My private cloud – granting federated access to cloud resources

David W Chadwick[1*]
[*] Corresponding author
Email: d.w.chadwick@kent.ac.uk

Matteo Casenove[1]
Email: m.casenove@gmail.com

Kristy Siu[1]
Email: kwss2@kent.ac.uk

[1] School of Computing, University of Kent, Canterbury CT2 7NF, UK

Abstract

We describe the research undertaken in the six month JISC/EPSRC funded My Private Cloud project, in which we built a demonstration cloud file storage service that allows users to login to it, by using their existing credentials from a configured trusted identity provider. Once authenticated, users are shown a set of accounts that they are the owners of, based on their identity attributes. Once users open one of their accounts, they can upload and download files to it. Not only that, but they can then grant access to their file resources to anyone else in the federated system, regardless of whether their chosen delegate has used the cloud service before or not. The system uses standard identity management protocols, attribute based access controls, and a delegation service. A set of APIs have been defined for the authentication, authorisation and delegation processes, and the software has been released as open source to the community. A public demonstration of the system is available online.

Keywords

Federated access, Attribute based access controls (ABAC), Delegation of authority, Identity management

Introduction

When anyone offers a new cloud service, one of the first things they typically do is register the new users and give them a new set of credentials. We believe this is sub-optimal practise for a number of reasons. Firstly it is burdensome on the user, who already has too many sets of credentials to remember. Secondly it is burdensome on the cloud software developer, who has to: determine what type of authentication scheme to use, build a secure store for the new credentials and ensure that the authentication mechanism works securely and correctly. Thirdly it is burdensome on the operational staff who may have to register and remove users from the system and deal with users losing their credentials and asking for new ones. Finally, and perhaps most importantly, it can lead to security vulnerabilities, since it now becomes a target for hackers to attack and steal the users' credentials [1], and users tend to lighten their mental load by either choosing weak and easy to remember passwords or using the same password for each of their systems.

All of these burdens can be reduced if we make use of trust within systems. Trust is known to lower the cost of doing business [2]. If you can get a trusted third party to provide a service you need, which they specialise in, at less cost to yourself, then you don't have to provide it, and you can both benefit from the transaction. You obtain a higher quality of service at a lower cost, and they profit from selling their expertise. Since cloud service providers want users to trust them and use their new services, isn't it reasonable to expect those same cloud service providers to trust other service providers who already specialise in authenticating and identifying users? These latter service providers are known as identity providers (IdPs), and there are already thousands of them in existence today, providing their services to existing end user service providers, through a process known as identity federation.

This trust based approach was adopted by the My Private Cloud project, a six month project funded by EPSRC/JISC, to see how trust based services from the EC funded Trusted Architecture for Securely Shared Services (TAS3) project [3] could be applied to cloud services. From our experiences of building federated services in the EC TAS3 project, we decided to apply this to the open source S3 cloud service offered by Eucalyptus [4]. TAS3's trust model is based on Ronald Reagan's famous saying "trust but verify". When applied to web service providers this means that web services' clients (WSCs) are able to trust web service providers (WSPs) to provide the security services they advertise, by virtue of both the contractual agreements they offer and the reputations they have earned from other service users. Moreover the clients can subsequently validate that the WSPs do provide these services according to their contractual legal agreements through an audit summary which is sent directly to them. In the context of the current project, the S3 cloud service is the WSC. Once we have acknowledged that WSPs can be trusted, we can start to build the trust, security and privacy preserving infrastructures that cloud service providers require, by relying on these trusted third parties to operate parts of the security infrastructure. As researchers we can turn our attention to the problems of designing and building tools and services for a large scale trusted federated privacy preserving cloud infrastructure, with fine grained access control, user accessible audit trails, and trust assurance mechanisms.

Current state of the Art

When users submit their personal data to web sites and cloud providers today, they have to trust that the provider will act in accordance with its advertised policies and contract terms. Users typically have no visibility about what happens to their data after it has been given to the provider. Users have little evidence to tell them which providers to trust and have no control over the policy that is applied by the service provider to their data. If users do not agree to the provider's terms and conditions, they are typically not allowed to use the service. In the case of Amazon S3, for example, the user is presented with a lengthy legal document that has the following clause buried within it [5]:

> *4.2 Other Security and Backup. You are responsible for properly configuring and using the Service Offerings and taking your own steps to maintain appropriate security, protection and backup of Your Content, which may include the use of encryption technology to protect Your Content from unauthorized access and routine archiving Your Content.*

This places a considerable burden on the user (data encryption, backup etc.) and seems to absolve the cloud service provider from all responsibility to keep the data secure, and from any liability should the data be lost, compromised or corrupted. This type of clause represents a significant inhibitor to many users which prevents them from using cloud computing today.

Many researchers similarly assume that cloud service providers cannot be trusted and that all data must be encrypted before it is submitted to the cloud, see for example [6]. Mowbray and Pearson [7] recognize the problem of privacy protection when submitting sensitive information to (untrusted) clouds. Their solution uses a client based privacy manager which obfuscates sensitive data before submitting it to the cloud.

However, organisations have been regularly outsourcing (unencrypted) data and processing to third party providers for decades [8]. Many UK government's IT services are already run by third party providers and this is expected to grow and migrate to cloud computing [9]. Consequently many organisations are routinely used to trusting third parties with their data and the processing of it. It therefore seems logical to assume that most cloud service providers will similarly assume the 'trusted data handler' label in due course. My Private Cloud aimed to facilitate this process by providing cloud service providers (cloud SPs) with the software tools, procedures and services to easily offer trusted cloud services to cloud users, by relying in part on external trusted WSPs.

Current cloud platforms (e.g. Amazon, Google, Eucalyptus etc.) do not support federated access, delegation of authority, or fine grained access control. They primarily use a simple Access Control List (ACL) to provide access to other cloud users. In general, these are coarse grained. Controlling who might invoke a virtual machine is often restricted to everyone or no-one. Amazon's S3's ACLs only allows users to specify the following access levels: anonymous access (everyone, including non-Amazon Web Services (AWS) users), all AWS users, or named individual AWS users. All existing ACL based systems are limited in only being able to grant controlled access to other registered users. Amazon has recently extended its Access Control system with Bucket Policies. These policies allow users to manage access to their S3 resources at the bucket level for both the buckets and the objects, providing a more fine grained access control for those resources, but still limited to other registered Amazon users. As pointed out in [10], this rather crude access control mechanism makes it difficult to use S3 as part of a (large) science project involving many collaborators. Also, the lack of a fine-grained delegation mechanism impedes the use of S3 in data-driven science projects where scientists often want to delegate access rights to a particular data set to an application/job running on their behalf.

