Extending Session Types to Model Security Properties

Julie Tollund

9th December 2018

Contents

1 | Introduction

The purpose of this report, is to establish a foundation for a future thesis by the author, in regards of extending session types to model security properties with the introduction of adversaries.

In the report I will first introduce the motivation for exploring this field, as well as the current research done in the area. Secondly, I will introduce different research areas, all related to the field, and how these work together as background knowledge for the further thesis, by which I will explain further about in the final section of this report.

1.1 Motivation

With IT becoming an ever bigger part of our lives, the need for stronger and better security measurements, has grown with it.
Voting machines
Confidentiality
Ever expanding field
Tighter restrictions (session types - adversaries)
TPM (new security measurement)

1.2 Current research

The project relies heavily on the research done within the field of Security protocols and Session types.
Mark Ryan's research on the TPM (citation).
ProVerif?

1.3 Intended outcome

"The intended outcome of the thesis, is to take the idea of session type and choreography programming, and consider them in an adversarial environment. This

will be done by extending session types to model properties, and from this produce protocols that are secure by construction." (Taken from Thesis prep agreement). Compiler.

2 | Work Done

This section describes the work carried out so far. Most of it will be highlighting research done within the different fields, and showcasing examples of its use. The section ends in a summery of how the different fields come together, and how they can be used further in the coming thesis.

2.1 Security Protocols

Security protocols is an abstract or concrete protocol, that characterise the security related functions and applies cryptographic methods. It describes how the algorithm should be used to ensure the security and integrity of data transmitted. The security protocol is a protocol that runs in an untrusted environment, where it usually assumes channels are untrusted and participants are dishonest. In academic examples, they are often described with the Alice and Bob notation, which will also be used in the following examples. (The Dolev-Yao model)

2.1.1 Needham-Schroeder Protocol

The Needham-Schroeder Public Key Protocol, was first proposed by Roger Needham and Michael Schroeder in 1978 (ref?), and will be used as a running example in this and the following two sections.

The Needham-Schroeder Public Key Protocol can be illustrated by the before mentioned Alice and Bob notation in the following way, as done by Cortier and Kremer

$$
\begin{aligned}
&\mathrm{A} \xrightarrow{\{|\langle A, N_a\rangle|\}^a_{\mathsf{pk}(B)}} \qquad \xrightarrow{\{|\langle x, y\rangle|\}^a_{\mathsf{pk}(B)}} \mathrm{B} \\
&\mathrm{A} \xleftarrow{\{|\langle N_a, z\rangle|\}^a_{\mathsf{pk}(A)}} \qquad \xleftarrow{\{|\langle y, N_b\rangle|\}^a_{\mathsf{pk}(x)}} \mathrm{B} \\
&\mathrm{A} \xrightarrow{\{|z|\}^a_{\mathsf{pk}(B)}} \qquad \xrightarrow{\{|N_b|\}^a_{\mathsf{pk}(B)}} \mathrm{B}
\end{aligned}
$$

The A and B each represent Alice and Bob, while the arrows indicates the direction of the sent and received messages, by which are illustrated above each line. The notation $\{|m|\}^a_{pk(B)}$ denotes that the message m is created with an asymmetric encryption of Bob's public key, while the $\langle m_1, m_2 \rangle$ illustrates a pairing, so a concatenation of the two messages. A and B in the messages, each represent the identities of Alice and Bob, while N_a denotes the freshly generated nonce, a random number generated each session. The variables x, y, z are used for the unknown values of the message.

In the first exchange, Alice sends her identity and nonce asymmetrically encrypted with Bob's public key. Bob receives the message, with variables x, y illustrating the unknown values of the message. Bob decrypts the message, to check that it is well-formed. He then generate his own nonce, pair it with Alice's nonce, and then encrypts it with Alice's public key. Alice then decrypts Bob's message, to verify that it contains her previously sent nonce, which proves that Bob received her first message. This way of sending and receiving nonces is often called a *challenge-response* authentication [7], and is also what you see when using passwords, where the challenger asks for a password and then checks that the response is valid.

