Cross-realm Kerberos implementations

Esan Wit
Esan.Wit at os3.nl

Mick Pouw
Mick.Pouw at os3.nl

Supervisors:
Michiel Leenaars
Rick van Rein

July 11, 2014

Acknowledgements

We would like to acknowledge Michiel Leenaars and Rick van Rein for their support and feedback during the course of this project. Offering clear insights and helpful commentary during this project to keep us focused and moving forward.

Abstract

When someone wants to use a service, a person has to provide credentials. These credentials have to be remembered by the user and also needs to be filled in everywhere. This is where Kerberos comes in. With the help of Kerberos its single sign-on possibilities and cross-realm capabilities, this could all be taken care of for the user. This would require several Kerberos KDCs configured with trust relationships. Due to the nature of Kerberos and its philosphy of shared keys this imposes a heavy administrative burden.

We look at four major implementations, namely Heimdal, MIT Kerberos 5, GNU Shishi and Active Directory. This paper describes the research that has been conducted to see how well these four implementations of Kerberos work together and highlight several points of interest when using these implementations in a mixed environment. Furthermore we discuss several focus points that play an important role in cross-realm behaviour and techniques that may assist in accomplishing dynamic configuration and easing the administrative complexity.

Contents

1 Introduction

Kerberos 5 is a very popular authentication protocol for internal networks. It is used by software like Samba and Active Directory. Kerberos was used to facilitate authentication in untrusted networks. Compared to current technologies it is comparable to the use of OAuth[1] on the internet when used only for authentication.

Kerberos can be used for cross-realm authentication in predefined configurations. This project looks into compatibility issues when attempting dynamic cross-realm configurations. It should be noted that as of this moment there is no proper support for dynamic cross-realm configuration. As such the project aims are to figure out the interoprability options between various implementations and the requirements for a dynamic cross-realm configuration.

Four implementations of Kerberos will be analysed, namely: MIT Kerberos 5[2], Heimdal[3], Active Directory[4] and GNU Shishi[5]. As mentioned this research will focus on the cross compatibility of these implementations and how to identify other realms and use these for cross-realm authentication.

The future goal is to pave way for a system which can be used to allow users the use of a variety of identity providers chosen by the user to authenticate themselves on services offered on the internet. Users would be able to choose an identity provider with conditions they themselves approve of. This is to avoid dependence on commercial companies such as Facebook, Google, Twitter, etc. as identity providers (often OAuth).

This report will detail some of the relevant research already performed after which the scope of this project will be explained further. Then some background information is included for readers not familiar with the basics of Kerberos. Afterwards the approach for the compatibility tests will be discussed along with the results. Finally we end with a discussion of the requirements for dynamically creating cross-realm configurations and a conclusion.

1.1 Related work

Much research has been performed on the Kerberos protocol, significantly less on the implementations. Some research on the implementations was performed in 2001 into the cross-realm options between MIT Kerberos 5 and Active Directory[1]. This research focused on the differences between the implementations and how cross-realm was implemented.

A protocol study in 2005 analysed the behaviour of cross-realm authentication and identified certain problem areas inherent to the Kerberos architecture[2]. When transitive trust is enabled, that is a chain of trust between

[1] `http://oauth.net`
[2] `http://web.mit.edu/kerberos/`
[3] `http://www.h5l.org/`
[4] `http://technet.microsoft.com/en-us/library/bb742437.aspx`
[5] `http://www.gnu.org/software/shishi/manual/shishi.html`

multiple Keberos realms, it is possible for a rogue implementation in the chain to fake request for trusted realms.

More recent research in 2007 highlighted some scalability issues within Kerberos due to the symmetric key nature of the protocol[3]. The administrative requirements to manage key changes and rollover procedures made it impossible to maintain in a large distributed manner.

In 2008, a survey was conducted to see how a good implementation of a decentralised access control mechanism in distributed file systems should function [4]. And although Kerberos was also analysed it was determined that web based authentication methods showed most potential to be implemented. Kerberos was rejected due to the administrative burden when running at a larger scale. This goes to show that Kerberos although a good protocol needs more tools to facilitate production environments.

Research performed by Kamada et al. in 2008 implies that the authentication path created by cross realm authentication may impose undue limits on client libraries[5]. As such an amendment was researched to allow the client to offload path verification to the TGS.

