Securing statically-verified communications protocols against timing attacks

Mikael Buchholtz [1]

Informatics and Mathematical Modelling
The Technical University of Denmark, Lyngby, Denmark

Stephen Gilmore [2]

Laboratory for Foundations of Computer Science
The University of Edinburgh, Edinburgh, Scotland

Jane Hillston [3]

Laboratory for Foundations of Computer Science
The University of Edinburgh, Edinburgh, Scotland

Flemming Nielson [4]

Informatics and Mathematical Modelling
The Technical University of Denmark, Lyngby, Denmark

Abstract

We present a federated analysis of communication protocols which considers both security properties and timing. These are not entirely independent observations of a protocol; by using timing observations of an executing protocol it is possible to calculate encryption keys which were intended to be secret or to deduce derived information about the nature of the communication even in the presence of unbreakable encryption. Our analysis is based on expressing the protocol as a high-level model and deriving from this process calculus models analysable by the Imperial PEPA Compiler and the LySatool.

[1] Email: mib@imm.dtu.dk
[2] Email: stg@inf.ed.ac.uk
[3] Email: jeh@inf.ed.ac.uk
[4] Email: nielson@imm.dtu.dk

1 Introduction

Through the use of strong encryption and well-designed communications protocols it is possible to engineer secure systems which allow a sender and a receiver to exchange confidential messages supported by authentication guarantees which confirm the identity of the principals in the communication. A very high level of confidence in the correctness of protocols such as these can be obtained by the application of static analysis methods to detect unwanted control flows. Such an analysis can ensure the absence of information flows from the sender or recipient to an attacker who is trying to eavesdrop on their conversation or to interfere through inserting bogus messages or otherwise subverting meaningful information flow between the sender and the receiver.

Supported by strong and justified belief in the encryption techniques used and the communication protocol deployed it might be tempting to conclude that no information could be leaked from the communication and gained by the attacker. However, digital communications are susceptible to a particularly subtle form of attack which is not always well defended against. Those attacks in which information is leaked through an inference obtained from timing a secure interaction are known as *timing attacks*.

The effects of a successful timing attack can be catastrophic. A successful timing attack can reveal the encryption keys which secure protocols depend upon. Messages believed to be private can now be read by an attacker. Additionally, an attacker can forge digitally signed messages and thereby entirely subvert the message authentication process.

It was previously believed to be the case that timing attacks could only be applied to hardware security tokens such as smartcards where they had been shown to be able to expose the secret keys used for RSA decryption [20]. In the context of a networked computing application it was hoped that the unpredictable delays which are inherent to networks would introduce sufficient "noise" to make timing attacks impractical in such a setting. Unfortunately this hope was shown to be misplaced when Brumley and Boneh were able to mount a remote timing attack against OpenSSL [8]. By measuring the time which a server takes to respond to decryption queries Brumley and Boneh's attack client is able to extract the private key stored on the server. Their attack applies in networked environments but also in interprocess and virtual machine environments as well.

Many cryptographic libraries completely ignore the problem of timing attacks and have no defenses implemented to prevent it, so this is not merely a problem with one particular implementation of a secure protocol but instead is a general problem which is not widely understood or one which is being methodically fought using well-founded analysis methods.

In the context where developers have access to the source code of the implementation of their encryption routine then it might be possible to add an implementation of *RSA blinding*, which draws from a random number distribution and applies encryption to this in order to add a random delay to the time taken per encryption step. Somewhat problematically, this method requires developers to have the ability and expertise to implement this operation correctly. In practice, many application programmers are unfamiliar with the details of encryption routines, and there is genuine reason for concern that they could compromise the integrity of the encryption in the process of modifying its implementation.

Timing attacks can be mounted by timing the exchange of messages during a session between two parties. The parts of the communication which do not require user interaction have sufficient repeatability that timings of these interactions can be used to derive information. Any user interaction, even a single keystroke or button click, introduces sufficient random delay to mask any information which could have been obtained from the timing information. Indeed, measuring the time between user keystrokes is used as a source of unrepeatable, weakly correlated random numbers in seeding many cryptographic routines.

Where a secure session between two parties can be monitored by an eavesdropper then information about the time of parts of the interaction can be recorded. In the case where there is a difference in the time taken by a successful interaction and one which fails then one bit of information is leaked from the communication, namely whether this session ended in success or failure. One scenario in which this single bit of information might be of some value is if an eavesdropper is timing the exchange of messages during a session between a buyer and a seller. The longer interaction could correspond to buying shares (successful completion) and the shorter one could correspond to getting a quote for a share price, with no attendant purchase (the failure outcome). If they are unable to break the encryption which is used then the eavesdropper can infer that some shares have been purchased, but not how much, or by whom.

An alternative method of combating timing attacks against secure sessions such as the one described above—and one which does not depend on a deep understanding of how to securely modify sophisticated encryption routines—is to introduce delays in the faster of the important interactions in order to mask the difference between a slower and a faster interaction.

It might seem strange to deliberately insert delays into a protocol. Surely we should be concerned with making the protocol run at top speed and therefore we should speed up the execution of whichever of the success or failure interactions takes the longer time. That might be possible to achieve if we have some additional insights into how the longer-running interaction can be made more efficient (using better algorithms for encryption or authentication, or compressing the transmitted data using lossless compression, or some other means). While that might be the most desirable option in an ideal world in practice we may not have better authentication protocols. In addition, to achieve a faster encryption rate we might be tempted to use lower grade encryption and so such an approach would even give rise to the possibility that the security of the system is compromised by trying to reduce the time taken by the slower of the two interactions.

