Trusted Virtual Infrastructure Bootstrapping for On Demand Services.

Abstract— **As cloud computing continues to gain traction, a great deal of effort is being expended in researching the most effective ways to build and manage secure and trustworthy clouds. Providing consistent security services in on-demand provisioned Cloud infrastructure services is of primary importance due to the multi-tenant and potentially multi-provider nature of Cloud Infrastructure. Cloud security infrastructure should address two aspects of the IaaS operation and dynamic security services provisioning: (1) provide security infrastructure for secure Cloud IaaS operation; (2) provisioning dynamic security services. Although the first task is a traditional task in security engineering, dynamic provisioning of managed security services in virtualized environment remains a problem and requires additional research. Entire frameworks have been proposed and demonstrated but although successful, there is a tendency to see such solutions as integrated 'all in one' infrastructures. This paper describes a light-weight mechanism and protocol for building trust between two machines that takes advantage of the Trusted Platform Module (TPM) to handle a key exchange and remote trusted deployment of a bootstrapping tool (referred to as the Bootstrapping Initiator (BI)). Once deployed, the BI can execute any arbitrary software required which could be (but is not limited to) solutions for advanced architecture management such as the Dynamic Access Control Infrastructure (DACI). The proposed solution provides a light-weight layer of trust backed by a TPM that additional systems can build upon as required by the individual use case without the requirement for a specific management or security infrastructure to be deployed along with it.**

Keywords—Cloud Security, Trusted Computing, Bootstrapping, Deployment.

I. INTRODUCTION

Cloud computing technologies [1, 2] are emerging as infrastructure services for provisioning computing and storage resources on-demand in a simple and uniform way. However, there is no well-defined architectural model for the Cloud Infrastructure a Service (IaaS) provisioning model despite its wide use among big Cloud providers such as Amazon, RackSpace, Google, and others. Recent research based on the first wave of Cloud Computing implementation have revealed a number of security issues both in actual services organizational, operational and business models [3, 4, 5]. The current Cloud' security model is based on the assumption that the user/customer should trust the provider. This is governed by the general Service Level Agreement (SLA) that defines mutual provider and user expectations and obligations for the whole provisioned service but doesn't allow dynamic Quality of Services (QoS) management for potentially changing resource availability due to changing resource demand and utilisation in the typically multi-user Cloud environment.

Although Cloud providers are investing significant resources into making their own infrastructure secure and complying to existing security management standards (e.g. Amazon Cloud recently achieved PCI compliance certification [6] and announced providing special services for governmental organisations [7], Microsoft Azure claims ISO27001 compliance [8]), still the overall security of Cloud based infrastructures and services will depend on two other factors: security services implementation in user applications and binding between virtualised services and Cloud based virtualisation platforms, that should also ensure protection against malicious users and risks related to possible Denial of Service (DoS) attacks.

Practical Cloud usage within one provider infrastructure creates illusion of unlimited availability, "elasticity" and "perfect" security (as claimed by the providers themselves), but in practice this is related only to limited range of services and with limited manageability. Currently implemented and offered security services are based on VPNs and provide only simple access control services based on users access over SSH channel. Recent improvements in GoogleApps allow SAML based Single Sign-On (SSO) [9] to connect/integrate Cloud based services and customer legacy access control infrastructure. More advanced security services and fine grained access control cannot be achieved without deeper integration with the Cloud virtualisation platform and incumbent security services, what in its own turn can be achieved with an open and well defined Cloud IaaS platform architecture to allow transparent interoperability and integration of heterogeneous multi-provider Cloud infrastructure services.

Current development of Cloud technologies demonstrate movement to developing inter-Cloud models, architectures and integration tools that could allow integrating Cloud based infrastructure services into existing enterprise and campus infrastructures, on one hand, and provide a common/interoperable environment for moving existing infrastructures and infrastructure services to a virtualised Cloud environment. More complex and community oriented use of Cloud

infrastructure services will require developing new service provisioning and security models that could allow creating complex project and group oriented infrastructures provisioned on-demand and across multiple providers.

Building a secure virtual infrastructure in such an environment is challenging. It is a requirement that not only can a service be trusted but also that the operating system and physical machine can also be trusted. When deploying infrastructure on-demand, the underlying hardware is generally accepted as trustworthy. However, if the trustworthiness of the underlying physical machine could be in question, then the trustworthiness of the entire infrastructure could be brought into question as well.