The gap left between the two participants illustrate the challenge of this protocol in an untrusted network, as an attacker may instigate a man-in-the-middle attack. This vulnerability of the protocol, was first described by Gavin Lowe in his paper published in 1995, where a fix was also purposed. Again the Alice and Bob notation is used, but now we introduce an adversary C, as illustrated here by Cortier and Kremer.

$$
\begin{array}{lcl}
\mathrm{A} & \xrightarrow{\{|\langle A, N_a \rangle|\}^a_{\mathsf{pk}(C)}} & \mathrm{C} \\
 & & \downarrow \\
 & & \mathrm{C(A)} \xrightarrow{\{|\langle A, N_a \rangle|\}^a_{\mathsf{pk}(B)}} \mathrm{B} \\
 & & \mathrm{C(A)} \xleftarrow{\{|\langle N_a, N_b \rangle|\}^a_{\mathsf{pk}(A)}} \mathrm{B} \\
\mathrm{A} & \xleftarrow{\{|\langle N_a, N_b \rangle|\}^a_{\mathsf{pk}(A)}} & \mathrm{C} \\
\mathrm{A} & \xrightarrow{\{|N_b|\}^a_{\mathsf{pk}(C)}} & \mathrm{C} \\
 & & \downarrow \\
 & & \mathrm{C(A)} \xrightarrow{\{|N_b|\}^a_{\mathsf{pk}(B)}} \mathrm{B}
\end{array}
$$

If the attacker can persuade A to initiate a session with him, he relay the messages to B, and thus convince him he is communicating with A. Lowe suggest

a simple solution to this problem, by adding the identity of the sender in the second message so that $\{|\langle N_a, N_b\rangle|\}^a_{pk(a)}$ would now look as such $\{|\langle N_a, \langle N_b, B\rangle\rangle|\}^a_{pk(a)}$

2.1.2 Message deduction

Message deduction is a formal way of figuring out wether a message can be deducted from a priori given set of messages through induction rules and derivation sequences. This

TODO: Terms

Inference rules(system)

Inference rules uses the following notation $\frac{u_l \quad ... \quad u_n}{u}$ with $u_l, ..., u_n, u$ as terms with variables. Having a set of inference rules is also called an inference system, and often contains both *composition rules* and *decomposition rules*. Below can be seen an inference system for the Dolev-Yao model ($\mathcal{I}_{DY}$) with the composition rules presented first in each line, and the rest being the decomposition rules.

$$\mathcal{I}_{DY} \ : \begin{cases} \dfrac{x, y}{\langle x, y\rangle} \qquad \dfrac{\langle x, y\rangle}{x} \qquad \dfrac{\langle x, y\rangle}{y} \\[2ex] \dfrac{x \ y}{\mathrm{senc}(x, y)} \qquad \dfrac{\mathrm{senc}(x, y) \ y}{x} \\[2ex] \dfrac{x \ y}{\mathrm{aenc}(x, y)} \qquad \dfrac{\mathrm{aenc}(x, \mathrm{pk}(y)) \ y}{x} \end{cases}$$

First line - anyone can concatenate terms and retrieve terms from a concatenation.
Second line - one can encrypt and decrypt symmetrically whenever he has the corresponding key
Third line - same as two, but with asymmetric encryptions

TODO: Derivation sequence (Deduction rules?)

Inference rules can be combined to compute or derive new messages (rewrite).

2.2 Applied π-Calculus

The applied pi-calculus (ref. Abadi and Fournet, 2001) is based upon the language pi-calculus, but offers a more convenient use for modelling security protocols to be specified, by allowing for a more wide variety of complex primitives. It is used for describing and analysing security protocols, as it provides a more intuitive process syntax for detailing the actions of the participants in a protocol [1]. This is done by

introducing a rich term algebra, for modelling the cryptographic operations used in security protocols, where function symbols represent cryptographic protocols.

