Verifying a Delegation Protocol for Grid Systems

Benjamin Aziz[a], Geoff Hamilton[b]

[a]*School of Computing, University of Portsmouth, Portsmouth, U.K.*
[b]*School of Computing, Dublin City University, Dublin, Ireland*

Abstract

In this paper, we design a non-uniform static analysis for formally verifying a protocol used in large-scale Grid systems for achieving delegations from users to critical system services. The analysis reveals a few shortcomings in the protocol, such as the lack of token integrity and the possibility of repudiating a delegation session. It also reveals the vulnerability of non-deterministic delegation chains that was detected as a result of adopting a more precise analysis, which allows for more participants in the protocol than did the original protocol designers envisage.

Keywords:
Delegation, Security, Protocol Verification, Static Analysis, Grid

1. Introduction

DToken is a lightweight delegation protocol [1] that has been recently proposed as one solution for the problem of delegation from users to software services and resources in large-scale Grid systems. Grid middleware systems, such as Globus[1] or GLite[2], allow users to access the computational and storage resources of the Grid. A typical model of Grids is often called Virtual Organisations, which permits users from one organisation to access and use resources belonging to another organisation under certain resource sharing policies. However, such cross-organisational provisioning of resources requires that critical issues of trust and security be managed. One issue is releted to delegations performed by the user to the gateway, and within the system on behalf of the user's original delegation.

[1]www.globus.org
[2]glite.web.cern.ch

Applying analysis techniques is one means by which critical services, such as a delegation service, can be verified and validated against vulnerabilities and incorrect usage therefore increasing the levels of confidence in the overall system functionality. In this paper, we apply formal analysis techniques that we previously developed in [2, 3, 4, 5] to verify the DToken protocol against certain core properties expected to hold in any robust delegation protocol. We show, through our analysis, that properties like basic token integrity validation, verifiable non-repudiation and deterministic delegation chains actually do not hold in the protocol. Our non-uniform analysis allows for different vulnerabilities to be discovered at different levels of abstraction. Indeed, the lowest level of abstraction (i.e. the most precise analysis) demonstrates a vulnerability, the lack of deterministic delegation chains, which was overlooked in the original design of the protocol as a result of adopting too abstract approaches.

In the rest of the paper, we give an overview of the DToken protocol in the next Section 2 and discuss three essential properties that we expect to hold of this protocol. In Section 3, we define the simple applied pi calculus language and give its operational semantics. In Section 4, we define a non-standard semantics that captures name substitutions as a result of communications, and in Section 5, we introduce a computable approximation of this non-standard semantics. In Section 6, we revisit the DToken protocol applying our static analysis to it. Finally, in Section 8, we discuss related work and in Section 9, we conclude the paper.

2. The Delegation Protocol

We give an overview here of the DToken delegation protocol as was defined in [1]. The protocol comprises secure communications between a *Delegator*, *Dor*, and a *Delegatee*, *Dee*. The following sequence of messages describes the interactions in the protocol:

$$\begin{array}{lll}
1. & \mathit{Dor} \rightarrow \mathit{Dee}: & C_{\mathit{Dor}}, C_{\mathit{Dee}}, V_{fr}, V_{to}, TS, P_{\mathit{Dor}\rightarrow \mathit{Dee}}, DS_{\mathit{Dor}\rightarrow \mathit{Dee0}}, Sig_{\mathit{Dor}\rightarrow \mathit{Dee}} \\
2. & \mathit{Dee} \rightarrow \mathit{Dor}: & C_{\mathit{Dor}}, C_{\mathit{Dee}}, V_{fr}, V_{to}, TS, P_{\mathit{Dor}\rightarrow \mathit{Dee}}, DS_{\mathit{Dor}\rightarrow \mathit{Dee}}, Sig_{\mathit{Dor}\rightarrow \mathit{Dee}}, \\
 & & Sig_{\mathit{Dee}\rightarrow \mathit{Dor}}, C_{\mathit{Dor}_{\mathrm{CAs}}}
\end{array}$$

where,
C_{Dor}: Long-term public key identity certificate of *Dor*,
C_{Dee}: Long-term public key identity certificate of *Dee*,

V_{fr}: The starting validity date of the delegation,
V_{to}: The expiry date of the delegation,
TS: A timestamp representing the time the message is generated,
$P_{Dor \rightarrow Dee}$: The delegated permissions from *Dor* to *Dee*, which include the delegation policies,
$DS_{Dor \rightarrow Dee}$: A number representing the delegation session identifier,
$DS_{Dor \rightarrow Dee0}$: Initial empty value of $DS_{Dor \rightarrow Dee}$, which for simplicity is assumed to be *Null*,
$Sig_{Dor \rightarrow Dee}$: The signature of the delegation information in the first message signed by the private key of *Dor*, K_{Dor}, where
$Sig_{Dor \rightarrow Dee} \overset{\mathtt{def}}{=} \{\!| C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee0} |\!\}_{K_{Dor}}$,
$Sig_{Dee \rightarrow Dor}$: The signature of *Dor*'s signature in the first message signed by the private key of *Dee*, K_{Dee}, where $Sig_{Dee \rightarrow Dor} \overset{\mathtt{def}}{=} \{\!| Sig_{Dor \rightarrow Dee} |\!\}_{K_{Dee}}$, and
$C_{Dor_{\mathrm{CAs}}}$: The list of subordinate CAs linking C_{Dor} to the trusted root authority.

There are a few points to note about the protocol as described in [1]. In the first message, $DS_{Dor \rightarrow Dee}$ has an empty value, which we assume to be some default value like *Null*. The choice of the delegatee's decision to assign the delegation session identifier rather than the delegator was not explained by the designers of the protocol. Timestamps in both messages are neglected in our analysis, as these are often non-reliable means of sequencing events in distributed systems due to the problem of clock synchronisation.

The second message is referred to as the DToken (Delegation Token) from *Dor* to *Dee*, written as $DT_{Dor \rightarrow Dee}$, which represents the mutual delegation agreement between *Dor* and *Dee*. In this message, *Dee* will update the value for $DS_{Dor \rightarrow Dee}$ assigning it the current delegation session identifier. Furthermore, in between the two messages, *Dee* performs some verification tests to ensure that *Dor* is authorised to delegate permissions to *Dee* and to ensure that the security information *Dor* has sent in the first message is indeed valid. For example, *Dee* will ensure that the certificates are valid and can be traced up to the root of trust and that the token has not expired.

Another main assumption in the protocol is that all the communications between *Dor* and *Dee* are carried over Secure Sockets Layer (SSL)-based channels [6]. This means that *Dor* and *Dee* are sure of each others identities and the privacy of messages is guaranteed against external intruders. However, communication security does not imply that such external intruders cannot participate in the protocol like any other agents.

The protocol is claimed to form chains of delegation. Once the last delegatee in the delegation chain decides to stop delegating, it is assumed that it will execute the delegated permissions, $P_{Dor \to Dee}$, by applying them to a *Delegation Enforcement Point* (DEP), typically a service or a resource. The DEP will perform a couple of validation steps to check the integrity of the DToken containing the permissions and other DTokens forming the full delegation chain. In [1], the authors give an example of a delegation chain in Grid systems as shown in Figure 1.

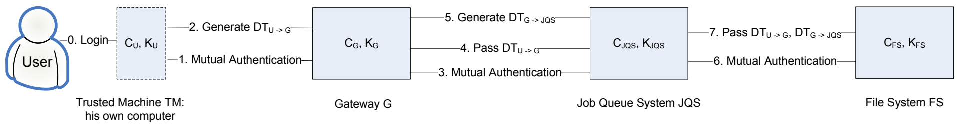

Figure 1: DToken Chained Delegation through a Gateway (cited from [1, Figure 5]).

This chain consists of the user as the delegation root, who then delegates some permissions to run a job to a gateway (a computer on which the user can login). Then the gateway delegates the job to a job queueing system, which itself is the end of the delegation chain. The job queueing system will then execute the job on a file system (the DEP). One aspect of the communication between the job queueing system and the file system is that the DTokens generated in the previous communications are passed as a set. We demonstrate later how this aspect introduces a vulnerability in the protocol.

2.1. Protocol Properties

The DToken protocol was designed to achieve lightweight delegation focusing on traceability of the participants rather than their privacy (in contrast to protocols such as [7]), therefore, it should sustain a few properties related to its purposes and functionality.

2.1.1. DToken Integrity.

This property refers to the success of a DEP in validating the integrity of a DToken. This implies that the two hash comparisons mentioned in Section IV in [1] must always succeed.

Property[DToken Integrity Validation] A DToken is said to be valid if the following equations are true:

$hash(C_U, C_G, V_{fr}, V_{to}, TS, P_{U\to G}, DS_{U\to G}) \;=\; decrypt(Sig_{U\to G}, C_U)$
$hash(Sig_{U\to G}) \;=\; decrypt(Sig_{G\to U}, C_G)$ □

The first of these compares the hash of the delegation information to the decryption of the signature of the delegator. The success of this validation implies the consent of the delegator to the delegation. The second compares the hash of the delegator's signature with the decryption of the delegatee's signature. This second validation implies the consent of the delegatee to the delegation. In general, the success of both comparisons ensures that the DToken's integrity is preserved.

