# Safe Simplifying Transformations for Security Protocols
or
Not Just the Needham Schroeder Public Key Protocol

Mei Lin Hui Gavin Lowe*

January 14, 1999

**Abstract**

Recent techniques for analyzing security protocols have tended to concentrate upon the small protocols that are typically found in the academic literature. However, there is a huge gulf between these and most large commercial protocols: the latter typically have many more fields, and much higher levels of nested encryption. As a result, existing techniques are difficult to apply directly to these large protocols.

In this paper we develop the notion of *safe simplifying transformations*: transformations that have the property of preserving insecurities; the effect of such transformations is that if we can verify the transformed protocol, then we will have verified the original protocol. We identify a number of such safe simplifying transformations, and use them in the analysis of a commercial protocol.

*Authors' address: Department of Mathematics and Computer Science, University of Leicester, University Road, Leicester, LE1 7RH, UK; e-mail: {mlh7,gl7}@mcs.le.ac.uk.

# Contents

# 1 Introduction

Many security protocols have been suggested, with various goals, such as establishing a cryptographic key, or achieving authentication (where an agent becomes sure of the identity of the other agent taking part in the protocol). These protocols are supposed to succeed even in the presence of a malicious agent, called an *intruder*; this intruder is assumed to have complete control over the communications network, and so can intercept messages, and introduce new messages into the system using information from messages he has previously seen. Unfortunately, a large proportion of the protocols that have been suggested do not succeed in their stated goals!

In recent years, a number of techniques have been developed for analysing protocols; the two main categories of such techniques are:

**Model checking techniques** where a model checker is used to search the state space of a small system running the protocol, looking for attacks; see, for example, [MCF87, KMM94, Mea96, Low96, LR97, Low98a, MMS97, MCJ97]

**Direct proofs** where the protocol is proved correct directly (possibly with the help of machine assistance), typically using some form of induction; see, for example, [Sch97, GM95, Pau97, THG98].

These techniques have proved successful at analysing the protocols that have appeared in the academic literature. Such protocols are normally small, typically having between two and seven messages, each message having between two and ten fields. However, most commercial protocols are considerably more complicated, typically containing several layers of nested encryptions, and significant amounts of redundancy. For example, the CyberCash Main Sequence Protocol—a protocol for carrying out commercial transactions over the Internet between a customer and a merchant—is given in Appendix A; this protocol contains dozens of fields, and in one place, six levels of nested cryptography.

This extra complexity of such protocols makes analysis much more difficult: for the model checkers, it leads to an explosion in the state space and the message space; for those doing direct proofs, the complexity just makes the protocol harder to understand—it is very hard to see how the protocol is supposed to work, let alone verify it.

However, we will often have a strong feeling that much of the complexity could be removed without altering the security of the protocol: for example, in the CyberCash Protocol, we can be reasonably sure that the field representing the customer's postal code is not relevant to the secrecy of the customer's credit card number (we might require that the postal code be authenticated, but that is a different question).

If we could prove that this complexity is indeed not needed for the security of the protocol, then it would be enough to analyse the protocol with this complexity removed. This is the question we address in this paper. We identify a number of *simplifying transformations* on protocols that have the property that if there is an attack upon the orignal protocol (i.e. it is insecure), then there is also an attack upon the simplified protocol; we call transformations with this property "*safe*". This means that if we can verify the simplified protocol, we will have verified the original protocol.

Note that the property of being safe is only one way round: it is possible to apply safe simplifying transformations in such a way as to introduce new attacks, by oversimplifying the protocol; we need to use our experience and intuition to avoid this. Quoting Albert Einstein:

> Everything should be made as simple as possible, but not simpler.

The idea will be, starting from the original protocol, to apply as many safe simplifying transformations as possible, trying to avoid introducing new attacks, and then to analyse the simplified protocol. If the simplified protocol is secure, then so is the original; if, however, our attack discloses an attack upon the simplified protocol, we have to consider whether there is a corresponding attack on the original protocol (by effectively undoing all the simplifying transformations); if there *is* such an attack, then clearly the original protocol is flawed; otherwise, we have oversimplified, so we have to try again, learning from our mistakes.

Our approach is to identify a function $f$ from the messages of the original protocol $P_1$ to the messages of the simplified protocol $P_2$, describing how messages are simplified. We then lift this function to traces (i.e. sequences of messages), by pointwise application. In order for the transformation to satisfy the condition outlined above, we require that if $tr$ is a trace representing an attack upon $P_1$, then $f(tr)$ is an attack upon $P_2$. Proving this for a particular $f$ will consist of two things: proving that for every trace $tr$ of $P_1$, $f(tr)$ is a trace of $P_2$; and that if a trace $tr$ constitutes an attack upon $P_1$, then $f(tr)$ constitutes an attack upon $P_2$.

Our formal analysis is based around the process algebra CSP [Hoa85, Ros97]. We should stress, however, that our results can be used in conjunction with *any* protocol analysis technique, not just those based on CSP.

In the next section, we give a CSP model of a system running a protocol, along with an intruder who can interact with that system; we also formalize some security properties. In Section 3 we formalize the class of simplifying transformations that we are interested in, and prove a theorem that shows that if such a transformation satisfies two properties, then it is safe. In Section 4 we show that a number of useful transformations satisfy these properties, namely transformations to: remove some encryptions, replacing encrypted components by their unencrypted bodies; remove some hash functions, replacing hashed components by their unhashed bodies; remove some fields of the protocol completely; rename some atoms; coalesce some adjacent pairs of atoms; swap some adjacent pairs of atoms; and move some fields out of enclosing encryptions. In Section 5 we illustrate these simplifying transformations by applying them to the CyberCash Protocol, simplifying the protocol to a stage where we can applying standard analysis techniques. We sum up in Section 6.

## 2 The model

We will use the process algebra CSP to reason about security protocols, to formalize the simplifying transformations, and to verify that they satisfy the desired condition that if there is an attack on the original protocol, then there is an attack on the simplified protocol. A brief introduction to CSP is included in Appendix C.

We use CSP because it is well suited for reasoning about systems of communicating agents. Further, it has been much used for security protocol analysis, and the translation from descriptions of protocols in "standard form" into CSP is well understood—in many ways, a description in CSP is far more formal than "standard form". Also, CSP has a well developed theory, which we may exploit. Finally, the transformations themselves are easy to formalize—they are simply CSP renamings!

In Section 2.1 we define the underlying space of messages; in Section 2.2 we describe how the honest agents are modelled in CSP, and in Section 2.3 we do the same for the intruder; then in Section 2.4 we formalize what we mean by an attack upon the protocol, thus formalizing the security properties that we are interested in.

### 2.1 Messages

Our model of a system running a protocol is based around a datatype *Message*, representing the messages that are sent or received by agents. We define the message space to be built up from some set *Atom* of atomic values, by pairing, encryption and hashing. We define $Key \subseteq Atom$ to be those atoms used for encrypting and decrypting messages; we write $K^{-1}$ for the inverse key of $K$. We let $HashFn$ be the set of cryptographic hash functions. The set $Message$ is defined by:

$$\begin{aligned} Message ::= \; & \textsc{Atom}\, Atom \mid \\ & \textsc{Pair}\, Message\ Message \mid \\ & \textsc{Encrypt}\, Message\ Key \mid \\ & \textsc{Hash}\, HashFn\ Message. \end{aligned}$$

We will use the syntactic sugar $(M, M')$ for $\textsc{Pair}\, M\, M'$, $\{M\}_K$ for $\textsc{Encrypt}\, M\, K$, and $g(\!|M|\!)$ for $\textsc{Hash}\, g\, M$.

In fact, all the results of the following section are independent of the definition of $Message$, and so would apply equally to other definitions of $Message$, for example with additional constructors, such as one representing Vernam encryption. The choice of the type $Message$ will be important only when we define specific transformations in Section 4. However, we give the definition here for clarity of exposition.

Note that implicit in the above datatype definition is that two messages are equivalent only if they have identical fields, so, for example, encryption is injective:

$$\{M\}_K = \{M'\}_{K'} \Leftrightarrow M = M' \wedge K = K'.$$

This is the *perfect encryption assumption*.

Further, the datatype definition means that some messages are distinguished that we would like to identify, for example $((\textsc{Atom}\, A, \textsc{Atom}\, B), \textsc{Atom}\, C)$ and $(\textsc{Atom}\, A, (\textsc{Atom}\, B, \textsc{Atom}\, C))$: the definition does not capture the associativity of pairing. We therefore identify a *normal form* for messages, and will ensure that only messages in normal form are passed in the system[1]. The normal form will have all pairing associating to the right, for example $(\textsc{Atom}\, A, (\textsc{Atom}\, B, \textsc{Atom}\, C))$. This means that for any pair in normal form, the first element will not itself be a pair. We write $pair(M)$ if $M$ is a pair (easily defined over the structure of the message space), and formally define a message to be in normal form if it satisfies the following predicate:

$$\begin{aligned} isNF \; &: \; Message \to \mathbf{B}, \\ isNF(\textsc{Atom}\, A) \; &\widehat{=} \; true, \end{aligned}$$

[1] Having to define a normal form in this way is very much orthogonal to the main thrust of the paper, but is unfortunately necessary; the reader is advised not to be distracted too much by the details.

$$
\begin{aligned}
isNF(M, M') &\mathrel{\widehat{=}} isNF(M) \wedge isNF(M') \wedge \neg\, pair(M),\\
isNF(\{M\}_K) &\mathrel{\widehat{=}} isNF(M),\\
isNF(g(\!|M|\!)) &\mathrel{\widehat{=}} isNF(M).
\end{aligned}
$$

We define $NFMessage$ to be the set of all messages in normal form:

$$NFMessage \mathrel{\widehat{=}} \{M \in Message \mid isNF(M)\},$$

and define a function to convert a message into normal form:

$$
\begin{aligned}
NF &: Message \to Message,\\
NF(\textsc{Atom}\, A) &\mathrel{\widehat{=}} \textsc{Atom}\, A,\\
NF((M_1, M_2), M_3) &\mathrel{\widehat{=}} \text{let } (M_1', M_2') = NF(M_1, M_2)\\
&\phantom{\mathrel{\widehat{=}}}\ \ \text{in } (M_1', NF(M_2', M_3)),\\
NF(M_1, M_2) &= (NF(M_1), NF(M_2)), \qquad \text{if } \neg\, pair(M_1),\\
NF(\{M\}_K) &\mathrel{\widehat{=}} \{NF(M)\}_K,\\
NF(g(\!|M|\!)) &= g(\!|NF(M)|\!).
\end{aligned}
$$

The following lemma relates the above two definitions.

**Lemma 1**

- $isNF(M) \Rightarrow NF(M) = M$;
- $isNF(NF(M))$.

### 2.2 Honest agents

We will model each agent taking part in the protocol by a CSP process, using the following events:

- events of the form $send.A.B.M$, representing agent $A$ sending (normal form) message $M$, with intended recipient $B$;
- events of the form $receive.A.B.M$, representing agent $B$ receiving message $M$, apparently from $A$;
- events of the form $claimSecret.A.B.M$, representing that $A$ thinks that $M$ is a secret shared only with $B$; if $B$ is not the intruder, then the intruder should not learn $M$;
- events of the form $running.A.B.Ms$, representing that $A$ thinks he is running the protocol with $B$; $Ms$ is a sequence of messages, recording some details about the run in question;

- events of the form $finished.A.B.Ms$, representing that $A$ thinks he has finished a run of the protocol with $B$; $Ms$ is a sequence of messages, recording some details about the run in question;

events of the first two forms are used for modelling the main communications of the protocol. The latter three forms will be used for writing specifications for the protocol; we discuss this further in Section 2.4.