Different approaches to the cloud access control problem have been published in the literature. V. Echevarrıa et.al [11] have developed a novel approach called Permission as a Service (PaaS) which provides a separate access control service from the other cloud services. In PaaS, user data is encrypted in the cloud, using attribute based encryption (ABE), to maintain confidentiality. Permissions are managed via decryption keys based on the attributes of the users being granted access. Our approach is similar to PaaS, in that access is granted based on the attributes of the user, but there the similarity ends. We assume the cloud service provider can be trusted to keep the information confidential, so encryption is not mandatory (though the cloud provider can encrypt the data, as an optional value added service, if it wishes to). The user is given confidence to do this through the ability to attach "sticky" policies to his data and receive audit summaries from the cloud SP. The permissions in our system are provided by ACLs decided by the user following the discretionary access control model. Moreover, our system permits delegation of access and allows the user to give permission to someone who is not an existing cloud user, as long as they can be authenticated by one of the trusted federation IdPs.

Recent work by Dongwan Shin [12] added fine grained role based access controls to IaaS, by introducing a trusted domain that is used to manage users, roles and access permissions. But all the users, roles and permissions are managed centrally within this domain, thus it does not provide federated access or allow roles and attributes to be assigned by external attribute authorities, as in for example the UK Access Management Federation [13]. We believe that these are essential features for scalability, and they are provided as part of the current project.

Federated identity management, for example, as typified in the Shibboleth implementation [14], comprises a trusted Identity Provider (IdP), Service Provider (SP) and user agent (usually a web browser). The user attempts to access the SP via his user agent, but the SP, not knowing who the user is, redirects the user to its trusted IdP for authentication. If the SP supports multiple IdPs, then it may first ask the user to choose which IdP he wants to use for authentication. When the user's agent contacts the IdP, the user is asked to authenticate, and if successful, the IdP redirects the user back to the SP, providing the agent with digitally signed assertions to present to the SP. These assertions say that the user has been authenticated by the IdP, and that he has the attached identity attributes. These attributes may include a unique persistent identifier (PID) for the user, so that the SP can provide a personal service for the user on repeated visits. When we apply this model to cloud services, the cloud service provider (cloud SP) becomes the SP, and the IdP can be any existing IdP which the cloud SP trusts, from any existing federation to which the cloud SP belongs.

Delegation of Authority allows a user (the delegator) to delegate any of his privileges to another user or application of his choice (the delegate) [15]. When role based or attribute based access controls are used, then the delegator may delegate a role or attribute instead of a privilege, since privileges are assigned to roles and attributes. This form of delegation is superior to existing grid based delegation that relies on proxy certificates [16], since i) the delegate must authenticate as himself, and not as a child of the delegator as in proxy certificates (which is a form of masquerade) and ii) fine grained delegation is automatically supported by the delegator delegating a subset of his identity attributes. Delegation in federated systems is made more difficult because the cloud SP may not have had contact with the delegate before, therefore does not know how to recognise which user is the chosen delegate.

Original project objectives

The main objective of the My Private Cloud project was to port existing research software, tools and methods from the European FP7 integrated project TAS3, to the cloud, in order to undertake the following proof-of-concept activities:

i. integrate a trust and reputation infrastructure into cloud services that will allow users to determine which cloud providers are the most trustworthy, and that will block untrustworthy users from gaining access to cloud resources;
ii. *allow the users to set their own fine grained privacy policies for controlling all accesses to their data whilst it is in the cloud;*
iii. *automatically enforce the user's privacy policy before any requestor is allowed access to*

the user's data;

iv. *ensure that the user's privacy policy is "stuck" to their data and follows their data around as it moves through the cloud;*

v. *allow the users to dynamically update their policies;*

vi. *ensure that legal and other policies (such as the cloud provider's) are also enforced as well as the user's privacy policy, and that appropriate conflict resolution strategies are adopted when conflicts arise;*

vii. provide users with the ability to delegate access to their data to any other entities of their choosing;

viii. provide federated access to cloud services and the user's data, thereby not constraining the set of users (to those only known to the cloud provider);

ix. include the level of assurance (LoA), a measure of trustworthiness, in the authentication assertion so that access decisions can be based on this;

x. *allow requestors to aggregate their roles and attributes from multiple sources of authority so that finer grained privacy policies can be created;*

xi. provide users with (configurable) audit trails that provide them with full visibility of what happens to their data whilst it is in the cloud;

xii. *provide an optional access over-ride capability (so called break the glass policies) which allow responsible requestors to over-ride the access control decision in an emergency situation, when they are willing to justify this later.*

The above were to be achieved by utilizing the software deliverables from the European FP7 TAS3 integrated project (http://www.tas3.eu/), which has provided open source software for the construction of a Trusted Architecture for Securely Shared Services.

Objectives shown in **bold** above were fully achieved, as well as some new originally unanticipated objectives described in section 4 below. Those shown in *italics* above were partially achieved, whilst those shown in normal font were not attempted due to the time constraints of the project.

New objectives

Whilst the original objectives remained valid, they could not all be achieved in the short duration of the project (6 months) due to the addition of new higher priority objectives which became apparent once the project had started. When developing the first prototype it soon became obvious that (sticky) policy based authorisation systems *on their own* cannot provide the user friendly access control functionality that is required. Specifically, a policy decision point (PDP), which evaluates the policies stuck to a resource, typically only answers Grant or Deny[1] to a request to access the resource. But when a delegated user logs into the cloud, he does not necessarily know which resources have been delegated to him by other users (and even if he did, one would not want to burden the user with the mental effort of having to remember all the resources' URLs). The only way the system could find out which resources were available to the delegate, using a PDP alone, would be to cycle through its entire list of resources asking the PDP if this user has access to them. This clearly is not feasible for cloud-scale resources, so another solution had to be sought. This introduced two new higher priority objectives into the project, namely:

- given the attributes of a user, enable the authorisation system to be able to instantly say

which resources the user potentially has access to, and
- given a resource, enable the authorisation system to be able to instantly say which users, based on their attributes, potentially have access to this resource.

When these two objectives are implemented as services, we can use them to locate a user's resources at login time, and after that use the PDP to perform fine grained access control on them.

Once we started the project implementation, we realised that in order to gain wide scale implementer acceptance, simply specifying the standard protocols and services that need to be implemented would not be sufficient. The EC TAS3 project had done this, but considerable implementation effort is still needed in order to utilise the security and trust web services that it provides. Many toolkits need to be used in order to create the secure SOAP messages that are needed to talk the various protocols that the different security web services use. A much simpler interface is needed, which led to another higher priority objective:

- can we specify a set of easy to use APIs that will make it very easy for implementers to integrate the security services that we are providing into their cloud applications?