In contrast to the above we can always make the faster route slower by introducing delays. This requires the invention of no novel algorithms with untested security and performance behaviour and no utilisation of previously-untried authentication protocols. For this reason we frame the problem in the terms of padding out a protocol with delays but our method is equally well applicable to the case where we do have the insight or skill to speed up the slower of the success and failure paths to come in line with the other.

The modification brought about by adding delays to a protocol gives rise to two interrelated questions:

(i) is the protocol with the additional delays still a secure protocol or have the delays themselves somehow compromised its security; and

(ii) do the delays which are inserted actually achieve the effect of masking the difference between a successfully completed session and one which was unsuccessful?

In order to be able to address these questions we need to have a way to efficiently re-run previously-developed security and performance analyses of a protocol.

It would be possible to take an engineering approach to attempting to determine the answer to these questions by padding an implementation of the system with delays and profiling it over multiple success and failure runs. Repeated testing in this way could give us some confidence that padding with delays had achieved the desired effect but it is difficult to believe that we could achieve the same level of confidence as could be achieved using a well-founded formal approach.

To perform formal analysis of security protocols we need a formal framework in order to describe the protocols and to conduct analysis of these descriptions. Rather than developing an entirely new framework, we make use of already-developed modelling formalisms and analysis tools. We use the process calculi LySa [10] and PEPA [18] for security and performance analysis, respectively. The software tools which we use to process our models in these calculi are the LySatool [11] and the Imperial PEPA Compiler (IPC) [6,5] with DNAmaca [19].

Structure of this paper:

We continue with an overview of the process calculi which we use in this paper, examining their similarities and differences. In Section 3 we describe our method of working with these calculi in order to achieve a federated analysis of security and performance. In Section 4 we present the case study of the paper, a secure mobile m-commerce application in which the transactions between the buyer and the seller are secured using the “Wide-Mouthed Frog” protocol. A discussion of related work follows in Section 5 and the conclusions of the paper are presented after that.

2 Overview of the calculi

The LySa and PEPA calculi are used for different analysis purposes. LySa models can be used to determine if a protocol is secure against malicious (or accidental) harm, guaranteeing trustworthy exchange of information even in the presence of the hardest attacker. PEPA models can be used to determine if a protocol with continue to perform under malicious (or accidental) overloading, guaranteeing timely exchange of information even in the presence of increasing network traffic.

Due to their different ends, LySa and PEPA offer the modeller different linguistic constructs. In this section we contrast the features of the PEPA and LySa process calculi through the use of a common example. The intention in this section is to give the reader sufficient familiarity with the notation to be comfortable with the remainder of the paper. If further details are needed they can be found in Appendix A and B.

We begin with a simple example which models a user printing documents. The user issues `lpr` print commands which send a document to the print daemon. The print daemon converts them to PostScript and spools them to a printer which prints them. In PEPA the rates at which these events happen are quantified. The producer and consumer roles in a communication are distinguished because the producer (generating the work) determines its rate and the consumer passively accepts this (without specifying the rate).

$$\begin{aligned}
\mathit{User} &\stackrel{\mathit{def}}{=} (\mathit{lpr}, r_l).\mathit{User} \\
\mathit{Daemon} &\stackrel{\mathit{def}}{=} (\mathit{lpr}, \top).(\mathit{toPS}, r_{PS}).(\mathit{spool}, r_s).\mathit{Daemon} \\
\mathit{Printer} &\stackrel{\mathit{def}}{=} (\mathit{spool}, \top).(\mathit{print}, r_p).\mathit{Printer} \\
\mathit{System} &\stackrel{\mathit{def}}{=} (\mathit{User} \underset{\{\mathit{lpr}\}}{\bowtie} \mathit{Daemon}) \underset{\{\mathit{spool}\}}{\bowtie} \mathit{Printer}
\end{aligned}$$

We present the LySa version of this example next. The LySa version looks very similar although there are a number of small differences in the presentation:

(i) LySa uses process replication (via the ! operator) instead of binding names to processes and implementing looping behaviour using recursion;

(ii) in PEPA co-operation is symmetric whereas in LySa inputs and outputs are syntactically distinguished, as in the π-calculus;

(iii) the parallel composition operator is not indexed in LySa, processes synchronise on all of their common names; and

(iv) there are no individual actions in LySa, so we take the step of introducing a process which models the PEPA environment and has the function of matching the individual activities to make an input/output pair.

