type *t_subs*$_1$, and this message is recorded in the field "subsComp", we can conclude from expression 5.20 that

   $$s_i \models \text{vseal}(m_1, m_2, k_b) \ \wedge \ m_1.\text{comp}_1 \not\in^* \Delta.$$

□

Bearing in mind what Φ^d_{B1} and Φ^d_{B2} specify, the proof of Prop. 5.34 shows that if B accepts a coin for deposit, then the coin is valid and has not been deposited before. This result is not surprising because, even though E may try to deposit invalid coins or coins that have been deposited before, B can check for these conditions, and reject the deposit if something is wrong.

Note that we analyzed all executions in $E(\Pi)^D_B$, and did not restrict ourselves to just trustworthy executions. In reality, all executions in $E(\Pi)^D_B$ are trustworthy in this case, because T_{B_d} is the only trust assumption here, and all executions in $E(\Pi)^D_B$ satisfy it.

Prop. 5.35 concerns maximal compliant executions. Our proof of Prop. 5.34 is readily applicable here because it only requeries that σ has compliant local executions of B, a condition satisfied by maximal compliant executions.

Proposition 5.35 *Let Π be Brands's deposit protocol, σ be an execution in $E(\Pi)^C$, and P_B^d be B's protection property as specified in Section 5.4.3 (p. 118). Then*

$$\sigma \models^* P_B^d.$$

From Prop. 5.34 and 5.35, we can derive the following

Corollary 5.36 *Brands's deposit protocol is B-protective under deception mode.*

Next we address E-protection. As with the analysis of B-protection, we need to analyze both deceptive and compliant executions. We start with deceptive ones.

Proposition 5.37 *Let Π be Brands's deposit protocol, σ be an execution in $E(\Pi)_E^D$, $\mathcal{T}$ be trust assumptions Π make, and P_E^d be E's protection property as specified in Section 5.4.3 (p. 118). Then*

$$\sigma \models^* \mathcal{T} \quad \textit{implies} \quad \sigma \models^* P_E^d.$$

Analysis: Let σ be a trustworthy execution in $E(\Pi)_E^D$, and s_i be a state in σ where there exist messages m_1 : *t_subs*, m_2 : *t_seal*, m_3: *t_cs* and m_4 : *t_rcert*, such that

$$s_i \models \Phi_{B3}^d[m_1/x_1, \ldots, m_4/x_4]. \tag{5.21}$$

We would like to prove that there exists a state s_j in $\sigma, j \geq i$, such that

$$s_j \models \text{credit}(ac_e) \in \mathtt{H}^{\mathtt{B}} \;\vee\; \theta_p \in^* \mathtt{MS}^{\mathtt{B}}.$$

1. Expression 5.21 says that

$$\begin{aligned} s_i \quad \models \quad & \text{receive}(E, m_1 \ldots m_4) \in \mathtt{H}^{\mathtt{B}} \;\wedge\; \text{vseal}(m_1, m_2, k_b) \;\wedge \\ & \text{vrcert}(m_4, \text{chal}(ac_e.\text{id}, m_3), m_1) \;\wedge\; m_4 \not\in^* \Delta. \end{aligned}$$

   Now, let $s_{i'}, i' \leq i$, be a state in σ such that

$$s_{i'} \models \text{last}(\mathtt{H}^{\mathtt{B}}) = \text{receive}(E, m_1 \ldots m_4).$$

   There are two possibilities:

   (a) $\nexists x_5 : \textit{t\_deposit} \in \Delta \mid x_5.\text{subsComp} = m_1.\text{comp}_1$, or

   (b) $\exists x_5 : \textit{t\_deposit} \in \Delta \mid x_5.\text{subsComp} = m_1.\text{comp}_1$.

   We examine each in turn.

2. If (1a) is true, then the event credit(ac_e) will eventually take place, according to trust assumption T_{B_d}. That is, there exists a state $s_{i''}$ in $\sigma, i'' > i'$, such that

$$s_{i''} \models \text{credit}(ac_e) \in \mathtt{H}^{\mathtt{B}}.$$

3. If (1b) is true, then B terminates its local execution without the event credit(ac_e). We show below that this scenario does not guarantee that θ_p be revealed.

4. Let e: *t_deposit* be the entry in the deposit database such that

$$s_{i'} \models e.\text{subsComp} = m_1.\text{comp}_1. \tag{5.22}$$

Given that $m_4 \notin^* \Delta$, it must be the case that

$$e.\text{rc} \neq m_4.$$

Apparently B can apply the function reveal_sec to messages e.rc, m_4, and θ_o to reveal P's account secret θ_p. But the equality reveal_sec(e.rc, m_4, θ_o) $= \theta_p$ holds only if e.rc and m_4 are rights certificates concerning the same coin, which is not guaranteed by the equality 5.22 if P can deceive.

We conclude that this proposition does not hold.

□

Prop. 5.37 does not hold. Bearing in mind what Φ^d_{B3} and Φ^d_E specify (Section 5.4.3), its analysis shows that, when B receives a valid coin and a not-yet-submitted rights certificate from a rightful E, it is not necessary that E's account will be credited or P's identity will be revealed. Intuitively, this means that a coin that passes all verifications recommended in the payment protocol can still be worthless.

This result is unexpected, because it contradicts what is claimed in [14]. According to [14], there are three possible outcomes when E tries to deposit a coin that he has rights to (Section 5.2.4). If the coin is not found in the deposit database, B simply deposits it, and E's account is credited. If the coin is found in the deposit database, but its accompanying rights certificate from the database differs from the one being submitted, then B uses the two rights certificates to reveal the account secret embedded in them, thus retrieving P's identity. Finally, if both the coin and the rights certificate being submitted are found in the deposit database, B simply ignores this deposit attempt. Since Prop. 5.37 assumes that the rights certificate being submitted had not been submitted before, we should have been able to conclude that the deposit attempt yields the first or the second outcomes. That is, either E's account is credited or P's identity is revealed.