2.1.2. Traceability and Accountability.

Traceability is defined in [1] as the ability of the delegatee to uniquely identify the identity of any of the previous delegators. Accountability, on the other hand, is verifiable traceability where such identity is cryptographically identifiable. Accountability is also called non-repudiation. More specifically, we define non-repudiation as the property that neither the delegator nor the delegatee can deny their acceptance of the delegation at the point of permission execution. This implies further that the delegatee must not be able to use the delegated permissions before at least having signed the DToken containing those permissions initiated by the delegator.

Formally, we define the property of verifiable non-repudiation as follows, assuming *Perms* is the set of all permission.

Property [Verifiable Non-repudiation] Given a delegator, *Dor*, and a delegatee, *Dee*, then we say that neither can deny the delegation if the following is true:
$\forall P_{Dor\to Dee} \in Perms : use(Dee, P_{Dor\to Dee}) \Rightarrow$
$(signed(Dee, P_{Dor\to Dee}) \;\wedge\; signed(Dor, P_{Dor\to Dee}))$ □

This property says that it must hold that each time a delegatee uses some permissions, then those permissions must have been signed by both the delegator and the delegatee.

2.1.3. Deterministic Delegation Chains.

The property of *deterministic delegation chains* implies the ability of determining the delegation chain ending at a DEP based on the set of DTokens received by that DEP. The property is deterministic since the delegation chain consists of a single trace of delegation events.

In order to formalise this property, we first need to establish what is meant by a delegation chain according to the DToken protocol, assuming

Agents is the set of all possible agents that can participate in the protocol.

Definition[Delegation Chains] Given a set of DTokens, $DT_{set} \in DTokens$, then we define the set of all DToken chains, DT_{chain}, that can be constructed from DT_{set} as follows:
$DT_{chain} = \{dc \mid (set(dc) = DT_{set}) \wedge (\forall a, b \in dc, \exists Dor, Dee \in Agents : (a = head(dc)) \wedge (b = tail(dc)) \wedge (b \neq [\,]) \wedge (a = DT_{Dor \to Dee}) \Rightarrow (\exists H \in Agents : b = DT_{Dee \to H}))\}$ □

According to this definition, a chain, *dc*, is essentially a *list* of DTokens whose underlying set is DT_{set} and whose adjacent elements have common adjacent participating agents. Now, we can define the property of deterministic delegation chains as follows.

Property[Deterministic Delegation Chains] Given a set of DTokens, DT_{set}, then a deterministic delegation chain implies that $|DT_{chain}| = 1$. □

If however, $|DT_{chain}| > 1$, then a DEP validating the delegation path from a specific root delegator will not be able to determine the exact chain of delegations leading to itself.

3. The Simple Applied Pi Calculus

The language that we adopt in the analysis to follow is a simple version of the applied pi calculus [8] as is shown in the syntax of Figure 2. This syntax

$L, M, N, T, U, V \in \mathcal{T}$	$::=$	Terms
	$a, b, c, n, m \in \mathcal{N}$	Names
	$x, y, z \in \mathcal{V}$	Variables
	$f(M_1, \ldots, M_n)$	Function application
$P, Q, R \in \mathcal{P}$	$::=$	Processes
	$\mathbf{0}$	Null process
	$P \mid Q$	Parallel composition
	$P + Q$	Non-deterministic Choice
	$!P$	Replication
	$(\nu n)P$	Name restriction
	$\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$	Conditional
	$\underline{M}(x).P$	Input
	$\overline{M}\langle N \rangle.P$	Output

Figure 2: The syntax of the applied pi calculus.

is described briefly as follows. Terms consist of names, which are constant values, and variables, which are names that can be instantiated to other terms. A term can also be complex as a result of the application of a function, $f(M_1, \ldots, M_n)$, to other terms, $M_1, \ldots, M_n$. We assume that functions are taken from a finite signature, Ξ, equipped with an equational theory used to infer when two terms are equal, $\Xi \vdash M = L$. For example, the symmetric-key decryption equation is written as $\Xi \vdash \texttt{decrypt}(\texttt{encrypt}(x, y), y) = x$ and digital signature verification is written as $\Xi \vdash \texttt{decrypt}(\texttt{sig}(x, K), C) = x$, where C is the public certificate corresponding the private key K and x is the hash of some message. We assume by default that $\Xi \vdash M = M$.

Using terms, processes are built according to the following syntax. A null process, $\mathbf{0}$, is an inactive process. A parallel composition of two processes, $P \mid Q$, allows P and Q to interact by interleaving, whereas the non-deterministic choice, $P + Q$, allows either P or Q to be chosen and the other is discarded. The replication of a process creates as many copies of the process as required by the context. The restriction of a name, $(\nu n)P$, creates a fresh name n bounded to the scope of P. The conditional process, *if* $M = N$ *then* P *else* Q, allows two terms to be compared under Ξ and if equal, then P is chosen, else Q. Finally, $M(x).P$ and $\overline{M}\langle N \rangle.P$ represent standard input and output actions. We shall omit trailing $\mathbf{0}$s as residues of actions and write for example, $M(x)$, instead of $M(x).\mathbf{0}$

We assume that the usual notions of free and bound names and free and bound variables (referred to as *fn*, *bn*, *fv*, *bv*, respectively) apply as usual to terms and processes. We refer to the combined set of bound names and variables for any process or a term, e, as $bnv(e) = bn(e) \cup bv(e)$, and the combined set of free names and variables of e as $fnv(e) = fn(e) \cup fv(e)$. In what follows, we only consider *standard processes*.

Property[Standard Processes] For any process, $P \in \mathcal{P}$, we say P is *standard* if and only if:

1. there are no occurrences of homonymous bound names and variables, such as for example, $a(x).P \mid b(x).Q$, or $\nu n.A \mid \nu n.B$,

2. $\forall s, s' \in \{bn(P), fn(P), bv(P), fv(P)\} : s \cap s' = \{\}$,

3. all standard processes are *closed* processes, i.e. $fv(P) = \{\}$, and finally,

4. in the following processes, $M(x).P$, $\overline{M}\langle N \rangle.P$ and *if* $M = M'$ *then* P *else* Q,

then M and M' are either names or variables that are instantiated to names.

$\square$

Renaming of bound names and variables, called α-conversion, can be used to initially achieve points [1.] and [2.]. On the other hand, [3.] allows us to avoid open systems and [4.] prevents constructing channel names or comparing terms that are complex.

The structural operational semantics of our simple applied pi calculus is defined in terms of the structural congruence ($\equiv$) relation, shown in Figure 3, and the reaction ($\xrightarrow{\tau}$) relation as shown in Figure 4. The rules of Figure 3

PAR-0	P	$\equiv$	$P \mid \mathbf{0}$
PAR-A	$P \mid (Q \mid R)$	$\equiv$	$(P \mid Q) \mid R$
PAR-C	$P \mid Q$	$\equiv$	$Q \mid P$
CHOICE-C	$P + Q$	$\equiv$	$Q + P$
REPL	$!P$	$\equiv$	$P \mid\ !P$
NEW-0	$\nu a.0$	$\equiv$	$\mathbf{0}$
NEW-C	$\nu a.\nu b.P$	$\equiv$	$\nu b.\nu a.P$
NEW-PAR	$P \mid \nu a.Q$	$\equiv$	$\nu a.(P \mid Q)$ *when* $a \notin fnv(P)$
EQN-OUT	$\overline{a}\langle M\rangle.P$	$\equiv$	$\overline{a}\langle N\rangle.P$ *such that* $\Xi \vdash M = N$

Figure 3: Rules of the structural congruence relation, $\equiv$.

COMM	$\overline{a}\langle M\rangle.P \mid a(x).Q \xrightarrow{\tau} P \mid Q[M/x]$		
THEN	*if* $M = N$ *then* P *else* $Q \xrightarrow{\tau} P$		*such that* $\Xi \vdash M = N$
ELSE	*if* $M = N$ *then* P *else* $Q \xrightarrow{\tau} Q$		*such that* $\Xi \nvdash M = N$
CHOICE	$P \xrightarrow{\tau} P'$	$\Rightarrow$	$P + Q \xrightarrow{\tau} P'$
STRUCT	$P \equiv Q \wedge P \xrightarrow{\tau} P' \wedge P' \equiv Q'$	$\Rightarrow$	$Q \xrightarrow{\tau} Q'$

Figure 4: Rules of the reaction relation, $\xrightarrow{\tau}$.

express how processes may change their shape in a commutative way. On the other hand, the rules of Figure 4 define process evolution in a non-reversible (non-commutative) manner.

