Correcting a Delegation Protocol for Grids

Benjamin Aziz

School of Computing
University of Portsmouth
Portsmouth PO1 3HE, United Kingdom
http://azizb.myweb.port.ac.uk/contact.html

Abstract. *Delegation is one important aspect of large-scale distributed systems where many processes and operations run on behalf of system users and clients in order to achieve highly computational and resource intensive tasks. As such, delegation is often synonymous with the concept of trust, in that the delegator would expect some degree of reliability regarding the delegatee's ability and predictability to perform the delegated task. The delegation protocol itself is expected to maintain certain basic properties, such as integrity, traceability, accountability and the ability to determine delegation chains. In this paper, we give an overview of the vulnerabilities that one such delegation protocol exhibits, namely DToken, a lightweight protocol for Grid systems, as interesting examples of design mistakes. We also propose an alternative protocol, DToken II, which fixes such vulnerabilities.*

Keywords: Delegation Protocols, Grid Systems, Security, Trust

1 Introduction

Delegation is a concept that is usually mentioned in the context of trust, where generally a delegator would hold some trust in the ability and predictability of the delegatee in carrying out some task on behalf of the delegator or performing some behaviour related to the purpose of the delegation. There have been numerous definitions of what delegation is in the context of computing systems (e.g. [1, 2, 5, 18]). In Grid systems, a common mechanism used to achieve delegation is via *proxy certificates* [20]. However, proxy certificates have often been criticised in the past [20–22] for their weak performance, the lack of symmetric non-repudiation (only the identity of the delegator is preserved in the delegation) and the various security implications arising from the fresh generation of encryption key pairs at each delegation level.

Hence, DToken was proposed as a lightweight protocol [22] that could replace proxy certificates as a reliable and secure solution for the problem of delegation in large-scale Grid systems. Grid middleware systems, such as Globus[1] or GLite[2], adopt a model of the Grid often referred to as Virtual Organisations (VOs), where in a VO, users from one organisation are permitted access

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

and usage of resources such as computational power, storage and network bandwidth, belonging to another organisation under certain constraints. However, such cross-organisational provisioning of resources requires critical issues of trust and security to be managed in a reliable manner. One such issue is related to delegations by users to Grid *gateways* that allow the gateways to perform tasks on behalf of the users within the scope of the delegated permissions. DToken was designed to achieve such delegations in an integrity-preserving, accountable, traceable and determinisitic manner.

In this paper, we provide an overview of the results of a formal analysis that applied to the DToken protocol in [4] and that uncovered serious vulnerabilities in the protocol related to the above properties. We furthermore propose a second corrected version of the protocol, DToken II, which we claim fixes the vulnerabilities of the original version by changing slightly the specification of the protocol to allow agreement on the session identity number between the delegator and the delegatee, prevent the immature passing of permissions from the delegator to the delegatee and finally, use ordered lists instead of sets to pass information about delegation chains.

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, namely integrity, traceability and accountability, and deterministic delegation chains. In Section 3, we demonstrate that the protocol in fact suffers from vulnerabilities that undermine all of these desirable properties. In Section 4, we propose a new version of the protocol, DToken II, which implements fixes to the vulnerabilities in the original version. Finally, in Section 5, we discuss related work and conclude in Section 6.

2 The Delegation Protocol

We give an overview here of the DToken delegation protocol as was defined in [22]. The protocol comprises secure communications between a *Delegator*, *Dor*, and a *Delegatee*, *Dee*. The following sequence of messages describes the interactions in the protocol:

1. $Dor \rightarrow Dee : C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee0}, Sig_{Dor \rightarrow Dee}$
2. $Dee \rightarrow Dor : C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee}, Sig_{Dor \rightarrow Dee}, Sig_{Dee \rightarrow Dor}, C_{Dor_{\mathrm{CAs}}}$

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
$Sig_{Dor \rightarrow Dee} \stackrel{\texttt{def}}{=} |\{C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee0}\}|_{K_{Dor}}$,
$Sig_{Dee \rightarrow Dor}$: The signature of *Dor*'s signature in the first message signed by the private key of *Dee*, K_{Dee}, where $Sig_{Dee \rightarrow Dor} \stackrel{\texttt{def}}{=} |\{Sig_{Dor \rightarrow Dee}\}|_{K_{Dee}}$, and
$C_{Dor_{\text{CAs}}}$: The list of subordinate CAs linking C_{Dor} to the trusted root authority.