This contradiction results from an inconsistency between what can actually happen in the system under deception mode and what Brands assumes. To follow the rest of this discussion, the reader should be familiar with the material presented in Sections 5.2.2, 5.2.3, and 5.2.4.

Concretely, Brands assumes that two valid coins are identical if the first components of their substrates are identical. That is, if (u_1, u_2) and (u'_1, u'_2) are respectively the substrates of two valid coins c and c', then

$$u_1 = u'_1 \;\rightarrow\; c = c'.$$

Or simply,

$$u_1 = u'_1 \;\rightarrow\; (u_1, u_2) = (u'_1, u'_2). \tag{5.23}$$

This is true if substrates are always generated as prescribed. According to the withdrawal protocol (Section 5.2.2), all coin-specific secrets (ν_{1p}, ν_{2p}, and ν_o) embedded in substrates should be randomly generated for each new substrate. Since no two randomly generated numbers are equal, it is impossible to have two different substrates sharing a same ν_{1p}, and therefore a same first component. That is,

$$(u_1, u_2) \neq (u'_1, u'_2) \;\rightarrow\; u_1 \neq u'_1,$$

which is equivalent to expression 5.23. We know that Brands makes the assumption above because B's deposit database only keeps the first component of a coin's substrate when the coin is deposited.

Withdrawers can deceive, and not generate substrates exactly as prescribed, however. For example, they may embed in a new substrate both secrets generated at random and ones already used in other substrates. Thus, it is possible for P to generate a substrate (u'_1, u'_2) where

$$\begin{cases} u'_1 = subs_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{ and} \\ u'_2 = subs_2(\nu_{1p}, \nu'_{2p}, \overline{\theta_o}, \overline{\nu'_o}), \end{cases}$$

with randomly generated ν'_{2p} and ν'_o, and a "recycled" ν_{1p} from a pre-existing substrate (u_1, u_2),

$$\begin{cases} u_1 = subs_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{ and} \\ u_2 = subs_2(\nu_{1p}, \nu_{2p}, \overline{\theta_o}, \overline{\nu_o}). \end{cases}$$

In this case, we have

$$(u_1, u_2) \neq (u'_1, u'_2) \ \wedge \ u_1 = u'_1.$$

Prop. 5.37 does not hold exactly because of the deception depicted above. More specifically, B would not credit E's account or be able to reveal P's identity if the coin being submitted for deposit has not been deposited before, but shares the first component of its substrate with an already deposited coin. Fig. 5.6 shows two such coins and their accompanying rights certificates.

$$c: \quad \text{Substrate}: (u_1, u_2), \begin{cases} u_1 = \mathrm{subs}_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{ and} \\ u_2 = \mathrm{subs}_2(\nu_{1p}, \nu_{2p}, \overline{\theta_o}, \overline{\nu_o}), \end{cases}$$

$$\text{Accompanying rights certificate}: rc = \mathrm{rcert}(\mathrm{pcert}(ch, \theta_o, \nu_o), \theta_p, \nu_{1p}, \nu_{2p})$$

$$c': \quad \text{Substrate}: (u'_1, u'_2), \begin{cases} u'_1 = \mathrm{subs}_1(\overline{\theta_p}, \nu_{1p}, \overline{\theta_o}), \text{ and} \\ u'_2 = \mathrm{subs}_2(\nu_{1p}, \nu'_{2p}, \overline{\theta_o}, \overline{\nu'_o}), \end{cases}$$

$$\text{Accompanying rights certificate}: rc' = \mathrm{rcert}(\mathrm{pcert}(ch', \theta_o, \nu'_o), \theta_p, \nu_{1p}, \nu'_{2p})$$

Figure 5.6: Examples of two different coins sharing the first component of their substrates.

If c has been deposited, then B will not accept c' for deposit and credit E's account because $u_1 = u'_1$. But B cannot apply the function reveal_sec (Def. 5.1) to rc and rc' to reveal θ_p either, because unless rc and rc' differ only in the challenge they embed, reveal_sec(rc, rc', θ_o) does not yield θ_p. In Fig. 5.6, rc and rc' embed not only different challenges ($ch \neq ch'$), but also some different coin-specific secrets ($\nu_o \neq \nu'_o$ and $\nu_{2p} \neq \nu'_{2p}$).

Because Prop. 5.37 does not hold, we can skip analyzing compliant executions, and conclude that

Corollary 5.38 *Brands's deposit protocol is not E-protective under deception mode.*

Theorem 5.39 summarizes the results in this subsection:

Theorem 5.39 *Brands's deposit protocol is not all-protective under deception mode.*

Proof: From Cor. 5.36 and 5.38.

□

Discussion

According to our analysis, Brands's deposit protocol is B-protective under deception mode. This is not surprising, because B has complete control over the depositing procedure, and cannot be tricked into depositing invalid coins or coins that have been deposited before.

What is surprising are that the deposit protocol is not E-protective and why it is not. According to our analysis, E-protection is not compromised by unreliable communication channels or B's deceptions. It is not affected by unreliable communication channels because E can keep a copy of what he submits to B, and re-submit it later if his submission gets lost in transmission. It is not affected by B's deceptions because B is trusted not to commit the only deception that could compromise E-protection.

Instead, E-protection is compromised by P's deception earlier in the withdrawal protocol, when the coin was withdrawn! More specifically, P can deceive and withdraw a perfectly valid coin that B *may* not accept for deposit, and whose accompanying rights certificate *may* not be useful to reveal P's identity when E tries to deposit the coin. The discussion following the analysis of Prop. 5.37 shows an example of such a coin. We say *may* instead of *will* because what actually happens depends on the result of a race condition involving coins c and c' in Fig. 5.6. If c is submitted for deposit first, c' is useless. However, if c' is submitted first, it will be deposited, and c will be the useless one.