Tools such as ProVerif [9], uses a syntax closely related to the applied pi-calculus, and offers a way of automated reasoning about the security properties found in cryptographic protocols.

TODO: Grammar for process as well, as it differs from the Pi-calculus

2.2.1 Grammar / Syntax

As mentioned, the applied pi-calculus uses Terms, and such differs from the Pi Calculus, where only names are used, to model messages that are exchanged during a protocol. Terms are built over a signature, names and variables[1]: (grammar - missing vertical bar)

L, M, N, T, U, V ::=	terms
a, b, c,...,k,...,m, n,..,s	names
x, y, z	variables
g(M_1,..,M_l)	function application

$P,\ Q,\ R ::=$	plain processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction
if $M = N$ then P else Q	conditional
$u(x).P$	message input
$\overline{u}\langle M\rangle.P$	message output

(Note on different notation!)

2.2.2 Re-visiting the Needham-Schroeder Protocol

Having established an understanding of how the the applied pi-calculus differentiate from the pi-calculus, we can now use it to get a better understanding of the Needham-Schroeder Protocol previously introduced.

First of, we look at the simple Handshake protocol used for setting the parameters for communication between two devices, such as the old dial-up modem or when connecting to a USB.
Handshake protocol with applied pi-calculus as illustred by Ryan and Smyth:

$$
\begin{aligned}
P &\triangleq \nu\, sk_{\mathcal{S}}.\nu\, sk_{\mathcal{C}}.\nu\, s.\\
&\quad \text{let } pk_S = \mathsf{pk}(sk_{\mathcal{S}}) \text{ in let } pk_C = \mathsf{pk}(sk_{\mathcal{C}}) \text{ in}\\
&\quad (\overline{c}\langle pk_S\rangle \mid \overline{c}\langle pk_C\rangle \mid !P_{\mathcal{S}} \mid !P_{\mathcal{C}})
\end{aligned}
$$

$$
\begin{aligned}
P_{\mathcal{S}} &\triangleq c(x\_pk).\nu\, k.\overline{c}\langle \mathsf{aenc}(x\_pk, \mathsf{sign}(sk_{\mathcal{S}}, k))\rangle.\\
&\quad c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k, z)) = tag \text{ then } Q
\end{aligned}
$$

$$
\begin{aligned}
P_{\mathcal{C}} &\triangleq c(y).\text{let } y' = \mathsf{adec}(sk_{\mathcal{C}}, y) \text{ in let } y\_k = \mathsf{getmsg}(y') \text{ in}\\
&\quad \text{if } \mathsf{checksign}(pk_S, y') = \mathsf{true} \text{ then}\\
&\quad \overline{c}\langle \mathsf{senc}(y\_k, \mathsf{pair}(tag, s))\rangle
\end{aligned}
$$

Description of Alice and Bob's processes (P_a + P_b):

$$
\begin{aligned}
P_A(sk_i, pk_r) &\hat{=} \nu n_a.\ \mathsf{out}(c, \mathsf{aenc}(\langle \mathsf{pk}(sk_i), n_a\rangle, \mathsf{pk}_r)).\\
&\quad \mathsf{in}(c, x).\\
&\quad \text{if } \mathsf{fst}(\mathsf{adec}(x, sk_i)) = n_a \text{ then}\\
&\quad \text{let } x_{nb} = \mathsf{snd}(\mathsf{adec}(x, sk_i)) \text{ in}\\
&\quad \mathsf{out}(c, \mathsf{aenc}(x_{nb}, \mathsf{pk}_r))
\end{aligned}
$$

$$
\begin{aligned}
P_B(sk_r) &\hat{=} \mathsf{in}(c, y).\\
&\quad \text{let } pk_i = \mathsf{fst}(\mathsf{adec}(y, sk_r)) \text{ in}\\
&\quad \text{let } y_{na} = \mathsf{snd}(\mathsf{adec}(y, sk_r)) \text{ in}\\
&\quad \nu n_b.\mathsf{out}(c, \mathsf{aenc}(\langle y_{na}, n_b\rangle, pk_i))\\
&\quad \mathsf{in}(c, z).\\
&\quad \text{if } \mathsf{adec}(z, sk_r)) = n_b \text{ then } Q
\end{aligned}
$$