To facilitate the comparison step-by-step we introduce syntactic abbreviations for process terms using $\equiv$ though technically this notation is not a part of LySa, which has nameless processes.

$$
\begin{aligned}
\mathit{User} &\equiv\ !\langle \mathit{lpr}, r_l\rangle.0\\
\mathit{Daemon} &\equiv\ !(\mathit{lpr}, r_l;\ ).\langle \mathit{toPS}, r_{PS}\rangle.\langle \mathit{spool}, r_s\rangle.0\\
\mathit{Printer} &\equiv\ !(\mathit{spool}, r_s;\ ).\langle \mathit{print}, r_p\rangle.0\\
\mathit{Environment} &\equiv\ !(\mathit{toPS}, r_{PS};\ ).0\ |!(\mathit{print}, r_p;\ ).0\\
\mathit{System} &\equiv \mathit{User} \mid \mathit{Daemon} \mid \mathit{Printer} \mid \mathit{Environment}
\end{aligned}
$$

Co-operations in PEPA have been replaced by pattern matching in the LySa model. Patterns such as those used above require inputs and outputs to be identical in order to match. More specifically, the LySa input requires everything before the semi-colon to be component-wise identical to the values in an output for the synchronisation to take place. PEPA's actions are made up of an activity type and an activity rate. These are represented by tuples of size two in the LySa model.

We could observe that individual activities should not be seen by an attacker (because they do not involve activities which cross the network). If we did not care about the timing behaviour of the protocol we could remove all of the individual activities and therefore the environment as well to have a much simpler LySa model.

$$
\begin{aligned}
\mathit{User} &\equiv\ !\langle \mathit{lpr}, r_l\rangle.0\\
\mathit{Daemon} &\equiv\ !(\mathit{lpr}, r_l;\ ).\langle \mathit{spool}, r_s\rangle.0\\
\mathit{Printer} &\equiv\ !(\mathit{spool}, r_s;\ ).0\\
\mathit{System} &\equiv \mathit{User} \mid \mathit{Daemon} \mid \mathit{Printer}
\end{aligned}
$$

However, in this paper we certainly do care about the timing behaviour of protocols and so we retain the individual activities of the PEPA model now and later.

In stochastic process algebras it is usual to abstract away the data values which would be passed from process to process. However, in an analysis of security and confidentiality it is crucial to know which messages are being passed between the principals in the interaction. For this reason we now progress to consider a notational extension of PEPA, PEPA with value passing and function symbols. In this version of our model of a user printing documents we supplement the action types with an annotation which gives the local name of the data which is being passed. Thus the user refers to his file as *myFile*, the daemon calls this the *userFile*, and the printer sees it after conversion to PostScript as the *spoolFile*.

$$\begin{aligned}
\mathit{User} &\overset{\mathit{def}}{=} (\mathit{lpr}(\mathit{myFile}), r_l).\mathit{User} \\
\mathit{Daemon} &\overset{\mathit{def}}{=} (\mathit{lpr}(\mathit{userFile}), \top). \\
&\qquad\qquad (\mathit{toPS}(\mathit{userFile}), r_{PS}).(\mathit{spool}(\mathit{userFile}), r_s).\mathit{Daemon} \\
\mathit{Printer} &\overset{\mathit{def}}{=} (\mathit{spool}(\mathit{spoolFile}), \top).(\mathit{print}(\mathit{spoolFile}), r_p).\mathit{Printer} \\
\mathit{System} &\overset{\mathit{def}}{=} (\mathit{User} \underset{\{\mathit{lpr}\}}{\bowtie} \mathit{Daemon}) \underset{\{\mathit{spool}\}}{\bowtie} \mathit{Printer}
\end{aligned}$$

In contrast to PEPA, LySa supports value passing in its syntax. Values are passed through input/output pairs where passing a value involves the exact *matching* of values to ensure that the correct inputs and outputs are being married and the *binding* of formal parameter names to actual parameter names. In the example below the match on the lpr, r_l activity gives rise to the binding $\{\, \mathit{userFile} \mapsto \mathit{myFile} \,\}$. The names used within a pattern in an input action prefix are syntactically separated from the variables used there by a semi-colon.

$$\begin{aligned}
\mathit{User} &\equiv\ !\langle \mathit{lpr}, r_l, \mathit{myFile}\rangle.0 \\
\mathit{Daemon} &\equiv\ !(\mathit{lpr}, r_l;\ \mathit{userFile}).\langle \mathit{toPS}, r_{PS}, \mathit{userFile}\rangle.\langle \mathit{spool}, r_s, \mathit{userFile}\rangle.0 \\
\mathit{Printer} &\equiv\ !(\mathit{spool}, r_s;\ \mathit{spoolFile}).\langle \mathit{print}, r_p, \mathit{spoolFile}\rangle.0 \\
\mathit{Environment} &\equiv\ !(\mathit{toPS}, r_{PS};\ \mathit{PSFile}).0 \mid !(\mathit{print}, r_p;\ \mathit{printFile}).0 \\
\mathit{System} &\equiv\ \mathit{User} \mid \mathit{Daemon} \mid \mathit{Printer} \mid \mathit{Environment}
\end{aligned}$$

This brief example has illustrated some differences between the two process calculi used here. We will discuss the encryption and decryption primitives in the LySa calculus in a later section.

3 Methodology

Distilled to their essence, process algebras record a causal ordering on the events in a discrete-event system formed as a composition of concurrently-active communicating processes. This view of the system allows the verification of properties of the model which are expressible as inter-dependencies between events. Richer analyses are facilitated by decorating the events from the model in a variety of ways. One type of decoration is an estimate of the duration of events; another is a record of the security classification of values. In their turn, these types of decoration lead us to *stochastic process algebras* and *calculi for security analysis.*

Different analyses are applicable to languages of different types. The analyses can be so different in their nature and their use that they deserve the separate labels and classifications of *static analysis* and *dynamic analysis*. We wish to apply both static analysis and dynamic analysis to a protocol and so we should ask which analysis should be done first. Static analysis is computationally cheaper than dynamic analysis so static analysis should be completed first. If the results from the static analysis process are encouraging then one would follow with dynamic analysis. To begin with dynamic analysis would be a less prudent course of action because the computationally more expensive results of the dynamic analysis of a model would have to be discarded if a subsequent static analysis found the model to be unsatisfactory.