Consequently significant effort was spent during the project in specifying and implementing these APIs (in both natural language and in our chosen implementation language, PHP).

Finally, we wanted to make the federated S3 service very easy to use by end users. Amazon's S3 service is web browser based, whereas Eucalytus's S3 uses a command line client. We preferred the browser interface as it is much easier to use, so another objective was added:

- can we to make the Eucalyptus S3 service accessible via a standard web browser so that no new software is needed by the end user?

The result of this, is that our project then focussed on creating a front end web application (our new proxyS3 software) that interfaces with the user, with the security APIs and with the Eucalyptus S3 service (Walrus) at the backend. This web application is called the proxyS3 server. Unfortunately a disproportionate amount of effort was spent during the project on this last objective, which we will discuss again in Lessons Learnt.

Architecture and design

The overall architecture is shown in Figure 1. The cloud service makes use of many external security services via a set of 3 security APIs: one for authentication, one for authorisation, and one for delegation. Behind each of the APIs lie infrastructure components that provide the appropriate service to the API. This architecture allows the infrastructure components to be changed, upgraded, and replaced etc. without affecting the functional service that the API provides to the application (although the quality of the service may vary with different backend components).

Figure 1 The overall architecture.

The Authn API acts as a federation SP which contacts the configured federation IdP asking it to authenticate the user and return his identity attributes, including a Persistent ID (PID). The

Authn API talks the standard SAML protocol, enhanced with the level of assurance, so that it can talk to any IdP that supports this. In our architecture we replace the (single) federation IdP with a proxyIdP that:

i. is capable of talking multiple IdP protocols such as SAML, OpenID, OAuth etc. to a variety of IdPs. This allows us to add new IdPs and new IdP protocols at any time, without effecting the cloud service, since its API continues to talk the enhanced SAML protocol to the proxy IdP;
ii. is configured with the level of assurance (LoA) of each federated IdP, so that it can inform the cloud service how trustworthy the IdP is;
iii. allows the user to choose which IdP he wishes to use for authentication, from the set that is available;
iv. allows the user to link together his various IdP accounts in order to aggregate his attributes and increase the LoA;
v. allows the cloud service provider to integrate its own corporate LDAP service in order to retrieve the identity attributes of its employees for finer grained access control;
vi. has a configurable Credential Validation Service (CVS) which contains policy rules saying which IdPs are trusted to issue which attributes to whom [17].

The proxy IdP is an enhancement of the SimpleSAMLphp [18] IdP. We added two new modules to it, and replaced the Facebook module which was out of date. Figure 2 shows the IdP selection screen that is presented to the user. After the user chooses his IdP, he is redirected to it, authenticates, and is then returned to the cloud service, which is given the full set of identity attributes of the user, including the PID and the LoA, by the proxy IdP. The design, implementation and user trials of the proxyIdP are described in [19]. A full description of the Authn API can be found in [20] and in the online documentation of the proxy S3 service [21].

Figure 2 The proxyIDP screen.

The Authz API is responsible for authorising the user to access resources. It has a backend PDP and a database that stores the access rights of the user, identified by their identity attributes, to the resources, also identified by their identity attributes. API calls are available to add rights, remove rights and get rights from the database, and obtain an authorisation decision from the PDP. As described in Section 4, it is currently not possible to use only a standard policy decision point (PDP), since additional supporting functionality is needed which has not yet been standardized e.g. the ability to determine what access rights a user has. For this reason, the database is used to obtain the initial potential set of user access rights to resources, then the PDP is called to determine any additional policy based constraints, such as limits on the time of access, or whether break the glass must be first enacted etc. A full description of this API can be found in [20] and [21].

The Delegation API allows one user, the delegator, to delegate any attribute to another user, the delegate. It has a Delegation Issuing Web Service (DIS) [15] to support it. This web service has a backend LDAP directory that holds the users, both delegators and the delegates, and the attributes that the former have delegated to the latter, along with any delegation constraints, such as the period of delegation and whether the attributes can be re-delegated etc. The delegated attributes and delegation constraints are held in digitally signed X.509 attribute certificates in the delegate's LDAP entry, so that they cannot be tampered with. The attribute certificates are signed by the delegation service itself. There were several obstacles to overcome in using our pre-existing DIS in the current project, and these are discussed in the next section.

API validation

In order to validate the security APIs, we built a proxyS3 service which front ends a normal S3 service through its APIs (which are provided by both Amazon's and Eucalyptus's S3 services). The proxyS3 service has been designed and built in a modular fashion, with configuration options specifying the resources and actions so that it can be adapted to other cloud storage services. The proxyS3 service comprises four parts: the security APIs (which can be used by any cloud service), the normal S3 APIs, a GUI (to capture input from the user and display results to the user) and the application logic which accesses both sets of APIs to upload, download files, and grant and enforce fine grained access to users etc. The application logic calls the security APIs and enforces the decisions they make, before calling the S3 APIs and displaying the results to the user via the GUI. In terms of policy based authorisation, the application logic is a policy enforcement point (PEP) which calls the security APIs in order to interact with the security services (authentication, authorisation and delegation) before enforcing their decisions when calling the S3 APIs. Figure 3 (top) shows the Welcome screen which the user is presented with when first contacting the cloud service.

Figure 3 Welcome screen and account chooser screen.

The proxyS3 uses the AWS library, which is written in php, to communicate with the cloud S3 service. It was therefore natural to write the entire proxyS3 service and security APIs in php. Whilst Eucalyptus provide a command line client, Amazon already uses a web browser to manage its S3 resources, but this is very limited in the authorisation and delegation services that it provides. Our proxyS3 provides a much richer set of capabilities via our defined security APIs, and in this way the user is able to provide finer grained control to the S3 resources by using a standard web browser.