Note that this coin is interesting because it is a perfectly valid new coin at its generation, even though P has deceived in its generation. P herself does not benefit from the deception, because she "pays for" the coin when she withdraws it (her account is deducted), and she is able to spend it. E is the only one hurt in this case because the coin he received is completely useless. Thus, this deception is effective only for sabotage, but not for bringing P monetary gains.

Finally, even though E-protection is compromised by P's deception in the withdrawal protocol, the real problem lies in the design of the deposit protocol, and can be fixed simply by using both components of a coin's substrate to tell whether or not a coin has been deposited. B would need to record whole substrates in the deposit database in the fixed protocol.

5.8.2 Protection under Compliance Mode

In this subsection, we analyze the protocol under compliance mode. To verify whether the protocol is all-protective under compliance mode, we need to verify whether both B's and E's interests are protected in maximal compliant executions. Prop. 5.35 (Section 5.8.1) shows that Brands's deposit protocol is B-protective in maximal compliant executions; Prop. 5.40 shows that it is E-protective.

Proposition 5.40 *Let Π be Brands's deposit protocol, σ be an execution in $E(\Pi)^C$, and P_E^d be E's protection property as specified in Section 5.4.3 (p. 118). Then*

$$\sigma \models^* P_E^d.$$

Proof: The first part (steps 1 - 3) of this analysis is identical to that of Prop 5.37. Under compliance mode, we reach a different conclusion:

4. Let e: *t_deposit* be the entry in the deposit database such that

$$s_{i'} \models e.\text{subsComp} = m_1.\text{comp}_1. \tag{5.24}$$

Given that $m_4 \not\in^* \Delta$, it must be the case that

$$e.\text{rc} \neq m_4.$$

B can now apply the function reveal_sec to messages e.rc, m_4, and θ_o to reveal P's account secret θ_p. We know that reveal_sec(e.rc, m_4, θ_o) $= \theta_p$ because e.rc and m_4 are rights certificates concerning the same coin. We know this last fact because of the equality in expression 5.24, which under compliance mode means that m_1 and the coin associated with entry e are identical.

□

From Prop 5.35 and 5.40, we can now conclude

Theorem 5.41 *Brands's deposit protocol is all-protective under compliance mode.*

Discussion

Brands's deposit protocol is all-protective under compliance mode. Unlike under deception mode, the protocol is E-protective here. It is E-protective because, when P behaves compliantly, two coins that share the first components of their substrates are always identical. This identity prevents the problem-causing scenario that was possible under deception mode from occurring here, and allows us to reach the following conclusion. When a coin is submitted for deposit, the first component of its substrate can or cannot be found in the deposit database. In the first case, the coin is accepted for deposit and E's account is credited. In the second case, the coin has actually been deposited before, and the two available rights certificates for this coin can be used to reveal θ_p.

5.8.3 Protection under Abortion Mode

In this subsection, we analyze the protocol under abortion mode. Like in the two previous subsections, we need to verify whether the protocol is both B-protective and E-protective. We start with B-protection.

According to Defs. 2.25 and 2.26 in Section 2.4.3, to verify whether Π protects B's interests under abortion mode, we need to examine all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_B$. Here we focus on abortive executions only, since we have analyzed compliant executions (Prop. 5.35) in Section 5.8.1.

Prop. 5.42 can be proven using our proof for Prop. 5.34, since that proof only depended on the fact that B behaves compliantly.

Proposition 5.42 *Let Π be Brands's deposit protocol, σ be an execution in $E(\Pi)^A_B$, and P^d_B be B's protection property as specified in Section 5.4.3 (p. 118). Then*

$$\sigma \models^* P^d_B.$$

From Prop. 5.35 (Section 5.8.1) and 5.42, which jointly address B-protection under abortion mode, we derive the following

Corollary 5.43 *Brands's deposit protocol is B-protective under abortion mode.*

We address E-protection next. According to Defs. 2.25 and 2.26 in Section 2.4.3, to verify whether Π protects E's interests under abortion mode, we need to examine all executions in $E(\Pi)^C$ and trustworthy executions in $E(\Pi)^A_E$. Here we focus on abortive executions only, since we have analyzed compliant executions (Prop. 5.40) in Section 5.8.2.

Prop. 5.44 can be proven using our proof for Prop. 5.40, since that proof only depended on B's behaving as trusted.

Proposition 5.44 *Let Π be Brands's deposit protocol, σ be an execution in $E(\Pi)^A_E$, and P^d_E be E's protection property as specified in Section 5.4.3 (p. 118). Then*

$$\sigma \models^* P^d_E.$$

E's protection property is satisfied by all executions in $E(\Pi)^A_E$ because, according to T_{B_d}, the only termination that could violate P^d_E (B's terminating its local execution right before it should credit E's account) does not happen.

From Prop. 5.40 (Section 5.8.2) and 5.44, which jointly address E-protection under abortion mode, we derive the following

Corollary 5.45 *Brands's deposit protocol is E-protective under abortion mode.*

Theorem 5.46 summarizes the results in this subsection.

Theorem 5.46 *Brands's deposit protocol is all-protective under abortion mode.*

Proof: From Cor. 5.43 and 5.45.

□

Discussion

Brands's deposit protocol is all-protective under abortion mode. B-protection depends only on B's compliance to the protocol, while E-protection relies on B's behaving as trusted. If B behaves as trusted, then it will not terminate its execution in the middle of processing deposit requests, and will always credit E's account if the coin being submitted has not been deposited before.

5.8.4 Summary

We summarize our findings in Table 5.3. All-protection is guaranteed in entries with a $\surd$; E-protection relies on T_{B_d} in entries with a $*$. The table shows that all-protection does not depend on reliable communication channels, and trust assumptions are needed only under abortion mode. Note that Brands's deposit protocol is not all-protective only under deception mode, where E's protection property can be violated.

	compliance	abortion	deception
Channel Failures	$\surd$	$\surd *$	(1)
No Channel Failures	$\surd$	$\surd *$	(1)

1. B-protection is guaranteed. E-protection is violated by P's deceptions earlier in the withdrawal protocol, when the coin was withdrawn.