In regards to dynamically configurable cross-realm authentication in Kerberos there has been a draft for PKCROSS specifying the use of public key cryptography for cross-realm operations[6]. This draft however never matured into a standard and for reasons unknown was allowed to expire in 2001. The PKCROSS standard would have specified an addendum to the Kerberos protocol allowing different realms to communicate using public key cryptography as opposed to a manually configured symmetric key.

1.2 Research questions

Our main goal is to analyse the options for dynamically configurable cross-realm authentication and find areas in which cross-realm behaviour can be improved. Before this can be answered there are several other topics that must be analysed.

- Is it possible to give one user access to a service from a different realm and from a different implementation?
- Is it possible to do cross-realm authentication between services? If not, how can we implement it?
- Can a service be authenticated to a user that uses a different Kerberos implementation? What if the user or service is a different realm
- How could a service identify itself to other Kerberos implementations?
- How can the Kerberos implementations share cryptographic algorithms?

1.2.1 Scope

This research is focused on Kerberos 5. Due to the time available for this research, the scope was narrowed down to exclude making changes to any of the implementations to be tested. Also default configurations were used for either source builds or versions stored in package repositories. Lastly the focus

is on the implementations "as is"; instead of how the implementations should be according to specifications.

The implementations looked at are listed below.

Active Directory
: This is the default utility used by many Windows domains to manage users due to built in support by Microsoft. It is built on the Kerberos standard but has several quirks in odd places.

GNU Shishi
: Shishi is an open source GNU implementation of the Kerberos protocol it is newer then other implementations analysed and holds great expectations.

Heimdal
: Heimdal is a European implementation of the Kerberos protocol it was developed at roughly the same time as MIT Kerberos which was unavailable due to U.S. export laws and the encryption used. Heimdal, like Shishi and MIT Kerberos 5, is also available open source.

MIT Kerberos 5
: The MIT Kerberos 5 implementation was developed by MIT and is available, as open source software, outside the US and Canada since October 2003[6].

[6] `http://web.mit.edu/kerberos/dist/index.html`

2 Background

For readers unfamiliar with the Kerberos 5[7] authentication flows and cross-realm authentication a small explanation is provided. This explanation is meant to provide essential knowledge for understanding this report and is not a full description of the Kerberos 5 protocol.

2.1 Kerberos protocol

Kerberos 5 is a seasoned authentication infrastructure that is used in large environments and enterprises. It offers a mechanism to establish authentication but not necessarily authorisation of clients. Authentication is based on shared secrets and makes use of tickets. These tickets are encrypted with the shared secret, most often a symmetric key, and are only meant to be decrypted by the intended receiver, the client, a service, or the key distribution center (KDC). The KDC will, when requested, send a ticket that proves the validity of a client. However the ticket is encrypted with the secret that is shared between client and KDC. As a result only the client is capable of decrypting the actual ticket upon receival. This design makes sure it can be used in untrusted networks like the internet, since the key is not send over the network. Kerberos also provides support for public key cryptography via PKINIT.

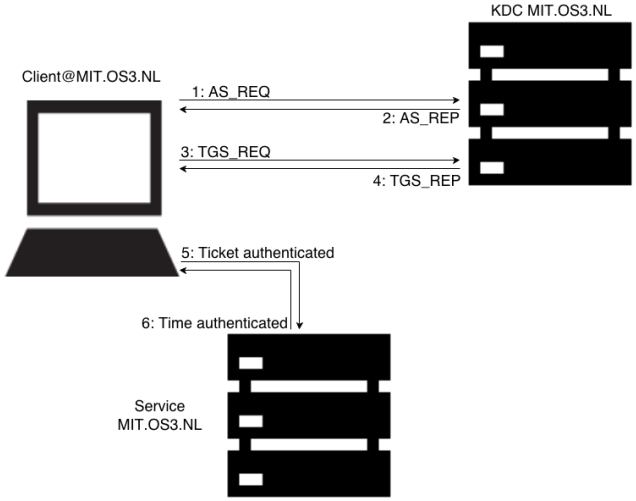

Figure 1: Kerberos 5 authentication flow

As illustrated in figure 1, The client first has to authenticate with the KDC. This is done by an authentication request(AS_REQ), after which the KDC replies with an authentication response(AS_REP). This response contains a ticket granting ticket (TGT) which can be used by the client to request new tickets without having to authenticate again.