4. A Name-Substitution Semantics

In this section, we define a non-standard semantics for the language of the previous section in which it is possible to express the meaning of processes in terms of name substitutions resulting from message passing only (we explicitly exclude other substitutions, such as those due to α-conversions). For example, consider the process,

$$!((\nu a)\bar{b}\langle a\rangle) \mid\ !b(x)$$

then we would like to have a meaning that captures the set of substitutions, $[a_1/x_1]$, $[a_2/x_2]$ etc., where a_i is a labelled[3] copy of the fresh name, a, and x_i is a labeled instance of the input variable x. The semantic meaning that we introduce to capture name substitutions is the environment, $\phi_{\mathcal{S}} : \mathcal{V} \to \wp(\mathcal{T})$, such that $L \in \phi_{\mathcal{S}}(x)$ implies that the term L replaces the input parameter, x, at runtime. From $\phi_{\mathcal{S}}$, a concrete semantic domain, $D_\perp : \mathcal{V} \to \wp(\mathcal{T})$, is formed with the following ordering:
$\forall \phi_{\mathcal{S}1}, \phi_{\mathcal{S}2} \in D_\perp : \ \phi_{\mathcal{S}1} \sqsubseteq_{D_\perp} \phi_{\mathcal{S}2} \quad \Leftrightarrow \quad \forall x \in \mathcal{V} : \ \phi_{\mathcal{S}1}(x) \subseteq \phi_{\mathcal{S}2}(x)$

where the bottom element, $\perp$, denotes the empty environment, $\phi_{\mathcal{S}0}$, which maps every variable in $\mathcal{V}$ to $\{\}$. The special union of $\phi_{\mathcal{S}}$ environments, $\cup_{\phi_{\mathcal{S}}}$, is defined as follows:
$\forall x \in \mathcal{V} : \ (\phi_{\mathcal{S}1} \cup_{\phi_{\mathcal{S}}} \phi_{\mathcal{S}2})(x) = \phi_{\mathcal{S}1}(x) \cup \phi_{\mathcal{S}2}(x)$

From the above definition of $D_\perp$, we can assign a meaning to process P as a function $\mathcal{S}([P])\ \rho\ \phi_{\mathcal{S}} \in D_\perp$, defined inductively over the structure of P by the rules of Figure 5, where ρ is a multiset representing processes running in parallel with P and $\phi_{\mathcal{S}}$ is the current value of $\phi_{\mathcal{S}}$ prior to the interpretation of P.

The two multiset operators, singleton $\{\!| - |\!\} : \mathcal{P} \to \wp(\mathcal{P})$ and multiset union $\uplus : \wp(\mathcal{P}) \times \wp(\mathcal{P}) \to \wp(\mathcal{P})$, are defined in the standard manner over ρ. The interpretation of the contents of ρ is given in rule $(\mathcal{R}0)$ using $\cup_{\phi_{\mathcal{S}}}$ over the meaning of every process P in ρ.

We describe next informally each of the name-substitution semantic rules, $(\mathcal{S}1)$ to $(\mathcal{S}7)$. In $(\mathcal{S}1)$, input actions are interpreted for each of the two following possibilities:

[3]Other labeling schemes can be used provided they maintain the uniqueness of bound names and variables and can cope with their infinite number.

$$
\begin{array}{lll}
(\mathcal{R}0) & \mathcal{R}\llbracket \rho \rrbracket\ \phi_{\mathcal{S}} & = \bigcup\limits_{P \in \rho}{}_{\phi_{\mathcal{S}}} \mathcal{S}\llbracket P \rrbracket\ (\rho \backslash \{\!|P|\!\})\ \phi_{\mathcal{S}} \\
(\mathcal{S}1) & \mathcal{S}\llbracket N(x).P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \\
 & \multicolumn{2}{l}{\Big(\bigcup\limits_{\overline{N'}\langle M\rangle.P' \in \rho}{}_{\phi_{\mathcal{S}}} \mathcal{R}\llbracket \rho \backslash \{\!|\overline{N'}\langle M\rangle.P'|\!\} \uplus \{\!|P|\!\} \uplus \{\!|P'|\!\} \rrbracket\ \phi_{\mathcal{S}}[y \mapsto (\phi_{\mathcal{S}}(y) \cup \{M\})] \Big) \quad \cup_{\phi_{\mathcal{S}}}\ \ \phi_{\mathcal{S}}} \\
 & \multicolumn{2}{l}{\textit{where, } N \overset{\phi_{\mathcal{S}}}{\approx} N'} \\
(\mathcal{S}2) & \mathcal{S}\llbracket \overline{N}\langle M\rangle.P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \phi_{\mathcal{S}} \\
(\mathcal{S}3) & \mathcal{S}\llbracket P \mid Q \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \mathcal{R}\llbracket \{\!|P|\!\} \uplus \{\!|Q|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}4) & \mathcal{S}\llbracket P + Q \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \mathcal{R}\llbracket \{\!|P|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}}\ \ \cup_{\phi_{\mathcal{S}}}\ \ \mathcal{R}\llbracket \{\!|Q|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}5) & \mathcal{S}\llbracket \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \begin{cases} \mathcal{R}\llbracket \{\!|P|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} & \mathit{if}\ M \overset{\phi_{\mathcal{S}}}{\approx} N \\ \mathcal{R}\llbracket \{\!|Q|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} & \mathit{otherwise} \end{cases} \\
(\mathcal{S}6) & \mathcal{S}\llbracket !P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \mathit{snd}(\mathit{fix}\ \mathcal{F}(1, \bot)) \\
 & \multicolumn{2}{l}{\text{where, } \mathcal{F} = \lambda f \lambda(j, \phi).f\ (\mathit{if} \phi = \mathcal{R}\llbracket (\biguplus\limits_{i=0}^{j} \{\!|P\sigma_i|\!\})\ \uplus\ \rho \rrbracket\ \phi_{\mathcal{S}}\ \mathit{then}\ j, \phi\ \mathit{else}} \\
 & \multicolumn{2}{l}{(j+1), (\mathcal{R}\llbracket (\biguplus\limits_{i=0}^{j} \{\!|P\sigma_i|\!\})\ \uplus\ \rho \rrbracket\ \phi_{\mathcal{S}}))} \\
 & \multicolumn{2}{l}{\text{and, } \sigma_i \overset{\mathsf{def}}{=} [\mathit{bnv}_i(P)/\mathit{bnv}(P)] \quad \text{and} \quad \mathit{bnv}_i(P) = \{x_i \mid x \in \mathit{bnv}(P)\}} \\
(\mathcal{S}7) & \mathcal{S}\llbracket (\nu n)P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \mathcal{R}\llbracket \{\!|P|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}8) & \mathcal{S}\llbracket \mathbf{0} \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \phi_{\mathcal{S}}
\end{array}
$$

Figure 5: The definition of $\mathcal{S}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{S}}$.

- The case of communications with matching output actions in ρ: In this case, the rule uses the equivalence of two terms, $\overset{\phi_{\mathcal{S}}}{\approx}$, parameterised by $\phi_{\mathcal{S}}$ to determine matching channel names. This is defined as follows: $M \overset{\phi_{\mathcal{S}}}{\approx} N \quad \Leftrightarrow \quad (\{a : a \in \mathcal{N} \wedge a \in \phi_{\mathcal{S}}(M)\} \cup \{M\}) \cap (\{b : b \in \mathcal{N} \wedge b \in \phi_{\mathcal{S}}(N)\} \cup \{N\}) \neq \emptyset$

  The definition considers the cases where M or N are variables by searching for all the names in their $\phi_{\mathcal{S}}$ values, and also for the cases where they may be names, by adding themselves to their value sets. A non-empty intersection of the two value sets then implies common channel names. For example, if $M = x \in \mathcal{V}$, $N = a \in \mathcal{N}$ and $\phi_{\mathcal{S}}(x) = \{a, b, c\}$, then we have that $(\{a, b, c\} \cup \{x\}) \cap (\{\} \cup \{a\}) = \{a\}$, which implies that $M \overset{\phi_{\mathcal{S}}}{\approx} N$. For each synchronisation between $N(x)$ and a matching $\overline{N'}\langle M\rangle$, the value of $\phi_{\mathcal{S}}$ is updated with the resulting substitution. The resulting ρ is also updated with the residues of the input and output actions.

- The case of no communications: In this case, no communication is assumed to take place and the same initial $\phi_{\mathcal{S}}$ is left unaffected.