Trust is built in layers, with each layer only as trustworthy as the layer beneath it. Generally when a machine is provided with an operating system, this is the first layer of trust. It is assumed that the operating system and the hardware are not compromised. However while this assumption could be considered acceptable in a hosting facility under direct control, it becomes severely stretched when machines are provided at remote locations by third parties. The situation is complicated further when multiple machines are involved. A cluster requiring a number of machines needs to ensure that each machine is trustworthy. If such a system needs to span data centers or even countries, the level of implied trust drops drastically. In these cases it would be beneficial to ensure that the remote machine that has been provided is in fact the machine that it is supposed to be. Specifically it would be desirable to be able to confirm the machine's identity and further to ensure that it is the state that it is supposed to be in.

Trust has long been associated with authentication [10], where one is tightly related to the other. Authenticating a machine and verifying its state can ensure that the machine has not been tampered with. However incorporating such a protocol directly into existing frameworks or software would increase their complexity and would likely create a number of disparate and incompatible systems. A generic bootstrapping protocol that can be adopted by frameworks (such as DACI proposed by the authors) is required. Such a protocol would have two key requirements. First it must be able to authenticate the remote machine and verify its trustworthiness. Second, it must provide a mechanism for transferring and executing the initializing the framework.

This paper proposes a new generic bootstrapping protocol called the Dynamic Infrastructure Trust Bootstrapping Protocol (DITBP). This includes supporting mechanisms and infrastructure that takes advantage of the TCG Reference Architecture (TCGRA) and Trusted Platform Module (TPM). The TPM is used to provide a root of trust that extends from the physical hardware itself. The TPM is used to generate a key pair in hardware where the private key is never revealed (the key pair is non-migratable). The key is only available when the machine is in a known and trusted state. The key pair is used to authenticate the machine and to decrypt the payload which is then executed to bootstrap the rest of the virtual infrastructure.

The paper is organized as follows. Section II analyzes two typical usecases that require trusted bootstrapping. Section III refers to the core features provided by the TGC Reference Architecture and analyzes current limitations. Section IV provides an overview of a model for trusted bootstrapping and Section V analyzes the individual components of such a model. Section VI analyzes the bootstrapping process and Section VII provides a suggested implementation. Section VIII provides a summary and a direction for future work.

II. USE CASE

Collaborative infrastructure where machines are provided by multiple providers are becoming more popular. They are especially useful when working in geographically disparate teams. Often datasets can span many hundreds of gigabytes and are potentially being updated in real-time. Social media feeds such as Twitter where hundreds of updates are received per second, produce datasets that are large, immobile and constantly growing. For a remote researcher to work with this data, they must have a machine in close proximity to that data. Being able to confirm the identity of the machine and that it is running in a trusted state allows for a higher level of confidence when working on sensitive data.

In the financial sector there is often the need to work with remote machines that are not under the direct control of the customer. Many financial exchanges offer high speed price feeds that clients need to respond to within microseconds in order to be competitive. Most exchanges are 'first come first served' and therefore response time is vital to a company's potential success. Even high quality fiber networks introduce latencies far above this threshold which makes trading from a remote location impractical. To counter this problem, many exchanges and brokers offer some form of hosting or colocation. This allows a company to have a presence in close proximity to the market gateway and thus keep latency to a minimum.

However, not all facilities offer colocation and in many cases, a prebuilt machine may be provided. Where colocation is allowed and the business provides their own machines, there is no guarantee that those machines have not been tampered with. Due to the nature of the business, these 'front end' machines are required to contain sensitive business information such as strategies in order to carry out their function. This in turn means that the value of the

data being placed at risk is quite considerable. While companies do this, it is because there is no alternative rather than being willing to trust the remote machines.

These machines tend to exist in a tightly controlled networks. It is likely that the machines would not have Internet access and will be severely segregated (such as with firewalls). This limits the applicability of many existing technologies and would effectively block the use of Direct Anonymous Attestation (DAA) [11] (although infrastructure to support this does not currently exist in any case). In addition, these machines should be as bare metal as possible. Financial applications are latency sensitive and require as little support software running as possible. As a consequence it is not uncommon for multiple machines to work together, often deployed at different times. When new machines are added, it is critical that their identity be verified before the customer transfers proprietary content to it.