Where we write $|\{M\}|_K$ to represent the signing of the message M by the private key K, which is also a syntactic sugar for applying an encryption $\{|-|\}_K$ to the hash of the message; i.e. $|\{M\}|_K \stackrel{\texttt{def}}{=} \{|hash(M)|\}_K$, such that $hash$(-) is any of the MD5 or SHA hashing functions.

From this definition of the protocol, there are a few points to note as highlighted in [22]. 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 [11]. 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, $P_{Dor \rightarrow Dee}$, 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 [22], the authors give an example of a delegation chain in Grid systems as shown in Figure 1.

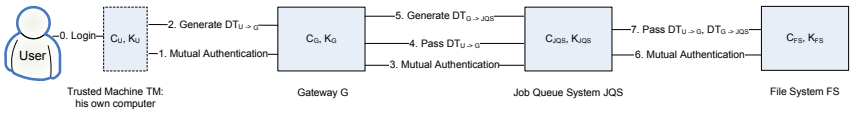

Fig. 1. DToken Chained Delegation through a Gateway (cited from [22, Figure 5]).

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 [20]), therefore, it should sustain a few properties related to its purposes and functionality.

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 [22] must always succeed.

Property 1 (DToken Integrity Validation). A DToken is said to be valid if the following equations are true:
$hash(C_U, C_G, V_{fr}, V_{to}, TS, P_{U \to G}, DS_{U \to G}) = decrypt(Sig_{U \to G}, C_U)$
$hash(Sig_{U \to G}) = decrypt(Sig_{G \to U}, C_G)$ □

Where $decrypt(\{\!|M|\!\}_{K_A}, C_A)$ is a decryption function that if applied to a message encrypted with the private key K_A of some agent A using the public certificate C_A for that agent, will yield back the plaintext M corresponding to the ciphertext.

The first of the above equations 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.

Traceability and Accountability. Traceability is defined in [22] 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.

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

Property 2 (Verifiable Non-repudiation). Given a delegator, *Dor*, and a delegatee, *Dee*, then we say that neither can deny the delegation if the following holds:
$\forall P_{Dor \to Dee} \in Perms : use(Dee, P_{Dor \to Dee}) \Rightarrow$
$(signed(Dee, P_{Dor \to Dee}) \ \wedge \ signed(Dor, P_{Dor \to Dee}))$ □

Where $use(Dee, P_{Dor \to Dee})$ is a predicate implying that *Dee is able to use* the permissions $P_{Dor \to Dee}$ it has received from the delegator *Dor*, and $signed(A, P_{A \to B})$ is another predicate meaning that the signature $Sig_{A \to B}$ exists and has been created by A. Hence, the above property says that it must hold that each time a delegatee is able to use some permissions, then those permissions must have been signed by both the delegator and the delegatee agents.

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 define 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 1 (Delegation Chains). *Given a set of DTokens, DT_{set}, then we name the set of all DToken chains that can be constructed from DT_{set} as DT_{chain}. Every element in DT_{chain} is a finite list, dc, where $|dc| > 1$ and such that for any two adjacent DToken elements, $a, b \in dc$, then the following holds:*
$\exists Dor, Dee, H \in Agents : \quad (a = DT_{Dor \to Dee}) \wedge (b = DT_{Dee \to H})$ □

Hence, in a delegation chain, adjacent elements have common adjacent participating agents (i.e. *Dee*). Now, we can define the property of deterministic delegation chains as follows.

Property 3 (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 Vulnerabilities of the Current Protocol

In [4], we carried out a verification of the DToken protocol using static analysis techniques based on abstract interpretation. The analysis revealed several vulnerabilities in the protocol, which we summarise in the following sections. The assumption we made regarding the nature of the intruder was that the intruder was just another protocol participant (same assumptions made in [22]) who has a well-known certified identity who is only able to divert the protocol via messages that make sense to other participants. The use of secure communications in the protocol prevented external intruders from interfering with the protocol messages. This assumption yields our intruder less powerful than Dolev-Yao's *most powerful attacker* [10, 8], 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.