Table 5.3: Summary table for the deposit protocol.

In all entries, the protocol is B-protective simply because B has complete control over the deposit procedure, and can detect all of E's deviations before accepting the deposit request. The protocol is E-protective under both compliance and abortion modes. Under deception mode, however, the protocol is not E-protective even if B behaves as trusted. In fact, E-protection is not compromised by B's deviations in this case. It is compromised by P's deceptions earlier in the withdrawal protocol, when the coin was withdrawn.

5.9 Conclusion

Of our three case studies, Brands's protocol is the most challenging. It is challenging because there has been no previous studies of ecash protocols with respect to fairness or protection of individuals' interests, and we had to devise our own approach to tackle this problem. For example, since protection of individuals' interests is a transaction-related concept, and Brands's protocol implements different types of transactions, we do not analyze the protocol as a monolithic unit. Instead, we focus on each of the different types of transactions, and analyze each of the subprotocols separately. Also, we had to propose protection properties for these subprotocols afresh.

Dividing up the protocol for analysis does not make the subsequent analysis effort and its results trivial or insignificant: the results we obtained and the insights we gained prove it. Besides standing on their own as protection results, the outcomes of these analyses provide insights as to whether or not the protocol satisfies certain other properties. For example, violation of P's interests in the payment protocol allows us to conclude immediately that Brands's protocol is not money-atomic [84]. We suspect that treating ecash subprotocols separately may be a feasible divide-and-conquer approach to master the complexity of defining protocol-wide properties, e.g., money atomicity, and analyzing protocols with respect to these properties.

As for the protection properties we proposed for the three subprotocols, we do not claim that they are absolute, but we believe that they capture the essence of what different parties see as their interests, and when these interests are protected. More than trying to establish a set of absolute protection properties for ecash protocols, we aimed to show that the notion of protection is applicable to more than just exchange protocols.

Of all the problems detected by our analysis, no one is more subtle than the violation of E's protection property in the deposit protocol. It is so subtle that our analysis is the first to reveal it since Brands's protocol [14] was first published. We attribute our success in revealing it to the use of our analysis framework.

The two trust assumptions we identified for B are not surprises. In withdrawal transactions, B is expected to generate signatures (T_{B_w}) correctly, while in deposit transactions B is expected to deposit coins that are worth money (T_{B_d}).

The trust assumptions we identified for O are somewhat unexpected. Used to restrain double-spending, it would seem more intuitive if the trust placed on O was about not enabling multiple spendings. However, since the protocol has a safety net to handle potential failures of the restraining mechanism, no trusted behavior is required of O in this respect. In contrast, P depends critically on O to generate a spendable coin and then to spend it. Thus, the trust assumed of O concerns O's generating its contribution to the substrates (T_{O_w}) properly, and O's providing the protocertificates (T_{O_p}) properly. Violation of either T_{O_w} or T_{O_p} will make a coin non-spendable, and P would lose money.

Finally, note that all-protection is not complete security. Even if all subprotocols of an ecash protocol were all-protective, the entire protocol may still be insecure. For example, none of our protection properties addresses whether payers are *eventually able to spend* spendable coins they own; a protocol is clearly insecure if it fails to guarantee that one can eventually spend a spendable coin that one withdraws.

5.10 Feedback from Stefan Brands

In this section, we report on Brands's comments on our analysis [17].

First, there seems to be different notions of what lies within the scope of a cryptographic protocol design. For Brands [17], fault-tolerance and the like are not core issues that should be addressed within the design of a cryptographic protocol. Because of this view, he argues that some of the weaknesses we point out are not weaknesses of the cryptographic protocol; instead, they relate to fault-tolerance. Violations of P-protection in both the withdrawal protocol and the payment protocol due to channel failures (entries (1) in Table 5.1 (p. 127) and Table 5.2 (p. 135)) fall into this category. Fault-tolerance measures needed to address these violations are discussed by Brands himself elsewhere [15].

Other violations of P-protection in the payment protocol (entries (2) and (3) in Table 5.2 (p. 135)) are due to E's abortion and deception. A return protocol [16] that allows P to return the affected coin is needed in these cases.

Finally, protection of E's interests in the deposit protocol (entries (1) in Table 5.3 (p. 143)) can be guaranteed if B always deposits valid coins that are accompanied by fresh rights certificates [15]. Under this mode of operation, problem coins that were rejected for deposit are now accepted, and E-protection is restored.

Chapter 6

Conclusions and Future Work

The general goal of this research is to answer questions such as: What do electronic commerce protocols try to achieve? What must they achieve? And how do they achieve it? My thesis in this dissertation is that 1) in electronic commerce transactions where participants have different interests to preserve, protection of individuals' interests is a concern of the participants, and should be guaranteed by the protocols; and 2) a protocol should protect a participant's interests whenever the participant behaves according to the protocol and trusted parties behave as trusted.

To make this thesis precise, we formulated a model for electronic commerce systems and gave a definition of protection of individuals' interests in this model. To demonstrate the applicability of our model and to investigate how well electronic commerce protocols do with respect to this requirement, we analyzed three protocols using our framework.

In the rest of this chapter, we first summarize and reflect on the results (Section 6.1). We then discuss future work (Section 6.2). Finally, we provide a few remarks on formal methods research as applied to electronic commerce protocols (Section 6.3).

6.1 Summary and Reflections of Results

We discuss our framework and the case studies in turn.

6.1.1 The Framework

Our framework for analyzing protocols with respect to protection of individuals' interests is model-theoretic. It consists of a protocol specification formalism, a model of electronic commerce systems, and a definition of p-protective protocols (where p is a protocol participant). Our protocol specification formalism is a standard rule-based formalism in which one can specify trust assumptions of a protocol in linear-time temporal logic. Our model consists of standard state machines with which we can distinguish some particular types of protocol executions: compliant, abortive, deceptive, and trustworthy. Finally, our definition of p-protective protocols establishes the set of executions one should consider in order to conclude whether a protocol protects the interests of a participant p.