This use case can be extended to any scenario where it is required that a remote machine be trusted but where it is also required that the infrastructure to allow this is as light weight and unobtrusive as possible. Thus an architecture that allows machines to develop a level of trust between them (which allows for trusted bootstrapping) but does not provide any additional layers (i.e. allowing the business to choose which additional infrastructure or software to run) is required. Such a solution would not replace any of the existing models, rather it would provide a low-level layer that other infrastructure solutions can build upon and extend. Such solutions can use this platform to provide their own services; for example Dynamic Access Control Infrastructure (DACI) [12].

III. TCG Reference Architecture (TCGRA)

One of the key components in the Trusted Computing Group Reference Architecture (TCGRA) is the Trusted Platform Module (TPM). The TPM is a physical device that provides cryptographic functionality in hardware. Two of the key features provided by the TPM are the generation of encryption keys and the ability to measure the current state of a given system.

One of the critical features provided by a TPM is the ability to create encryption key pairs in hardware. A TPM can create both migratable and non-migratable key pairs. If a non-migratable key pair is generated, the private key may never leave the TPM. Therefore only the one machine (or more specifically the particular TPM) is able to use the key pair for encryption and decryption operations. The TPM does not provide acceleration facilities and has limited processing power. It is generally only used for key pair creation and storage.

The Platform Configuration Registers (PCRs) are a set of registers inside the TPM that store SHA1 hashes. The TPM measures data such as the Master Boot Record (MBR), boot loader and kernel. The PCRs use a ratcheting mechanism. When a file is measured, the SHA1 hash is computed. This is concatenated with the existing value in the PCR. The TPM calculates the SHA1 of the combined value and stores the result in the PCR. The effect of this mechanism is that each new state depends on the previous state. If any of the previous states are different, all of the hashes generated from that point on will be different as well. This allows a root of trust to be created from system boot to the operating system itself.

When a non-migratable key pair is created, it can be bound to the values stored in the PCRs. Which PCRs a key is bound to can be configured when the key itself is generated. A bound key can only be accessed when the PCRs are in the state that they were in when the key was created. This in effect allows a key pair to be only available to the system if that system is in a known and trusted state. For example, a machine that has been tampered with during the boot process would generate different PCR hash values. As the PCR values are different, the TPM will not be able to access the private key and so will not be able to conduct encryption / decryption operations and hence will not be able to authenticate using the key pair. Binding a non-migratable key pair to a known system state provides a method for authenticating a particular machine and ensuring it is in a trusted state.

Direct Anonymous Attestation (DAA) was added in the TPM 1.2 specification. It provides a mechanism where a TPM can verify its authenticity without giving away its identity. However at present, the infrastructure to actually support DAA does not exist and the most popular Open Source implementation of the TCS stack (TrouSers) doesn't support it either. DAA is also a fairly complicated protocol involving numerous different entities. It would theoretically require an Internet connection which in some restricted networks might not be available. The key benefit of DAA is that a TPM can prove that it is real without disclosing its identity but in a cloud environment, the main purpose of using a TPM is to identify a particular machine. As the infrastructure is not currently readily available and the main benefit is not useful in this context, a simpler system that does not depend on the Internet or external entities, would be more appropriate.

IV. General Security Scenario

It is generally accepted that a trusted root is required in order for any trust relationships to be built. That is, for a client to authenticate a server, the client must already trust something (such as a Certificate Authority) in order for it to determine whether a server can be trusted (when it presents a signed certificate). When the provisioned

machine is provided by the end user, they are in a position to verify the identity of the machine directly. However as provisioned machines both virtual and physical are most likely provided by a third party who controls and is ultimately responsible for the service, they are an ideal candidate to provide the authentication service.

The Domain Authentication Server (DAS) manages two distinct services. First, the service needs to handle the registration and initial authentication of newly provisioned machines. Second, it needs to handle authentication requests from client machines that wish to to authenticate and then bootstrap a given machine. Two key pieces of information need to be held by the authentication service for each machine that it provides authentication for. The DAS requires two key pieces of information. First, it stores the public SSL certificate used for authenticating the machine during the initial handshake and the machine's public key which is used to encrypt the payload. Second, the DAS holds configuration data that describes the state of the machine such as the state of the Platform Configuration Registers (PCRs) which describe the machine's trusted state.