Each honest agent can be represented by a CSP process, describing the way the agent sends and receives messages. The precise details will depend upon the protocol in question; a number of previous papers (e.g. [Low96, Sch96, LR97, Low98a]) have described the translation from a protocol description in "standard form" to a CSP representation. However, it turns out that we don't need to assume anything else about the CSP representation of the honest agents (apart from the fact that such a representation exists)—our results apply to arbitrary CSP processes, not just those that represent participants in a security protocol.

We write $P_A$ for the CSP process representing the honest agent with identity $A$; this process will have the following alphabet:

$$\begin{aligned} \alpha(P_A) \ \widehat{=} \ & \{send.A.B.M \mid B \in Agent, M \in NFMessage\} \\ & \cup \{receive.B.A.M \mid B \in Agent, M \in NFMessage\} \\ & \cup \{claimSecret.A.M \mid M \in NFMessage\} \\ & \cup \{running.A.B.M \mid B \in Agent, M \in NFMessage^*\} \\ & \cup \{finished.A.B.M \mid B \in Agent, M \in NFMessage^*\}. \end{aligned}$$

### 2.3 The intruder

We now define the most general intruder who can interact with a system running the protocol. Following [Sch96], the definition is based around a relation $\vdash$, which defines the ways in which the intruder can deduce new messages; we write $B \vdash M$ if message $M$ can be deduced from the set of messages $B$; the relation is defined by the following six rules:

**member** $M \in B \Rightarrow B \vdash M$;

**pairing** $B \vdash M \land B \vdash M' \land \neg\, pair(M) \Rightarrow B \vdash (M, M')$;

**splitting** $B \vdash (M, M') \Rightarrow B \vdash M \land B \vdash M'$;

**encryption** $B \vdash M \land B \vdash \textsc{Atom}\, K \land K \in Key \Rightarrow B \vdash \{M\}_K$;

**decryption** $B \vdash \{M\}_K \land B \vdash \textsc{Atom}\, K^{-1} \Rightarrow B \vdash M$;

**hashing** $B \vdash M \wedge g \in HashFn \Rightarrow B \vdash g(\!|M|\!)$.

The intruder can create new messages by creating or splitting pairs, by encrypting or decrypting with keys he can produce, and by hashing. The **pairing** rule is formulated to ensure that the pair created is in normal form, provided the components are. The following lemma shows that from normal form messages, only normal form messages can be created:

**Lemma 2**

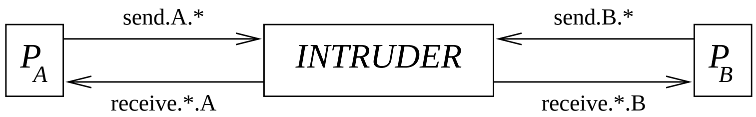
$$(\forall M' \in B \bullet isNF(M')) \wedge B \vdash M \;\Rightarrow\; isNF(M).$$

We can show that, despite the limitation in the **pairing** rule above, the intruder can create and split pairs as one would expect:

**Lemma 3**

$$\begin{aligned} B \vdash M \wedge B \vdash M' \;&\Rightarrow\; B \vdash NF(M, M'), \\ isNF(M) \wedge isNF(M') \wedge B \vdash NF(M, M') \;&\Rightarrow\; B \vdash M \wedge B \vdash M'. \end{aligned}$$

We now define a CSP process representing the intruder. Our model of the system will ensure that the intruder receives every message sent by an honest agent, and can then choose what messages are received by honest agents. A system with two honest agents, $A$ and $B$, can therefore be pictured as below:




In effect, the intruder process also models the effect of the communication medium.

The process representing the intruder is parameterized by the set $S$ of messages that he either initially knew, or has subsequently seen (recall that the *send* channel represents messages that are sent by honest agents and so are received by the intruder, and that the *receive* channel represents messages that are received by honest agents and are sent by the intruder):

$$\begin{aligned} & INTRUDER(S) \mathrel{\widehat=} \\ & \quad \Box_{M \in Message}\, send?A?B!M \to INTRUDER(S \cup \{M\}) \\ & \quad \Box \\ & \quad \Box_{M \in Message, S \vdash M}\, receive?A?B!M \to INTRUDER(S) \\ & \quad \Box \\ & \quad \Box_{M \in Message, S \vdash M}\, leak.M \to INTRUDER(S). \end{aligned}$$

The intruder may:

- see messages sent by honest agents, and so learn the contents;

- send messages with contents that can be derived from his current knowledge, such messages being received by an honest agent;

- signal that he knows any message that can be derived from his current knowledge.

We assume that the intruder initially knows some set $IK_0$ of messages, so the initial state of the intruder is $INTRUDER(IK_0)$. The complete system is then represented by the CSP process:

$$SYSTEM \mathrel{\widehat{=}} \left( |||_{A \in Honest} P_A \right) \parallel INTRUDER(IK_0),$$

(where $Honest$ is the set of honest agents' identities) that is, the parallel composition of the honest agents and the intruder, synchronizing on their common events, i.e. the *send* and *receive* events.

### 2.4 Attacks

In this subsection we consider possible requirements of a protocol. We will produce various *trace specifications* representing different security properties that we would like systems running the protocol to satisfy. A trace specification is a predicate $SPEC(tr)$ whose free variable $tr$ represents a trace. A process satisfies such a specification if $SPEC(tr)$ is true for all its traces:

$$P \textbf{ sat } SPEC \Leftrightarrow \forall tr \in traces(P) \cdot SPEC(tr).$$

Thus, we will be interested in whether $SYSTEM$ **sat** $SPEC$, for various specifications representing security properties.

First of all, we define a breach of secrecy to occur when an honest agent thinks a certain message (typically an atomic message) is secret, but the intruder learns it. We formalize secrecy by specifying that if an agent $A$ performs a $claimSecret.A.B.M$ event (representing that $A$ believes $M$ to be a secret shared only with $B$), where $B$ is not the intruder, then the intruder should never perform $leak.M$ (i.e. should never learn $M$). We write this as a trace specification, parameterized by the set $Secs$ of messages that can be considered secret:

$$\begin{array}{l} Secrecy_{Secs}(tr) \mathrel{\widehat{=}} \\ \quad \forall A \in Agent\,;\, B \in Honest\,;\, M \in Secs \cdot \\ \qquad claimSecret.A.B.M \text{ in } tr \Rightarrow \neg\, leak.M \text{ in } tr. \end{array}$$

We now define some entity authentication specifications. As explained in [Low97], we believe that the term "authentication" refers to a collection of related requirements, rather than there being a single generic authentication specification; other authors have made similar observations, e.g. [Boy97, SvO94]. The following is one of the more common forms of authentication:

> Whenever $A$ completes a run of the protocol, apparently with $B$, then $B$ has previously been running the protocol, apparently with $A$, and the two agents agreed on some collection $Ms$ of messages, and each such run of $A$ corresponds to a *unique* run of $B$.

To formalize this requirement, we specify that the number of $finished$ events performed by $A$ concerning $B$ with some sequence of messages $Ms$ must be no more than the number of corresponding $running$ events performed by $B$. We write this requirement as a trace specification, parameterized by the set $AgreementSet$ of sequences of messages upon which the agents should agree (for example, if it is required that the agents agree upon a key and a nonce, then the agreement set will be all sequences of the form $\langle K, N\rangle$ for keys $K$ and nonces $N$):

$$
\begin{array}{l}
Agreement_{AgreementSet}(tr) \mathrel{\widehat=} \\
\quad \forall\, A \in Agent\,;\, B \in Honest\,;\, Ms \in AgreementSet \bullet \\
\qquad tr \downarrow finished.A.B.Ms \le tr \downarrow running.B.A.Ms.
\end{array}
$$

If $tr$ is a trace, and $e$ is an event, then $tr \downarrow e$ represents the number of occurrences of $e$ in $tr$. Of course, in order to ensure that this specification captures the requirements outlined above, we need to ensure that the $finished$ and $running$ events correctly capture the informally descriptions given to them above: the $finished$ events should be performed by honest agents when they complete a protocol run, and the $running$ event should be performed just before the last $send$ event; see [Low97] for more details.

We believe it would be completely straightforward to extend the results of this paper to other authentication specifications, such as the aliveness and non-injective agreement specifications of [Low97]. It might also be possible to deal with completely different specifications such as non-repudiation or anonymity.

## 3 Transformations on protocols

In this section we formalize how we represent transformations upon protocols; we then prove a result, giving sufficient conditions for a transformation to be safe.

### 3.1 Formalizing transformations

A transformation will be defined via a function

$$f : Message \to Message,$$

which defines how messages in the original protocol are replaced by messages in the simplified protocol. In the following, we will overload $f$ to apply to several different types of arguments.

We lift the function $f$ to events as follows:

$$\begin{aligned}
f(send.A.B.M) &\mathrel{\widehat{=}} send.A.B.f(M),\\
f(receive.A.B.M) &\mathrel{\widehat{=}} receive.A.B.f(M),\\
f(claimSecret.A.B.M) &\mathrel{\widehat{=}} claimSecret.A.B.f(M),\\
f(running.A.B.Ms) &\mathrel{\widehat{=}} running.A.B.f(Ms),\\
f(finished.A.B.Ms) &\mathrel{\widehat{=}} finished.A.B.f(Ms),
\end{aligned}$$

and lift it to traces by:

$$f(tr) \mathrel{\widehat{=}} \langle f(e) \mid e \leftarrow tr\rangle.$$

We now define the CSP representation of an honest agent $A$ in the transformed system, corresponding to the process $P_A$ in the original system. We need a process that sends or receives the message $f(M)$ whenever the process $P_A$ sends or receives $M$. We therefore define the new representation to be the process $f(P_A)$ (recall that the CSP process $f(P)$ is the process that acts like $P$, except where every event $e$ is renamed to $f(e)$). This is certainly the right model for the honest agents when one considers only the sending and receiving of messages. Some protocols, however, involve more than just a straightforward sending and receiving of messages; for example, in many cases agents perform tests upon the messages they receive; in these cases, it's not immediately obvious how the simplifying transformations affect the protocol. This definition gives the answer: the behaviour of the agent in the transformed protocol is described by the process $f(P_A)$.

When we transform the protocol, we similarly have to transform the initial knowledge of the intruder; we will let $IK'_0$ be this transformed initial knowledge. Below we will prove various results relating $IK_0$ and $IK'_0$: these results will tell us what initial knowledge ($IK'_0$) we have to give the intruder in the transformed system, in order to prove properties of the original system for a particular choice ($IK_0$) of the intruder's initial knowledge there.

The complete transformed system is modelled, therefore, by:

$$SYSTEM' \;\widehat{=}\; \left(|||_{A \in Honest}\, f(P_A)\right) \parallel INTRUDER(IK_0').$$

### 3.2 A general result for safe transformations

We now prove a general result that gives sufficient considerations under which a transformation is safe. In the next section we give particular examples of such transformations.

In order to show that a particular transformation $f$ is safe, we need to show that if there is an attack upon $SYSTEM'$, then there is an attack upon $SYSTEM$. We begin by showing that if $tr \in traces(SYSTEM)$ then $f(tr) \in traces(SYSTEM')$; we then go on to consider attacks.

It turns out that we will need two conditions upon the transformation function $f$ and the intruder's initial knowledge in the two systems. The first condition is that if the intruder can deduce a particular message $M$ in the original protocol, then he can deduce the corresponding message $f(M)$ in the corresponding setting in the transformed protocol:

$$\begin{array}{l} \forall\, B \in \mathbf{P}(NFMessage)\,;\, M \in NFMessage \bullet \\ \qquad B \vdash M \Rightarrow f(B) \cup IK_0' \vdash f(M). \end{array} \tag{1}$$

We call this property the *distribution* property. (This property is very similar to Roscoe's positive property from [Ros98].)