The next rule, $(\mathcal{S}2)$, considers the case of output actions, which are interpreted as having no effect on $\phi_{\mathcal{S}}$ since communications are taken care of in rule $(\mathcal{S}1)$. In rule, $(\mathcal{S}3)$, parallel composition of two processes is interpreted straightforwardly by joining the two processes to ρ. Rule $(\mathcal{S}4)$ deals with the non-deterministic choice of two processes by interpreting both choices separately and then joining the results. Rule $(\mathcal{S}5)$ considers conditional processes, where $M \overset{\phi_{\mathcal{S}}}{\approx} N$ is used to compare M to N. Rule $(\mathcal{S}6)$ deals with replicated processes using a fixed-point calculation of the higher order functional, $\mathcal{F}$, with a fixed-point $\mathit{fix}\mathcal{F} = \mathcal{F}\mathcal{F}$. The rule allows for as many copies of P to be spawned and the number of each copy is used to subscript its bounded names and variables in order to maintain their uniqueness. This renaming is achieved through applying the σ_i substitution parameterised by the number of the process copy. Rule $(\mathcal{S}7)$ removes the new name operator on restrictions, $(\nu n)P$, thanks to the fact that the interpretation of replicated processes (in rule $(\mathcal{S}6)$) maintains the uniqueness of bound names and variables ([1.] in Property 3). Finally, rule $(\mathcal{S}8)$ interprets the null process as having no effect on $\phi_{\mathcal{S}}$.

The following soundness theorem states that any name substitutions in the structural operational semantics are captured in the non-standard semantics.

Theorem[**Soundness of the non-standard semantics**]
$\forall P, Q, M, y : \ P \xrightarrow{\tau}^{*} Q[M/y] \ \Leftrightarrow \ M \in \phi'_{\mathcal{S}}(y)$ where, $\phi'_{\mathcal{S}} = \mathcal{S}(\![P]\!)\ \rho\ \phi_{\mathcal{S}0}$

Proof sketch. The proof is by induction on the rules of the structural operational semantics in Figures 3 and 4 and the rules of the non-standard semantics in Figure 5. The most interesting case is for rule $(\mathcal{S}1)$, which we can show that substitutions captured in that rule correspond to communications occurring in rule Comm. $\square$

5. An Approximated Semantics

The computation of the name substitution semantics of the previous section may lead to infinite sizes of $D_{\perp}$ as a result of the presence of infinitely-bounded replication in processes. Hence, it may be the case that the number of instances of bound names and variables needs to be restricted to a finite number that would maintain a finite size of the semantic domain. Therefore, we introduce α_m, an approximation function, which limits the number of copies of bounded names and variables to an upper limit of $m \in \mathbb{N}$ copies. This approximation function is defined as follows.

Definition[**The α_m approximation**] Define $\alpha_m : (\mathcal{N} \cup \mathcal{V}) \to (\mathcal{N}^\sharp \cup \mathcal{V}^\sharp)$ as follows, where $\mathcal{N}^\sharp = \mathcal{N} \backslash \{a_i \mid i > m\}$ and $\mathcal{V}^\sharp = \mathcal{V} \backslash \{x_i \mid i > m\}$:
$\forall u \in (\mathcal{N} \cup \mathcal{V}) : \alpha_m(u) = \begin{cases} u_m, & \text{if } u = u_i \ \wedge \ i > m \\ u, & \text{otherwise} \end{cases}$

$\Box$

We shall write $\alpha_m(\{u, u', \ldots\})$ to mean $\{\alpha_m(u), \alpha_m(u'), \ldots\}$. The selection of m will determine how precise the analysis is. The higher the value of m, the more precise it is. In the extreme case of $m = \infty$, the semantics converges to the case of the concrete semantics of the previous section. On the contrary, selecting $m = 1$ results in a linear analysis in which all copies of bound names and variables are identified with one another. In most cases, trial and error will lead to some value probably in between the two extremes.

The α_m approximation function leads naturally to the appearance of the abstract environment, $\phi_\mathcal{A} : \mathcal{V}^\sharp \to \wp(\mathcal{N}^\sharp)$ and the abstract domain, $D^\sharp_\perp$, ordered as follows:
$\forall \phi_{\mathcal{A}1}, \phi_{\mathcal{A}2} \in D^\sharp_\perp : \ \phi_{\mathcal{A}1} \sqsubseteq_{D^\sharp_\perp} \phi_{\mathcal{A}2} \quad \Leftrightarrow \quad \forall x \in \mathcal{V}^\sharp : \ \phi_{\mathcal{A}1}(x) \subseteq \phi_{\mathcal{A}2}(x)$

with the expected definitions for $\phi_{\mathcal{A}0}$ and $\cup_{\phi_\mathcal{A}}$. Based on $D^\sharp_\perp$, we can interpret processes using the approximated semantic function, $\mathcal{A}^{\alpha_m}([\![P]\!]) \ \rho \ \phi_\mathcal{A} \in D^\sharp_\perp$, defined as:
$\mathcal{A}^{\alpha_m}([\![P]\!]) \ \rho \ \phi_\mathcal{A} \stackrel{\text{def}}{=} \quad \texttt{let } \sigma_i = \sigma_{\alpha_m} \texttt{ in let } \phi_\mathcal{S} = \phi_\mathcal{A} \texttt{ in } \mathcal{S}([\![P]\!]) \ \rho \ \phi_\mathcal{S}$

which uses the same algorithm for $\mathcal{S}([\![P]\!]) \ \rho \ \phi_\mathcal{S}$ defined in Figure 5 but replacing $\phi_\mathcal{S}$ and σ_i with their abstract counterparts. The σ_{α_m} operator is defined for all P as follows:
$\sigma_{\alpha_m} \stackrel{\text{def}}{=} \ [\alpha_m(bnv_i(P))/bnv(P)]$

Let's consider next an example of how this approximation is used.

Example Consider the replicated process,

$$(!(\nu K)a(x).\overline{b}\langle enc(x, K)\rangle) \quad | \quad \overline{a}\langle msg\rangle.!(b(y).\overline{a}\langle y\rangle)$$

Then the concrete semantics of this process according to the rules of Figure 5 would yield the following infinite environment:
$\phi_{\mathcal{S}}[x_1 \mapsto \{msg\}, x_2 \mapsto \{y_1\}, x_3 \mapsto \{y_2\}, \ldots,$
$\quad y_1 \mapsto \{enc(x_1, K_1)\}, y_2 \mapsto \{enc(x_2), K_2)\}, y_3 \mapsto \{enc(x_3, K_3)\}, \ldots]$

Now, assuming an approximation function α_1, we obtain the following linear analysis:
$\phi_{\mathcal{A}}[x_1 \mapsto \{msg, y_1\}, y_1 \mapsto \{enc(x_1, K_1)\}]$

which shows that *msg* is protected by encryption using the key K. On the other hand, selecting α_2 will result in the following non-linear analysis:
$\phi_{\mathcal{A}}[x_1 \mapsto \{msg\}, x_2 \mapsto \{y_1, y_2\}, y_1 \mapsto \{enc(x_1, K_1)\}, y_2 \mapsto \{enc(x_2), K_2)\}\}]$

This result, though, provides better information in that it demonstrates that the process generates higher-order encryptions (encryptions of encryptions), in this case more robust than single encryptions since new fresh keys are used for each encryption. □

Finally, the following termination result can be shown to hold.

Theorem[**Termination of the abstract semantics**] For any process, P, the computation of $\mathcal{A}[\![P]\!]\ \{\!|\,|\!\}\ \bot_{D^\sharp_\bot}$ terminates.

Proof sketch. The proof relies on two requirements: First, to show that $D^\sharp_\bot$ is finite. This is true from the definition of α_m. The second is to show that the abstract meaning of a process is monotonic with respect to the number of copies of a replicated process:

$$\mathcal{R}[\![(\biguplus_{i=0}^{j}\{\!|P\sigma_{\alpha_m}|\!\})\ \uplus\ \rho]\!]\ \phi_{\mathcal{A}} \quad \sqsubseteq_{D^\sharp_\bot} \quad \mathcal{R}[\![(\biguplus_{i=0}^{j+1}\{\!|P\sigma_{\alpha_m}|\!\})\ \uplus\ \rho]\!]\ \phi_{\mathcal{A}}$$

This latter requirement is proved by showing that the extra copy of P can *only* induce more communications and not less, which will be ultimately bounded by α_m. This can be shown by considering every case of P in the abstract version of the semantics of Figure 5. From this semantics, we can see that the only rule affecting the value of $\phi_{\mathcal{A}}$ will be rule ($\mathcal{S}1$), which only adds elements to $\phi_{\mathcal{A}}$ rather than removing ones. □

6. Revisiting the DToken Protocol

Naturally, the definition of any security protocol written in the informal Alice/Bob notation leaves the protocol under-specified with regards to the intermediate steps that the participants in the protocol perform. This applies to the case of the DToken protocol of Section sect:protocol and to its informal specification given by its designers in [1]. Therefore, one immediate benefit resulting from the modelling of the protocol in a formal language (such as the simple applied pi calculus) is that it leads to the removal of any under-specification or ambiguity of behaviour in the protocol.