Using our framework, one can examine a protocol under three deviation modes: 1) compliance mode, where the participants execute according to the protocol; 2) abortion mode, where they can terminate their executions prematurely; and 3) deception mode, where they can send bogus messages. Trusted parties can deviate as well, but they do not violate a protocol's trust assumptions.

That is, their executions are always trustworthy. Under each deviation mode, one can consider both reliable and unreliable communication links. These deviation modes do not cover all possible deviations, but they model attacks that do not require many resources or much sophistication, which constitute most of the attacks on electronic commerce protocols [1].

This framework is useful because it builds on well-known and simple models and formalisms and enables investigation of an assortment of electronic commerce protocols with respect to a novel and critical property.

Also, since protection of individual interests is a very general notion and we do not specify what exactly can be protection properties, our framework is potentially widely applicable. For example, it seems to be readily applicable to formalize and analyze protocols with respect to abuse-freeness [43], a property recently introduced for contract signing protocols. A contract signing protocol is abuse-free if no party can ever prove to a third party that he or she is capable of choosing whether to validate or invalidate a contract. Abuse-freeness is clearly an instance of protection of individuals' interests, and to analyze a protocol with respect to this property, one simply needs to specify the corresponding protection properties in temporal logic. Abstractly, protection properties corresponding to abuse-freeness have the following general pattern:

> Let p and q be parties signing a contract. If p signs the contract at a state s, then there cannot be a state in the future in which q can prove that she can either validate the contract or invalidate it.

6.1.2 The Case Studies

We analyzed three protocols using our framework.

Franklin and Reiter's Protocol

This protocol is the simplest among the three we analyzed, in terms of both its functionality and its protection properties. Our analysis did not reveal any surprises: the protocol is all-protective under all three deviation modes, as long as communication links are reliable.

Our main contributions in this case study is a formalization of semi-trusted third parties. Franklin and Reiter introduced the notion of semi-trustworthiness in electronic commerce protocols [42], but they did not fully develop it. In particular, they did not take it into account in their (informal) analysis of the protocol. Using our framework, we could formalize the notions of semi-trustworthiness and conspiracy behaviors, and provide a clear-cut analysis of the protocol.

NetBill

This protocol is the second in complexity (among the three we analyzed), but its protection properties are the most complex. The complexity stems from the fact that NetBill is a protocol that supports dispute resolution, and its protection properties need to take into account not only on-line exchanges of money and goods, but also how disputes are resolved.

In terms of protection of individuals' interests, our analysis did not reveal any surprises. NetBill is both customer-protective and merchant-protective under all three deviation modes, even when communication links can fail. NetBill is not affected by link failures because the NetBill server is assumed to be permanently reachable.

Our analysis, however, did make explicit interesting points about how the protocol guarantees all-protection, and what the NetBill server is trusted to do. Under compliance and abortion modes, all-protection is guaranteed by the server's transaction capabilities alone; certified delivery is needed only under deception mode. More interestingly, certified delivery achieves its goals only if the NetBill server satisfies a small set of trust assumptions. Basically, the server is entrusted to handle accounts and keys honestly; to provide unique and non-repudiable proofs of what happens on-line through transaction slips; and to allow merchants to learn what really happens with a transaction – so that keys are not given to customers without their knowledge. Even though it may seem counter-intuitive at first, NetBill's all-protection does not depend on the server's releasing the keys sent to it by merchants, or its retaining transaction requests.

Brands's Protocol

This case study is the most interesting for three reasons. First, we had to derive an abstract version of the protocol from its purely mathematical formulation [14]. This exercise is challenging because the protocol is very complex (not only compared to the other two protocols, but in absolute terms), and it is not immediately clear how abstract we should model the protocol.

Second, to our knowledge, there has not been analysis of ecash protocols with respect to protection of individuals' interests. This means that there is no standard protection properties for these protocols, and we had to propose them afresh. The properties we proposed might not be definitive, but we believe that they capture the essence of what different parties see as their interests. More than trying to establish a set of absolute protection properties for ecash protocols, we aimed to show that parties in such protocols do have different interests to preserve, and the notion of protection of individuals' interests is, in fact, applicable to more than just exchange protocols.

Finally, our analysis revealed a number of weaknesses in the protocol. Some of the weaknesses are well-known and far from subtle; for example, that ecoins can get lost during transmission. Others, however, are quite subtle, and have not been found before. For example, a payee can either abort or deceive, and effectively make the payer's money disappear, even if communication links are reliable. Also, withdrawers can deceive, and withdraw perfectly valid coins that they can spend, but will be neither accepted for deposit, nor usable for tracing the identity of the withdrawer.

Brands does not make clear his assumptions about how different parties can misbehave, and arguably did not design the protocol to counteract the attacks we found. In any case, these attacks are realistic, pose real threats to the users, and should be taken into account.

We were able to unveil these weaknesses because we have a well-defined deviation model in which protocols can be systematically analyzed. This testifies to the importance of systematization and formalization that go into formulating frameworks such as ours.

Our analysis gave us other interesting insights. For example, it showed how vulnerable payers become when observers are corrupted. While there is a whole mechanism for tracing double-spenders should an observer fail to prevent double-spending, there is no recourse for the payer to recover her money, if the observer fails to authorize a rightful spending of a valid coin.

The discussion above is based on our analysis of Brands's protocol as it was presented in his Crypto '93 paper [14]. According to our recent personal communication with Brands [17], that paper presents only the core cryptographic protocol; it does not include fault tolerance/recovery measures discussed in Section 5.10, which Brands considers outside the scope of protocol designs.

Even though the protocol we analyzed is incomplete and the weaknesses we found are not present in the complete protocol, our analysis is still valuable. It shows that protection of individuals' interests indeed captures a notion of security that should be guaranteed by electronic cash protocols. It also shows that our framework can be effectively used to find problems related to this type of security in protocols.