The second condition is that if the intruder initially knows a particular message in the original system, then he knows the corresponding message in the transformed system:

$$f(IK_0) \;\subseteq\; IK_0'. \tag{2}$$

This property tells us how to choose the initial knowledge for the intruder in the transformed system, given his initial knowledge in the original system.

The following lemma relates the intruders in the two systems:

**Lemma 4** If the transformation function $f$ satisfies equations (1) and (2), then

$$INTRUDER(IK_0')) \;\sqsubseteq\; f(INTRUDER(IK_0)).$$

**Proof:** We will prove:

$$\forall\, S \in \mathbf{P}(Message) \bullet INTRUDER(f(S) \cup IK_0') \sqsubseteq f(INTRUDER(S)),$$

which implies the result by taking $S = IK_0$, and making use of the inclusion $IK_0' \supseteq f(IK_0)$. For convenience, we define $f^+(S) \mathrel{\widehat{=}} f(S) \cup IK_0'$, and calculate as follows:

$$\begin{array}{ll}
& INTRUDER(f^+(S)) \\
= & \left\langle \text{definition} \right\rangle \\
& \Box_{M' \in Message}\, send?A?B!M' \to INTRUDER(f^+(S) \cup \{M'\}) \\
& \Box \\
& \Box_{M' \in Message, f^+(S) \vdash M'}\, receive?A?B!M' \to INTRUDER(f^+(S)) \\
& \Box \\
& \Box_{M' \in Message, f^+(S) \vdash M'}\, leak.M' \to INTRUDER(f^+(S)) \\
\sqsubseteq & \left\langle \{f(M) \mid M \in Message\} \subseteq Message \right\rangle \\
& \Box_{M \in Message}\, send?A?B!f(M) \to INTRUDER(f^+(S) \cup \{f(M)\}) \\
& \Box \\
& \Box_{M \in Message, f^+(S) \vdash f(M)}\, receive?A?B!f(M) \to INTRUDER(f^+(S)) \\
& \Box \\
& \Box_{M \in Message, f^+(S) \vdash f(M)}\, leak.f(M) \to INTRUDER(f^+(S)) \\
\sqsubseteq & \left\langle f^+(S) \cup \{f(M)\} = f^+(S \cup \{M\});\ \text{distribution property (1)} \right\rangle \\
& \Box_{M \in Message}\, send?A?B!f(M) \to INTRUDER(f^+(S \cup \{M\})) \\
& \Box \\
& \Box_{M \in Message, S \vdash M}\, receive?A?B!f(M) \to INTRUDER(f^+(S)) \\
& \Box \\
& \Box_{M \in Message, S \vdash M}\, leak.f(M) \to INTRUDER(f^+(S)).
\end{array}$$

Also,

$$\begin{array}{ll}
& f(INTRUDER(S)) \\
= & \left\langle \text{definition; laws of CSP} \right\rangle
\end{array}$$

$$\Box_{M \in Message} send?A?B!f(M) \rightarrow f(INTRUDER(S \cup \{M\}))$$
$$\Box$$
$$\Box_{M \in Message, S \vdash M} receive?A?B!f(M) \rightarrow f(INTRUDER(S))$$
$$\Box$$
$$\Box_{M \in Message, S \vdash M} leak.f(M) \rightarrow f(INTRUDER(S)).$$

Note the similarity of the right hand sides of the above equations. Hence we have shown that each $INTRUDER(f^+(S))$ process is refined by the result of applying a defining equation for the vector of processes $\langle f(INTRUDER(S')) \mid S' \in \mathbf{P}(Message)\rangle$ to the vector of processes $\langle INTRUDER(f^+(S')) \mid S' \in \mathbf{P}(Message)\rangle$. We thus deduce that

$$INTRUDER(f^+(S)) \sqsubseteq f(INTRUDER(S)),$$

using fixed-point induction [Ros97, page 122]. $\Box$

We now prove our main theorem, which relates the traces of the two systems.

**Theorem 5** If the transformation function $f$ satisfies equations (1) and (2), then

$$\forall tr \in traces(SYSTEM) \bullet f(tr) \in traces(SYSTEM').$$

**Proof:** The result of the theorem can be rewritten as:

$$SYSTEM' \sqsubseteq f(SYSTEM),$$

which we prove as follows:

$$\begin{aligned}
& f(SYSTEM) \\
= & \left\langle \text{definition} \right\rangle \\
& f\left(\left(|||_{A \in Agent} P_A\right) \parallel INTRUDER(IK_0)\right) \\
\sqsupseteq & \left\langle \text{CSP laws 1 and 2 from Appendix C} \right\rangle \\
& \left(|||_{A \in Agent} f(P_A)\right) \parallel f(INTRUDER(IK_0)) \\
\sqsupseteq & \left\langle \text{from Lemma 4} \right\rangle \\
& \left(|||_{A \in Agent} f(P_A)\right) \parallel INTRUDER(IK_0')
\end{aligned}$$

$$= \left\langle \text{definition} \right\rangle$$
$$SYSTEM'.$$

□

The above theorem shows that every trace of the original system is "matched" by one in the new system. We want to show that every *attack* on the original system is "matched" by one on the new system. We produce sufficient conditions such that if $tr$ constitutes an attack upon the original protocol, then $f(tr)$ constitutes an attack on the transformed protocol.

We begin with the secrecy specification. It turns out that if the secrets of the original protocol are given by the set $Secrets$, then we should take the secrets of the adapted protocol to be $f(Secrets)$:

**Lemma 6** If a particular trace $tr$ constitutes a breach of secrecy on the original protocol, then $f(tr)$ is a breach of secrecy on the simplified protocol:

$$\neg\, Secrecy_{Secrets}(tr) \;\Rightarrow\; \neg\, Secrecy_{f(Secrets)}(f(tr)) \;.$$

**Proof:** Suppose $tr$ constitutes a breach of secrecy on the original protocol, i.e., $\neg\, Secrecy_{Secrets}(tr)$. Then for some $A \in Agent$, $B \in Honest$, $M \in Secrets$, we have $claimSecret.A.B.M$ in $tr$ and $leak.M$ in $tr$. Hence $claimSecret.A.B.f(M)$ in $f(tr)$ and $leak.f(M)$ in $f(tr)$. But $f(M) \in f(Secrets)$, and so this constitutes a breach of secrecy on the simplified protocol: $\neg\, Secrecy_{f(Secrets)}(f(tr))$. □

This result allows us to show that such transformations are safe for the secrecy property:

**Corollary 7** If equations (1) and (2) are satisfied, then:

$$SYSTEM' \;\mathbf{sat}\; Secrecy_{f(Secrets)} \;\Rightarrow\; SYSTEM \;\mathbf{sat}\; Secrecy_{Secrets}.$$

**Proof:** Suppose $SYSTEM'$ **sat** $Secrecy_{f(Secrets)}$, and suppose for a contradiction that $\neg\, (SYSTEM$ **sat** $Secrecy_{Secrets})$. Then there is some trace $tr \in traces(SYSTEM)$ such that $\neg\, Secrecy_{Secrets}(tr)$. Then by Theorem 5, $f(tr) \in traces(SYSTEM')$, and by Lemma 6, $\neg\, Secrecy_{f(Secrets)}(f(tr))$. But then $f(tr)$ constitutes a breach of secrecy on $SYSTEM'$, which contradicts the assumption that $SYSTEM'$ **sat** $Secrecy_{f(Secrets)}$. □

We now consider the agreement authentication specification. It turns out that if the messages upon which agreement is intended in the original protocol is *AgreementSet*, then we should take the messages upon which agreement is intended in the transformed protocol to be $f(AgreementSet)$. Further, we require the following condition:

$$\begin{array}{l} \forall\, Ms \in AgreementSet\,;\, Ms' \in Message^* \bullet \\ \qquad Ms \neq Ms' \Rightarrow f(Ms) \neq f(Ms'). \end{array} \tag{3}$$

That is, $f$ does not map a different sequence $Ms'$ of messages to the same value as a member $Ms$ of *AgreementSet*.

**Lemma 8** If equation 3 is satisfied, then if a particular trace $tr$ constitutes a failure of agreement authentication on the original protocol, then $f(tr)$ constitutes a failure of agreement authentication on the simplified protocol:

$$\neg\, Agreement_{AgreementSet}(tr) \;\Rightarrow\; \neg\, Agreement_{f(AgreementSet)}(f(tr)).$$

**Proof:** Suppose $tr$ constitutes a failure of agreement authentication on the original protocol: $\neg\, Agreement_{AgreementSet}(tr)$. Then for some $A \in Agent$, $B \in Honest$, $Ms \in AgreementSet$, we have:

$$tr \downarrow finished.A.B.Ms \;>\; tr \downarrow running.B.A.Ms.$$

By the condition (3) on $f$:

$$\begin{aligned} f(tr) \downarrow finished.A.B.f(Ms) \;&=\; tr \downarrow finished.A.B.Ms, \\ f(tr) \downarrow running.B.A.f(Ms) \;&=\; tr \downarrow running.B.A.Ms, \end{aligned}$$

and so

$$f(tr) \downarrow finished.A.B.f(Ms) \;>\; f(tr) \downarrow running.B.A.f(Ms).$$

That is, $\neg\, Agreement_{f(AgreementSet)}(f(tr))$. □

The reason for condition (3) about $f$ can be explained by the following example: suppose the protocol aims to achieve agreement upon a key, so *AgreementSet* is the set of all singleton sequences of keys; and suppose the function $f$ maps two different keys $K_1$ and $K_2$ to the same value, so $f(\langle K_1 \rangle) = f(\langle K_2 \rangle)$ and condition (3) is not satisfied; then there might be some attack on the original protocol where the two honest agents end up accepting these different keys $K_1$ and $K_2$, but this would not be recognized as an attack in the simplified protocol, because the two agents would use the same key $f(K_1) = f(K_2)$.

The following corollary shows that such transformations are safe with respect to the agreement property; it is proved analogously to Corollary 7:

**Corollary 9** If equations (1), (2) and (3) are satisfied, then:

$$\begin{aligned}&SYSTEM' \ \mathbf{sat}\ Agreement_{f(AgreementSet)} \Rightarrow \\ &\qquad SYSTEM \ \mathbf{sat}\ Agreement_{AgreementSet}\ .\end{aligned}$$

# 4 Examples of safe simplifying transformations

In this section we present a number of transformations that satisfy the conditions from the previous section, and are hence safe.

## 4.1 Removing encryptions

We begin by defining a transformation that removes some of the encryptions from the protocol. We identify a set $Encs$ of encrypted messages such that all encrypted components $\{M\}_K$ from $Encs$ that appear in the protocol should be replaced with the body $M$. We define the transformation function $f$ as follows:

$$\begin{aligned} f(\textsc{Atom}\,A) &\mathrel{\widehat{=}} \textsc{Atom}\,A, \\ f(M, M') &\mathrel{\widehat{=}} NF(f(M), f(M')), \\ f(\{M\}_K) &\mathrel{\widehat{=}} \begin{cases} f(M), & \text{if } \{M\}_K \in Encs, \\ \{f(M)\}_K, & \text{otherwise,} \end{cases} \\ f(g(\!|M|\!)) &\mathrel{\widehat{=}} g(\!|f(M)|\!). \end{aligned}$$

The interesting case is the case for $\{M\}_K$ with $\{M\}_K \in Encs$, where the outermost encryption is stripped off; most of the other cases simply apply the function recursively; the "$NF$" in the definition for pairs is necessary, to ensure that the messages returned by the function are in normal form.