The formal model of the DToken delegation protocol that we provide here is shown in Figure 6.

$$
\begin{array}{ll}
 & \mathit{Protocol} \overset{\mathtt{def}}{=} \\
1 & (!(\nu A)\overline{c_{\mathit{protocol}}}\langle A\rangle) \;| \\
2 & (c_{\mathit{protocol}}(z1)\ldots c_{\mathit{protocol}}(zn).\overline{c_{\mathit{Agent}}}\langle(z1,\ldots,zn)\rangle\ldots\overline{c_{\mathit{Agent}}}\langle(zn,z1\ldots,z[n-1])\rangle)\;| \\
3 & \mathit{Agent} \mid \mathit{DEP} \\
 & \\
 & \mathit{Agent} \overset{\mathtt{def}}{=} \\
4 & !(\nu c_{\mathit{fdb}})(c_{\mathit{Agent}}(\mathit{myName}, \mathit{xagent2},\ldots,\mathit{xagentn}).\overline{c_{\mathit{fdb}}}\langle\{\}\rangle \;| \\
5 & \qquad\qquad !(\nu c_{\mathit{DorS}})(\nu c_{\mathit{DeeS}})c_{\mathit{fdb}}(x_{\mathit{knowledge}}).(\\
6 & \qquad\qquad\quad (\sum_{i=1}^{n}\overline{c_{\mathit{DorS}}}\langle \mathit{xagenti}\rangle \mid \mathit{Dor}) + (\overline{\mathit{myName}}\langle c_{\mathit{DeeS}}\rangle \mid \mathit{Dee}) + \prod_{dt\in x_{\mathit{knowledge}}}\overline{c_{\mathit{DEP}}}\langle dt\rangle)) \\
 & \\
 & \mathit{Dor} \overset{\mathtt{def}}{=} \\
7 & c_{\mathit{DorS}}(\mathit{deeName}).\mathit{deeName}(\mathit{deeSC}).\overline{\mathit{deeSC}}\langle(\mathit{myName}, c_{\mathit{DorS}}, x_{\mathit{knowledge}})\rangle. \\
8 & \overline{\mathit{deeSC}}\langle(\mathit{Crt}(\mathit{myName}), \mathit{Crt}(\mathit{deeName}), V_{\mathit{fr}}, V_{\mathit{to}}, \mathit{TS}, \mathit{Prm}(\mathit{myName}, \mathit{deeName}), \mathit{Null}, \\
9 & \qquad\qquad \mathit{sig}(\mathit{Crt}(\mathit{myName}), \mathit{Crt}(\mathit{deeName}), V_{\mathit{fr}}, V_{\mathit{to}}, \mathit{TS}, \mathit{Prm}(\mathit{myName}, \mathit{deeName}), \mathit{Null}, \\
10 & \qquad\qquad\quad K(\mathit{myName})))\rangle.c_{\mathit{DorS}}(x_{\mathit{dtoken}}).\overline{c_{\mathit{fdb}}}\langle\cup(x_{\mathit{dtoken}}, x_{\mathit{knowledge}})\rangle \\
 & \\
 & \mathit{Dee} \overset{\mathtt{def}}{=} \\
11 & (\nu DS_{\mathit{DorDee}})(\nu c_{\mathit{intern}})c_{\mathit{DeeS}}(\mathit{dorName}, \mathit{dorSC}, y_{\mathit{knowledge}}). \\
12 & (c_{\mathit{DeeS}}(y1,y2,y3,y4,y5,y6,y7,y_{\mathit{dorSig}}). \\
13 & (\overline{c_{\mathit{intern}}}\langle \mathit{decrypt}(y_{\mathit{dorSig}}, \mathit{Crt}(\mathit{dorName}))\rangle \mid c_{\mathit{intern}}(u1,u2,u3,u4,u5,u6,u7). \\
14 & (\overline{c_{\mathit{intern}}}\langle(y1,y2,y3,y4,y5,y6, DS_{\mathit{DorDee}}, y_{\mathit{dorSig}}, \mathit{sig}(y_{\mathit{dorSig}}, K(\mathit{myName})), \mathit{CrtChain}(\mathit{dorName}))\rangle \;| \\
15 & \; c_{\mathit{intern}}(y_{\mathit{dtoken}}).(\overline{\mathit{dorSC}}\langle y_{\mathit{dtoken}}\rangle \mid \overline{c_{\mathit{fdb}}}\langle\cup(y_{\mathit{dtoken}}, y_{\mathit{knowledge}})\rangle))))) \\
 & \\
 & \mathit{DEP} \overset{\mathtt{def}}{=} \\
16 & !(\nu c_{\mathit{DEPint}})(c_{\mathit{DEP}}(w).(\overline{c_{\mathit{DEPint}}}\langle w\rangle \mid c_{\mathit{DEPint}}(w1,w2,w3,w4,w5,w6,w7,w8,w9,w10). \\
17 & (\overline{c_{\mathit{DEPint}}}\langle \mathit{hash}(w1,w2,w3,w4,w5,w6,w7)\rangle.\overline{c_{\mathit{DEPint}}}\langle \mathit{hash}(w8)\rangle. \\
18 & \;\overline{c_{\mathit{DEPint}}}\langle \mathit{decrypt}(w9,w2)\rangle.\overline{c_{\mathit{DEPint}}}\langle \mathit{decrypt}(w8,w1)\rangle \;| \\
19 & \; c_{\mathit{DEPint}}(w11).c_{\mathit{DEPint}}(w12).c_{\mathit{DEPint}}(w13).c_{\mathit{DEPint}}(w14). \\
20 & \;\; (\mathit{if}\ w12 = w13\ \mathit{then}\ (\mathit{if}\ w11 = w14\ \mathit{then}\ \overline{c_{\mathit{DEPint}}}\langle \mathit{OK}\rangle\ \mathit{else}\ \overline{c_{\mathit{DEPint}}}\langle \mathit{ER2}\rangle)\ \mathit{else}\ \overline{c_{\mathit{DEPint}}}\langle \mathit{ER1}\rangle \;| \\
21 & \;\;\; c_{\mathit{DEPint}}(w15)))))
\end{array}
$$

Figure 6: The definition of the DToken protocol in the simple applied pi calculus.

In this model, lines 1-3 describe the main protocol where $n \in \mathbb{N}$ number of agents is generated. These are then transmitted to instances of the *Agent* process, where each instance gets the names of all the agents. The ordering of the names in the output message is such that the first name represents the recipient's own name, whereas the others represents names of other agents in the system. This is altered for each instance such that each *Agent* instance gets a different own name from the others. Although we left n as a general number, we emphasize here that for delegation to be interesting (more than self-delegation), at least two different agents must be present and hence, $n \geq 2$.

Lines 4-6 describe an agent process. This process receives its name and the name of the other agents in the system. It then spawns an agent in line 4 by outputting a signal with empty knowledge on c_{fdb}. In line 5, the new agent is created and in line 6, it chooses one of three possible behaviours: to run as a delegator process (Dor), a delegatee process (Dee) or to apply the permissions it has in its knowledge to the DEP process (DEP). We have used the special operators, Σ and Π, as concise forms of the non-deterministic choice and the parallel composition of several processes, respectively. Additionally, we used in lines 10 and 15 the $\cup(e, s)$ operator to represent the union of element e to set s.

Lines 7-10 describe the delegator process and lines 11-15 describe the delegatee process. At the end of each of these process, the accumulated knowledge of the current set of DTokens, represented by the $x_{knowledge}$ and $y_{knowledge}$ variables, along with the newly generated DToken in variables x_{dtoken} and y_{dtoken} is returned back to a new instance of the *Agent* process. Finally, the *DEP* process in lines 16-21 receives any number of DTokens and attempts to validate their integrity by comparing hashes of the delegated and signed information. These validation steps were defined in [1] as the main verification of DTokens. If the two tests are successful, the process sends an *OK* signal. Otherwise, it will send either *ER1* or *ER2* depending on where the validation failure occurred.

6.1. The Abstract Interpretation of the Protocol

We carried out an abstract interpretation of the DToken protocol by applying the abstract semantic function, $\mathcal{A}^{\alpha_m}(\llbracket Protocol \rrbracket)\ \{\!|\!|\!\}\ \perp$, on the model of the protocol as defined in the previous section and for the approximation cases of α_1, α_2, α_3 and α_4.

6.1.1. The Case of α_1.

This case does not make much sense for a delegation protocol since it identifies all agents in the model. This has the effect of having multiple instances of the same agent, who will be able to only delegate permissions to itself (i.e. it will assume the roles of *Dor* and *Dee* at the same time). Therefore, we ignore this case as it does not give us much insight into the protocol.