The proxyS3 service has an account database (DB) of S3 keys (user authentication credentials) for all the registered accounts so that the proxy S3 application can make requests to S3 on the account holder's behalf. There is a many to many mapping between users and accounts. Users are identified by a set of identity attributes, similarly account holders. Thus one user may have a set of attributes which entitles him to own several different accounts (e.g. a professor from the University of Kent with PID abc, could be the owner of three S3 accounts: one for professors from the University of Kent, one for PID abc, and one for all professors). Likewise many different users may possess the attributes required to be the owner of any particular S3 account (e.g. all professors at the University of Kent can be the owners of a single S3 account). See the bottom screen shot of Figure 3 for the Account Chooser window which allows the user to choose which account he wishes to create or open. When a federated user wishes to create an account on the S3 system for the first time, the proxy S3 service can make an 'add user' request to the S3 service, acting in the admin role, in order to add a new user to the cloud S3 system. It can then download the S3 user's credentials from the cloud and store them in the DB. Whenever a federated user now wishes to access this S3 account, the proxyS3 service can extract the credentials from the DB and access the account on behalf of the user. This design hides the use of keys from the users and all the use of keys is performed inside the proxyS3 service. In order to maximise security, the DB should be stored separately from the proxyS3 module so that it can be separately protected, e.g. by encrypting the keys if necessary.

Whilst our intention was for the proxyS3 service to automatically register new users (i.e. accounts) with the S3 service, unfortunately the Eucalyptus API did not allow the entire process to be fully automated. Human administrator involvement is required in the registration process, so we had to mark new accounts as "pending" when the user asked for them to be created. They had to stay in the pending state until our administrator activated Eucalyptus's new user registration process. Once the account has been activated, it can then be opened by the user as described above.

All the security functionality of the proxyS3 service is carried out by the three security APIs. Users do not need new cloud provided credentials (keys) to access the proxyS3 service. Instead, they use their existing authentication credentials to login via a federated identity provider which is trusted by the proxyS3 service. This trusted IdP then sends the user's attributes to the proxyS3 service which allows it to determine the accounts this user is the owner of. In our demonstration proxyS3 service, we have configured the proxyS3 to trust our proxyIdP, rather than an actual IdP, since this provides greater flexibility and protocol support as described earlier. The proxyIdP is configured to trust a whole set of identity providers from the UK Access Management Federation, Google, Facebook, Twitter and OpenID. This was simply to ensure that a maximal set of users could try out our pilot service and experiment with it, since most people have a login account with at least one of these identity providers. We would not expect valuable cloud services to have such a lax trust policy in an operational environment!

Implementing the delegation API raised several challenges. The main problem we encountered is that neither the delegators nor the delegates are initially known to the delegation issuing service (DIS), so the LDAP directory is initially empty. We solved this problem by dynamically populating the LDAP directory at the time of delegation, with the Persistent Identifiers (PIDs) and IdPs of both users. The combination of IdP name and PID is globally unique throughout a federation, so no two users will ever have the same LDAP distinguished name built from these two components. Since the delegator is already logged in at the time of delegation, the proxyS3 service has access to his PID and IdP. Unfortunately, neither the proxyS3 service nor the delegator will know the PID of the delegate. They also may not know which IdP the delegate is registered with. So how is the delegator to refer to his chosen delegate? We solved this problem through a process known as "delegation by invitation". This splits the delegation process into two sub processes. In the first phase, delegation request, the delegator requests to delegate an attribute to a delegate. The top window of Figure 4 shows the screen for this, in which the delegator wishes to delegate the Family attribute to his Dad. Since the delegator is already known to the cloud service, his IdP/PID can be stored in the DIS's LDAP directory. The DIS now issues an invitation token for the delegator to give to the delegate. It stores a copy of this along with the delegated attributes and any delegation constraints. The invitation token is actually a randomly generated 128 bit secret number, so it is reasonably strong to brute force or guessing attacks. The cloud service converts this into a "secret" URL which is passed back to the delegator. This is shown in the bottom screen shot of Figure 4. The delegator now gives this URL to his chosen delegate, by any out of band means e.g. via SMS, email or memory stick. Once the delegate clicks on the URL, phase 2, delegation acceptance, begins. The cloud service now presents the delegate with the federated authentication screen e.g. Figure 2, which requires the user to choose her IdP and authenticate. Once authenticated, the delegate is identified by his PID and IdP, and can be added to the DIS's LDAP service. A new delegation attribute certificate can be created for her by the DIS, and stored in her LDAP entry. Whenever a user authenticates to the cloud service, it should ask the DIS for any attributes which have been delegated to this user, and add them to the identity attributes provided by the IdP. In this way the user can be given access to both the accounts that he or she owns, and the resources that have been delegated to him or her. So from now on the delegate may authenticate to the cloud service and access the delegated resources as often as he or she wishes, until either the delegation expires, or the delegator revokes the delegation, in which case the attribute certificate is removed from his/her LDAP entry.

Figure 4 The delegation screens.

Another issue we had to solve, is which attribute should be delegated to the delegate by the delegator. It would not be appropriate to allow the delegator to delegate one of his IdP provided attributes to anyone, since he has no authority to do this e.g. the first author of this paper should not be allowed by the cloud service to delegate his University of Kent professor attribute to anyone. The solution we conceived of, was to make the delegator an attribute authority for his own attributes, and to allow him to specify his own attribute values such as: friend, family, project X member etc. The attribute type is automatically created by the delegation issuing service based on the identity of the delegator, thereby ensuring global uniqueness of the delegated attribute type and value.

Achievements

Overall the project was very successful and achieved most of its original objectives as well as several new ones. Its main achievements were the specification of a set of three security APIs for cloud services and a pilot cloud storage service that demonstrated how to use them. These security APIs have been specified in natural language as well as in PHP. They have been implemented in PHP as open source code and are publicly available from the project's web site [21].

We have made the proxyS3 service into a permanent public demonstration, available at https://authz.tas3.kent.ac.uk/proxyS3 which anyone can log into and use if they already have an account at either: Google, Facebook, Twitter, the UK Access Management Federation, or any OpenID provider. This should cover the vast majority of Internet users. We distribute the open source code, documentation, published papers and several powerpoint presentations via the public pages of the cloud service. In this way customers can experiment with the service, and if they like it, use the service to download its code and documentation.

Of the original objectives that were not fully achieved, only two were not attempted: adding a trust and reputation service and auditing. This was simply to do with a lack of time in the original project. It would be possible, given a new project, to integrate these functionalities in an application independent way via some new APIs.

With respect to the partially achieved objectives, most of these are to do with using policies for fine grained privacy control. Whilst the policy infrastructure itself was integrated into the cloud security APIs, there was insufficient time to implement the front end GUIs that would allow users to set their privacy policies and update them. Whilst we have built GUIs that allow users to set their access control rules, based on the attributes of the requestor, this does not provide an interface for setting more sophisticated features such as retention periods, purpose of use, or conditions such as time constraints. In the TAS3 project we experimented with several different policy GUIs – a natural language GUI, described in [22], and a simple matrix GUI of roles vs. resources in which the user clicks the various cells to indicate which users (roles) are granted access to which resources. The natural language GUI provides more functionality than the implemented attribute based GUI, but is more difficult to use. The matrix GUI provides less functionality than the attribute based GUI since the matrix's roles and resources are statically configured, whilst the attribute based GUI dynamically changes the user's choices as more attribute values become known to the proxyS3 service. Consequently neither of the GUIs developed within the TAS3 project were used further.