3.1 Non-Matching Hash Validation

The first vulnerability we discovered was in the case of a single delegation step, i.e. where there is one delegator, *Dor*, and one delegatee, *Dee*, though it is also applicable to the more general case of n delegation steps. The vulnerability is simply an incorrect specification of the protocol that prevents the integrity of a DToken from being validated. This is caused by the fact that the delegator (*Dor*) agent always signs a *Null* value for the delegation session identifier, which is $DS_{Dor \rightarrow Dee0}$ in Message 1. This is, in Message 2., assigned a different value by the delegatee (*Dee*), which is the value of the identifier $DS_{Dor \rightarrow Dee}$. In the protocol of Section 2, this is equivalent to the first integrity check:

$$\begin{aligned} &hash(C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee}) \ = \\ &decrypt(\{\!|\{C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee0}\}|\!|_{K_{Dor}}, C_{Dor}) \end{aligned}$$

Applying the decryption of the signature, we further simplify the equation to:

$$\begin{aligned} &hash(C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee}) \ = \\ &hash(C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee0}) \end{aligned}$$

which clearly does not hold, due to the difference in the value of $DS_{Dor \rightarrow Dee}$ in both messages. This reveals a lack of agreement on the delegation session identifier values as assigned by the delegator and the delegatee, and it is a significant result as it undermines the claims in [22] 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.

3.2 Delegation Repudiation

The second vulnerability that we uncovered was that delegations in the DToken protocol can be repudiated. This is relevant to the case of a delegation where the delegatee attempts to execute permissions received from the delegator on a DEP. For example, in the case of a Grid system, the delegator could be the user U and the delegatee the gateway G. G then attempts to execute permissions received from U say on a file system. The case assumes that both U and DEP play their normal roles in an honest manner, whereas G is playing a man-in-the-middle role where in addition to being able to run its normal protocol behaviour (delegatee for U and user of DEP), it is also running extra code that attempts to subvert the protocol. A "robust" protocol, hence, would be expected to withstand such subversive behaviour.

Assuming that U delegates to G and G attempts to execute the delegated permissions on the DEP, the attack occurs with the following run of messages from two sessions:

$$1.\ U \rightarrow G : C_U, C_G, V_{fr}, V_{to}, TS, P_{U\rightarrow G}, DS_{U\rightarrow G0}, Sig_{U\rightarrow G}$$
$$1'.\ G \rightarrow G : C_G, C_G, V_{fr}, V_{to}, TS, P_{U\rightarrow G}, DS_{U\rightarrow G0}, Sig_{G\rightarrow G}$$
$$2'.\ G \rightarrow G : C_G, C_G, V_{fr}, V_{to}, TS, P_{U\rightarrow G}, DS_{U\rightarrow G}, Sig_{G\rightarrow G}, |\{Sig_{G\rightarrow G}\}|_{K_G}, C_{G_{\mathrm{CAs}}}$$

In the first session, the user U attempts to delegate some permissions $P_{U\rightarrow G}$ in Message 1 to G. This session is left incomplete by G, and so it does not sign anything (i.e. there is no Message 2 for the first session). In the second session, G simply delegates the received permissions to itself in order to create a syntactically valid but semantically dummy DToken. The DEP will successfully validate (on the condition that the integrity validation vulnerability of the previous section is fixed) the token as it is syntactically correct and will assume that G has delegated the permissions to itself. Therefore, the delegation that occurred in the first session can easily be repudiated by the delegatee.

The vulnerability works because it is possible, with the above run of messages, to show that:

$$\exists P_{U\rightarrow G} \in Perms : use(G, P_{U\rightarrow G}) \ \Rightarrow\ \neg(signed(G, P_{U\rightarrow G}) \ \wedge\ signed(U, P_{U\rightarrow G}))$$

As a result, the right hand side of the implication will always be false in either of the two sessions. In the first session, G does not sign the delegation token and so can repudiate the delegation, and in the second session, U has not signed the delegation, and so it can repudiate the delegation as well. The main reason behind this vulnerability is that the delegatee always receives permissions prematurely from the delegator, therefore, it is able to subvert its part on signing

the DToken, whereas it is too late for the delegator to sign those permissions in the second session.

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 $P_{U \to G}$. 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.