When a client attempts to authenticate to a service, the client requests a new ticket from the KDC that authenticates the client to the service using the TGT(TGS_REQ). The KDC will check the TGT for a matching identity and then issue a ticket that can be authenticated by the service(TGS_REP). The client can then simply send the received ticket to the service which can verify the users identity.

2.2 Cross-realm authentication

When adding a remote realm, in which there is a service for which we wish to authenticate, the flow is adjusted a little, which can be seen in figure 2.

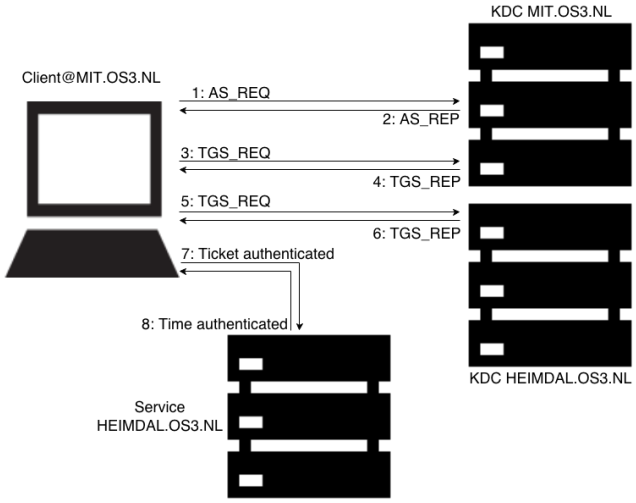

Figure 2: Kerberos 5 cross-realm authentication flow

With the remote KDC being in a new realm, and the service only accepting clients authenticated by the the remote KDC, the client needs a way to authenticate with the service using the local KDC. This is called cross-realm authentication. The local and remote KDCs are configured to perform cross-realm authentication in which a client from MIT.OS3.NL can authenticate itself to use a service from HEIMDAL.OS3.NL.

This is achieved by authenticating the user to its local KDC, which in this case is the MIT.OS3.NL KDC. First an authentication request must be done in which the KDC responds with an authentication reply, AS_REQ and AS_REP same as in 1. But the service the client wants to use is in a different realm. In a cross-realm setup the client would request a ticket from the local KDC (TGS_REQ 1). The local KDC will provide a ticket which the client can provide to the remote KDC to authenticate as a client from the local realm (TGS_REP 1). The client can then contact the KDC in the remote realm and provide

the ticket retrieved in exchange for a ticket that authenticates the user to the service(TGS_REQ 2 and TGS_REP 2).

Once this ticket is granted the client can authenticate to the remote service and has successfully completed the cross-realm authentication.

3 Approach

The research is split into several experiments and smaller research parts. These parts are discussed below and should offer more insights into the total compatibility of the various implementations. First the use of DNS search paths by the implementations to find Kerberos servers is checked. Then the interoperability between implementations for basic Kerberos functionality is analysed. Building on that we analyse public-key cryptography support for client authentication. The behaviour concerning cross-realm configurations is also test between the various implementations.

3.1 Search paths

Kerberos stores a lot of information for path-finding in DNS SRV and TXT records. TXT records can be used to indicate which realm a host resides in and SRV records can be used to define where the KDC of a particular realm is located. The behaviour of these search paths is paramount when working with poorly configured clients. As such the behaviour of the various clients is analysed and checked using Wireshark.

3.2 Cross compatibility

The first thing that needs to be examined, is how the different implementations work together. It requires to check each client side implementation of Kerberos against the possible KDC and TGS implementations. To achieve this, we have split the test in two parts. The first part includes the client authenticating with the Kerberos service and getting a TGT ticket. For test the client authenticates using a password.

The second part is testing if a client holding such a ticket is capable of using a service that handles authentication via Kerberos. This requires the service to be able to retrieve the key used by the Kerberos server to encrypt the tickets intended for the service.

The services use Kerberos using the GSSAPI[8]. GSSAPI in turn uses the implementation available on the system which is one of implementations we are focusing on for this research. Because the services are not able to enter a password they use the de facto standard for services of using a `krb5.keytab` file. This is a file that most implementations can read in an automated fashion to enable host and service authentication.