Lessons learnt

User interfaces are extremely important as this is how a software product is finally judged by end users. However, intuitive, easy to use GUIs are very time consuming and resource intensive to produce. Research projects often do not have the time or resources to spend on this feature, as often there is little "research" in providing this functionality. It is primarily an engineering task. When we decided to make the Eucalyptus S3 service accessible via a standard web browser, we effectively had to build the entire S3 front end again as dynamic web pages served from an Apache server. This consumed a large amount of development effort from the short six month project, and, in retrospect, was probably not the best use of our very limited resources. Consequently, in a subsequent project (Sticky Policy Based Open Source Security APIs for the Cloud), in which we have added federated access to OpenStack, we decided to enhance the existing OpenStack command line interfaces, rather than produce new browser based interfaces. This was certainly less time consuming, but it was not without its problems. Specifically, all of the existing IdPs expect users to be using web browsers for authentication, and therefore do not provide command line interfaces for this. Furthermore their web based login pages are all formatted differently and present different content to their users. So it would have been impractical to try to parse these and extract the username and password fields from them. We solved this problem by invoking the web browser from the command line interface, which allows the user to login using the existing IdP browser based interface with which he is familiar, and then asking the user to continue with the command line interface, now that he has been authenticated.

Providing an appropriate GUI for setting security and privacy policies is also a very difficult task. Even providing an application dependent GUI, where many of the parameters of the policies are already known, and some can be fixed, it is still very difficult, due to the number of possibilities and combinations that still remain. Providing an application independent security policy GUI where none of the parameters are known or fixed beforehand, is orders of magnitude more difficult. We have had a previous EPSRC research project which researched this topic and which formed the basis of our current natural language GUI. This is capable of creating simple RBAC policies for any application. But this still is not sophisticated enough to cater for all the different policy constructs that may be needed (such as time constraints and arbitrary conditions), nor for all the different ways that users naturally use to specify the same thing. At the same time it is still more difficult to use the application independent constrained natural language GUI than a specially tailored and constrained application dependent GUI. We conclude that constructing application dependent policy GUIs are at the limit of our current computer science abilities, and that much more research is needed into building application independent policy management GUIs.

Further work is still required in a number of other areas. Ontologies of attribute types and identity provider classes are needed in order to simplify the security APIs and make them scalable to Internet scale. The API programmer will then be able to ask for classes of attributes from classes of identity provider rather than having to specifically list them all e.g. request a credit card attribute from a bank, rather than a Visa card attribute from HSBC or a MasterCard attribute from Barclays Bank.

Providing an implementation of the security APIs in PHP has provided a good initial proof of concept, but one language alone is not sufficient for all cloud developers to use since cloud applications may be developed in other programming languages such as Python, Java or C. Developing security APIs in other languages requires development effort rather than research effort.

The current PHP implementation works well and has been stress tested, but it may not be as scalable as some cloud applications require, and certainly some of the existing back end services that it uses, such as LDAP, the PERMIS delegation service, the PDP, and MySQL database may not be as elastic as many cloud applications require. To make the security APIs and their supporting services horizontally and massively scalable is a huge research and development task in itself and should not be underestimated.

Limitations and conclusions

The proxyS3 service is a proof of concept cloud application that shows how the application independent security APIs can be used by any cloud application to provide federated access, enhanced fine grained access controls, and delegation of authority. The security APIs use attribute based access controls, so that access to a cloud resource is based on a user's validated attributes from one or more trusted identity providers, instead of on a locally issued account ID. In this way users do not need to register for a new account with the cloud service before they can be granted access to it. Public access can be given to a cloud resource by simply not requiring the user to have any valid attributes. Delegation of authority is supported by the account holder becoming an attribute authority, and creating his own unique attributes.

The current project did not attempt to integrate several trust based services due to a lack of time. In particular, no audit services were introduced, and no reputation service was incorporated. Thus cloud users are not able to validate that the trusted services are behaving as expected, nor are they able to determine which cloud providers are the most trustworthy.

Whilst the authz API does support a call out to a PDP holding authorisation and privacy policies, we did not have time to implement the passing of sticky policies to the PDP via the cloud service and the browser interface. As explained above, this interface would have to be application specific, with many parameters being fixed by the application, so that users have an easy, but limited, set of policy choices to make.

In conclusion, the 6 month My Private Cloud project gave us good insights into how to provide a rich set of security services to cloud developers, through the production of a set of security APIs. Providing open source security APIs to cloud developers should both increase the security of the developed cloud applications whilst simultaneously reducing the development effort that is needed to do this. We believe that this work should continue and that further security features should be added via new security APIs, as well as additional implementations of the existing APIs be developed in further languages such as Python.

Endnotes

[1]More sophisticated PDPs may be able to return Not Applicable, Indeterminate and Break The Glass responses as well.

Abbreviations

ABAC, Attribute based access controls; ABE, Attribute based encryption; ACL, Access control list; API, Application programmable interface; AWS, Amazon web services; CVS, Credential validation service; DIS, Delegation issuing service; EPSRC, Engineering and physical sciences research council; FP7, Framework programme 7; GUI, Graphical user interface; IaaS, Infrastructure as a service; IdP, Identity provider; JISC, Joint information services committe; LDAP, Lightweight directory access protocol; PDP, Policy decision point; PEP, Policy enforcement point; PHP, PHP: Hypertext preprocessor; PID, Persistent identifier; RBAC, Role based access controls; S3, Simple storage service; SAML, Security assertions markup language; SOAP, Simple object access protocol; SMS, Short messaging service; SP, Service provider; TAS3, Trusted architecture for securely shared services; URL, Uniform resource locator; WAYF, Where are you from; WSC, Web services consumer/client; WSP, Web services provider

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

DWC – wrote the paper, intellectual and conceptual design. MC – Detailed design and implementation of proxyS3 system. KS – Implementation and design of proxyIdP, installation and user instructions. All authors read and approved the final manuscript.