6.2 Future Work

There are several directions along which we can further develop this research. The main ones are discussed below.

Analysis Automation

The most pressing future work is automation. We analyzed all three protocols by hand, and the process is tedious and slow. An automatic (or semi-automatic) analyzer would greatly improve the efficiency of the analysis process, in addition to minimizing proof errors that are common in manual analyses.

It is possible to model our framework in existing general-purpose tools. Schneider's analysis [78] of Zhou and Gollman's non-repudiation protocol [89] using CSP [52] is an indication that our analyses can be carried out, in principle, in FDR [64]. However, since FDR is a trace-based model checker, mapping what we have developed to FDR is not straightforward. A state-based model checker that uses temporal logic as specification language would require less effort.

Model checkers give us a greater degree of automation, but theorem provers allow/force us to be more explicit and precise in our models and specifications. Since our framework emphasizes explicitly making assumptions that usually remain implicit, automation based on theorem proving is preferable for our purposes.

Alternatively, we can build a special-purpose tool, custom-built for investigating protection of individuals' interests. A good argument for building such a tool is that it will incorporate our model and definition, and, in the case of model checkers, automatically generate different sets of executions that need to be considered.

Further Exploration Using Our Framework

A different direction for future work is to use the framework that we now have in place to explore further the issue of protection of individuals' interests in electronic commerce protocols. One possibility is to investigate classes of protocols that we have not looked at. Voting and auction protocols are types of protocols we would investigate next.

Another possibility is to deepen the analyses we have done, possibly by strengthening or adding to current protection properties. For example, does the server in NetBill have interests to preserve in a transaction? If yes, what are these interests? We suspect that, as a service provider, the NetBill server may or may not have monetary or material interests in transactions; however, it certainly has its reputation to preserve. In the case of ecash protocols, are there other interests that payers may want to preserve in addition to preserving the value of their coins? Protection of privacy, which we did not address in this dissertation, is something that payers certainly care about. But is privacy a different type of property altogether? Or is it simply a subtype of individuals' interest? These questions are worth investigating.

Enriching the Framework

The framework itself can be further extended. One obvious extension is to add other deviation modes. For example, we can add a more permissive type of deception.

The type of deception we defined in this dissertation (Definition 2.16) is stringent in that a deceptive step does not violate the conditions for the firing of the protocol rule. These conditions typically include two components: 1) a specification of protocol steps that should or should not have taken place, and 2) a specification of conditions that these previous steps must satisfy. In rule R_{Z_4} in Franklin and Reiter's protocol (page 37), for example, everything except line 3 falls into the first component, while line 3 itself falls into the second.

A more permissive type of deception would allow the initial state of a state transition to violate the second component, for instance. This new type of deception models the cases where a principal takes steps in the order prescribed by the protocol, without checking whether he or she should actually take the steps. A concrete example is when Z in Franklin and Reiter's protocol fires the rule R_{Z_4} and forwards a message to X even if the secret sharings between X and Z, and Y and Z (as specified in line 3 in R_{Z_4}) cannot be verified.

The notion of trust assumptions itself requires better understanding. For example, given a protocol and a protection property, is there a unique set of trust assumptions that would allow the protocol to guarantee the property? If not, how can we compare two different sets of trust assumptions? Ideally, the weaker the trust assumptions of a protocol, the less vulnerable it is, and possibly the cheaper it is to build.

Also, the types of trust assumptions required by a protocol depend on under which deviation modes the protocol is supposed to function. The set of trust assumptions we devised in our case studies addressed abortive and deceptive deviations. As we add other deviation modes to our model, we will also need to add additional trust assumptions. Will these additional assumptions be formalizable in temporal logic? Or will we need a different formalism?

Currently, our model handles only finite executions. Not all protocols have finite executions, however. For example, there can be servers that can respond to requests infinitely often. We can extend our model to handle these cases. For example, we can consider both maximal executions and infinite executions, and require that infinite executions have finite prefixes that satisfy appropriate protection properties. (The requirements on maximal executions remains the same.)

Relationship to Other Properties

Identifying a new property is important. Just as important is establishing the relationship between a new property and the existing ones. In this dissertation, we have established the relationship between fairness and protection of individuals' interests. There are others to be explored. For example, what is the relationship between protection of individuals' interests in ecash protocols and money-atomicity [84]? In Brands's protocol, violation of payer's interests allows us to conclude immediately that the protocol is not money-atomic. But can money-atomicity be defined in terms of protection of individuals' interests and other more "primitive" properties? If so, then we will be taking a step towards understanding these more complex, protocol-wide properties.

Finally, we can try to identify other classes of properties that are relevant to electronic commerce protocols.

6.3 Closing Remarks

In this dissertation, we identified protection of individuals' interests as a critical requirement for electronic commerce protocols where participants have different interests to preserve. We then formulated a model-theoretic framework in which protocols can be analyzed with respect to this requirement.

Two related factors make this work a significant step forward towards understanding electronic commerce protocols. First, it has extracted the essence of a property that is expected in a diverse range of electronic commerce protocols. Second, because the framework captures only what is essential, it is in turn applicable to a wide range of electronic commerce protocols. This generality is highly desirable because it will save us from rediscovering what others have discovered in similar contexts. For instance, a number of challenges faced by Shmatikov and Mitchell [80] in analyzing a contract signing protocol were faced by Heintze *et al* [51] in analyzing a payment protocol and an ecash protocol. These same challenges were also encountered by Schneider [78] in analyzing a non-repudiation protocol. Now, they can all be handled in our framework.

Our framework is appealing because it uses well-known and simple models and formalisms. Thus, it can be readily applied as is, or be mapped into one's favorite formalism.

Despite its simplicity, our framework can handle complex protocols, as is demonstrated by our analysis results. The framework provides not only the low-level apparatus for formalization and analysis, but also a high-level frame of reference for protocol design.