Active Directory offers a `ktpass.exe` executable whilst Heimdal `ktutil` utility to create or export keytab files containing specific credentials. These files function as a binary password store used by the services to decrypt the passwords supplied by clients in service tickets. Shishi does not support keytab files but instead uses a text alternative in which key information is stored in an encoded format. It should also be noted that these files are meant to be access-limited as they contain passwords in the clear.

Testing in the first part is done by attempting to request a ticket from the TGS using the provided utilities. For MIT Kerberos 5 and Heimdal the `kinit` utility is used. For Shishi the `shishi` utility is used. Testing is considered successful once a ticket is obtained. Table 3.1 contains a list of all combinations tested for obtaining a ticket for a client. It should be noted that the Active Directory client is used to refer to the Windows implementation. We refer to this as Active Directory because other implementations, for example MIT Kerberos for Windows, exist and thus referring to Windows could be confusing.

Client	Server
Active Directory	Active Directory
Heimdal	Active Directory
Heimdal	Heimdal
Heimdal	MIT Kerberos 5
Heidmal	Shishi
MIT Kerberos 5	Active Directory
MIT Kerberos 5	Heimdal
MIT Kerberos 5	MIT Kerberos 5
MIT Kerberos 5	Shishi
Shishi	Active Directory
Shishi	Heimdal
Shishi	MIT Kerberos 5
Shishi	Shishi

Table 3.1: Implementations of clients and servers tested for protocol compliance

For the second part tests are performed by attempting to use a service which delegates authentication to Kerberos. The service chosen is SSH. Because many services use Kerberos via the GSSAPI, SSH included, it was decided to not test other applications since the purpose of this research is to determine compatibility of Kerberos implementations themselves and not GSSAPI compliance. Table 3.2 contains a list of all combinations tested for authenticating a client for SSH.

For Shishi this turned out to be impossible because it turned out that Shishi had not yet implemented support for GSSAPI, or any other API. Attempts were made to use a telnet utility patched to work with Shishi however the utility did not communicate with Shishi.

Due to the integration of Active Directory and Windows we were unable to find a suitable service to test the Windows client side implementation against implementations other then Active Directory. As such Active Directory was tested solely for compliance to the Kerberos 5 protocol and interoperability with other clients. Active Directory was correctly used for authentication of a Windows 7 workstation but other tools were not found to test the required functionality.

Client	Server
Active Directory	Active Directory
Heimdal	Active Directory
Heimdal	Heimdal
Heimdal	MIT Kerberos 5
Heidmal	Shishi
MIT Kerberos 5	Active Directory
MIT Kerberos 5	Heimdal
MIT Kerberos 5	MIT Kerberos 5
MIT Kerberos 5	Shishi

Table 3.2: Combinations tested for protocol compliance between services and kerberos implementations

3.2.1 Cryptographic algorithms

Kerberos uses many different cryptographic algorithms and these are not always compatible. Some implementations may not yet possess the newest cryptographic algorithms added to the standard this could impose limits on the compatibility between clients and servers of different implementations. The opposite is also true some implementations may support outdated and broken algorithms as a means of providing backward compatibility. As such an overview will be provided of the supported algorithms.

3.3 Public-key cryptography

Kerberos also includes support for public-key cryptography for initial authentication(PKINIT) [9]. This means that a client can authenticate using an X.509 certificate instead of a static password. Public-key cryptography can be used to separate the security of the communications from password complexity. Also public-key cryptography allows the use of Diffie-Hellman to determine a secure session key. Furthermore authenticity of the KDC and client can easily be confirmed by checking signatures on the arrival of requests.

PKINIT supports this for the initial authentication of clients. Initial authentication is the authentication used by a client to request the initial Ticket Granting Ticket (TGT) from the KDC.

Tests were performed to analyse the behaviour of the various implementations and their compatibility for PKINIT.

We were unable to test compatibility with Active Directory and Shishi. The former due to complications in the certificates used by Active Directory and the second because Shishi has not yet implemented PKINIT support. Hence these tests are limited to Heimdal and MIT Kerberos 5. Both support the same form of certificate which uses the Extended Key Usage option of the X.509 certificates. X.509 certificates are used to maintain a chain of trust and all relevant information for use in PKINIT is contained in defined extension fields.