Acknowledgements

This project has received funding from the UK JISC and EPSRC under grant ref. no. EP/1034181/1 (My Private Cloud) and the EC FP7 under grand agreement no. 216287 (Trusted Architecture for Securely Shared Services)

References

1. See for example: "Update: Over 450,000 emails and passwords allegedly stolen from Yahoo" at http://www.infoworld.com/d/security/over-450000-emails-and-passwords-allegedly-stolen-yahoo-197609 and "6 Million Passwords Stolen from LinkedIn" at http://www.techcessgroup.com/newsletter-content/item/111-6-million-passwords-stolen-from-linkedin.html (last accessed 16 Oct. 12)

2. Zak PJ, Knack S (1998) “Trust and growth”. Available at SSRN: http://ssrn.com/abstract = 136961 or http://dx.doi.org/10.2139/ssrn.136961 (last accessed 16 Oct. 12)

3. TAS3, see http://www.tas3.eu/. (last accessed 16 Oct. 12)

4. See “AWS and Eucalyptus” at http://www.eucalyptus.com/learn/amazon-aws-compatibility. (last accessed 7 Feb 2013)

5. See “AWS Customer Agreement” at http://aws.amazon.com/agreement/last accessed 15 October 2012)

6. Wang W, Li Z, Owens R, Bhargava B (2009) Secure and efficient access to outsourced data, *Proceedings of the 2009 ACM workshop on cloud computing security* (CCSW ’09). ACM, New York, NY, USA, pp 55–66

7. Mowbray M, Pearson S (2009) “A client-based privacy manager for cloud computing”, *Proceedings of the fourth international ICST conference on COMmunication system softWAre and middlewaRE* (COMSWARE ’09). ACM, New York, NY, USA, p 8, Article 5

8. Applegate LM, Montealegre R (1991) “Eastman Kodak organization: managing information systems through strategic alliances”. Harvard Business School Case 9-192-030, Boston, MA

9. See “Efficiency drive will grow government outsourcing and cloud usage, says Ovum” at http://www.computerworlduk.com/news/public-sector/3256210/efficiency-drive-will-grow-government-outsourcing-and-cloud-usage-says-ovum/ Last accessed 15 October 2012)

10. Palankar MR, Iamnitchi A, Ripeanu M, Garfinkel S (2008) “Amazon S3 for science grids: a viable solution?”, Proceedings of the 2008 international workshop on Data-aware distributed computing (DADC '08). ACM, New York, NY, USA, pp 55–64. doi:10.1145/1383519.1383526, http://doi.acm.org/10.1145/1383519.1383526

11. Echevarrıa V, Liebrock LM, Dongwan S (2010) “Permission Management System: Permission as a Service in Cloud Computing”, IEEE 34th Annual Computer Software and Applications Conference Workshop (COMPSACW). pp 371–375, 10.1109/COMPSACW.2010.71

12. Dongwan S, Akkan H (2010) “Domain-based virtualized resource management in cloud computing”, 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp 1–6

13. UKAMF, see http://www.ukfederation.org.uk/ (last accessed 15 Oct 2012)

14. Bob Morgan RL, Scott C, Steven C, Walter H, Ken K (2004) “Federated security: the shibboleth approach”. Educ Q 27:4

15. Chadwick DW (2008) “Dynamic delegation of authority in Web services”. In: Periorellis P (ed) “Securing Web services: practical usage of standards and specifications”. Newcastle University. Idea Group Inc, Newcastle, pp 111–137

16. Tuecke S, Welch V, Engert D, Pearlman L, Thompson M (2004) “Internet X.509 Public Key infrastructure (PKI) proxy certificate profile”. RFC3820, RFC3820, June 2004. Available from http://datatracker.ietf.org/doc/rfc3820/

17. Chadwick DW, Otenko S, Nguyen TA (2009) “Adding support to XACML for multi-domain user to user dynamic delegation of authority”. Int J Inf Secur 8(2):137–152

18. SimpleSAMLphp is available from http://simplesamlphp.org (last accessed 7 February 2013)

19. Chadwick DW, Inman G, Siu KWS, Ferdous MS (2011) “Leveraging social networks to gain access to organisational resources”. To appear in Proc. ACM DIM’11, Chicago, Illinois, USA, pp 43–52, http://dx.doi.org/10.1145/2046642.2046653

20. Chadwick DW, Casenove M (2011) “Security APIs for My private cloud - granting access to anyone, from anywhere at any time”. IEEE CloudCom 2011, Athens Greece, pp 792–798. doi:10.1109/CloudCom.2011.122. ISBN 978-1-4673-0090-2

21. The online demonstration “Welcome to the Cloud Storage Service” is available at https://authz.tas3.kent.ac.uk/proxyS3/. The online documentation and open source code is available by clicking the Public Resources link. (last accessed 15 Oct 12)

22. Shi L, Chadwick DW (2011) “A controlled natural language interface for authoring access control policies”. Proceedings of the 2011 ACM Symposium on Applied Computing, TaiChung, Taiwan, pp 1524–1530, Available from http://portal.acm.org/citation.cfm?id = 1982510#