We define the intruder's initial knowledge in the transformed protocol as the result of applying the simplifying function to his knowledge in the original protocol:

$$IK_0' \mathrel{\widehat{=}} f(IK_0).$$

This clearly satisfies condition (2). In Appendix B.1 we show that this transformation also satisfies the distribution property:

$$\forall B \in \mathbf{P}(NFMessage) \,;\, M \in NFMessage \bullet \\ \quad B \vdash M \Rightarrow f(B) \cup IK_0' \vdash f(M).$$

Hence, we can use Corollary 7 to deduce that this transformation is a safe transformation with respect to secrecy:

$$SYSTEM' \ \mathbf{sat} \ Secrecy_{f(Secrets)} \ \Rightarrow \ SYSTEM \ \mathbf{sat} \ Secrecy_{Secrets}.$$

Further, $f$ is clearly injective on atoms, so we can use Corollary 9 to deduce that this transformation is a safe transformation with respect to agreement on atoms:

If $AgreementSet$ is a set of sequences of atoms, then

$$SYSTEM' \ \mathbf{sat} \ Agreement_{f(AgreementSet)} \Rightarrow \\ \quad SYSTEM \ \mathbf{sat} \ Agreement_{AgreementSet}.$$

The way this simplifying transformation will be used in practice is to identify a particular encryption in the protocol description that we think is unnecessary, and to take $f$ to be the function that removes this encryption; that is, we take $Encs$ to be the set of all instances of the encrypted component in question. Note though, that if there is a second encrypted component in the protocol description whose instances are of the same type as the instances of the original encryption, then these instances of the second encrypted component are also in $Encs$, and so we must also strip off the encryptions on these instances. One way to think of this condition is that in the original protocol it is possible for the intruder to replay an instance of one of these two encrypted components and have it interpreted as an instance of the other; if we simplify the protocol to remove only one of the encryptions, then we remove this capability from the intruder.

## 4.2 Removing hash functions

We now define a transformation that removes some of the hash functions from the protocol. The definition is very similar to that for removing encryptions: we identify a set $Encs$ of applications of hash functions such that all components $g(\!|M|\!)$ from $Encs$ should be replaced with the body $M$. We define the transformation function $f$ as follows:

$$\begin{aligned}
f(\textsc{Atom}\,A) &\mathrel{\widehat{=}} \textsc{Atom}\,A,\\
f(M, M') &\mathrel{\widehat{=}} NF(f(M), f(M')),\\
f(\{M\}_K) &\mathrel{\widehat{=}} \{f(M)\}_K,\\
f(g(\!|M|\!)) &\mathrel{\widehat{=}} \begin{cases} f(M), & \text{if } g(\!|M|\!) \in Encs,\\ g(\!|f(M)|\!), & \text{otherwise.}\end{cases}
\end{aligned}$$

We again define the intruder's initial knowledge in the transformed protocol by:

$$IK_0' \mathrel{\widehat{=}} f(IK_0),$$

which satisfies condition (2). In Appendix B.2, we prove that the transformation satisfies the distribution property, condition (1). Hence, as with the previous transformation, we can deduce that this is a safe transformation with respect to secrecy, and with respect to agreement on atoms.

### 4.3 Removing some atomic or hashed fields

We now define a transformation that completely removes some fields from the protocol. We only allow these fields to be either atoms that are not keys, or the application of hash functions. Formally, we let

$$Ms \subseteq Message$$

be the set of such messages, and assume:

$$\begin{aligned}
\forall M \in Ms \bullet\ &(\exists A \in Atom \bullet M = \textsc{Atom}\,A \wedge A \notin Key)\\
&\vee\\
&(\exists g \in HashFn\,;\, M' \in Msg \bullet M = g(\!|M'|\!)).
\end{aligned}$$

The reason we allow only atoms and applications of hash functions to be removed is that the intruder would be unable to use such messages to deduce any other messages other than those containing this message itself. We do not allow keys to be removed, for clearly this would reduce the intruder's abilities.

In order to define this transformation, we need to specify how it acts upon a message that is to be removed, leaving an "empty message". The $Message$ datatype has no concept of an "empty message", so we model such "empty messages" by a distinguished atomic value $\mathsf{nil} \in Atom$ (distinct from all the atoms in the protocol).

We define the transformation as follows:

$$
\begin{aligned}
f(M) &\mathrel{\widehat{=}} \textsc{Atom}\,\mathsf{nil}, && \text{if } M \in Ms,\\
f(\textsc{Atom}\,A) &\mathrel{\widehat{=}} \textsc{Atom}\,A,\\
f(M, M') &\mathrel{\widehat{=}} \begin{cases} f(M'), & \text{if } f(M) = \textsc{Atom}\,\mathsf{nil},\\ f(M), & \text{if } f(M') = \textsc{Atom}\,\mathsf{nil},\\ (f(M), f(M')), & \text{otherwise,} \end{cases}\\
f(\{M\}_K) &\mathrel{\widehat{=}} \begin{cases} \textsc{Atom}\,\mathsf{nil}, & \text{if } f(M) = \textsc{Atom}\,\mathsf{nil},\\ \{f(M)\}_K, & \text{otherwise,} \end{cases}\\
f(g(\!|M|\!)) &\mathrel{\widehat{=}} \begin{cases} \textsc{Atom}\,\mathsf{nil}, & \text{if } f(M) = \textsc{Atom}\,\mathsf{nil},\\ g(\!|f(M)|\!), & \text{otherwise.} \end{cases}
\end{aligned}
$$

We adopt a convention from functional programming languages such as Haskell, namely that the patterns in the definition are matched from top to bottom, so that a definition further down the page will be applied only if no earlier definition applies; this means that the line with the pattern "$\textsc{Atom}\,A$" applies only to atoms that are not matched by the first line, i.e. only to atoms not in $Ms$.

For composite messages, the definition gets applied recursively to the subcomponents; if either member of a pair gets removed in this way, then we are left with the other member of the pair; if all of the body of an encryption or a hash function application gets removed, then the encryption or hash function is removed.

We define the intruder's initial knowledge in the transformed protocol by:

$$IK_0' \mathrel{\widehat{=}} f(IK_0) \cup \{\textsc{Atom}\,\mathsf{nil}\}.$$

Note that we assume the intruder can produce $\textsc{Atom}\,\mathsf{nil}$, i.e. an empty message. This definition clearly satisfies condition (2).

In Appendix B.3, we prove that the transformation satisfies the distribution property. We hence deduce that this is a safe transformation with respect to secrecy. Note that the transformation is safe even if we remove a field that we are trying to prove is secret; however, in this case the secret will be renamed to $\textsc{Atom}\,\mathsf{nil}$, which is in the intruder's initial knowledge, and so this transformation will introduce a new attack: such a transformation is safe but stupid.

If all the values upon which agreement is intended are atoms, and none are removed, i.e.:

$$
\begin{aligned}
&\forall\, Ms' \in \mathit{AgreementSet} \bullet \forall\, M \text{ in } Ms' \bullet\\
&\quad \exists\, A \in \mathit{Atom} \bullet M = \textsc{Atom}\,A \wedge M \notin Ms,
\end{aligned}
$$

then the transformation satisfies condition (3), and so we deduce that this is a safe transformation with respect to agreement on such values. The above condition simply states that if you want to be sure that the protocol guarantees agreement on a particular field, then your simplification shouldn't remove that field!

In practice, we will use this transformation to remove *all* fields of a particular type: to remove one field we would have to define $Ms$ to be all values with which that field could be instantiated, i.e. all values of that type; this would have the effect of removing all fields of the type.

### 4.4 Renaming atoms

We now consider a simplifying transformation that uniformly renames atoms in protocol messages. Let $f$ be a renaming function on atoms:

$$f \;:\; Atom \rightarrow Atom,$$

which respects inverses of keys:

$$\forall K \in Key \bullet f(K)^{-1} = f(K^{-1}).$$

We can lift $f$ to operate over complete messages, in the obvious way:

$$\begin{aligned} f(\textsc{Atom}\, A) \;&\widehat{=}\; \textsc{Atom}\, f(A), \\ f(M, M') \;&\widehat{=}\; (f(M), f(M')), \\ f(\{M\}_K) \;&\widehat{=}\; \{f(M)\}_{f(K)}, \\ f(g(\!|M|\!)) \;&\widehat{=}\; g(\!|f(M)|\!). \end{aligned}$$

And we can define the intruder's initial knowledge in the transformed protocol in the obvious way:

$$IK_0' \;\widehat{=}\; f(IK_0).$$

This definition clearly satisfies condition (2). We sketch the proof of condition (1) in Appendix B.4. Hence we deduce that this is a safe transformation with respect to secrecy. It is also a safe transformation with respect to the agreement property provided the function $f$ over messages is injective on the messages upon which agreement in intended.

### 4.5 Coalescing atoms

Our next transformation is one that coalesces certain pairs of atoms, replacing them by the first. We define the transformation with respect to a set of pairs of atoms:

$$Pairs \; : \; \mathbf{P}(Atom \times Atom),$$

the idea being that for each $(A, A') \in Pairs$, any occurrence of the pair $(\textsc{Atom}\, A, \textsc{Atom}\, A')$ is replaced by $\textsc{Atom}\, A$.

We need the following condition upon $Pairs$:

$$\forall (A, A') \in Pairs \bullet IK_0' \cup \{\textsc{Atom}\, A\} \vdash \textsc{Atom}\, A', \tag{4}$$

that is, the atom removed can be deduced from the atom remaining and the intruder's initial knowledge. If we did not have this condition, then removing the atom $A'$ would remove a possible source of information for the intruder. We will therefore tend to use this transformation in two situations: where the atom removed is in the intruder's initial knowledge, and where the atom removed is equal to the one remaining.

We formalize the transformation as follows:

$$\begin{array}{l}
f(\textsc{Atom}\, A) \mathrel{\widehat{=}} \textsc{Atom}\, A, \\
f(\textsc{Atom}\, A, \textsc{Atom}\, A') \mathrel{\widehat{=}} \\
\qquad \begin{cases} \textsc{Atom}\, A, & \text{if } (A, A') \in Pairs, \\ (\textsc{Atom}\, A, \textsc{Atom}\, A'), & \text{otherwise,} \end{cases} \\
f(\textsc{Atom}\, A, (\textsc{Atom}\, A', M)) \mathrel{\widehat{=}} \\
\qquad \begin{cases} f(\textsc{Atom}\, A, M), & \text{if } (A, A') \in Pairs, \\ (\textsc{Atom}\, A, f(\textsc{Atom}\, A', M)), & \text{otherwise,} \end{cases} \\
f(M, M') \mathrel{\widehat{=}} (f(M), f(M')), \\
f(\{M\}_K) \mathrel{\widehat{=}} \{f(M)\}_K, \\
f(g(\!|M|\!)) \mathrel{\widehat{=}} g(\!|f(M)|\!).
\end{array}$$

We define the intruder's initial knowledge in the transformed system by:

$$IK_0' \;\mathrel{\widehat{=}}\; f(IK_0).$$

This definition clearly satisfies condition (2). We prove condition (1) in Appendix B.4. Hence we deduce that this is a safe transformation with respect to secrecy. It is also a safe transformation with respect to the agreement property provided $f$ is injective on the messages upon which agreement in intended.