3.3 Non-deterministic Delegation Chains

This case is an interesting one since it did not feature in the original design of the DToken protocol presented in [22] and it assumes the presence of four agents, two of which are playing men-in-the-middle roles, though the vulnerability also applies to the general case of $n \geq 4$ number of agents. The original protocol of [22] assumes in several occasions[3] 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 1. 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: $\{DT_{A \to B}, DT_{B \to C}, DT_{C \to A}, DT_{A \to C}, DT_{C \to D}\}$

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. □

From our analysis in [4], we were able to show that $|DT_{chain}| > 1$ in the case of four agents or more participating in the protocol. Since delegation is a form of trust, this vulnerability breaks the trust chain and does not preserve the deterministic delegation chain property of Section 2. Chains of trust are common in many sensitive scenarios, such as in the case of digital forensics evidence preservation, where Chains of Custody (CoC) require that every step in the handling of a criminal evidence is well documented and its integrity can be proven for the evidence to be acceptable in a court. Hence, the CoC (*A delegates to B, B delegates to C, C delegates to A, A delegates to C and C delegates to D*) may be trusted in a court, whereas the CoC (*A delegates to C, C delegates to A, A delegates to B, B delegates to C and C delegates to D*) may not.

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

Technically, this vulnerability arises from the fact that DTokens are passed as a set, rather than as a list as is expected from the definition of delegation chains in Property 3. 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 DToken II: The Corrected Version

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

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

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. Next, we discuss the three properties we introduced in Section 2.1 in light of this new protocol.

4.1 DToken Integrity Validation

One suggestion to fix the vulnerability of non-matching hash validation in the current protocol 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 of the identifier as the one chosen by the delegatee. Hence, in DToken II, both messages 3 and 4 (corresponding to messages 1 and 2 in the original protocol) use the same value for delegation session identifier, $DS_{Dor \rightarrow Dee}$, therefore both validation steps of Property 1 involving the comparison of the hashes of the two signatures will succeed, since they both are applied to the same value of $DS_{Dor \rightarrow Dee}$.

4.2 Verifiable Non-repudiation

Fixing the vulnerability of delegation repudiation in the original protol, 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.

To be able to prove Property 2 holds, the definition of RfD_{Dor} in Message 1 of the new protocol will need to include some reference to $P_{Dor \rightarrow Dee}$. Once this is signed in Message 2 by the delegatee, the latter cannot repudiate its acceptance of the session, even though this is proven outside the structure of the DToken itself (i.e. within RfD_{Dor}). This is because the implication in Property 2 will hold true. Additionally, since the delegatee has signed the session identifier also in Message 2, the delegator can now prove the association between the delegated permissions and the delegation session through this identifier.

4.3 Deterministic Delegation Chains

Finally, in order to achieve deterministic delegation chains, we propose that DTokens be passed in a *list* structure as opposed to the *set* structure as it was done in the original protocol, when performing second and further level delegations. Let's consider the same example we discussed in Section 3.3. The set of DTokens passed will be a list $[DT_{A \rightarrow B}, DT_{B \rightarrow C}, DT_{C \rightarrow A}, DT_{A \rightarrow C}, DT_{C \rightarrow D}]$, which is unique, i.e. adding new elements or changing the ordering sequence of elements will result in a new list (chain). From a practical point of view, this implies that in the case of the original design of Figure 1, we propose that the job queueing system passes the list [DTU $\rightarrow$ G; DTG $\rightarrow$ JQS] instead of {DTU $\rightarrow$ G, DTG $\rightarrow$ JQS}. This will ensure that DTokens are ordered in their temporal sequencing and so non-determinism does not arise when building chains.

5 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 [14]. A recent taxonomy of delegation methods has recently been published in [15], 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 [23] and in the context of Web-based social networking [16].

Literature provides several protocols for achieving delegation. In [7], 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 [9]. In this work, the authors propose a number of delegation protocols based on the Schnorr signature scheme [17], which are either key-based, identity-based or a combination of the two.

The work in [4] is not the first case where formal analysis techniques have been applied to delegation protocols. In [3], the authors verify the delegation scheme in the SESAME protocol, a compatible extension version of Kerberos [14], using

the Coq theorem prover [6]. In [19], the authors provide a formalisation of the security of proxy signature schemes and analyse one such scheme, namely the Kim, Park and Won scheme [12].

6 Conclusion and Future Work

We presented in this paper an overview of the vulnerabilities that were formally proven in [4] to exist in the DToken protocol [22]; a lightweight delegation protocol for Grid systems. As a result 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, which are essential when chains of trust are to be preserved, for example in cases where forensic evidence on the usage of Grid resources must be maintained over the chain of custody handling the evidence. We also presented DToken II, an alternative protocol, which include fixes to the above vulnerabilities. In the future, we plan to formally verify the new DToken II protocol to analyse its behaviour against the three properties described in this paper.