For Active Directory this used to be in the `dNSName` Subject Alternative Name and the official standard uses a specially designed field for PKINIT, `id-pkinit-san`. Newer version of Active Directory should support both standards but only give out certificates following the official specification. However due to time constraints and configuration difficulties we were unable to test PKINIT support with Active Directory. According to mailing lists Active Directory has been in compliance with the standard in Windows 2008 and newer versions[7].

The tests performed were to create valid certificates according to documented procedures and then subsequently using these certificates in an attempt to retrieve a TGT, see appendix A for the extensions used. This resulted in four tests to cover all scenarios with only MIT Kerberos 5 and Heimdal participating, namely MIT Kerberos 5 to MIT Kerberos 5, Heimdal to Heimdal, Heimdal to MIT Kerberos 5, and MIT Kerberos 5 to Heimdal.

3.4 Cross-realm compatibility

Having established which implementations can operate together the next phase is to check support and options for cross-realm trusts. This is tested by configuring a trust relationship in each way between the various KDCs and analysing their behaviour.

Tests are performed by letting a client of one realm use a service registered in another realm. The service used is once more SSH and success is defined as the correct login. It should be noted that it is important to include a `.k5login` file to allow clients from another realm to authenticate as a user. By default the SSH server only allows clients from the default realm to authenticate provided that the username matches the client principal.

GNU Shishi was found to not yet support cross-realm authentication, although noted that it is expected by the developers that one week is required to implement this functionality. Thus a cross-realm trust is established between combinations of Active Directory, Heimdal, and MIT Kerberos 5. A two-way trust is configured and then the connection is tested in both ways. Table 3.3 shows the combinations tested.

[7] `http://mailman.mit.edu/pipermail/kerberos/2010-August/016343.html`

Local KDC	Remote KDC
Active Directory	Active Directory
Active Directory	Heimdal
Active Directory	MIT Kerberos 5
Heimdal	Active Directory
Heimdal	Heimdal
Heimdal	MIT Kerberos 5
MIT Kerberos 5	Active Directory
MIT Kerberos 5	Heimdal
MIT Kerberos 5	MIT Kerberos 5

Table 3.3: Configurations tested for cross-realm authentications

4 Results

4.1 Search Paths

To work with Kerberos, the client must first be configured on what realm to use. This can be achieved in two ways which will be discussed. The first is by statically configuring what realm to use and where the realm can be located. This is done via configuration files. The second method is by specifying the appropriate DNS records.

4.1.1 Configuration files

Both Heimdal and MIT Kerberos 5 use the same configuration file, namely `krb5.conf` which can be found in the `/etc` folder. When configuring to connect to a realm, one must add the realm to this configuration file in a specific format.

```
AD.OS3.NL = {
    kdc = ad.os3.nl
    admin_server = ad.os3.nl
}
```

4.1.2 DNS search

When a Kerberos KDC is not configured at the client side, the Kerberos client must find a way to the specified Kerberos realm. Without any configuration Kerberos will default to DNS. More specific to `SRV`[10] and `TXT` records. An example of such `SRV` records are given below.

```
_kerberos._udp.mit.os3.nl   IN   SRV 88   0    1    mit.os3.nl.
_kerberos._tcp.mit.os3.nl   IN   SRV 88   0    1    mit.os3.nl.
```

Tests have shown that the `SRV` records are used by all four implementations if no configuration is present.

The `TXT` records are used to locate what realm a certain domain is in. For this a record is used at `_kerberos.DOMAIN` for the domain in question. When a KDC has these DNS records, the clients will no longer have to configure the realm location in the configuration file. Instead specifying the the realm when attempting to gain the initial ticket is possible. Without client configuration the line below would function, using default settings, if the DNS is properly configured.

```
kinit client@SOME.REALM.OS3.NL
```

One observation showed that the Heimdal kinit utility attempts to contact the known TGS for a service ticket if configuration was lacking. However due to time constraints it could not be determined if this was intentional or not.

4.2 Interoperability

As described in section 3.2 several tests were performed to determine the interoperability between existing Kerberos implementations. Table 4.1 contains a summary of the findings which indicates the interoperability and what form of communication can take place between different implementations.