## 4.6 Swapping pairs

We now define a transformation that swaps particular pairs of atoms when they appear next to one another. Let $Pairs$ be a set of pairs of atoms:

$$Pairs \; : \; \mathbf{P}(Atom \times Atom).$$

We define a transformation that, for every pair $(A, A') \in Pairs$, swaps the atoms $A$ and $A'$ whenever they occur adjacently within a message:

$$
\begin{aligned}
& f(\textsc{Atom}\, A) \mathrel{\widehat=} \textsc{Atom}\, A, \\
& f(\textsc{Atom}\, A, \textsc{Atom}\, A') \mathrel{\widehat=} \\
& \qquad \begin{cases} (\textsc{Atom}\, A', \textsc{Atom}\, A), & \text{if } (A, A') \in Pairs, \\ (\textsc{Atom}\, A, \textsc{Atom}\, A'), & \text{otherwise,} \end{cases} \\
& f(\textsc{Atom}\, A, (\textsc{Atom}\, A', M)) \mathrel{\widehat=} \\
& \qquad \begin{cases} (\textsc{Atom}\, A', f(\textsc{Atom}\, A, M)), & \text{if } (A, A') \in Pairs, \\ (\textsc{Atom}\, A, f(\textsc{Atom}\, A', M)), & \text{otherwise,} \end{cases} \\
& f(M, M') \mathrel{\widehat=} (f(M), f(M')), \\
& f(\{M\}_K) \mathrel{\widehat=} \{f(M)\}_K, \\
& f(g(\!|M|\!)) \mathrel{\widehat=} g(\!|f(M)|\!).
\end{aligned}
$$

We again use the convention that patterns within the left hand sides of definitions like the above are matched from top to bottom, so that a particular pair will be matched against the fourth equation of the definition only if its form does not match either of the patterns in the second and third equations. Note that the form of the third equation means that within a sequence of adjacent atoms, the swaps are applied from left to right.

We can define the intruder's initial knowledge in the transformed protocol in the obvious way:

$$IK'_0 \mathrel{\widehat=} f(IK_0).$$

Clearly this definition satisfies condition (2). In Appendix B.6 we show that condition (1) is also satisfied. Hence we deduce that this is a safe transformation with respect to secrecy. Further, $f$ is injective on atoms, so this is a safe transformation with respect to agreement on atoms.

## 4.7 Removing fields from the bodies of encryptions

Finally, we define a transformation that allows some fields to be pulled out of the body of an encryption; for example a message of the form $\{M_1, M_2\}_K$

could be simplified to $M_1, \{M_2\}_K$. We define the transformation with respect to a set of messages *Encs*:

$$Encs \subseteq Message,$$

each member of which is an encryption of a pair:

$$\forall M \in Encs \bullet \exists M_1, M_2 \in Message\,; K \in Key \bullet M = \{M_1, M_2\}_K.$$

The transformation is then defined as follows:

$$\begin{aligned}
f(\textsc{Atom}\, A) &\mathrel{\hat=} \textsc{Atom}\, A, \\
f(M, M') &\mathrel{\hat=} NF(f(M), f(M')), \\
f(\{M\}_K) &\mathrel{\hat=} \begin{cases} NF(f(M_1), \{f(M_2)\}_K), & \\ & \text{if } \{M\}_K \in Encs \text{ and } M = (M_1, M_2), \\ \{f(M)\}_K, & \text{otherwise,} \end{cases} \\
f(g(\!|M|\!)) &\mathrel{\hat=} g(\!|f(M)|\!).
\end{aligned}$$

We define the intruder's initial knowledge in the transformed protocol in the normal way:

$$IK_0' \mathrel{\hat=} f(IK_0).$$

Clearly this definition satisfies condition (2). In Appendix B.7 we show that condition (1) is also satisfied. Hence we deduce that this is a safe transformation with respect to secrecy, and as $f$ is injective on atoms, this is a safe transformation with respect to agreement on atoms.

# 5 Case study: The CyberCash Main Sequence Protocol

We illustrate our simplifying transformations by applying them to the CyberCash Main Sequence Protocol. The protocol is designed to allow credit card purchases to take place securely between a customer and merchant, with the aid of CyberCash who acts as a trusted third party. The customer and merchant both send the relevant details of the transaction to CyberCash who verifies that both principals are indeed participating in the same transaction and agree on specific details.

A complete description of the protocol is included in Appendix A; the description is based upon Steve Brackin's description from [Bra97]. To help the reader understand the basis of the protocol, we give an overview below:

$$
\begin{array}{lrl}
1. & M \rightarrow C & : \textit{payment\_details} \\
2. & C \rightarrow M & : \{kcs\}_{PKCyberKey}, \{info\}_{kcs} \\
3. & M \rightarrow CB & : \{kcs\}_{PKCyberKey}, \{info\}_{kcs}, \\
 & & \quad \{kms\}_{PKMerchantKey}, \{more\_info\}_{kms} \\
4. & CB \rightarrow M & : \{reply1\}_{kms}, \{reply2\}_{kcs} \\
5. & M \rightarrow C & : \{reply2\}_{kcs},
\end{array}
$$

where $M$ is the merchant, $C$ is the customer and $CB$ is CyberCash.

The payment details in the first message include a description of the purchase, and the price of the goods. The customer generates a symmetric key $kcs$, which is used in subsequent communications with CyberCash; this key is sent to CyberCash, via the merchant, encrypted with the key $PKCyberKey$, which is a secret asymmetric key shared between the customer and CyberCash; the customer also sends some information about the transaction, encrypted with $kcs$. Similarly, the merchant generates a symmetric key $kms$, and sends it to CyberCash, encrypted with a secret shared asymmetric key $PKMerchantKey$; the merchant also sends some information about the transaction, encrypted with $kms$. CyberCash verifies that the messages are in agreement, and sends replies in messages 4 and 5.

## 5.1 Tool Support

In order to make the simplifying transformations easier to apply, we have produced tool support as an extension to Casper [Low98a]. The tool takes in a description of the complete protocol and a specification of a sequence of transformations to be applied. The syntax for the protocol description is precisely as in Casper; we describe the syntax for defining the transformations below.

The transformations in Section 4 were defined using values; however, protocols are normally described in terms of variables, which can be instantiated with different values (of the appropriate types) in different runs. We want to define transformations that are applied uniformly, independent of the values with which variables are instantiated. Therefore, the tool supports transformations defined in terms of types, and applies these transformations to all variables and values of the appropriate types.

Here are some examples of simplifying transformations as they should appear in the input file:

```
#Simplifications

RemoveFields [Nonce, TimeStamp]
RemoveEncryption {Nonce, Agent}{PublicKey}
RemoveHash f(Nonce, Nonce, TimeStamp)
Coalesce (Agent, Agent)
SwapPairs (Nonce, Agent)
```

The effect of these transformations is as follows:

`RemoveFields [Nonce, TimeStamp]` This removes every variable of type `Nonce` or `TimeStamp` from the protocol description; it corresponds to the transformation of Section 4.3.

`RemoveEncryption {Nonce, Agent}{PublicKey}` This transformation removes a level of encryption from the protocol description. The notation "`{Nonce, Agent}{PublicKey}`" is intended to represent the set of all messages of the form $\{N, A\}_K$ for $N$ in `Nonce`, $A$ in `Agent` and $K$ in `PublicKey`. This is the transformation of Section 4.1 where *Encs* is taken to be this set.

`RemoveHash f(Nonce, Nonce, TimeStamp)` This transformation removes some hash functions from the protocol description. This is the transformation of Section 4.2 where *Encs* is taken to be the set of all messages of the form $\mathtt{f}(N, N', T)$ for $N$, $N'$ in `Nonce`, $T$ in `TimeStamp`.

`Coalesce (Agent,Agent)` This transformation coalesces adjacent pairs of agent identities. This is the transformation described in Section 4.5, where *Pairs* is taken to be the set of all pairs of agent identities. At present, the tool only coalesces atoms that are the same type.

`SwapPairs (Nonce, Agent)` This transformation swaps adjacent pairs of nonces and atoms. It is the transformation of Section 4.6 where *Pairs* is taken to be the the set of all (`Nonce`, `Agent`) pairs.

Not all the transformations from Section 4 are currently supported: only those for which we have found a need in our analysis of the CyberCash protocol.

### 5.2 Simplifying the protocol

The complete protocol contains dozens of fields, but very few seem directly relevant to the security of the protocol. Many fields are included solely for reasons of functionality, for example the fields representing the customer's address (if it is the intention that such fields are authenticated, then we could perform extra analyses, retaining each such field in turn). We describe, below, those fields that we think are necessary; the rest can be removed from our analysis.

We want to check whether the protocol leads to the agents agreeing upon the price paid, and so we keep the fields ($Amount$ and $MerchantAmount$) recording the price. We also want to test whether the protocol keeps the customer's credit card number secret, so we retain the relevant field ($CardNumber$).

The agents' identities and all the keys are clearly necessary, and so are retained. Further, we suspect that the timestamps and transaction numbers (which act as a kind of run identifier) are necessary to prevent replay attacks, so we retain them.

Our first simplification is therefore a "remove fields" simplification, to remove all atomic fields of types other than those mentioned above.

We also remove all fields that are hashes of public keys, since these fields seem to carry no information (these fields are included to make the implementation more efficient).

Applying these simplifications, the protocol from Appendix A becomes:

$$
\begin{array}{lcl}
1. & M \rightarrow C & : MerchantAmount, MerchantCcId, MerchantDate, \\
 & & \quad MerchantSignedHash\_ \\
2. & C \rightarrow M & : Date, Id, MerchantCcId, MerchantDate, \\
 & & \quad Transaction, PrHash\_, PrSignedHash\_, \\
 & & \quad OpaqPrefixCH1\_, OpaqueCH1\_ \\
3. & M \rightarrow CB & : MerchantCcId, MerchantDate, \\
 & & \quad MerchantTransaction, OpaqPrefixCH1\_, \\
 & & \quad OpaqueCH1\_, MerchantOpaqPrefixCM1\_, \\
 & & \quad MerchantOpaqueCM1\_ \\
4. & CB \rightarrow M & : MerchantCcId, MerchantTransaction, \\
 & & \quad MerchantDate, MerchantOpaqueCM6\_, \\
 & & \quad OpaqueCM6\_ \\
5. & M \rightarrow C & : Date, MerchantCcId, MerchantDate, Id, \\
 & & \quad Transaction, PrHash\_, PrSignedHash\_, \\
 & & \quad OpaqueCM6\_
\end{array}
$$

The fields with names ending with an underscore are abbreviations, which are simplified by these transformations to:

$$MerchantSignedHash\_ \mathrel{\widehat{=}} \{MD5(MerchantDate, MerchantAmount, MerchantCcId)\}_{SK(M)}$$

$$PrHash\_ \mathrel{\widehat{=}} MD5(Date, MerchantAmount, MerchantCcId)$$

$$PrSignedHash\_ \mathrel{\widehat{=}} MerchantSignedHash\_$$

$$OpaqPrefixCH1\_ \mathrel{\widehat{=}} \{kcs\}_{PKCyberKey}$$

$$OpaqueCH1\_ \mathrel{\widehat{=}} \{Amount, CardNumber, SignatureCH1\_\}_{kcs}$$

$$SignatureCH1\_ \mathrel{\widehat{=}} \{MD5(Amount, CardNumber, Date, Id, MerchantCcId, PrHash\_, PrSignedHash\_, Transaction)\}_{SK(C)}$$

$$MerchantOpaqPrefixCM1\_ \mathrel{\widehat{=}} \{kms\}_{PKMerchantCyberKey}$$

$$MerchantOpaqueCM1\_ \mathrel{\widehat{=}} \{Date, Id, MerchantAmount, PrHash\_, PrSignedHash\_, Transaction, MerchantSignatureCM1\_\}_{kms}$$

$$
\begin{array}{l}
MerchantSignatureCM1\_ \mathrel{\widehat{=}} \\
\quad \{MD5(Date, Id, MerchantAmount, MerchantCcId, \\
\qquad\quad MerchantDate, MerchantTransaction, PrHash\_, \\
\qquad\quad PrSignedHash\_, Transaction) \\
\quad \}_{SK(M)} \\
MerchantOpaqueCM6\_ \mathrel{\widehat{=}} \\
\quad \{CardNumber, Date, Id, PrHash\_, PrSignedHash, \\
\quad\; ServerDate, Transaction \\
\quad \}_{kms} \\
OpaqueCM6\_ \mathrel{\widehat{=}} \{Amount, CardNumber, ServerDate\}_{kcs}
\end{array}
$$

The protocol description contains many occurrences of the fields *PrSignedHash_* (also known as $MerchantSignedHash\_$) and $PrHash\_$, both of which are formed as applications of hash functions. These fields appear to duplicate security provided by other parts of the protocol. We therefore remove them, as an instance of the transformation of Section 4.3 to remove atomic or hashed fields.