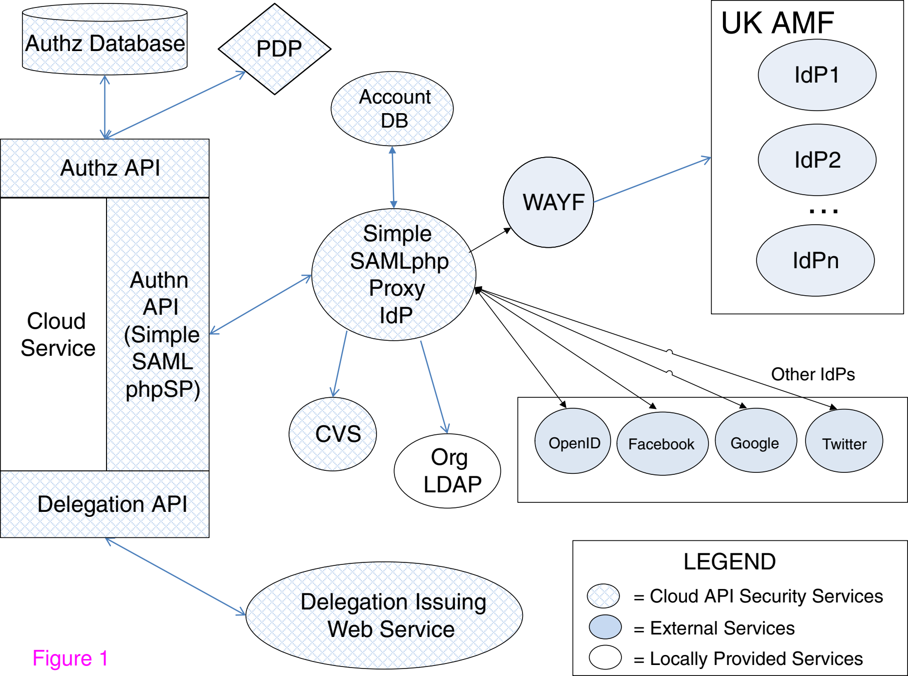
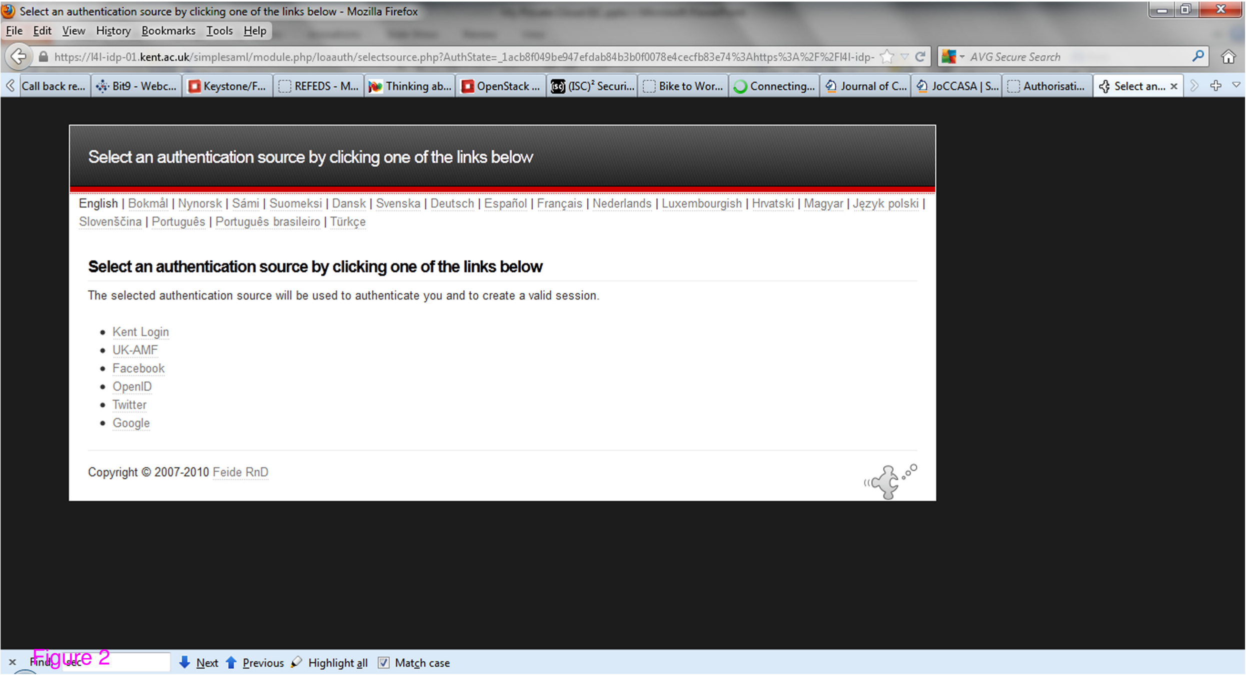

Authz Database
PDP
Account DB
UK AMF
IdP1
IdP2
. . .
IdPn
WAYF
Authz API
Cloud Service
Authn API (Simple SAML phpSP)
Simple SAMLphp Proxy IdP
CVS
Org LDAP
Other IdPs
OpenID
Facebook
Google
Twitter
Delegation API
Delegation Issuing Web Service
LEGEND
= Cloud API Security Services
= External Services
= Locally Provided Services

Figure 1



https://l4l-idp-01.kent.ac.uk/simplesaml/module.php/loaauth/selectsource.php?AuthState=_1acb8f049be947efdab84b3b0f0078e4cecfb83e74%3Ahttps%3A%2F%2Fl4l-idp-

Select an authentication source by clicking one of the links below

English | Bokmål | Nynorsk | Sámi | Suomeksi | Dansk | Svenska | Deutsch | Español | Français | Nederlands | Luxembourgish | Hrvatski | Magyar | Język polski | Slovenščina | Português | Português brasileiro | Türkçe

Select an authentication source by clicking one of the links below

The selected authentication source will be used to authenticate you and to create a valid session.

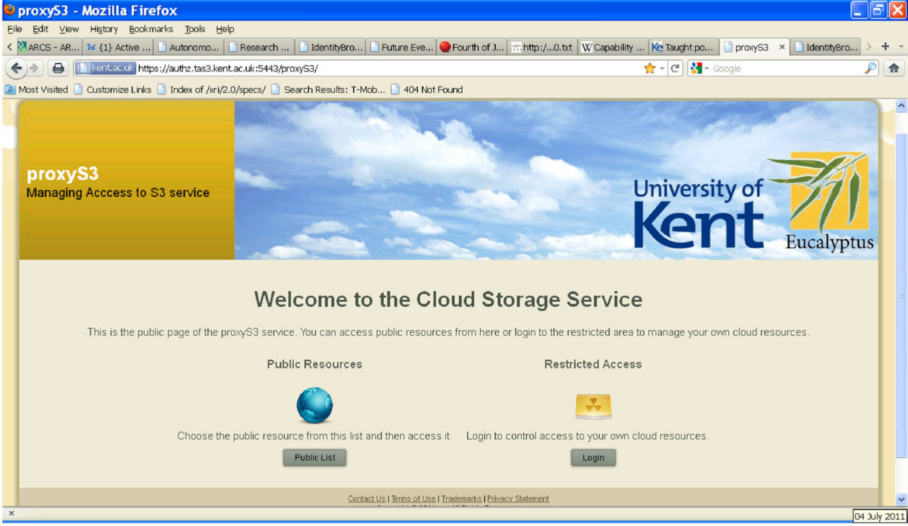
- Kent Login
- UK-AMF
- Facebook
- OpenID
- Twitter
- Google

Copyright © 2007-2010 Feide RnD

Find: sec

☑ Match case

Figure 2


proxyS3 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
ARCS - AR... {1} Active ... Autonomo... Research ... IdentityBro... Future Eve... Fourth of J... http:/...0.txt Capability ... Taught po... proxyS3 IdentityBro...
kent.ac.uk https://authz.tas3.kent.ac.uk:5443/proxyS3/
Google
Most Visited Customize Links Index of /xri/2.0/specs/ Search Results: T-Mob... 404 Not Found
proxyS3
Managing Acccess to S3 service
University of Kent
Eucalyptus
Public List
Login
Contact Us | Terms of Use | Trademarks | Privacy Statement
04 July 2011

Welcome to the Cloud Storage Service

This is the public page of the proxyS3 service. You can access public resources from here or login to the restricted area to manage your own cloud resources.