References

1. Atluri, V., Warner, J.: Supporting conditional delegation in secure workflow management systems. In: Proceedings of the tenth ACM symposium on Access control models and technologies. pp. 49–58. SACMAT '05, ACM, New York, NY, USA (2005)
2. Aura, T.: On the structure of delegation networks. In: Proceedings of the 11th IEEE workshop on Computer Security Foundations. pp. 14–26. IEEE Computer Society, Washington, DC, USA (1998)
3. Ayadi, M., Bolignano, D.: On the formal verification of delegation in SESAME. In: Proceedings of the 12th Annual Conference on Computer Assurance (COMPASS'97). pp. 23–34. IEEE Computer Society (Jun 1997)
4. Aziz, B., Hamilton, G.: Verifying a delegation protocol for grid systems. Future Generation Computer Systems: The International Journal of Grid Computing and eScience 27(5), 476–485 (2011)
5. Barka, E., Sandhu, R.: Framework for role-based delegation models. In: Proceedings of the 16th Annual Computer Security Applications Conference. pp. 168–176. ACSAC '00, IEEE Computer Society, Washington, DC, USA (2000)
6. Bertot, Y., Castéran, P.: Coq'Art: The Calculus of Inductive Constructions. Springer (2004)
7. Broadfoot, P., Lowe, G.: Architectures for Secure Delegation within Grids. Tech. Rep. PGR-RR-03-19, Oxford University Computing Laboratory (2003)
8. Cervesato, I.: The dolev-yao intruder is the most powerful attacker. In: Halpern, J. (ed.) Proceedings of the 16^{th} Annual Symposium on Logic in Computer Science. pp. 246–265. IEEE Computer Society Press, Boston, MA, U.S.A. (Jun 2001)
9. Ding, Y., Petersen, H.: A New Approach for Delegation using Hierarchical Delegation Tokens. Tech. Rep. TR-95-5-E, University of Technology Chemnitz-Zwickau (1995)

10. Dolev, D., Yao, A.: On the security of public key protocols. In: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science. pp. 350–357 (Oct 1981)
11. Group, T.L.S.W.: The ssl protocol version 3.0 (Nov 1996)
12. Kim, S., Park, S., Won, D.: Proxy signatures, revisited. In: Han, Y., Okamoto, T., Qing, S. (eds.) ICICS. Lecture Notes in Computer Science, vol. 1334, pp. 223–232. Springer (1997)
13. Masi, M., Maurer, R.: 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. Miller, S.P., Neuman, C., Schiller, J.I., Saltzer, J.H.: Kerberos authentication and authorization system - project athena technical plan. Tech. Rep. Section E.2.1, MIT, USA (Oct 1987)
15. Pham, Q., Reid, J., McCullagh, A., Dawson, E.: On a Taxonomy of Delegation. Challenges for Security, Privacy and Trust 29(5), 565–579 (2010)
16. Schiffman, J., Zhang, X., Gibbs, S.: Dauth: Fine-grained authorization delegation for distributed web application consumers. IEEE International Workshop on Policies for Distributed Systems and Networks pp. 95–102 (2010)
17. Schnorr, C.P.: Effecient Signature Generation by Smart Cards. Journal of Cryptology 4, 161–174 (1991)
18. Stein, L.A.: Delegation is inheritance. SIGPLAN Not. 22, 138–146 (December 1987)
19. Tan, Z., Liu, Z.: Provably secure delegation-by-certification proxy signature schemes. In: InfoSecu '04: Proceedings of the 3rd international conference on Information security. pp. 38–43. ACM, New York, NY, USA (2004)
20. Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, M.: Internet x.509 public key infrastructure (pki): Proxy certificate profile. RFC 3820 (Jun 2004)
21. Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Gawor, J., Meder, S., Siebenlist, F.: X.509 proxy certificates for dynamic delegation. In: In Proceedings of the 3rd Annual PKI Research and Development Workshop (2004)
22. Yang, E.Y., Matthews, B.: 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. pp. 107–116. IEEE Computer Society, Washington, DC, USA (2009)
23. Zhang, Y., Chen, J.L.: A Delegation Solution for Universal Identity Management in SOA. IEEE Transactions on Services Computing 99 (2010)