The next simplification is to remove all applications of the hash function $MD5$. Many messages in the protocol are hashed, to reduce the amount of data transmitted. However, this hashing appears to have no security implications, so we remove the hashes for ease of analysis.

Analyzing the effects of this transformation, we see that $OpaqueCH1\_$ changes to:

$$
\begin{array}{l}
\{Amount, CardNumber, \\
\;\{Amount, CardNumber, Date, Id, MerchantCcId, Transaction\}_{SK(C)} \\
\}_{kcs}
\end{array}
$$

and $MerchantOpaqueCM1\_$ changes to

$$
\begin{array}{l}
\{Date, Id, MerchantAmount, Transaction \\
\;\{Date, Id, MerchantAmount, MerchantCcId, MerchantDate, \\
\;\; MerchantTransaction, Transaction \\
\;\}_{SK(M)} \\
\}_{kms}
\end{array}
$$

There is nested encryption within $OpaqueCH1\_$ and *MerchantOpaqueCM1_*, which appears unnecessary. We therefore perform a removing encryptions transformation to strip off the inner encryptions.

Several data items are repeated within some encryptions. By swapping pairs of atoms around we can bring together like atoms, which can then be coalesced to a single atom. Doing this, produces a fully simplified protocol as follows:

$$\begin{array}{lll}
1. & M \rightarrow C & : MerchantAmount, MerchantDate, MerchantCcId \\
2. & C \rightarrow M & : Transaction, Date, Id, MerchantDate, \\
 & & \quad MerchantCcId, \{kcs\}_{PKCyberKey}, \\
 & & \quad \{Amount, CardNumber, Transaction, Date, Id, \\
 & & \qquad MerchantCcId\}_{kcs} \\
3. & M \rightarrow CB & : MerchantTransaction, MerchantDate, \\
 & & \quad MerchantCcId, \{kcs\}_{PKCyberKey}, \\
 & & \quad \{Amount, CardNumber, Transaction, Date, \\
 & & \qquad Id, MerchantCcId\}_{kcs}, \\
 & & \quad \{kms\}_{PKMerchantCyberKey}, \\
 & & \quad \{Transaction, MerchantTransaction, Date, Id, \\
 & & \qquad MerchantAmount, MerchantDate, \\
 & & \qquad MerchantCcId\}_{kms} \\
4. & CB \rightarrow M & : MerchantTransaction, MerchantDate, \\
 & & \quad MerchantCcId, \\
 & & \quad \{CardNumber, Transaction, Date, Id\}_{kms}, \\
 & & \quad \{Amount, CardNumber, ServerDate\}_{kcs} \\
5. & M \rightarrow C & : Transaction, Date, Id, MerchantCcId, \\
 & & \quad \{Amount, CardNumber, ServerDate\}_{kcs} .
\end{array}$$

## 5.3 Analysis of the simplified protocol

We have analyzed the simplified version of the Main Sequence Protocol using Casper and FDR [Low98a], testing whether the protocol kept the customer's credit card number and the session keys secret, and testing whether the two agents were authenticated to one another with agreement on the amount of the transaction.

We found a rather weak authentication attack upon the protocol, where the intruder watches a run of the protocol up until message 4, intercepts the message 4, and then fakes the message 5: the intruder is able to produce a suitable message 5 from what he has seen in previous messages, both in the simplified protocol, and in the original protocol. The effect of this attack is that the customer thinks he has completed the protocol run successfully, but the merchant thinks the run was broken off after message 3. It is arguable whether this should be classified as an attack; however, it is certainly a behaviour that we should be aware of. This is an attack according to our definition of authentication [Low97], which is that when one agent completes a run of the protocol, the other agent should at least have progressed as far as the last message it sends; this is captured in our formal model by the position of the $finished$ and $running$ events (see Section 2.4). Other definitions of authentication also classify this as an attack (for example, Roscoe's intensional specifications [Ros96]). If the merchant's $running$ event is moved to precede its message 3, we no longer find any attack.

We found another attack, because we oversimplified the protocol. We removed the field $MerchantCcId$, because we didn't realize that this was effectively the merchant's identity; unsurprisingly, this introduced an authentication attack. Fortunately, it proved fairly easy to undo all of our simplifications, and to realize that there was no corresponding attack on the original protocol, and hence that we had oversimplified.

Failure to find an attack using a model checker does not (in the absence of arguments like those in [Low98b, Ros98]) necessarily imply that the protocol is secure: there might be attacks upon systems running the protocol that are larger than those analysed. However, the model checking results certainly improve our confidence in the security of the protocol, to the point where we think it would be worth performing a formal proof; such a proof could be performed on the simplified protocol, and, if successful, the results would also apply to the original protocol.

The overall effect of these simplifications is to reduce the protocol to a version which can be easily understood at first glance. The main aspects of the protocol have been retained to ensure it still fulfills its original function. By applying the simplification transformations defined in this paper the protocol has been reduced to a simpler version without omitting attacks.

Our analysis also suggests the protocol could be considerably simplified, while retaining the same security and functionality: for example, the field $PrSignedHash\_$, also known as $MerchantSignedHash\_$, appears unnecessary, as do the nested encryptions within $OpaqueCH1\_$ and $MerchantOpaqueCM1\_$.

# 6 Conclusions

In this paper we have introduced the notion of *safe simplifying transformations*: transformations that have the property of preserving attacks. We have produced sufficient conditions for a transformation to be safe, and used this result to show that a number of transformations are indeed safe. We have illustrated these techniques by applying them to a large commercial protocol.

We close by giving brief pointers to future work. We intend to apply this techniques to more commercial protocols, for example SET [SET96] and SSL [SSL96]. It is likely that such case studies will reveal further useful safe simplifying transformations.

We would like to extend this work to deal with other forms of authentication specification, and also other security properties such as non-repudiation.

One class of simplifying transformation that is not captured by the results of this paper is what we are terming *redirecting transformations*. Consider the Otway-Rees Protocol [OR87]:

$$\begin{array}{lll}
1. & A \to B & : M, A, B, \{N_a, M, A, B\}_{K_{as}} \\
2. & B \to S & : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}} \\
3. & S \to B & : M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}} \\
4. & B \to A & : M, \{N_a, K_{ab}\}_{K_{as}} .
\end{array}$$

Here, $A$ sends an encrypted message intended for $S$ to $B$ in message 1, which $B$ forwards to $S$ in message 2. Similarly, $S$ sends an encrypted message intended for $A$ to $B$ in message 3, which $B$ forwards to $A$ in message 4. It seems clear to us that sending these encrypted messages directly to the intended recipient would give an equally secure protocol; in other words, we believe it would be safe to transform the protocol to the following:

$$\begin{array}{lll}
1a. & A \to B & : M, A, B \\
1b. & A \to S & : M, A, B, \{N_a, M, A, B\}_{K_{as}} \\
2. & B \to S & : M, A, B, \{N_b, M, A, B\}_{K_{bs}} \\
3. & S \to B & : M, \{N_b, K_{ab}\}_{K_{bs}} \\
4. & S \to A & : M, \{N_a, K_{ab}\}_{K_{as}} .
\end{array}$$

## Acknowledgements

This research was supported by grants from the United Kingdom Engineering and Physical Sciences Research Council, and the United States Office of Naval Research.

## References


[Boy97] Colin Boyd. Extensional goals in authentication protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[Bra97] Stephen Brackin. Automatic formal analysis of two large commercial protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[GM95] James W. Gray, III and John McLean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, 1995.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[KMM94] Richard Kemmerer, Catherine Meadows, and Jonathan Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2), 1994.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997.

[Low98a] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html`.

[Low98b] Gavin Lowe. Towards a completeness result for model checking of security protocols (extended abstract). In *11th Computer Security Foundations Workshop*, 1998.

[LR97] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[MCF87] J. K. Millen, S. C. Clark, and S. B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on software Engineering*, 13(2), 1987.

[MCJ97] Will Marrero, Edmund Clarke, and Somesh Jha. A model checker for authentication protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[Mea96] Catherine Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Mur$\phi$. In *IEEE Symposium on Security and Privacy*, 1997.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[Pau97] Lawrence C. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, 1997.

[Ros96] A. W. Roscoe. Intensional specification of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38, 1996.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[Ros98] A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *11th Computer Security Foundations Workshop*, pages 84–95, 1998.

[Sch96] Steve Schneider. Security properties and CSP. In *IEEE Computer Society Symposium on Security and Privacy*, Oakland, 1996.

[Sch97] Steve Schneider. Verifying authentication protocols with CSP. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, 1997.

[SET96] Secure Electronic Transaction (SET) specification. Available from http://www.visa.com/cgi-bin/vee/sf/set/intro.html?2+0, 1996.

[SSL96] The SSL protocol. Available via http://home.netscape.com/newsref/std/SSL.html, 1996.

[SvO94] Paul Syverson and Paul van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Symposium on Research in Security and Privacy*, 1994.

[THG98] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *1998 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1998.