3.1 Extracting LySa and PEPA models from UML diagrams

To drive the static and dynamic analyses of the protocol under study, we start with a root description of the protocol expressed as a UML project and from this we draw out a LySa model (which is used in the static, security analysis) and a PEPA model (which is used in the dynamic, performance analysis).

The UML project which is the root of the federated analysis of security and performance which we perform contains a number of diagrams of different types. Among them are included class diagrams, sequence diagrams, state diagrams and collaboration diagrams. In our case these contain additional annotations from which the formal objects of the LySa and PEPA models can be extracted. In the DEGAS project we have implemented software components called *extractors* which automate this process. The extraction process is discussed in prior publications on the ForLySa extractor [9] and the PEPA extractor [13] so we do not discuss it further here.

3.2 Relationship between the PEPA and LySa models

We learned in the previous section that the process calculus models which we use have a common root (they come from the same high-level model expressed as a UML project). We now discuss the relationship between these derived models in formal language terms when they are viewed as processes which generate labelled transition systems.

The LySa model represents an interaction with an unbounded number of copies of the principals at work. In contrast, the PEPA model represents an interaction with a bounded number of copies of the principals at work. The components are fixed in number and processes are not dynamically spawned in PEPA.

The reason for the difference between the two models is due to the different types of analysis which are to be performed, as follows.

- When analyzing for security, we want to be certain that we do not exclude attacks which are possible when many runs of the protocol are concurrently active. Hence, we model the protocol using process replication in LySa and the LySatool computes its best approximation to this using parameterised replication instantiated at a sufficiently large bound.

- When analyzing for performance, we can compute the measures of interest as long as we have at least one of the principals of each type in the model. Additional copies of the components would incur greater state-space generation and model solution costs without changing the measures which can be computed. Therefore, in the PEPA model, we work with the least number of principals necessary.

A consequence of this difference is that the strong relationships which might naively be expected to hold between the two models (such as bisimulation equivalence) do *not* hold. However, the weaker relationship of *trace subsumption* does hold: every observable sequence of transitions from the PEPA model is an observable trace of the LySa model (but not the other way around). Further, if additional copies of the principals are added to the PEPA model then the traces generated by the PEPA model continue to be subsumed by those of the LySa model. Thus, the observations of the LySa model represent the behaviour of the PEPA model in the limit, as more and more principals participate in the interaction described by the protocol.

4 Case study: secure m-commerce

We consider a small case study of a buyer and a seller communicating securely using the "Wide-Mouthed Frog" (WMF) protocol due to Michael Burrows and studied by Burrows, Abadi and Needham in their paper on the BAN logic for authentication analysis [12].

Before the protocol commences the buyer makes an initial contact with the seller to set up the connection. After the protocol finishes there is a final handshake which closes the connection. Both the initial contact and the final handshake can be seen by an attacker, and this is the basis of the timing attack.

The design of such a system might seem to be fundamentally flawed. Why is it not the case that all of the communication between the buyer and the seller is protected by a secure protocol? The reason for this is that the application which we consider is a secure m-commerce (mobile e-commerce) application where buyers and sellers spontaneously meet and establish a connection. In consequence, the business topology is not static and so virtual introductions are necessary. Small devices such as PDAs and handhelds are the essential execution platform for a mobile computing application but devices such as these are limited in their computing and storage capabilities and cannot support many simultaneous connections. This makes virtual goodbyes a necessary part of our application software model.

Our concern is whether or not an attacker can draw success/failure inferences from a timing analysis of an implementation of the secure m-commerce application, under the assumption that they can time a significantly large number of interactions between buyers and sellers, through playing the role of a buyer and compiling tables of timings of success and failure outcomes.

4.1 Static analysis

From the point of view of authentication analysis the salient points of the description of the WMF protocol used in a mobile e-commerce setting are that it aims to establish a secure session under a secret session key K_{AB} between a buyer A and a seller B, who are the two principals in the dialogue. A trusted server is used to establish the secure session and A has registered a master key with the server, K_{AS}. Similarly, B has registered K_{BS} with the server.

The protocol has three steps, which we describe informally first.

- Principal A sends a message to the server including the name of B and the new session key K_{AB}, encrypted under K_{AS}.
- The server decrypts this and sends the name of A and the new key K_{AB} to B, encrypted under K_{BS}.
- Principal A sends a *message* to B encrypted under K_{AB}.

In LySa this protocol can be modelled as three parallel processes representing each of the principals A, B, and S. In Figure 1, we model the source and destination addresses of each message as the first two values of the message. Notice that these addresses are sent in the clear, thus allowing an attacker to modify or fake addressing information at will.