6.1.2. The Case of α_2.

This case differentiates two agents, and we consider the interesting case where one of them assumes the role of the delegator and the other the role of the delegatee (i.e. one chooses to run *Dor* and the other *Dee*). This case corresponds to single-level delegation between two agents, where some of the results of the abstract interpretation using $\mathcal{A}^{\alpha_2}(\llbracket Protocol \rrbracket)\ \{\!|\!|\!\}\ \perp$ are shown in Figure 7.

$$
\begin{array}{ll}
\phi_{\mathcal{A}}[& w1_1 \mapsto \{y1_2\}, w2_1 \mapsto \{y2_2\}, w3_1 \mapsto \{y3_2\}, w4_1 \mapsto \{y4_2\}, w5_1 \mapsto \{y5_2\}, w6_1 \mapsto \{y6_2\}, \\
 & w7_1 \mapsto \{DS_{DorDee2}\}, w8_1 \mapsto \{y_{dorSig2}\}, w11_1 \mapsto \{hash(w1_1, w2_1, w3_1, w4_1, w5_1, w6_1, w7_1)\}, \\
 & w14_1 \mapsto \{decrypt(w8_1, w1_1)\}, w15_1 \mapsto \{ER2\} \\
 & y1_2 \mapsto \{Crt(myName_1)\}, y2_2 \mapsto \{Crt(deeName_1)\}, y3_2 \mapsto \{V_{fr}\}, y4_2 \mapsto \{V_{to}\} \\
 & y5_2 \mapsto \{TS\}, y6_2 \mapsto \{Prm(myName_1, deeName_1)\}, y7_2 \mapsto \{Null\} \\
 & y_{dorSig2} \mapsto \{sig(Crt(myName_1), Crt(deeName_1), V_{fr}, V_{to}, TS, Prm(myName_1, deeName_1), \\
 & \qquad Null, K(myName_1))\} \\
 & deeName_1 \mapsto \{xagent2_1\}, xagent2_1 \mapsto \{z2\}, z2 \mapsto \{A_2\}, myName_1 \mapsto \{z1\}, z1 \mapsto \{A_1\} \quad \ldots]
\end{array}
$$

Figure 7: Some of the results for the case of α_2.

Vulnerability 1 (Non-matching Hash Validation.). *The main result one notices is that $w15_1$ catches an error message ER2. If we examine the specification of Figure 6, we find that this error message is due to the fact that $w11_1 \neq w14_1$. Applying the results of $\phi_{\mathcal{A}}$ to this equation to resolve its variables, we arrive at the following implications:*

$w11 \neq w14$

$\Rightarrow hash(w1_1, w2_1, w3_1, w4_1, w5_1, w6_1, w7_1) \neq decrypt(w8_1, w1_1)$

$\Rightarrow hash(y1_2, y2_2, y3_2, y4_2, y5_2, y6_2, DS_{DorDee2}) \neq decrypt(y_{dorSig2}, y1_2)$

$\Rightarrow hash(Crt(myName_1), Crt(deeName_1), V_{fr}, V_{to}, TS, Prm(myName_1, deeName_1),$
$\quad DS_{DorDee2}) \neq$
$\quad decrypt(sig(Crt(myName_1), Crt(deeName_1), V_{fr}, V_{to}, TS, Prm(myName_1, deeName_1),$
$\quad Null, K(myName_1)), Crt(myName_1))$

$\Rightarrow hash(Crt(myName_1), Crt(deeName_1), V_{fr}, V_{to}, TS, Prm(myName_1, deeName_1),$

$DS_{DorDee2}) \neq$
$hash(Crt(myName_1), Crt(deeName_1), V_{fr}, V_{to}, TS, Prm(myName_1, deeName_1), Null)$
$\Rightarrow DS_{DorDee2} \neq Null$ $\square$

This result implies an incorrect validation of the DToken, which is caused by the fact that the delegator (Dor) agent always signs a $Null$ value for the delegation session identifier. This is later assigned by the delegatee (Dee) the value of DS_{DorDee}. In the protocol of Section 2, this is due to the fact that the value of $DS_{U \rightarrow G}$ is different in the two messages, or in other words, there is a lack of agreement between the delegator and the delegatee on the delegation session identifier as they both sign different values.

The significance of this result is that it further undermines the claims in [1] of the ability of the DEP to validate the integrity property of any DTokens it receives and further brings in to question some of the evaluation results presented for the case of chained delegations, since any such chains could not possibly have been successfully validated since the first DToken validation will always fail.

Fixing the Vulnerability.. One suggestion to fix this vulnerability is to simply allow the delegator to choose the session identifier instead of the delegatee. In this way, the delegator will agree on the same value with the delegatee.

6.1.3. The Case of α_3.

This case corresponds to a two-level delegation with three agents. The most interesting case here is where two agents play the normal roles of the delegator and the delegatee, and the third agent plays a man-in-the-middle role, which means that the agent in addition to being able to run the normal protocol behaviour, can also run any extra behaviour that may subvert the protocol. Therefore, any "robust" protocol is expected to be able to withstand such subversive behaviour.

Since we start from the assumption that one agent is the man-in-the-middle intruder, we replace line 15 by the following new line of code for the delegatee process and assume that the malicious agent (only) will run this new version:
15. $c_{intern}(y_{dtoken}).(\overline{dorSC}\langle y_{dtoken}\rangle \mid \overline{c_{fdb}}\langle \cup(y_{dtoken}, y_{knowledge})\rangle +$ $DeeMiM))))$

where $DeeMiM$ is the extra man-in-the-middle code defined as follows:

$$\begin{aligned}&\mathit{DeeMiM} \overset{\mathsf{def}}{=} \overline{c_{\mathit{DEP}}}\langle(\mathit{Crt}(\mathit{myName}), y2, y3, y4, y5, y6, \mathit{DS}_{\mathit{DorDee}},\\ &\mathit{sig}(\mathit{Crt}(\mathit{myName}), y2, y3, y4, y5, y6, \mathit{DS}_{\mathit{DorDee}}, K(\mathit{myName})),\\ &\mathit{sig}(\mathit{sig}(\mathit{Crt}(\mathit{myName}), y2, y3, y4, y5, y6, \mathit{DS}_{\mathit{DorDee}}, K(\mathit{myName})), K(\mathit{myName})),\\ &\mathit{CrtChain}(\mathit{myName}))\rangle\end{aligned}$$

This extra behaviour simply takes the permissions $y6$ received from the delegator as well as the other delegation information such as the time validity etc., and builds a new, but *dummy*, DToken. This dummy DToken is perfectly syntactically valid and it expresses self-delegation from the delegatee to itself. The delegatee then sends this DToken to the DEP in order to use the permissions that it received in reality from the delegator. The DEP will successfully validate the token as it is syntactically correct and will assume that the delegatee has delegated the permissions to itself.

Applying our abstract interpretation using $\mathcal{A}^{\alpha_2}([\![\mathit{Protocol}]\!])\ \{\![\,]\!\}\ \bot$ to the protocol enhanced with the above new man-in-the-middle definition of *Dee*, we arrive at the results of Figure 8.

$$\boxed{\begin{array}{l}\phi_{\mathcal{A}}[\\ w1_1 \mapsto \{y1_2, \mathit{Crt}(\mathit{myName}_2)\}, w2_1 \mapsto \{y2_2\}, w3_1 \mapsto \{y3_2\}, w4_1 \mapsto \{y4_2\}, w5_1 \mapsto \{y5_2\},\\ w6_1 \mapsto \{y6_2\},\\ w7_1 \mapsto \{\mathit{DS}_{\mathit{DorDee2}}\}, w8_1 \mapsto \{y_{\mathit{dorSig2}}, \mathit{sig}(\mathit{Crt}(\mathit{myName}_2), y2, y3, y4, y5, y6, \mathit{DS}_{\mathit{DorDee}_2}, K(\mathit{myName}_2))\},\\ w9_1 \mapsto \{\mathit{sig}(y_{\mathit{dorSig2}}, K(\mathit{myName}_2)),\\ \qquad\qquad \mathit{sig}(\mathit{sig}(\mathit{Crt}(\mathit{myName}_2), y2, y3, y4, y5, y6, \mathit{DS}_{\mathit{DorDee}_2}, K(\mathit{myName}_2)), K(\mathit{myName}_2))\},\\ w10_1 \mapsto \{\mathit{CrtChain}(\mathit{dorName}_2), \mathit{CrtChain}(\mathit{myName}_2)\},\\ w11_1 \mapsto \{\mathit{hash}(w1_1, w2_1, w3_1, w4_1, w5_1, w6_1, w7_1)\},\\ w14_1 \mapsto \{\mathit{decrypt}(w8_1, w1_1)\}, w15_1 \mapsto \{\mathit{ER2}\}\\ y1_2 \mapsto \{\mathit{Crt}(\mathit{myName}_1)\}, y2_2 \mapsto \{\mathit{Crt}(\mathit{deeName}_1)\}, y3_2 \mapsto \{V_{\mathit{fr}}\}, y4_2 \mapsto \{V_{\mathit{to}}\}\\ y5_2 \mapsto \{\mathit{TS}\}, y6_2 \mapsto \{\mathit{Prm}(\mathit{myName}_1, \mathit{deeName}_1)\}, y7_2 \mapsto \{\mathit{Null}\}\\ y_{\mathit{dorSig2}} \mapsto \{\mathit{sig}(\mathit{Crt}(\mathit{myName}_1), \mathit{Crt}(\mathit{deeName}_1), V_{\mathit{fr}}, V_{\mathit{to}}, \mathit{TS}, \mathit{Prm}(\mathit{myName}_1, \mathit{deeName}_1),\\ \qquad\qquad \mathit{Null}, K(\mathit{myName}_1))\}\\ \mathit{deeName}_1 \mapsto \{\mathit{xagent2}_1\}, \mathit{xagent2}_1 \mapsto \{z2\}, z2 \mapsto \{A_2\}, \mathit{myName}_1 \mapsto \{z1\}, z1 \mapsto \{A_1\},\\ \mathit{myName}_2 \mapsto \{z2\} \hfill \ldots]\end{array}}$$

Figure 8: Some of the results for the case of α_3.