# A The CyberCash Main Sequence Protocol

The CyberCash Main Sequence Protocol can be described as follows:

$$
\begin{array}{rrcl}
1 & M \to C & : & Accepts, MerchantAmount, \\
 & & & MerchantAmount2Optional, MerchantCcId, \\
 & & & MerchantOrderId, MerchantDate, \\
 & & & MerchantSwVersion, Note, Payload, \\
 & & & PayloadNote, Type, UrlCancel, UrlFail, \\
 & & & UrlPayTo, UrlSuccess, MD5(Payload), \\
 & & & MerchantSignedHashKey\_, \\
 & & & MerchantSignedHash\_ \\
2 & C \to M & : & CyberKey, Date, Id, MerchantCcId, \\
 & & & MerchantDate, MerchantSignedHashKey\_, \\
 & & & OrderId, ServiceCategory, Transaction, Type, \\
 & & & PrHash\_, PrSignedHash\_, OpaqPrefixCH1\_, \\
 & & & OpaqueCH1\_ \\
3 & M \to CB & : & CyberKey, MerchantCcId, MerchantCyberKey, \\
 & & & MerchantDate, MerchantTransaction, \\
 & & & ServiceCategory, OpaqPrefixCH1\_, \\
 & & & OpaqueCH1\_, MerchantOpaqPrefixCM1\_, \\
 & & & MerchantOpaqueCM1\_ \\
4 & CB \to M & : & MerchantCcId, MerchantTransaction, \\
 & & & MerchantDate, ServiceCategory, \\
 & & & MerchantOpaqueCM6\_, OpaqueCM6\_ \\
5 & M \to C & : & Date, MerchantCcId, MerchantDate, \\
 & & & MerchantMessage, MerchantResponseCode, \\
 & & & MerchantSwVersion, Id, ServiceCategory, \\
 & & & Transaction, Type, PrHash\_, PrSignedHash\_, \\
 & & & OpaqueCM6\_
\end{array}
$$

Those fields whose names finish with an underscore are abbreviations, defined as follows:

$$MerchantSignedHashKey\_ \mathrel{\widehat{=}} MD5(PK(M)),$$

$$
\begin{aligned}
&MerchantSignedHash\_ \mathrel{\widehat{=}} \\
&\quad \{MD5(Accepts, MerchantDate, MerchantAmount, \\
&\qquad\quad MerchantCcId, MerchantOrderId, \\
&\qquad\quad MerchantSignedHashKey\_, Note, Type, \\
&\qquad\quad UrlCancel, UrlFail, UrlPayto, UrlSuccess)\}_{SK(M)},
\end{aligned}
$$

$$
\begin{aligned}
&PrHash\_ \mathrel{\widehat{=}} \\
&\quad MD5(Accepts, Date, MerchantAmount, MerchantCcId, \\
&\qquad\quad MerchantOrderId, MerchantSignedHashKey\_, Note, \\
&\qquad\quad Type, UrlCancel, UrlFail, UrlPayTo, UrlSuccess),
\end{aligned}
$$

$$PrSignedHash\_ \mathrel{\widehat{=}} MerchantSignedHash\_,$$

$$OpaqPrefixCH1\_ \mathrel{\widehat{=}} \{kcs\}_{PKCyberKey},$$

$$
\begin{aligned}
&OpaqueCH1\_ \mathrel{\widehat{=}} \\
&\quad \{Amount, CardCIdOptional, CardCityOptional, \\
&\quad\ CardCountryOptional, CardExpirationDate, CardName, \\
&\quad\ CardNumber, CardOtherFieldsOptional, \\
&\quad\ CardPostalCodeOptional, CardPrefixOptional, CardSalt, \\
&\quad\ CardStateOptional, CardStreetOptional, CardType, \\
&\quad\ SwVersion, MD5(PK(C)), SignatureCH1\_\}_{kcs},
\end{aligned}
$$

$$
\begin{aligned}
&SignatureCH1\_ \mathrel{\widehat{=}} \\
&\quad \{MD5(Amount, CardCIdOptional, CardCityOptional, \\
&\qquad\quad CardCountryOptional, CardExpirationDate, CardName, \\
&\qquad\quad CardNumber, CardOtherFieldsOptional, \\
&\qquad\quad CardPostalCodeOptional, CardPrefixOptional, \\
&\qquad\quad CardSalt, CardStateOptional, CardStreetOptional, \\
&\qquad\quad CardType, CyberKey, Date, Id, MerchantCcId, \\
&\qquad\quad MerchantSignedHashKey\_, OrderId, PrHash\_, \\
&\qquad\quad PrSignedHash\_, SwVersion, Transaction, Type)\}_{SK(C)},
\end{aligned}
$$

$$MerchantOpaqPrefixCM1\_ \mathrel{\widehat{=}} \{kms\}_{PKMerchantCyberKey},$$

$MerchantOpaqueCM1\_ \mathrel{\widehat{=}}$
$\{Date, DescriptionListOptional, Id, MerchantAmount,$
$MerchantDba, MerchantLocationOptional, MerchantMessage,$
$MerchantSignedHashKey\_, MerchantSwMessageOptional,$
$MerchantSwServerOptional, MerchantSwVersion,$
$MerchantUrlOptional, OrderId, PrHash\_, PrSignedHash\_,$
$RetrievalReferenceNumberOptional,$
$ServerDateMerchantOptional, TerminalIdFuture,$
$Transaction, TransactionDescriptionOptional, Type,$
$MD5(PKMerchantCyberKey), MerchantSignatureCM1\_\}_{kms},$

$MerchantSignatureCM1\_ \mathrel{\widehat{=}}$
$\{MD5($
$CyberKey, Date, Id, MerchantAmount, MerchantCcId,$
$MerchantCyberKey, MerchantDate, MerchantTransaction,$
$OrderId, PrHash\_, PrSignedHash\_,$
$ServerDateMerchantOptional, Transaction, Type)\}_{SK(M)},$

$MerchantOpaqueCM6\_ \mathrel{\widehat{=}}$
$\{AcquirerRefDataOptional, ActionCode,$
$Addn1ResponseDataOptional, AuthorizationCode,$
$AvsInfoOptional, CardCIdOptional, CardCityOptional,$
$CardCountryOptional, CardExpirationDate, CardName,$
$CardNumber, CardPostalCodeOptional, CardPrefixOptional,$
$CardStateOptional, CardStreetOptional, CardType, Date,$
$DebuggingInfoOptional, Id, MerchantMessage,$
$MerchantSignedHashKey\_, MerchantSwMessageOptional,$
$MerchantSwSeverityOptional, OrderId,$
$ProcessorErrorCodeFuture, PrHash\_, PrSignedHash\_,$
$MerchantResponseCode, ResponseDetailCodeFuture,$
$RetrievalReferenceNumberOptional, ServerDate,$
$TerminalIdFuture, Transaction, Type\}_{kms},$

$$
\begin{aligned}
&OpaqueCM6\_ \mathrel{\widehat=}\\
&\quad \{Amount, AuthorizationCode, CardCIdOptional,\\
&\quad\ CardCityOptional, CardCountryOptional, CardExpirationDate,\\
&\quad\ CardName, CardNumber, CardOtherFieldsOptional,\\
&\quad\ CardPostalCodeOptional, CardPrefixOptional, CardSalt,\\
&\quad\ CardStateOptional, CardStreetOptional, CardTypeOptional,\\
&\quad\ MerchantDba, MerchantLocationOptional, MerchantUrlOptional,\\
&\quad\ Message, OrderId, ResponseCode, ServerDate,\\
&\quad\ SwMessageOptional, SwSeverityOptional,\\
&\quad\ TransactionDescriptionOptional\}_{kcs}.
\end{aligned}
$$

# B Some proofs of transformations

In this appendix we prove that some of the transformations defined in Section 4 satisfy the distribution property:

$$\forall B \in \mathbf{P}(Message) \; ; M \in Message \bullet B \vdash M \Rightarrow f(B) \cup IK_0' \vdash f(M).$$

We begin by stating two extra properties of the derivation relation:

**Lemma 10**

$$
\begin{aligned}
B \vdash M \wedge B \subseteq B' \;&\Rightarrow\; B' \vdash M,\\
B \vdash M \wedge B \cup \{M\} \vdash M' \;&\Rightarrow\; B \vdash M'.
\end{aligned}
$$

If the intruder can deduce a message $M$ from some set $B$, then he can certainly deduce it from some larger set $B'$; and if the intruder can deduce a message $M$, then he can use that message in subsequent deductions. These properties can be proved by induction over the defining rules for $\vdash$.

## B.1 Removing encryptions

In this section we prove that the simplifying transformation for removing encryptions, defined in Section 4.1, satisfies the distribution property. The proof is by rule induction over the derivation $B \vdash M$, and proceeds by a case analysis on the form of this derivation.

**member** If $M \in B$ then $f(M) \in f(B)$, so $f(B) \cup IK_0' \vdash f(M)$.

**pairing** If $M = (M_1, M_2)$ and $B \vdash M_1$ and $B \vdash M_2$, then by the inductive hypotheses, $f(B) \cup IK_0' \vdash f(M_1)$ and $f(B) \cup IK_0' \vdash f(M_2)$, so $f(B) \cup IK_0' \vdash NF(f(M_1), f(M_2))$ by Lemma 3; i.e. $f(B) \cup IK_0' \vdash f(M)$.

**splitting** If $B \vdash (M, M')$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M, M') = NF(f(M), f(M'))$; so $f(B) \cup IK'_0 \vdash f(M)$ by Lemma 3; the symmetric case is similar.

**encryption** If $M = \{M'\}_K$ and $B \vdash M' \wedge B \vdash \textsc{Atom}\,K$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M')$ and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K) = \textsc{Atom}\,K$. There are two subcases to consider.

- Case $M \in \mathit{Encs}$: then $f(M) = f(M')$, so $f(B) \cup IK'_0 \vdash f(M)$.
- Case $M \notin \mathit{Encs}$: then $f(M) = \{f(M')\}_K$ so $f(B) \cup IK'_0 \vdash f(M)$, by an application of the **encryption** rule for $\vdash$.

**decryption** If $B \vdash \{M\}_K$ and $B \vdash \textsc{Atom}\,K^{-1}$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(\{M\}_K)$ and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K^{-1}) = \textsc{Atom}\,K^{-1}$. Again, there are two cases to consider:

- Case $\{M\}_K \in \mathit{Encs}$. Then $f(\{M\}_K) = f(M)$ so $f(B) \cup IK'_0 \vdash f(M)$.
- Case $\{M\}_K \notin \mathit{Encs}$. Then $f(B) \cup IK'_0 \vdash f(\{M\}_K) = \{f(M)\}_K$, and so $f(B) \cup IK'_0 \vdash f(M)$ by an application of the **decryption** rule for $\vdash$.

**hashing** If $M = g(\!|M'|\!)$ and $B \vdash M'$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M')$, and so $f(B) \cup IK'_0 \vdash g(\!|f(M')|\!) = f(g(\!|M'|\!)) = f(M)$.

## B.2 Removing hash functions

We now prove that the transformation for removing hash functions, defined in Section 4.2, satisfies the distribution property. The proof is again by rule induction, and most of the cases are very similar to those in the previous section; the interesting case is the following:

**hashing** If $M = g(\!|M'|\!) \wedge B \vdash M'$ then by the inductive hypothesis $f(B) \cup IK'_0 \vdash f(M')$. There are two subcases to consider.

- Case $g(\!|M'|\!) \in \mathit{Encs}$: then $f(M) = f(M')$ so $f(B) \cup IK'_0 \vdash f(M)$.
- Case $g(\!|M'|\!) \notin \mathit{Encs}$: then $f(M) = g(\!|f(M')|\!)$, so by an application of the **hashing** rule for $\vdash$, $f(B) \cup IK'_0 \vdash g(\!|f(M')|\!) = f(M)$.

## B.3 Removing some atomic or hashed fields

We now go on to consider the transformation for removing some atomic or hashed fields, defined in Section 4.3. We prove the distribution condition by rule induction, as follows:

**member** If $M \in B$ then $f(M) \in f(B)$ so $f(B) \cup IK'_0 \vdash f(M)$.

**pairing** If $M = (M_1, M_2)$ and $B \vdash M_1 \wedge B \vdash M_2$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M_1) \wedge f(B) \cup IK'_0 \vdash f(M_2)$. Applying the **pairing** rule we get $f(B) \cup IK'_0 \vdash (f(M_1), f(M_2))$; we can perform a simple case analysis to deduce that $f(B) \cup IK'_0 \vdash f(M)$.

**splitting** If $B \vdash (M, M')$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M, M')$. Note that $(M, M') \notin Ms$ due to the definition of $Ms$. We perform a case analysis on $(M, M')$.

- Case $f(M) = \textsc{Atom}\,\mathsf{nil}$: then $f(B) \cup IK'_0 \vdash f(M)$, from the definition of $IK'_0$.
- Case $f(M') = \textsc{Atom}\,\mathsf{nil}$: then $f(M, M') = f(M)$ and so $f(B) \cup IK'_0 \vdash f(M)$.
- Case $f(M) \neq \textsc{Atom}\,\mathsf{nil} \wedge f(M') \neq \textsc{Atom}\,\mathsf{nil}$: then $f(B) \cup IK'_0 \vdash (f(M), f(M'))$; applying the **splitting** rule gives $f(B) \cup IK'_0 \vdash f(M)$ and $f(B) \cup IK'_0 \vdash f(M')$.