Symmetric key encryption in LySa is written as $\{B, K_{AB}\}_{K_{AS}}$ denoting that the name of principal B and the session key is encrypted under the key which principal A shares with the server. The corresponding decryption of this message takes place at the server in the process P_S in a separate language construct. It checks whether the correct key has been used for encryption and additionally uses matching to ensure that the first value inside the encryption is B before binding the second value to the variable zk. The ν operator, familiar from the π-calculus, is used to generate new session keys and new message content. In addition to symmetric key encryption the LySatool also supports public/private key pair encryption, but that is not used in this protocol.

The LySa model of the Wide-Mouthed-Frog protocol has been annotated with information about the authentication properties that the protocol is intended to have. The static analysis conducted by the LySatool is able to tell whether these properties are satisfied when executing the protocol in parallel with a "standard" network attacker in the style of Dolev and Yao [14].

To specify these authentication properties we first introduce labels or *crypto-points*, such as $[\mathsf{at}\, a1\,]$ and $[\mathsf{at}\, s1\,]$, to denote the places where information is encrypted or decrypted. Second, at each encryption point we describe a set of destination crypto-points where this message is intended to be decrypted. In the case, such as above, where we know exactly the intended recipient of the message this set has only a single element. Third, at each decryption point we assert where we believe that this message was encrypted by giving a set of possible origin points.

The static analysis performed by the LySatool works by computing over-approximations to what a LySa process can do in *all executions* of the process executed in parallel with an arbitrary attacker. The analysis will compute information about which messages may be sent over the network and compute information about which values may become bound to the variables used in the model. The latter includes the variables which may occur within the attacker and thus the analysis will report whether a specific value will be kept out of reach of the attacker i.e. whether the value is confidential or not.

The LySa processes are additionally annotated with authentication properties specifying intended origin and destinations of messages. Violations of these properties will also be reported by the static analysis. Since the analysis computes over-approximations of the behaviour of a process it may also report too many errors. In [4] it is proven that, with respect to the formal semantics of LySa, the analysis *cannot report too few errors* and also illustrated that reporting too many errors does not pose a big problem in practice.

Technically, the static analysis is specified as a flow logic, which gives a logical description of how the behaviour of a process is represented in the analysis result. For example, the analysis result contains a relation κ which is used to keep track of the possible message tuples which may be transmitted over the network. To generate a formula that describes e.g. how an output of a LySa process affects the analysis result, we use a constraint generation function $\mathcal{G}$ of the following form.

$$\mathcal{G}(\langle a, b, c\rangle.P) = (a, b, c) \in \kappa \wedge \mathcal{G}(P)$$

This states that to have an analysis of the LySa process $\langle a, b, c\rangle.P$ it must hold that the tuple (a, b, c) is present in κ because it is possible that this tuple is sent in some run of the process. Furthermore, there must be an analysis of the process P, which will be executed after the output. A similar constraint generation takes place for the other language constructs of LySa.

The actual analysis result is computed by finding a relation κ (and other analysis components), which satisfies the generated formula. This is done by using standard tools for solving systems of logical constraints. To perform the analysis process the LySa model is encoded into a fragment of first order logic and the solution to the constraints can be found by the Succinct Solver [21]. This computation of the analysis result is guaranteed to terminate in a time bounded by a polynomial in the size of the LySa process. The full analysis of LySa may be found in [4] while a thorough introduction to flow logics may be found in [22].

The LySa model of the WMF protocol is presented in Figure 1. The LySatool confirms that the message sent from A to B in the WMF protocol is confidential as is the session key K_{AB}, since neither can come into the posses-

$$\begin{aligned}
P_A &\equiv\ !(\nu\, K_{AB})\,\langle A, S, A, \{B, K_{AB}\}_{K_{AS}}[\mathsf{at}\, a1\ \mathsf{dest}\, s1\,]\rangle.\\
&\qquad (\nu\, message)\,\langle A, B, \{message\}_{K_{AB}}[\mathsf{at}\, a2\ \mathsf{dest}\, b2\,]\rangle.0\\
P_B &\equiv\ !(S, B;\ x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{A;\ xk\}_{K_{BS}}[\mathsf{at}\, b1\ \mathsf{orig}\, s2\,]\ \mathsf{in}\\
&\qquad (A, B;\ y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{;\ ym\}_{xk}[\mathsf{at}\, b2\ \mathsf{orig}\, a2\,]\ \mathsf{in}\ 0\\
P_S &\equiv\ !(A, S, A;\ z).\mathsf{decrypt}\ z\ \mathsf{as}\ \{B;\ zk\}_{K_{AS}}[\mathsf{at}\, s1\ \mathsf{orig}\, a1\,]\ \mathsf{in}\\
&\qquad \langle S, B, \{A, zk\}_{K_{BS}}[\mathsf{at}\, s2\ \mathsf{dest}\, b1\,]\rangle.0\\
WMF_{LySa} &\equiv P_A \mid P_B \mid P_S
\end{aligned}$$

Fig. 1. LySa model of the Wide-Mouthed Frog protocol

sion of an attacker in any run of the protocol. Furthermore, the authentication properties do hold for this version of the protocol though small variants of the model are vulnerable to parallel session attacks. More information on these LySa encodings of the WMF protocol and its analysis can be found in [4].