Finally, protection of individuals' interests is just one of the properties of interest in electronic commerce protocols. Many others still await our investigation.

Bibliography

[1] Ross Anderson. Liability and computer security: Nine principles. In *Proceedings of the Third European Symposium on Research in Computer Security - ESORICS 94*, pages 231–245, Brighton, United Kingdom, November 1994.

[2] Andre Arnold. *Finite Transition Systems*. Prentice-Hall, 1994.

[3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. Technical Report RZ 2858, IBM Zurich, 1996.

[4] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[5] A. Bahreman and J. D. Tygar. Certified electronic mail. In *Proceedings of the 1994 Internet Society Symposium on Network and Distributed System Security*, February 1994.

[6] Mihir Bellare, Juan A. Garay, Ralf Hauser, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, and Michael Waidner. *i*KP – a family of secure electronic payment protocols. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 89–106, July 1995.

[7] M. Ben-Or, O. Goldreich, S. Micali, and R.L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, January 1990.

[8] Thomas Beth, Malte Borcherding, and Birgit Klein. Valuation of trust in open networks. In *Proceedings of the Third European Symposium on Research in Computer Security - ESORICS 94*, pages 3–18, Brighton, United Kingdom, November 1994.

[9] Andrew Birrell, Butler Lampson, Roger Needham, and Michael Schroeder. A global authentication service without global trust. In *IEEE Symposium on Research in Security and Privacy*, pages 223–230, 1986.

[10] M. Blum. *Three applications of the oblivious transfer: Part I: Coin flipping by telephone; Part II: How to exchange secrets; Part III: How to send certified electronic mail.* Department of EECS, University of California, Berkeley, CA, 1981.

[11] M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(175–193), 83.

[12] M. Blum and S. Goldwasser. An efficient probabilistic public key encryption scheme which hides all partial information. In *Advances in Cryptology - CRYPTO '84*. Springer Verlag, 1984. LNCS no. 196.

[13] Dominique Bolignano. Towards the formal verification of electronic commerce protocol. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 133–146, June 1997.

[14] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology - CRYPTO '93*, pages 302–318, 1993.

[15] Stefan Brands. Electronic cash. In Michael Atallah, editor, *Handbook on Algorithms and Theory of Computation*, chapter 44. CRC Press, November 1998.

[16] Stefan Brands. *Rethinking public key infrastructures and digital certificates — building in privacy*. PhD thesis, CWI - The Netherlands, 1999.

[17] Stefan Brands, 2000. Private communication.

[18] Martha Branstad, W. Curtis Barker, and Pamela Cochrane. The role of trust in protected mail. In *IEEE Symposium on Research in Security and Privacy*, pages 210–215, 1990.

[19] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. In *Proceedings of the Twelfth ACM Symposium on Operation Systems Principles*, 1989.

[20] Jean Camp, Michael Harkavy, J.D. Tygar, and Bennet Yee. Anonymous atomic transactions. In *Proceedings of the Second USENIX Workshop in Electronic Commerce*, pages 123–133, November 1996.

[21] D. Chaum. Achieving electronic privacy. *Scientific American*, 267(2):96–101, Aug 1992.

[22] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology — CRYPTO '88 Proceedings*, pages 200–212. Springer-Verlag, 1990.

[23] D. Chaum and T. Pederson. Transferred cash grows in size. In *Advances in Cryptology - EUROCRYPT '92 Proceedings*, pages 391–407. Springer-Verlag, 1993.

[24] D. Chaum and T. Pederson. Wallet databases with observers. In *Advances in Cryptology - CRYPTO '92 Proceedings*, pages 89–105. Springer-Verlag, 1993.

[25] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[26] Edmund Clarke, Somesh Jha, and Will Marrero. A machine checkable logic of knowledge for specifying security properties of electronic commerce protocols. In *Workshop on Formal Methods and Security Protocols*, 1998.

[27] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *Advances in Cryptology – CRYPTO '89 Proceedings (Lecture Notes in Computer Science 435)*, pages 573–588, 1990.

[28] B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings fo the 1st USENIX Workshop on Electronic Commerce*, July 1995.

[29] R. Cramer and T. Pederson. Improved privacy in wallets with observers. In *Advances in Cryptology - EUROCRYPT '93 Proceedings*, pages 329–343. Springer-Verlag, 1994.

[30] I.B. Damgard. Payment systems and credential mechanisms with provable security against abuse by individuals. In *Advances in Cryptology - CRYPTO '88*, pages 328–335, 1988.

[31] R. H. Deng, L. Gong, A. A. Lazar, and W. Wang. Practical protocols for certified electronic mail. In *Journal of Network and Systems Management 4(3)*, 1996.

[32] D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[33] Groupement des Cartes Bancaires. C-SET architecture de securite, June 1996.

[34] David L. Dill. The murϕ verification system. In *Proceedings of the 8th International Conference on Computer Aided Verification*, pages 390–393, 1996.

[35] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The coq proof assistant user guide. Technical Report Rapport INRIA 154, INRIA, 1993.

[36] Sape Mullender (editor). *Distributed Systems*. Addision-Wesley, 1993.

[37] T. Eng and T. Okamoto. Single-term divisible electronic coins. In *Advances in Cryptology — EUROCRYPTO '94 Proceedings*. Springer-Verlag, 1995.

[38] S. Even, O. Goldreich, and A. Lempel. Randomizing protocols for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[39] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. In *Journal of Cryptology*, pages 9(1):35–67, 1996.

[40] N.T. Ferguson. Extensions of single-term coins. In *Advances in Cryptology — CRYPTO '93 Proceedings*, pages 292–301. Springer-Verlag, 1994.

[41] N.T. Ferguson. Single-term off-line coins. In *Advances in Cryptology — CRYPTO '93 Proceedings*, pages 318–328. Springer-Verlag, 1994.

[42] Matthew Franklin and Michael Reiter. Fair exchange with a semi-trusted third party. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, April 1997.