**encryption** If $M = \{M'\}_K$, $B \vdash M'$ and $B \vdash \textsc{Atom}\,K$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M')$ and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K) = \textsc{Atom}\,K$, as no encrypting keys are in $Ms$. Again, $M \notin Ms$, because of the definition of $Ms$. If $f(M') = \textsc{Atom}\,\mathsf{nil}$ then $f(B) \cup IK'_0 \vdash \textsc{Atom}\,\mathsf{nil} = f(M)$. If $f(M') \neq \textsc{Atom}\,\mathsf{nil}$ then applying the **encryption** rule we get $f(B) \cup IK'_0 \vdash \{f(M')\}_K = f(M)$.

**decryption** If $B \vdash \{M\}_K$ and $B \vdash \textsc{Atom}\,K^{-1}$, then by the inductive hypothesis $f(B) \cup IK'_0 \vdash f(\{M\}_K)$, and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K^{-1}) = \textsc{Atom}\,K^{-1}$, because no decrypting keys are in $Ms$. Note also that $\{M\}_K \notin Ms$, from the definition of $Ms$. If $f(M) = \textsc{Atom}\,\mathsf{nil}$ then $f(B) \cup IK'_0 \vdash \textsc{Atom}\,\mathsf{nil} = f(M)$. If $f(M) \neq \textsc{Atom}\,\mathsf{nil}$ then $B \cup IK'_0 \vdash f(\{M\}_K) = \{f(M)\}_K$; applying the **decryption** rule we get $f(B) \cup IK'_0 \vdash f(M)$.

**hashing** If $B \vdash M'$ and $M = g(\!|M'|\!)$, then by the inductive hypothesis $f(B) \cup IK'_0 \vdash f(M')$. If $f(M') = \textsc{Atom}\,\mathsf{nil}$ then $f(B) \cup IK'_0 \vdash$

$\textsc{Atom}\,\mathsf{nil} = f(M)$. If $f(M') \neq \textsc{Atom}\,\mathsf{nil}$ then applying the **hashing** rule gives $f(B) \cup IK'_0 \vdash g(\!|f(M')|\!) = f(M)$.

### B.4 Renaming atoms

We now sketch the proof for the transformation from Section 4.4, which renames atomic fields uniformly. Most of the cases in the rule induction are very straightforward, so we only include the following:

**decryption** If $B \vdash \{M\}_K$ and $B \vdash \textsc{Atom}\,K^{-1}$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(\{M\}_K) = \{f(M)\}_{f(K)}$ and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K^{-1}) = \textsc{Atom}(f(K))^{-1}$, because $f$ respects inverses; hence, applying the **decryption** rule gives $f(B) \cup IK'_0 \vdash f(M)$.

### B.5 Coalescing atoms

We now consider the coalescing atoms transformation from Section 4.5. We begin by proving a subsidiary lemma about this transformation:

**Lemma 11**

$$IK'_0 \cup \{f(M, M')\} \;\vdash\; f(M), \tag{5}$$

$$IK'_0 \cup \{f(M, M')\} \;\vdash\; f(M'), \tag{6}$$

$$IK'_0 \cup \{f(M), f(M')\} \;\vdash\; f(M, M'). \tag{7}$$

**Proof:** By induction on the structure of $(M, M')$.

- Case $M = \textsc{Atom}\,A$, $M' = \textsc{Atom}\,A'$ and $(A, A') \in \mathit{Pairs}$: then $f(M, M') = \textsc{Atom}\,A = f(M)$ so (5) and (7) are proven. Using the assumption (4), we get $IK'_0 \cup \{\textsc{Atom}\,A\} \vdash \textsc{Atom}\,A' = f(M')$ so $IK'_0 \cup f(M, M') \vdash \textsc{Atom}\,A' = f(M')$ and (6) is proven.
- Case $M = \textsc{Atom}\,A$, $M' = (\textsc{Atom}\,A', M'')$ and $(A, A') \in \mathit{Pairs}$: then $f(M, M') = f(\textsc{Atom}\,A, M'')$. Applying part (5) of the inductive hypothesis we get

  $$IK'_0 \cup \{f(M, M')\} = IK'_0 \cup \{f(\textsc{Atom}\,A, M'')\} \vdash f(\textsc{Atom}\,A),$$

  and (5) is proven. Using (4) and the previous equation we get

  $$IK'_0 \cup \{f(M, M')\} \vdash \textsc{Atom}\,A' = f(\textsc{Atom}\,A').$$

  By applying part (6) of the inductive hypothesis we get

  $$IK'_0 \cup \{f(M, M')\} = IK'_0 \cup \{f(\textsc{Atom}\,A, M'')\} \vdash f(M'').$$

By applying part (7) of the inductive hypothesis we get

$$IK_0' \cup \{f(\textsc{Atom}\,A'), f(M'')\} \vdash f(\textsc{Atom}\,A', M'').$$

Combining the last three equations we get

$$IK_0' \cup \{f(M, M')\} \vdash f(\textsc{Atom}\,A', M'') = f(M'),$$

proving (6). Finally, part (6) of the inductive hypothesis gives

$$IK_0' \cup \{f(M')\} = IK_0' \cup \{f(\textsc{Atom}\,A', M'')\} \vdash f(M'').$$

And part (7) of the inductive hypothesis gives

$$IK_0' \cup \{f(M), f(M'')\} \vdash f(M, M'').$$

So

$$IK_0' \cup \{f(M), f(M')\} \vdash f(M, M'),$$

thus proving (7).

- In all other cases, $f(M, M') = (f(M), f(M'))$ and the result follows immediately from the **splitting** and **pairing** rules.

□

We can now prove that the transformation satisfies the distribution condition. Most of the cases are very similar to those for previous transformations. We include only the proofs for **pairing** and **splitting**.

**pairing** If $M = (M_1, M_2)$ and $B \vdash M_1 \wedge B \vdash M_2$, then by the inductive hypothesis, $f(B) \cup IK_0' \vdash f(M_1) \wedge f(B) \cup IK_0' \vdash f(M_2)$. Then by part (7) of Lemma 11, $f(B) \cup IK_0' \vdash f(M_1, M_2) = f(M)$.

**splitting** If $B \vdash (M, M')$, then by the inductive hypothesis, $f(B) \cup IK_0' \vdash f(M, M')$. Then by part (5) of Lemma 11, $f(B) \cup IK_0' \vdash f(M)$, and by part (6), $f(B) \cup IK_0' \vdash f(M')$.

## B.6 Swapping pairs

We now consider the transformation for swapping adjacent pairs of atoms, from Section 4.6. We again need an extra lemma about the transformation.

**Lemma 12**

$$IK_0' \cup \{f(M, M')\} \vdash f(M), \tag{8}$$

$$IK_0' \cup \{f(M, M')\} \vdash f(M'), \tag{9}$$

$$IK_0' \cup \{f(M), f(M')\} \vdash f(M, M'). \tag{10}$$

This is proved by induction over the structure of $(M, M')$, in essentially the same way as for Lemma 11.

The proof of the distribution property is now straightforward. Most of the cases are similar to in previous proofs; the **pairing** and **splitting** cases use Lemma 12, and are similar to the corresponding cases for the coalescing pairs transformation.

### B.7 Removing fields from the bodies of encryptions

We now consider the transformation from Section 4.7, which moves fields out of the bodies of certain encryptions. We show that this transformation satisfies the distribution property, by rule induction. Most of the cases are very similar to as in previous proofs; we include only the following cases:

**encryption** If $M = \{M'\}_K$ and $B \vdash M'$ and $B \vdash K$, then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(M')$ and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K) = \textsc{Atom}\,K$. The case where $\{M'\}_K \notin \mathit{Encs}$ is as in previous proofs, so suppose $\{M'\}_K \in \mathit{Encs}$ and let $M' = (M_1, M_2)$. Then $f(B) \cup IK'_0 \vdash f(M_1, M_2) = NF(f(M_1), f(M_2))$, and so $f(B) \cup IK'_0 \vdash f(M_1)$ and $f(B) \cup IK'_0 \vdash f(M_2)$ by Lemma 3. Hence $f(B) \cup IK'_0 \vdash NF(f(M_1), \{f(M_2)\}_K) = f(M)$ by an application of the **encryption** rule for $\vdash$ and Lemma 3.

**decryption** If $B \vdash \{M\}_K$ and $B \vdash \textsc{Atom}\,K^{-1}$ then by the inductive hypothesis, $f(B) \cup IK'_0 \vdash f(\{M\}_K)$ and $f(B) \cup IK'_0 \vdash f(\textsc{Atom}\,K^{-1}) = \textsc{Atom}\,K^{-1}$. The case where $\{M\}_K \notin \mathit{Encs}$ is as in previous proofs, so suppose $\{M\}_K \in \mathit{Encs}$ and let $M = (M_1, M_2)$. Then $f(B) \cup IK'_0 \vdash f(\{M_1, M_2\}_K) = NF(f(M_1), \{f(M_2)\}_K)$. By applying Lemma 3 and the **decryption** rule, we get $f(B) \cup IK'_0 \vdash f(M_1)$ and $f(B) \cup IK'_0 \vdash f(M_2)$. Hence, using Lemma 3, $f(B) \cup IK'_0 \vdash NF(f(M_1), f(M_2)) = f(M)$.

## C A brief overview of CSP

In this section we give a brief overview of CSP. More details can be obtained from [Hoa85, Ros97].

An event represents an atomic communication; this might either be between two processes or between a process and the environment. Channels carry sets of events; for example, $\mathit{send}.A.B.\{NA, Nb\}_{Kb}$ is an event of channel *send*. $\Sigma$ represents the set of all events. The notation $\{|a, b|\}$ represents the set of all events over channels $a$ and $b$. We write $\alpha P$ for the alphabet of $P$ (i.e. the set of events $P$ can perform).

In this paper we use processes defined using the following syntax:

| | |
|---|---|
| $a \rightarrow P$ | process that can perform the event $a$, and then act like $P$. |
| $P \square Q$ | external choice; the process can act like either $P$ or $Q$; the choice is made by the environment. |
| $\square_{i \in I} P_i$ | external choice between the processes $P_i$, where $i$ ranges over $I$. |
| $f(P)$ | process that acts like $P$, except every event $a$ is renamed to $f(a)$. |
| $P \parallel Q$ | parallel composition of $P$ and $Q$, synchronizing on events from the intersection of their alphabets. |
| $\vert\vert\vert_{i \in I} P_i$ | interleaving of the processes $P_i$, where $i$ ranges over $I$; i.e. parallel composition with no synchronization. |

The traces model of CSP represents a process by the set of traces it can perform, where a trace is a finite sequence of events. We write $traces(P)$ for the set of traces of $P$. We say that process $P$ is (trace) refined by process $Q$ if the traces of $P$ are a superset of the traces of $Q$:

$$P \sqsubseteq Q \Leftrightarrow traces(P) \supseteq traces(Q).$$

We use the following laws concerning refinement in this paper:

1. $f(P \parallel Q) \sqsupseteq f(P) \parallel f(Q)$ provided $f(\alpha P \cap \alpha Q) = f(\alpha P) \cap f(\alpha Q)$.
2. $f(|||_{i \in I} P_i) \sqsupseteq |||_{i \in I} f(P_i)$.