Vulnerability 2 (Delegation Repudiation.). *Going back to the property of verifiable non-repudiation defined in Section 2.1 and assuming that $\phi_{\mathcal{A}}$ is as defined in Figure 8, then we provide the following definitions of the* use *and* signed *predicates for any $i, j, k \in \mathbb{N}$ numbers, where $k < j$, and any two agents A_1 and A_2:*
$U \overset{def}{=} A_1$, $G \overset{def}{=} A_2$, $P_{U\to G} \overset{def}{=} \mathit{Prm}(A_1, A_2)$,
use(G,$P_{U\to G}$)$\overset{def}{=}$ $P_{U\to G} \in \phi_{\mathcal{A}}(w6_i)$,

signed(G,$P_{U\rightarrow G}$*)*$\overset{def}{=}\ \exists dt \in w_i :$
$dt = (\mathit{Crt}(U), \mathit{Crt}(G), V_{\mathit{fr}}, V_{\mathit{to}}, \mathit{TS}, \mathit{Prm}(U,G), \mathit{DS}_{U\rightarrow G}, \mathit{Sig}_{U\rightarrow G}, \mathit{sig}(\mathit{Sig}_{U\rightarrow G}, K(G)),$
$\qquad \mathit{CrtChain}(U)) \ \wedge \ y6_j = \mathit{Prm}(U,G),$
signed(U,$P_{U\rightarrow G}$*)*$\overset{def}{=}\ \exists dt \in w_i :$
$dt = (\mathit{Crt}(U), \mathit{Crt}(G), V_{\mathit{fr}}, V_{\mathit{to}}, \mathit{TS}, \mathit{Prm}(U,G), \mathit{DS}_{U\rightarrow G}, \mathit{Sig}_{U\rightarrow G}, \mathit{sig}(\mathit{Sig}_{U\rightarrow G}, K(G)),$
$\qquad \mathit{CrtChain}(U)) \ \wedge \ y6_j = \mathit{Prm}(U,G) \ \wedge$
$\mathit{Sig}_{U\rightarrow G} = \mathit{sig}(\mathit{Crt}(U), \mathit{Crt}(G), V_{\mathit{fr}}, V_{\mathit{to}}, \mathit{TS}, \mathit{Prm}(U,G), \mathit{Null}, K(U))$

then we find that the property does not always hold, since if we choose $w9_1$ *to be:*
$\mathit{sig}(\mathit{sig}(\mathit{Crt}(\mathit{myName}_2), y2, y3, y4, y5, y6, \mathit{DS}_{\mathit{DorDee}_2}, K(\mathit{myName}_2)), K(\mathit{myName}_2))$
then we can show, after applying the full substitutions of variables in $\phi_{\mathcal{A}}$*, that* $\mathit{use}(A_2, \mathit{Prm}(A_1, A_2)) = \boldsymbol{T}$ *and* $\mathit{signed}(A_2, \mathit{Prm}(A_1, A_2)) = \boldsymbol{F}$. □

This implies that the delegatee is able to use the permissions without having to sign a DToken proving that it got those permissions from the delegator. The main reason behind this vulnerability is that the delegatee receives these permissions prematurely from the delegator, therefore, it is able to subvert its part on signing the DToken. These results are equivalent to replacing Message 2 in the original protocol by the message:
$2'.G \rightarrow \mathit{DEP} : C_G, C_G, V_{fr}, V_{to}, TS, P_{U\rightarrow G}, DS_{U\rightarrow G}, Sig_{G\rightarrow G}, \{\!|Sig_{G\rightarrow G}|\!\}_{K_G}, C_{G_{\mathrm{CAs}}}$

This message allows G to create a dummy DToken in order to use the permissions it received from the user U. One argument against the validity of such a vulnerability is that the local policies at the DEP should be able to prevent G from using $P_{U\rightarrow G}$. However, we consider this argument to be weak as it associates the robustness of the protocol with the expressivity of the DEP policies. There are simply no guarantees that the DEP will enforce such policies, specifically in scenarios where the anonymity of the agents is required or where the DEP is a stateless Web service.

Fixing the Vulnerability.. The fix to this vulnerability requires the modification of the protocol and the introduction of extra steps. The delegator should only send the permissions to the delegatee after the latter has agreed (by signing the delegation information) to participate in the delegation session. In this way, the delegatee has no means of denying its participation in the protocol. Also, a monitoring service should be introduced to the architecture to record the signatures and provide evidence whenever required.

6.1.4. The Case of α_4.

This case is an interesting one since it did not feature in the design of the DToken protocol presented in [1] and it assumes the presence of four agents, two of which are playing men-in-the-middle roles. The original protocol of [1] assumes in several occasions[4] that the protocol is indeed able to form deterministic delegation chains. This is true in the specific case where the number of participating agents is less than or equal three. However, in the case of four agents, the possibility of internal circular delegations arises. In order to explain this, consider the following example.

Example Assume agents A, B, C and D with the following scenario of delegation: *A delegates to B, B delegates to C, C delegates to A, A delegates to C and C delegates to D.* This scenario results in D receiving the following set of DTokens: $\{DT_{A\rightarrow B}, DT_{B\rightarrow C}, DT_{C\rightarrow A}, DT_{A\rightarrow C}, DT_{C\rightarrow D}\}$

However, due to the presence of the delegation cycle - C delegates to A and A delegates to C-, D is able to form the following chain using the same set of tokens: *A delegates to C, C delegates to A, A delegates to B, B delegates to C and C delegates to D.* This is clearly different chain from the actual one above. The implication of this is that D will not be able to *determine* the exact set of delegations leading to itself. Since delegation is a form of trust, this problem breaks the trust chain and does not preserve the deterministic delegation chain property of Section 2. $\square$

Applying our analysis for α_4 along the lines of this example, where there are four agents A_1, A_2, A_3 and A_4, we were able to show that $|DT_{chain}| > 1$ assuming that the underlying set is $x_{knowledge}$. Our analysis does not capture temporal ordering of messages based on DToken timestamps (TS) since we consider that timestamps in a large-scale distributed system with several possibly non-synchronised clocks are a bad mechanism for imposing temporal ordering on events.

Fixing the Vulnerability.. This vulnerability arises from the fact that Dtokens are passed as a set. A set has no notion of ordering or indeed multiplicity. A richer structure, like lists, is needed when grouping and passing DTokens, such that some reasoning on their temporal ordering can be achieved.

[4]See, for example, Verification 3 of page 7 and Section V of page 8.

6.2. A Note on Modelling the Intruder

Due to the assumptions on the nature of the intruder in [1], the intruder in this paper is assumed to be another participant in the protocol with a well-known identity (through the use of certificates) and who is only able to divert the protocol via messages that make sense to other participants. The use of secure communications in the protocol prevents external intruders, who are not participating actively in the protocol, from interfering with the protocol messages. This assumption yields the intruder less powerful than Dolev-Yao's *most powerful attacker* [9, 10], since for example, we do not assume that the intruder is capable of listening passively to communications among other participants or injecting data into the exchanged messages without participating in a protocol session.

Therefore, the only case where we model an intruder is that of α_3, in which the delegatee is assumed to run some extra "malicious" code in order to exploit the repudiation vulnerability in the protocol. Apart from that, the delegatee is not assumed to be running any other malicious behaviour. For both of the other cases of α_2 and α_4, the vulnerabilities are related to the correct functionality of the protocol as claimed in [1], which are independent of the model of the attacker adopted.

7. DToken II: The Corrected Version

We now propose a new version of the DToken protocol, which does not suffer from any of the above vulnerabilities in the original protocol. The protocol consists of the following steps:

1. $Dor \rightarrow Dee: \quad RfD_{Dor}$
2. $Dee \rightarrow Dor: \quad \{\!|DS_{Dor\rightarrow Dee}, RfD_{Dor}|\!\}_{K_{Dee}}$
3. $Dor \rightarrow Dee: \quad C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor\rightarrow Dee}, DS_{Dor\rightarrow Dee}, Sig_{Dor\rightarrow Dee}$
4. $Dee \rightarrow Dor: \quad C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor\rightarrow Dee}, DS_{Dor\rightarrow Dee}, Sig_{Dor\rightarrow Dee}, Sig_{Dee\rightarrow Dor}, C_{Dor_{\mathrm{CAs}}}$

In this corrected version, the delegator commences the protocol by sending a *request for delegation* message RfD_{Dor}. This messages can be considered as a negotiation message, which may include description of the delegated permissions or any other delegation information. If the delegatee accepts the request, it will reply by proposing the delegation session identifier signed with its signature along with the original request from the delegator. In this manner, both parties will agree on the session identifier hence eliminating the validation bug in the original protocol. Similarly, since the delegatee has signed the request and the session identifier, the delegator can now prove the association between the delegated permissions and the delegation session. Hence, the delegatee cannot repudiate its acceptance of the session, even though this is proven outside the structure of the DToken itself.