[43] Juan A. Garay, Markus Jakobsson, and Philip MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology - CRYPTO '99 Proceedings*, pages 449–466. Springer-Verlag, 1999.

[44] V.D. Gligor, S.-W. Luan, and J.N. Pato. On inter-realm authentication in large distributed systems. In *IEEE Conference on Security and Privacy*, pages 2–17, 1992.

[45] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 1982.

[46] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–248, 1991.

[47] James N. Gray. The transaction concept: virtues and limitations. In *Proceedings of the Very Large Database Conference*, pages 144–154, September 1981.

[48] S. Haber and W. S. Stornetta. How to time-stamp a digital document. In *Journal of Cryptology*, pages 3(2):99–111, 1991.

[49] N. M. Haller. The S/KEY^{TM} one-time password system. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems*, 1994.

[50] B. Hayes. Anonymous one-time signatures and flexible untraceable electronic cash. In *Advances in Cryptology — AUSCRYPT '90 Proceedings*, pages 294–305. Springer-Verlag, 1990.

[51] N. Heintze, J.D. Tygar, J. Wing, and H. Chi Wong. Model checking electronic commerce protocols. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, November 1996.

[52] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[53] M. Jakobsson. Ripping coins for a fair exchange. In *Advances in Cryptology - EUROCRYPT '95 Proceedings*, pages 220–230. Springer-Verlag, 1995.

[54] Rajashekar Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 22(5):313–328, May 96.

[55] Volker Kessler and Heike Neumann. A sound logic for analyzing electronic commerce protocols. In *Proceedings of the Fifth European Symposium on Research in Computer Security - ESORICS 98*, pages 345–360, Louvain-la-Neuve, Belgium, September 1998.

[56] Volker Kessler and Gabriele Wedel. Autolog - an advanced logic of authentication. In *Proceedings of the VII Computer Security Foundations Workshop*, pages 90–99, June 1994.

[57] S. Ketchpel. Transaction protection for information buyers and sellers. In *Proceedings of the Darmouth Institute for Advanced Graduate Studies '95*, 1995.

[58] S. Ketchpel and H. Garcia-Molina. Making trust explicit in distributed commerce transactions. In *Stanford Digital Library Project Working Paper SIDL-WP-1995-0018*, October 1995.

[59] Darrell Kindred. Theory generation for security protocols. Technical Report CMU-CS-99-130, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1999. Ph.D. thesis.

[60] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[61] B. Lampson, M. Abadi, and E. Wobber M. Burrows. Authentication in distributed systems: Theory and practice. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, October 1991.

[62] C.H. Lim and P.J. Lee. A practical electronic cash system for smart cards. In *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, pages 34–47, 1993.

[63] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS '96*, pages 147–166, March 1996.

[64] Formal Systems (Europe) Ltd. *Failures Divergence Refinement—User Manual and Tutorial*, 1993. Version 1.3.

[65] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, pages 11–21, 1984.

[66] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. *Atomic Transactions*. Morgan Kaufmann, San Mateo, CA, 1994.

[67] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer-Verlag, 1992.

[68] MasterCard and VISA Corporations. Secure Electronic Transaction (SET), June 1996.

[69] Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of DISCEX 2000*, pages 237–250. IEEE Computer Security Press, January 2000.

[70] Catherine Meadows and Paul Syverson. A formal specification of requirements for payment transactions in the set protocol. In *Preproceedings of Financial Cryptography 98*, 1998.

[71] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978. Also Xerox Research Report, CSL-78-4, Xerox Research Center, Palo Alto, CA.

[72] T. Okamoto and K. Ohta. Disposable zero-knowledge authentication and their applications to untraceable electronic cash. In *Advances in Cryptology — CRYPTO '89 Proceedings*, pages 134–149. Springer-Verlag, 1990.

[73] T. Okamoto and K. Ohta. Universal electronic cash. In *Advances in Cryptology — CRYPTO '91 Proceedings*, pages 324–337. Springer-Verlag, 1992.

[74] D. Otway and O Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 87.

[75] Michael O Rabin. How to exchange secrets by oblivious transfer. Technical report, Harvard Center for Research in Computer Technology, 1981.

[76] P. Venkat Rangan. An axiomatic basis of trust in distributed systems. In *IEEE Symposium on Research in Security and Privacy*, pages 204–211, 1988.

[77] R. Rivest and A. Shamir. Payword and micromint: Two simple micropayment protocols. In *Proc. Security Protocols*, pages 69–88. Springer, 1996. LNCS 1189.

[78] Steve Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 54–65, June 1998.

[79] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, 1996.

[80] V. Shmatikov and J.C. Mitchell. Analysis of a fair exchange protocol. In *Proceedings of the Seventh Annual Symposium on Network and Distributed System Security (NDSS 2000)*, 2000. To appear.

[81] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings*, pages 191–200, Winter 1988.

[82] Paul Syverson and Catherine Meadows. A formal language for cryptographic protocol requirements. *Design, Codes, and Cryptography*, 7(1 and 2):27–59, 96.

[83] T. Tedrick. Fair exchange of secrets. In *Advances in Cryptology - CRYPTO '84*, pages 434–438, 1985.

[84] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 8–26, May 1996.

[85] U. Vazirani and V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 23–30, 1983.

[86] Gabriele Wedel and Volker Kessler. Formal semantics for authentication logics. In *Proceedings of the Fourth European Symposium on Research in Computer Security - ESORICS 96*, pages 219–241, Rome, Italy, 1996.

[87] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems. In *Proceedings of the 1993 Symposium on Security and Privacy*, pages 150–164, 1993.

[88] A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[89] J. Zhou and D. Gollman. A fair non-repudiation protocol. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 55–61, May 1996.

[90] J. Zhou and D. Gollman. An efficient non-repudiation protocol. In *Proceedings of the 1997 IEEE Computer Security Foundations Workshop*, pages 126–132, June 1997.