4.2 Dynamic analysis

Figure 2 shows a PEPA model of the Wide-Mouthed Frog protocol. This records the causality information of the protocol (by requiring A to communicate with the server first, and subsequent dependencies) and quantifies the average cost in time of each of the exchanges through the use of a random variable to use as the parameter of a negative exponential distribution.

$$\begin{aligned}
P_A &\overset{def}{=} (as, r_{as}).(ab, r_{ab}).P_A\\
P_B &\overset{def}{=} (sb, \top).(ab, \top).P_B\\
P_S &\overset{def}{=} (as, \top).(sb, r_{sb}).P_S\\
WMF_{PEPA} &\overset{def}{=} P_A \underset{\{\,as, ab\,\}}{\bowtie} (P_S \underset{\{sb\}}{\bowtie} P_B)
\end{aligned}$$

Fig. 2. PEPA model of the Wide-Mouthed Frog protocol

Through the use of the IPC/DNAmaca tool chain we are able to analyse a PEPA model such as that in Figure 2 via the calculation of steady-state, transient and passage-time distributions. The DNAmaca solver provides direct and iterative solution methods, Krylov subspace techniques, and decomposition-based methods for both Markovian and semi-Markov processes. All PEPA models generate Continuous-Time Markov Chains (CTMCs) so we are only using the Markovian analysis capabilities provided by DNAmaca. The version of DNAmaca which extracts passage-times from purely Markov systems (release version HYDRA) uses uniformization to calculate both passage-time densities and cumulative distributions.

The advantages of such an exact solution method over the approximation offered by discrete-event simulation include the greater efficiency of the solution and the fact that the results do not need to be interpreted against difficult-to-compute confidence intervals which estimate a bound on the errors in simulation results. The methods implemented by the DNAmaca tool have been shown to scale to very complex models which generate state-spaces in excess of 15 million states [7].

Here we are using the IPC/DNAmaca tool chain to compute the difference between the passage-time distributions for successful and failing runs of the protocol. Our measure of a successful obfuscation of the difference between a success run and a failure run will be if the relative difference between their observed completion rates is less than a chosen small tolerance. The observed completion rates are plotted as a cumulative density function (CDF) showing how the probability of completion increases as time progresses. The CDF of the model before and after padding with delays are computed using IPC/DNAmaca.

Figure 3 shows the determination of a suitable value for the delay to be inserted into the execution of the protocol. The smooth bezier curve in the figure illustrates the protocol in a success run acting as the reference point for our experimentation. We wish to move the other plots as close to this reference curve as is required by varying the amount of delay which is inserted into the protocol.

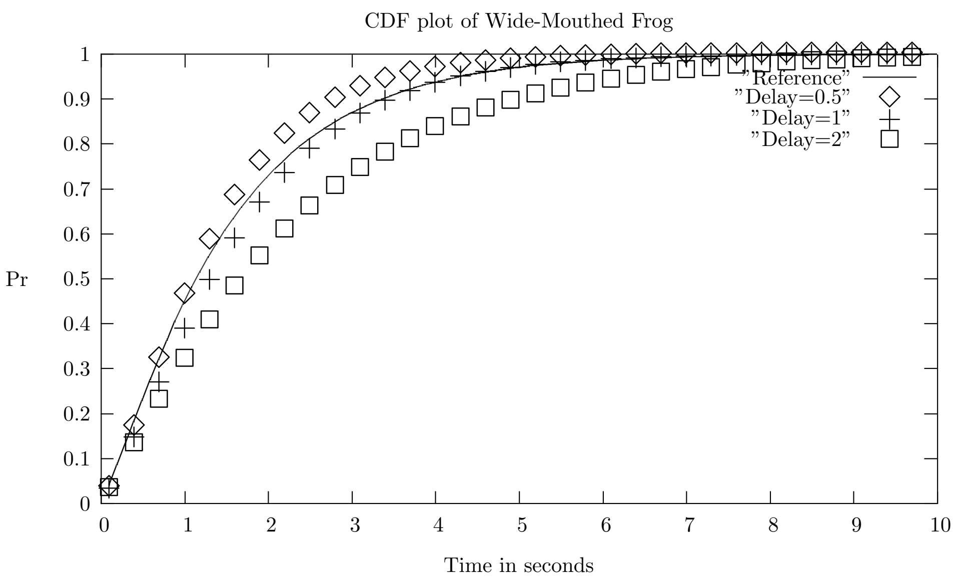

Fig. 3. CDF plot of passage time from start to completion of WMF protocol

Three estimates are taken for the value of the delay to be inserted into the failure scenario, 0.5, 1.0 and 2.0 seconds (plotted as diamond, plus and box symbols respectively). A delay of 0.5 seconds is insufficient and means that the faster path remains faster than the slower one. A delay of 2.0 seconds overcompensates and errs in the other direction, making the modification of the faster path slower than the slow one, and still perhaps detectably different. Inserting a delay of 1.0 seconds would seem to be a reasonable compensation. Further tuning might identify a better value somewhere between 0.5 and 1.0.

Using the Imperial PEPA Compiler and DNAmaca together with the method of *stochastic probes* [3] to select different start and end points for passages it is possible to automate the process of repeated passage-time analysis. The objective of repeated analyses is to rule out attacks which are based on correlating the timing of one path through the system with the timing of another, and making an indirect inference from that. Demonstrating that these passage-time distributions are acceptably close for all observable paths through the system which do not involve user interaction or another source of random stochastic delay is sufficient to show that a timing attack based on aggregating the results of timed observations is infeasible.