In order to achieve deterministic delegation chains, we propose that DTokens be passed in a list structure as opposed to the set structure as done in the original protocol, when performing second and further level delegations. Hence, for example in Figure 1, we propose that the job queueing system passes the list [DTU $\rightarrow$ G; DTG $\rightarrow$ JQS] instead of passing $\{$DTU $\rightarrow$ G, DTG $\rightarrow$ JQS$\}$. This will ensure that DTokens are ordered in their temporal sequencing and so no non-determinism arises when building chains.

8. Related Work

The use of tokens for achieving delegation in distributed systems is a common technique that has been used in many popular systems throughout the years, such as for example, Kerberos [11]. A recent taxonomy of delegation methods has recently been published in [12], where various types of delegation tokens and credentials are discussed. Further uses of delegation tokens in collaborative applications include healthcare [13], identity management in service-oriented architectures [14] and in the context of Web-based social networking [15].

The analysis carried out in this paper is based on a well-established methodology for formally verifying cryptographic protocols presented in previous works [2, 3, 4, 5]. In [2], a static analysis was defined for capturing message-passing communications in cryptographic processes defined in the language of the spi calculus [16]. The analysis was non-uniform in that it could distinguish different instances of the newly created names and was based on the denotational semantics approach. In [4], the same analysis was extended to PKI-based systems. In [3], a different variation of the analysis was defined to detect quantitative aspects of security, in particular, the speed of information leakage in a system. In [5], the same analysis was enhanced to detect men-in-the-middle attacks by precise timing of communications.

Literature provides several protocols for achieving delegation. In [17], the security implications when adopting delegation solutions in Grids are considered. These implications are discussed in the scope of two delegation schemes for Grids; delegation chaining and call-back delegations. Another research work closely related to the DToken protocol is the hierarchical delegation tokens architecture and protocols proposed by Ding and Petersen [18]. In this work, the authors propose a number of delegation protocols based on the Schnorr signature scheme [19], which are either key-based, identity-based or a combination of the two.

Not many of these protocols have been formally verified. In [20], the authors verify the delegation scheme in the SESAME protocol, a compatible extension version of Kerberos [11], using the Coq theorem prover [21]. In [22], the authors provide a formalisation of the security of proxy signature schemes and analyse one such scheme, namely the Kim, Park and Won scheme [23].

9. Conclusion

We presented in this paper a formal verification using static analysis methods for detecting several vulnerabilities in the DToken delegation protocol designed for Grid systems. The analysis revealed that the protocol suffers from vulnerabilities related to the integrity of its messages, the possibility of delegation repudiation by agents and the lack of deterministic delegation chains. Our conclusion regarding the protocol is that it is not suitable for delegations in scenarios where the delegated permissions refer to stateless Web services or require the anonymity of the participants. Also, it is limited by its lack of an essential feature in delegations, i.e. deterministic delegation chains.

In fact, the method of discovering this last vulnerability raises an interesting question of the precision of the design of a security protocol and indeed of the design of any system: In [1], the designers of the protocol considered only the cases of first and second level delegations. These simple cases could not reveal the problem of the lack of ability of constructing deterministic delegation chains, since they are not *precise* enough with the number of agents in the system. However, our analysis, which raised the level of precision of the system by introducing a third level of delegation, was able to reveal the vulnerability. This raises the need for the integration of robust formal methods at all stages of the software development cycle, particularly, for those components of the system that are deemed most critical.

References

[1] E. Y. Yang, B. Matthews, Dtoken: A lightweight and traceable delegation architecture for distributed systems, in: SRDS '09: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems, IEEE Computer Society, Washington, DC, USA, 2009, pp. 107–116. doi:http://dx.doi.org/10.1109/SRDS.2009.31.

[2] B. Aziz, G. Hamilton, D. Gray, A static analysis of cryptographic processes: The denotational approach, Journal of Logic and Algebraic Programming 64(2) (2005) 285–320.

[3] B. Aziz, Measuring the speed of information leakage in mobile processes, in: Proceedings of the 11$^{\text{th}}$ International Conference on Algebraic Methodology and Software Technology, Vol. 4019 of Lecture Notes in Computer Science, Springer Verlag, Kuressaare, Estonia, 2006, pp. 36–50.

[4] B. Aziz, G. Hamilton, The modelling and analysis of pki-based systems using process calculi, International Journal of Foundations of Computer Science 18 (3) (2007) 593–618.

[5] B. Aziz, G. Hamilton, Detecting Man-in-the-Middle Attacks by Precise Timing, in: Proceedings of the The 3rd International Conference on Emerging Security Information, Systems and Technologies (Securware 2009), Athens, Greece, 2009.

[6] T. L. S. W. Group, The ssl protocol version 3.0 (Nov. 1996).

[7] S. Tuecke, V. Welch, D. Engert, L. Pearlman, M. Thompson, Internet x.509 public key infrastructure (pki): Proxy certificate profile, RFC 3820 (Jun. 2004).

[8] M. Abadi, C. Fournet, Mobile Values, New Names, and Secure Communication, in: Proceedings of the 28$^{\text{th}}$ ACM Symposium on Principles of Programming Languages, ACM Press, London, UK, 2001, pp. 104–115.

[9] D. Dolev, A. Yao, On the security of public key protocols, in: Proceedings of the 22$^{\text{nd}}$ Annual Symposium on Foundations of Computer Science, 1981, pp. 350–357.

[10] I. Cervesato, The dolev-yao intruder is the most powerful attacker, in: J. Halpern (Ed.), Proceedings of the 16th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, Boston, MA, U.S.A., 2001, pp. 246–265.

[11] S. P. Miller, C. Neuman, J. I. Schiller, J. H. Saltzer, Kerberos authentication and authorization system - project athena technical plan, Tech. Rep. Section E.2.1, MIT, USA (Oct. 1987).

[12] Q. Pham, J. Reid, A. McCullagh, E. Dawson, On a Taxonomy of Delegation, Challenges for Security, Privacy and Trust 29(5) (2010) 565–579.

[13] M. Masi, R. Maurer, On the usage of SAML delegate assertions in an healthcare scenario with federated communities, Tech. rep., Dipartimento di Sistemi e Informatica, Univ. Firenze (2010).

[14] Y. Zhang, J.-L. Chen, A Delegation Solution for Universal Identity Management in SOA, IEEE Transactions on Services Computing 99.

[15] J. Schiffman, X. Zhang, S. Gibbs, Dauth: Fine-grained authorization delegation for distributed web application consumers, IEEE International Workshop on Policies for Distributed Systems and Networks (2010) 95–102.

[16] M. Abadi, A. Gordon, A calculus for cryptographic protocols: The spi calculus, in: Proceedings of the 4th ACM Conference on Computer and Communications Security, ACM Press, Zurich, Switzerland, 1997, pp. 36–47.

[17] P. Broadfoot, G. Lowe, Architectures for Secure Delegation within Grids, Tech. Rep. PGR-RR-03-19, Oxford University Computing Laboratory (2003).

[18] Y. Ding, H. Petersen, A New Approach for Delegation using Hierarchical Delegation Tokens, Tech. Rep. TR-95-5-E, University of Technology Chemnitz-Zwickau (1995).

[19] C. P. Schnorr, Effecient Signature Generation by Smart Cards, Journal of Cryptology 4 (1991) 161–174.

[20] M. Ayadi, D. Bolignano, On the formal verification of delegation in SESAME, in: Proceedings of the 12th Annual Conference on Computer Assurance (COMPASS'97), IEEE Computer Society, 1997, pp. 23–34.

[21] Y. Bertot, P. Castéran, Coq'Art: The Calculus of Inductive Constructions, Springer, 2004.

[22] Z. Tan, Z. Liu, Provably secure delegation-by-certification proxy signature schemes, in: InfoSecu '04: Proceedings of the 3rd international conference on Information security, ACM, New York, NY, USA, 2004, pp. 38–43. doi:http://doi.acm.org/10.1145/1046290.1046299.

[23] S. Kim, S. Park, D. Won, Proxy signatures, revisited, in: Y. Han, T. Okamoto, S. Qing (Eds.), ICICS, Vol. 1334 of Lecture Notes in Computer Science, Springer, 1997, pp. 223–232.