Needham-Schroeder Public key protocol without the Lowe modification, written in applied pi-calculus by Cortier and Kremer (Describe P_A and P_B first):

$$
\begin{aligned}
P^1_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(&P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_B(sk_b) \parallel\\
&\mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))
\end{aligned}
$$

Needham-Schroeder Lowe Public key protocol written in applied pi-calculus by Cortier and Kremer (Describe P_A and P_B first):

$$
\begin{aligned}
P^5_{\mathsf{nspk}} \hat{=} !\nu sk_a, sk_b.(&!\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel !P_B(sk_a) \parallel\\
&!\mathsf{in}(c, x_{pk}).P_A(sk_b, x_{pk}) \parallel !P_B(sk_b) \parallel\\
&\mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))
\end{aligned}
$$

TODO: Do description and explanation of both

2.3 Session Types (and choreography programming)

Maybe Sections instead:
- Behavioural Types
- Session Types
- Examples of global types

TODO: Description of session types and what they are used for (behavioural types - which is also behavioural contracts)

2.3.1 Research within the field

TODO:
Binary Sessions (exactly two participants)
Global Session (multiparty interaction)

2.3.2 Global Session Types

TODO: + Examples

2.4 TPM

- Description of the TPM - what it is, and what it's used for
- Examples of TPM commands with applied pi-calculus

The Trusted Platform Module (TPM) is a specialised chip that stores RSA encryption keys specific for the host system for hardware authentication. It is used as a component on an endpoint device and is used for the Windows BitLocker. The TPM contains an RSA key pair called the Endorsement Key (EK), together with an owner-specified password. The Storage Root Key (SRK) is created when a user or administrator takes ownership of the system (rephrase), and works in a tree like structure to store TPM generated RSA keys used.

The TPM offers two authorisation sessions:

- Object Independent Authorisation Protocol (OIAP)
- Object Specific Authorisation Protocol (OSAP)

The OIAP creates a session that can manipulate any object, but will only work with certain commands. The OSAP creates a session that can only manipulate a specific object, specified at the session start.

Note: remote attestation

2.5 Evaluation

- How they all come together
- Examples of TPM commands with session types

2.5.1 TPM as Session Types

$$\begin{cases} \textit{Read}: \text{Alice} \to \text{TPM} : \text{Read}.\ \text{PCR} \to \text{TPM} : \text{vlaue}.\ \text{TPM} \to \text{PCR} : \text{value}.\ \text{TPM} \to \text{Alice} : \text{va} \\ \textit{Quote}: \text{Alice} \to \text{TPM} : \text{Quote}.\ \text{PCR} \to \text{TPM} : \text{value}.\ \text{TPM} \to \text{PCR} : \text{value TPM} \to \text{Alice} : \text{C} \\ \textit{CreateWrapKey}: \text{Alice} \to \text{TPM} : \text{CreateWrapKey}.\ \text{PCR} \to \text{TPm} : \text{value}.\ \text{TPM} \to \text{PCR} : \text{valu} \\ \textit{LoadKey2}: \text{Alice} \to \text{TPM} : \text{LoadKey2}.\ \text{PCR} \to \text{TPM} : \text{value}.\ \text{TPM} \to \text{PCR} : \text{value}.\ \text{KeyTabl} \\ \textit{CertifyKey}: \text{Alice} \to \text{TPM} : \text{data}.\ \text{KeyTable} \to \text{TPM} : \text{KeyLoaded}.\ \text{TPM} \to \text{Alice} : \text{Cert} \\ \textit{Unbind}: \text{Alice} \to \text{TPM} : \text{unbind}.\ \text{PCR} \to \text{TPM} : \text{value}.\ \text{TPM} \to \text{PCR} : \text{value}.\ \text{KeyTable} \to \text{T} \\ \textit{Seal}: \text{Alice} \to \text{TPM} : \text{data}.\ \text{PCR} \to \text{TPM} : \text{value}.\ \text{TPM} \to \text{PCR} : \text{value}.\ \text{KeyTable} \to \text{TPM} : \\ \textit{Unseal}: \text{Alice} \to \text{TPM} : \text{unseal}.\ \text{PCR} \to \text{TPM} : \text{value}.\ \text{TPM} \to \text{PCR} : \text{value}.\ \text{KeyTable} \to \text{TP} \\ \textit{Extend}: \text{Alice} \to \text{TPM} : \text{extend}.\ \text{PCR} \to \text{TPM} : \text{value}.\ \text{TPm} \to \text{PCR} : \text{hpcr} \end{cases}$$