By selecting a value for the maximum tolerated difference between the distributions for the original and the version of the protocol which is padded with delays it is possible to increase the difficulty of a timing attack to the point where the users of the networked application which uses this protocol are satisfied that a timing attack is sufficiently difficult to mount that it would be uneconomic for an attacker to do so.

5 Related work

The problem of timing information leakage in security protocols has previously been studied by Focardi *et. al.* [16], and in the context of cryptographic protocols in [17]. In this work the authors use an real-time extension of Security Process Algebra [15]. A discrete time domain is used and processes are assumed to be synchronous. In [16] the authors develop a timed version of the property of bisimulation-based nondeducibility on compositions and show that this is useful to check for the absence of covert information flows. In [17] this work is extended to cryptographic protocol analysis.

One approach which has some similarities to ours and has similar aims of reconciling security and timing behaviour has been taken in the work of Volpano and Smith on probabilistic noninterference for concurrent languages [23]. That work uses Markov chains to study timing leakages between concurrent threads arising from the probabilistic behaviour of the scheduler.

Papers by Agat and Sands [1,2] have considered introducing dummy steps into an implementation in order to ensure that the fast path is no faster than the slow path. They construct type and transformation systems which transform a sequential imperative program P into a secure version P' which is semantically equivalent in a time-ignorant semantics and which would not allow leaks with respect to a time-observant semantics (expressed by a simulation result). Our work differs from theirs in that we consider protocols expressed at a high-level of design using annotated process algebras in contrast to the imperative language programs of Agat and Sands. Thus our work has the potential to apply to a range of satisfactory implementations of the protocol used instead of only a single instance.

6 Conclusions

In the innovative and experimental design of novel communications protocols it is necessary to consider both the security and performance of the new design, where the objective of the performance analysis may be to supplement the strong security provided by encryption and authentication.

Formal languages are helpful here because they provide an unambiguous statement of the protocol and because they are amenable to automated analyses. These analyses depend also on an unambiguous expression of the property which is being checked. In such a setting both the security and performance analyses can be re-run after a change to the protocol in order to ensure that the change has had no detrimental effect. We have applied this methodology here in the analysis of a high-level model of secure communications between a buyer and a seller.

Security and performance concerns are often at variance. It is frequently the case that application developers who are concerned with achieving peak performance view the security measures built into layers of application software as an overhead on execution time. However, in the networked world of today security is not an optional extra and so performance analysis and security analysis must work harmoniously together. In the example considered in the present paper we have shown how performance analysis can supplement a static analysis of system security, and how a control-flow analysis can add value to a stochastic model of system behaviour. Thus each analysis reinforces the usefulness of the other providing much-needed assistance in the design of reliable systems which give guarantees of security and performance properties.

Acknowledgements

The authors are supported by the Design Environments for Global ApplicationS project (DEGAS) IST-2001-32072 funded by the Future and Emerging Technologies Proactive Initiative on Global Computing. The authors thank the anonymous reviewers of the PASM workshop for their helpful comments. Monika Maidl provided detailed comments on an earlier version of this paper which helped us to improve the exposition throughout. The Imperial PEPA Compiler was developed by Jeremy Bradley and is available from http://www.doc.ic.ac.uk/ipc. The LySatool and the LySa extractor are available from

http://www.imm.dtu.dk/cs_LySa/. The PEPA extractor is a component of the PEPA Workbench, available from http://www.dcs.ed.ac.uk/pepa.

References

[1] Johan Agat. Transforming out timing leaks. In *Proceedings of the 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, 2000.

[2] Johan Agat and David Sands. On confidentiality and algorithms. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 64–77, Oakland, California, May 2001. IEEE Computer Society Press.

[3] A. Argent-Katwala, J.T. Bradley, and N.J. Dingle. Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models. In *Proceedings of the Fourth International Workshop on Software and Performance*, pages 49–58, Redwood Shores, California, USA, January 2004. ACM Press.

[4] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H.R. Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 1–15. IEEE Computer Society, 2003.

[5] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In G Kotsis, editor, *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 344–351, University of Central Florida, October 2003. IEEE Computer Society Press.

[6] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Extracting passage times from PEPA models with the HYDRA tool: A case study. In S. Jarvis, editor, *Proceedings of the Nineteenth annual UK Performance Engineering Workshop*, pages 79–90, University of Warwick, July 2003.

[7] J.T. Bradley, N.J. Dingle, P.G. Harrison, and W.J. Knottenbelt. Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models. In *Proceedings of International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 2003), Nice, France, April 26 2003*, pages 281–289, April 2003.

[8] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Conference proceedings of the 12th USENIX Security Symposium*, pages 1–14, Washington, DC, August 2003. USENIX Association. Available online at http://www.usenix.org/publications/library/proceedings/sec03/tech/brumley.html.

[9] M. Buchholtz, C. Montangero, L. Perrone, and S. Semprini. For-LySa: UML for authentication analysis. 2004. To appear.

[10] M. Buchholtz, H.R. Nielson, and F. Nielson. A calculus for control flow analysis of security protocols. *International Journal of Information Security*, 2004. Available online at http://dx.doi.org/10.1007/s10207-004-0036-x.