The provisioned machine on initial boot generates an SSL certificate and a TPM backed non-migratable key pair. As the private key can never leave the TPM, data encrypted using this public key can only be decrypted on this specific machine. Another machine could potentially use the SSL certificate, but without the TPM, it would not be able to decrypt the payload and authenticate itself successfully. As part of the registration process, the machine will transfer the SSL certificate and the public key to the DAS. The machine then waits for a bootstrapping request from the DAS.

The client machine initiates the bootstrapping procedure by making an authentication request to the DAS. It must provide a unique identifier such as an IP address, hostname or service identifier (this is liable to be implementation and or context specific). The DAS will then send information on that machine to the client. This is the same information provided to the DAS when the provisioned machine initially registered. The client performs a similar key generation process to that of the provisioned machine. The client encrypts its public key, certificate and a nonce value using the target machine's public key and submits them to the DAS. The DAS then sends a bootstrapping request to the provisioned machine which decrypts and verifies the request. The certificate will be used to verify the client's identity during the initial handshake and the key will be used to authenticate the payload's signature. Once configured, the provisioned machine informs the DAS that it is ready for the bootstrapping process to commence. The DAS then informs the client, that it may proceed.

At this stage in the process, the client machine will only connect to a machine that has the certificate provided by the DAS. The provisioned machine will only accept a connection from a machine that identifies itself using the client certificate it received from the DAS. Once the machines have established an authenticated communication channel, the payload can be encrypted with the remote machine's public key and sent to the remote machine for execution.

As a trust anchor, the DAS is attesting that the public key provided to the client is a non-migratable, bound key pair. Therefore as only the machine with the correct TPM can decrypt the payload, the client can be assured that the target machine is in a known and trusted state.

V. Bootstrapping Infrastructure Components

DITBP uses TCP to communicate between nodes and TLS to provide end point security. Where a client connects to the target machine (the one to be bootstrapped), both client and server mutually authenticate each other based on the public keys provided to them via the DAS. Communication with the DAS also occurs over TLS, where the cloud vendor has either provided the certificate to the cloud customer or a third party trusted Certificate Authority is used. The DAS may or may not require mutual authentication via TLS.

Message exchange is event driven. Each message or payload is tied to a particular event. A node may send or receive events in a request / response design pattern. When a client sends a bootstrapping request to the DAS, it should wait for a response from the DAS before sending any further messages to it. Communicating between nodes follows the same pattern, however a client can communicate with any number of nodes simultaneously. Messages are sent `point to point', that is the architecture does not route messages across multiple nodes. The actual message format depends on the specific implementation.
A WebSockets based architecture for example, provides authenticated message channels. This would allow the nodes to communicate with the DAS or each other in a secure, event-driven, message based architecture. Such a solution scales well, is able to function via HTTP proxies and is generally firewall friendly.

There are four key components to the bootstrapping process. The process enables a client machine to authenticate a remote machine, determine that the machine is in a trusted state and begin the bootstrapping process.

The **Domain Authentication Server** (DAS) provides a trusted root for the third party's domain. It contains relevant information such as the public key for that machine's non-migratable key pair. The tickets issued by the DAS allow the client to encrypt data specifically for a particular machine in a specified state. A new non-migratable key pair should be created each time the machine is redeployed and therefore it is not possible to create a 'one off' list containing the key information. Further, by requiring the client to request the ticket directly from the DAS, the freshness and validity of the key data is increased.

The **Bootstrap Initiator** (BI) is the application that is transferred to the remote machine in order to confirm the machine's status before any infrastructure or software is deployed. The BI is responsible for Stage 2 of the bootstrapping process and begins the actual deployment of the core virtual infrastructure.

The **Bootstrap Requester** (BREQ) is a client application that runs on the machine responsible for provisioning remote infrastructure. It communicates with its counterpart on the remote machine and handles Stage 1 of the boot strapping process. This involves four key parts. First, the BREQ authenticates the remote end-point and authenticates itself. Second, the BREQ creates the BI payload (a combination of the BI application, key pairs and other necessary files) by compressing the bundle and encrypting it with the remote machine's public key. Third, the BREQ manages the transfer of the payload to the remote machine for execution. Lastly, it maintains an initial communication channel used by the BI to send any necessary deployment information back to the client.