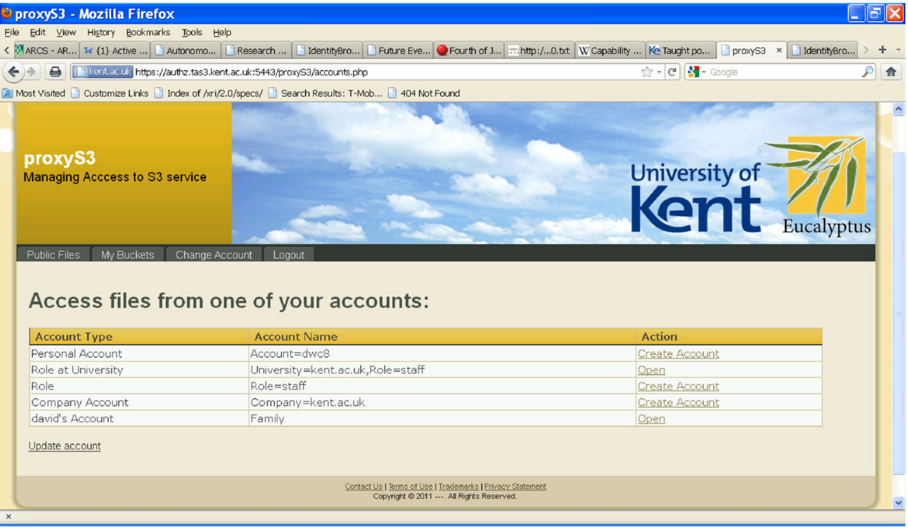
Public Resources

Choose the public resource from this list and then access it.

Restricted Access

Login to control access to your own cloud resources.

Top. The Welcome Screen

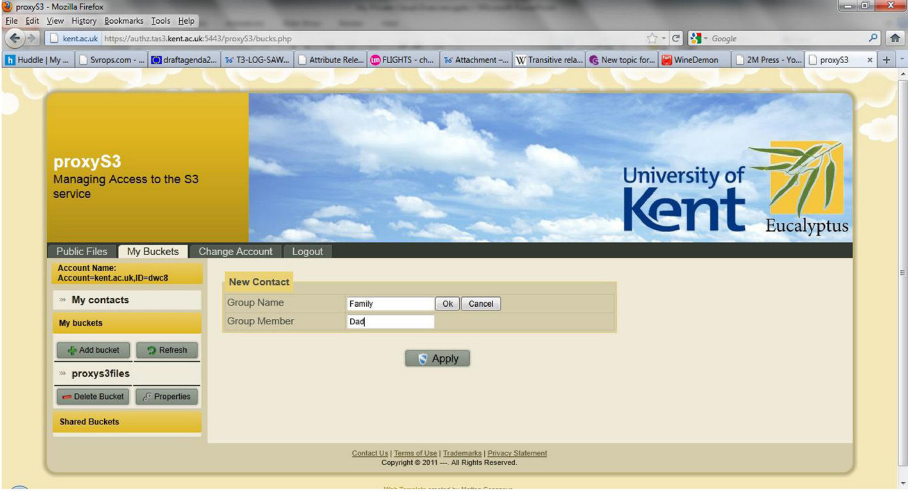

proxyS3 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
ARCS - AR... {1} Active ... Autonomo... Research ... IdentityBro... Future Eve... Fourth of J... http:/...0.txt Capability ... Taught po... proxyS3 IdentityBro...
kent.ac.uk https://authz.tas3.kent.ac.uk:5443/proxyS3/accounts.php
Google
Most Visited Customize Links Index of /xri/2.0/specs/ Search Results: T-Mob... 404 Not Found
proxyS3
Managing Acccess to S3 service
University of Kent
Eucalyptus
Public Files
My Buckets
Change Account
Logout
Contact Us | Terms of Use | Trademarks | Privacy Statement
Copyright © 2011 ---. All Rights Reserved.

Access files from one of your accounts:

Account Type	Account Name	Action
Personal Account	Account=dwc8	Create Account
Role at University	University=kent.ac.uk,Role=staff	Open
Role	Role=staff	Create Account
Company Account	Company=kent.ac.uk	Create Account
david's Account	Family	Open

Update account

Figure 3 Bottom. The Account Chooser Screen.



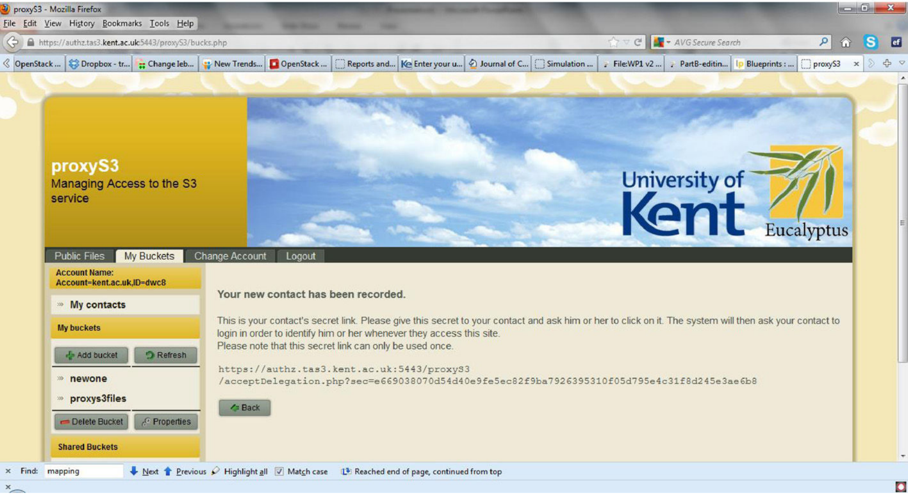
Account Name:
Account=kent.ac.uk,ID=dwc8

New Contact

Group Name	Family
Group Member	Dad

Copyright © 2011 ---. All Rights Reserved.

Top. Creating a New Delegate



Account Name:
Account=kent.ac.uk,ID=dwc8

Your new contact has been recorded.

This is your contact's secret link. Please give this secret to your contact and ask him or her to click on it. The system will then ask your contact to login in order to identify him or her whenever they access this site.
Please note that this secret link can only be used once.

```
https://authz.tas3.kent.ac.uk:5443/proxyS3
/acceptDelegation.php?sec=e669038070d54d40e9fe5ec82f9ba7926395310f05d795e4c31f8d245e3ae6b8
```

Figure 4 Bottom. Getting the Secret URL