[11] Mikael Buchholtz. LySa — a process calculus. Web site hosted by Informatics and Mathematical Modelling at the Technical University of Denmark, April 2004. http://www.imm.dtu.dk/cs_LySa/.

[12] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *ACM Transactions on Computing Systems*, 8(1):18–36, February 1990.

[13] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. *IEE Proceedings: Computers and Digital Techniques*, 150(2):107–120, March 2003.

[14] D. Dolev and A.C. Yao. On the security of public key protocols. In *22nd Annual Symposium on Foundations of Computer Science*, pages 350–357. IEEE, 1981.

[15] R. Focardi and R. Gorrieri. Classification of security properties. Part I: Information flow. In *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*, pages 331–396. Springer-Verlag, 2001.

[16] R. Focardi, R. Gorrieri, and F. Martinelli. Real-time information flow analysis. *IEEE Journal on Selected Areas in Communications*, 21(1):20–35, January 2003.

[17] R. Gorrieri and F. Martinelli. A simple framework for real-time cryptographic protocol analysis with compositional proof rules. *Science of Computer Programming*, 50:23–49, 2004.

[18] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[19] W.J. Knottenbelt. Generalised Markovian analysis of timed transition systems. Master's thesis, University of Cape Town, 1996.

[20] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology*, pages 104–113. Springer-Verlag, 1996.

[21] F. Nielson, H. Riis Nielson, and H. Seidl. A succinct solver for ALFP. *Nordic Journal of Computing*, 9:335–372, 2002.

[22] H. Riis Nielson and F. Nielson. Flow logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, volume 2566 of *LNCS*, pages 223–244. Springer, 2002.

[23] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 34–43, June 1998.

A LySa

In this appendix we give an informal introduction to the LySa calculus. We focus on the features of the calculus which were useful in the present study. This appendix is not a definitive reference for LySa. The reader is referred to [10] for a more comprehensive introduction to the calculus and its capabilities.

Briefly, LySa is a variant of the Spi-calculus which includes pattern matching on values at input and decryption. LySa identifies the syntactic categories of *terms* and *processes*. Terms include *variables*; *names* used to represent keys, nonces, and uninterpreted data; and *encrypted values* $\{t_1, \ldots, t_k\}_{t_0}$ representing perfect symmetric key cryptography where a tuple of k terms are encrypted under a key t_0.

Process in LySa include the following syntactic forms:

Parallel: $P_1 \mid P_2$ is the parallel composition operator which evaluates its left and right operand in parallel, allowing them to match inputs and outputs.

Replication: $!P$ replicates as many copies of P as necessary and in this way provides a mechanism to generate long runs from terminating processes.

Terminated process: $\mathsf{0}$ is the terminated process, used to end a process term.

Restriction/name generation $(\nu\, n)\, P$ is the construct which introduces a new name n, to be used in the process term P. New names are needed for session keys and message content in protocol descriptions.

Output $\langle t_1, \ldots, t_k\rangle.P$ outputs the tuple $t_1, \ldots, t_k$ and continues as P.

Input $(t_1, \ldots, t_j;\, x_{j+1}, \ldots x_k).P$ describes a pattern match input. Input will synchronise with an output of length k but only if the first j values of the output are equal to the first j values of the input. In this case, each of the remaining $k - j$ values of the output will be bound component-wise to the variables placed syntactically after the semi-colon in the input. These values will be used for the variables in the continuation P.

Decryption $\mathsf{decrypt}\ t\ \mathsf{as}\ \{t_1, \ldots, t_j;\, x_{j+1}, \ldots, x_k\}_{t_0}\ \mathsf{in}\, P$ attempts to decrypt and pattern match the term t. Decryption succeeds when t is a tuple of length k encrypted under the key t_0. In that case, pattern matching takes place on the first j values and possibly results in binding of the last $k - j$ values, similarly to input.

B PEPA

This appendix provides a brief introduction to PEPA in order to make the paper self-contained. It can safely be skipped by anyone who already knows the PEPA language. For a full explanation which complements the brief description presented here the reader is referred to [18].

Prefix: The basic mechanism for describing the behaviour of a system with a PEPA model is to give a component a designated first action using the prefix combinator, denoted by a full stop. For example, $(\alpha, r).S$ carries out activity (α, r), which has action type α and an exponentially distributed duration with parameter r, and it subsequently behaves as S.

Choice: The component $P + Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

Constant: It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \stackrel{\mathit{def}}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for example, $X \stackrel{\mathit{def}}{=} (\alpha, r).X$ performs α at rate r forever.

Hiding: The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted P/L. Here, the set L identifies those activities which are to be considered internal or private to the component and which will appear as the unknown type τ.

Cooperation: We write $P \underset{L}{\bowtie} Q$ to denote cooperation between P and Q over L. The set which is used as the subscript to the cooperation symbol, the *cooperation set* L, determines those activities on which the *cooperands* are forced to synchronise. For action types not in L, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty.

However, if a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity.* The rate of the shared activity may be altered to reflect the work carried out by both components to complete the activity (for details see [18]).

In some cases, when an activity is known to be carried out in cooperation with another component, a component may be *passive* with respect to that activity. This means that the rate of the activity is left unspecified (denoted $\top$) and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.