The **Bootstrap Responder** (BRES) is the counterpart server application. It is responsible for authenticating the machine to a remote client and verifying that the client is authorized to bootstrap the machine. Once each end point has been authenticated, the BRES will receive, decrypt and decompress the payload sent by the client. Once done, the BI application is executed.

VI. Bootstrapping Process

The BREQ application will connect to the DAS and request authentication data for the remote machine that the infrastructure is to be deployed on. This contains the public key and certificate for the machine as well as other relevant data such as expected PCR configuration. BREQ will then create its own set of keys and certificates, and initiates a bootstrap request to the DAS. The DAS initiates a bootstrap request to the BRES on the target machine which contains the key pairs generated by the BREQ. BRES uses the keys to configure itself for the bootstrapping process. BRES signals the DAS that it is ready and the DAS then signals the BREQ.

The BREQ will then connect to the remote server and both BREQ and BRES will mutually authenticate each other. BREQ then prepares the BI payload. This consists of at least the BI application and some form of unique identifier such as a key or nonce. The payload is then archived, compressed and encrypted using the BRES's public key. Once the payload has been generated, BREQ will transfer the file to the remote server for deployment.

After receiving the payload, BRES will decrypt the file with using the TPM non-migratable private key. This will allow the BI application and its support files to be decompressed and extracted. BRES will report a success message to BREQ and will execute the BI application.

After executing the BI can set up the machine and run any additional tests that it requires. At this stage the trusted nature of the machine has already been established by the fact that the remote machine could decrypt the payload. However once running the BI can execute additional tests such as measuring other deployment files, verifying the network environment and so forth.

Once the BI application is satisfied with the state of the machine, it can communicate with the client to report that the system is ready. The BI application then downloads the infrastructure payload (the BI is a required component of DITBP but its implementation is infrastructure specific). It then determines the authenticity of the infrastructure received before the infrastructure is configured and deployed.

After all the deployment files have been transferred to the remote machine and have been verified and authenticated, the BI will execute the infrastructure framework and control will then pass to the framework. At this stage, the client's infrastructure management system should be able to communicate directly with the infrastructure now running on the remote server. The infrastructure can then shutdown the BI and BRES instances as required.

VII. Implementation Suggestions

The NodeJS and SocketIO frameworks provide a secure, event driven message passing architecture. Predominantly used in web browsers to allow real-time 'push' updates, SocketIO can operate as both a client and a server when running on NodeJS. As SSL/TLS encryption can be easily used, (HTTPS is the most common method for communicating via SocketIO), channel authentication and data security are available as standard. By building the prototype implementation on top of these mature libraries,

the prototype can focus on the messages being passed and the general bootstrapping protocol without needing to concern itself with low-level message handling infrastructure.

NodeJS has bindings for NaCL (pronounced salt) which provides a wide range of cryptographic functions. However at present there is no native binding for TPM functionality. For research and testing purposes, NodeJS can create the equivalent using traditional software based methods.

While NodeJS makes an ideal candidate for prototyping the implementation, it would likely have too many dependencies to be deployed in production for either the BREQ or the BRES. It is anticipated that the DITBP will be integrated into the GAAA framework and as such a simpler implementation in a system programming language such as C or C++ might be more appropriate.

A. Yin (BREQ) and Yang (BRES)

Yin and Yang are discussed together as their implementations are similar. It is anticipated that rather than two separate applications both Yin and Yang will be implemented as a single application that runs in either BREQ or BRES mode. The key generation and authentication is very similar regardless of whether the application is functioning as a client or a server.

Using the framework discussed previously, Yin and Yang can be built by focusing on message-based events and their payload. The exchange of data and the protocol are straight forward and synchronous at this stage. However in future, the protocol might add additional features that require more flexibility from the protocol. Following a message driven format allows for easy extension and rapid prototyping.

B. Vanguard - Bootstrap Initiator

The BI can be implemented in numerous ways. Vanguard was prototyped in Python. As long as the target machine can execute the application, there is considerable flexibility in the form that the application can take. Vanguard is a military term denoting a unit that travels ahead of the main force in order to determine whether or not the way ahead is safe for passage. In this case the application is executed on the remote machine, downloads a sample payload from a secure site and executes it. The functionality of the BI is inherently application and context specific, although it is likely that a particular framework will use only one BI application.