TPM → KeyStorage (parameters)
TPM → Alice: PrivateKey. Rec a

Read: Alice → TPM : Read. PCR → TPM : value. TPM → PCR : value. TPM → Alice : value

Quote: Alice → TPM : Quote. PCR → TPM : vale. TPM → PCR : value. TPM → Alice : CertPCR

CreateWrapKey: Alice → TPM : CreateWrapKey. PCR → TPM : value. TPM → PCR : value.
KeyTable → TPM : KeyLoaded. ?(new key)?. TPM → Alice : data?

LoadKey2: Alice → TPM : LoadKey2. PCR → TPM : value. TPM → PCR : value. KeyTable → TPM : KeyLoaded.
TPM → KeyTable : LeyLoaded

CertifyKey: Alice → TPM : data. KeyTable → TPM : KeyLoaded. TPM → Alice : Cert

Unbind: Alice → TPM : unbind. PCR → TPM : value. TPM → PCR : value. KeyTable → TPM : KeyLoaded.
TPM → Alice : adec

Seal: Alice → TPM : data. PCR → TPM : value. TPm → PCR : value. KeyTable → TPM : KeyLoaded. TPM → Alice : data

Unseal: Alice → TPM : unseal. PCR → TPM : value. TPM → PCR : value. KeyTable → TPM : KeyLoaded.
TPM → Alice : KeyLoaded.

Extend: Alice → TPM : extend. PCR → TPM : value. TPM → PCR : hpcr.

TODO: Description of above (+ code in appendix? for refs.)

3 | Future Plan

Farewell

References

[1] M. D. Ryan and B. Smyth, "Applied pi calculus", 2010.

[2] A. Mukhamedov, A. D. Gordon and M. Ryan, "Towards a verified reference implementation of a trusted platform module", in *Security Protocols XVII, 17th International Workshop, Cambridge, UK, April 1-3, 2009. Revised Selected Papers*, 2009.

[3] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts and R. Plaga, "Security evaluation of scenarios based on the tcg's TPM specification", in *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, 2007.

[4] S. Delaune, S. Kremer, M. D. Ryan and G. Steel, "A formal analysis of authentication in the TPM", in *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010, Pisa, Italy, September 16-17, 2010. Revised Selected Papers*, 2010.

[5] H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira and G. Zavattaro, "Foundations of session types and behavioural contracts", *ACM Comput. Surv.*, vol. 49, no. 1, 3:1–3:36, 2016.

[6] V. T. Vasconcelos, "Fundamentals of session types", *Inf. Comput.*, vol. 217, pp. 52–70, 2012.

[7] V. Cortier and S. Kremer, "Formal models and techniques for analyzing security protocols: A tutorial", *Foundations and Trends in Programming Languages*, vol. 1, no. 3, pp. 151–267, 2014.

[8] S. Delaune, S. Kremer, M. D. Ryan and G. Steel, "Formal analysis of protocols based on TPM state registers", in *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, 2011, pp. 66–80.

Online references

[9] B. Blanchet, *Proverif*. [Online]. Available: `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/` (visited on 03/12/2018).