C. KeyStone - DAS Server

The DAS server can also be implemented using NodeJS. As it will run on an independent machine and does not need to be integrated into a framework, the design requirements for this component allow for some flexibility. Using SocketIO for the prototype allows KeyStone to easily interact with both Yin and Yang over the same transport mechanism. Storing and managing the key and configuration data are implementation specific and depends on the goals of the infrastructure vendor.

D. Integrating with the Common Security Services Interfaces (CSSI) framework

For trusted bootstrapping to prove valuable, it is vital that the security context is kept consistent between the physical and virtual infrastructures. In order to do this, DITBP needs to be integrated with the bootstrapped framework (such as DACI). However this will require special mechanisms to be developed to provide an interface between these two layers.

The GAAA-TK (developed by the authors) has a rich set of functionality that could be extended to support the use of Dynamic Infrastructure Bootstrapping (DIBP). CSSI (also proposed by the authors) currently includes authentication, authorization, session and security data as part of the Security Context (SC). This would need to be extended to include the additional trusted bootstrapping information.

VIII. SUMMARY AND FUTURE WORK

This paper presents the ongoing research on developing a trusted bootstrapping protocol for dynamically provisioned infrastructure to support high confidence computing in modern distributed computing environments.

The paper analyzes the use case of sensitive computing requirements on disparate infrastructure and identifies required protocols and mechanisms to allow greater confidence using remote machines. This includes both bootstrapping and preparing a remote system for use and generic processing on such a system where sensitive data or intellectual property might be exposed.

The paper proposes the Dynamic Infrastructure Trusted Bootstrapping Protocol (DITBP) and suggests a generic implementation that will provide a trusted bootstrapping layer for other frameworks (such as DACI) to leverage as a trust anchor.

The paper refers to the DACI framework which provides a general implementation for dynamically provisioned access control infrastructure as well as the GAAA Toolkit library that provides security context management.

The authors believe that concepts proposed in this paper will provide a good basis for further discussion among researchers about defining architectural models for dynamically provisioned virtualized security services as part of the general on-demand infrastructure services provisioning.

References

[1] NIST SP 800-145, "A NIST definition of cloud computing",[online] Available: http://csrc.nist.gov/publications/drafts/ 800-145/Draft-SP-800-145_cloud-definition.pdf

[2] GFD.150 Using Clouds to Provide Grids Higher-Levels of Abstraction and Explicit Support for Usage Modes. [Online]. http://www.ogf.org/documents/GFD.150.pdf

[3] Security Guidance for Critical Areas of Focus in Cloud Computing
V2.1. Cloud Security Alliance, December 2009. http://www.cloudsecurityalliance.org/csaguide.pdf

[4] Cloud Computing: Benefits, risks and recommendations for information security, Editors Daniele Catteddu, Giles Hogben, November 2009. http://www.enisa.europa.eu/ act/rm/files/deliverables/cloud-computing-risk-assessment

[5] Securing the Cloud: Designing Security for a New Age, Dec. 10,
2009. [Online] http://i.zdnet.com/whitepapers/ eflorida_ Securing_Cloud_Designing_Security_New_ Age.pdf

[6] Amazon AWS Security Center. Certification and Accreditation. [Online] http://aws.amazon.com/security/#certifications

[7] Amazon Boosts Web Services Security for Government Agencies.
PCWorld Business Center. April 17, 2011. [Online] http://www.pcworld.com/businesscenter/article/238276/amazon_boos ts_web_services_security_for_government_agencies.html

[8] 8. Kaufman, C., R. Venkatapathy. Windows Azure Security Overview. [Online] http://download.microsoft.com/download/6/0/2/6028B1AE-4AEE- 46CE-9187-641DA97FC1EE/Windows%20Azure%20 Security%20 Overview%20v1.01.pdf

[9] SAML Single Sign-On (SSO) Service for Google Apps. [Online] http://code.google.com/googleapps/domain/sso/saml_reference_imple mentation.html

[10] R. Yahalom, B. Klein, and T. Beth, "Trust relationships in secure systems-a dis- tributed authentication perspective," in Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on. IEEE, 1993, pp. 150–164.

[11] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation,"
. . . of the 11th ACM conference on . . . , 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1030083.1030103

[12] Y. Demchenko, C. Ngo, and C. de Laat, "Access control infrastructure for on- demand provisioned virtualised infrastructure services," in Collaboration Technologies and Systems (CTS), 2011 International Conference on. IEEE, 2011, pp. 466–475.