	KDC			
Client	Active Directory	Heimdal	MIT Kerberos 5	Shishi
Active Directory	✓			
Heimdal	✓	✓	✓	✓
MIT Kerberos 5	✓	✓	✓	✓
Shishi	✓	✓	✓	✓
Service	Active Directory	Heimdal	MIT Kerberos 5	Shishi
Active Directory	✓			
Heimdal	✓	✓	✓	✓
MIT Kerberos 5	✓	✓	✓	✓
Shishi	✗	✗	✗	✗

Table 4.1: Compatibility between implementations

The different implementations when concerning client and KDC communication appears to be following protocol and can be achieved almost out of the box. The first half of the table shows the possibility to communicate between a client[8] and the KDC implementation of a given producer. Please note that Active Directory could not be properly tested as described in 3.2.

It should be noted that the Windows `ktpass.exe` executable creates a keytab file which was incompatible with the MIT Kerberos 5 and Heimdal implementations. A working keytab file was created using the MIT Kerberos 5 `ktutil` utility using identical settings as provided to Active Directory.

When looking at the second half of the table, which concerns a service with a Kerberos implementation, one can deduce that Shishi does not work with any of the implementations. This is due to the fact that Shishi does not have the GSSAPI implemented as of version 1.0.2[9]. Due to the lack of GSSAPI, many services that use Kerberos in combination with GSSAPI will be unable to communicate with Shishi. A service that provided a compatible interface with Shishi could not be found. As such we were unable to test Shishi using a live service. However it was possible to use a keytab file, converted to the Shishi format using `keytab2shishi` utility, to retrieve a service ticket which implies that once GSSAPI support is implemented it should work as expected.

During the interoperability tests, we encountered issues regarding the Heimdal kinit client. The kinit utility ignored the `/etc/hosts` file and required a correctly configured DNS in order to locate the KDC. This behaviour was observed only under Ubuntu 12.04 and was not exhibited under OS X 10.9. The

[8] e.g. Kinit or similar

[9] `http://www.gnu.org/software/shishi/manual/shishi.html#Features-and-Status`

reason for this behaviour could not be ascertained.

4.2.1 Cryptography

Kerberos uses cryptography to encrypt tickets. The different implementations also use different cryptographic algorithms. These algorithms can be found in table 4.2.

	Active Directory	Heimdal	MIT Kerberos 5	Shishi
AES128/256-SHA1	✓	✓	✓	✓
CAMELLIA128/256-CTS-CMAC			✓	
DES3-CBC-SHA1		✓	✓	✓
DES-CBC-CRC	(✓)		(✓)	(✓)
DES-CBC-MD5	(✓)		(✓)	(✓)
DES-CBC-MD4			(✓)	(✓)
RC4-HMAC-EXP			(✓)	(✓)
RC4-HMAC	✓	✓	✓	✓

Table 4.2: Ciphers implemented, ✓in parentheses are off by default

This table explains that there are two encryption types common for every implementation of Kerberos. These are the AES128/256-SHA and the RC4-HMAC encryption types. Windows uses RC4-HMAC and AES128/256-SHA by default but disabled the DES encryption since Windows server 2008[10]. This can be enabled by explicitly configuring this.

During cross-realm configuration, Microsoft's Active Directory uses the RC4-HMAC encryption type for compatibility reasons and can only activate AES128/256-SHA during the trust setup.

The DES and RC4-HMAC-EXP ciphers are considered to be weak[11]. Therefore MIT Kerberos 5 and Active Directory have turned this cryptography off by default. Heimdal takes it one step further by removing it from source code[11].

MIT Kerberos 5 also added support for the relatively new CAMELLIA ciphers[12]. It won't work with cross compatibility since none of the other implementations allow for this encryption type, yet.

4.3 Public-key cryptography

Kerberos also provides support for public-key cryptography that can be used in, for example, smart cards. Shishi, version 1.0.2, is the only implementation of Kerberos that does not support PKINIT at the time of writing.

Tests were limited to Heimdal and MIT Kerberos 5 using the specified EKU as documented in [9]. We were unable to get Active Directory to function and

[10]Windows enabled encryption types `http://technet.microsoft.com/en-us/library/dd560670(v=ws.10).aspx`

[11]Heimdal git `https://github.com/heimdal/heimdal/pull/91`

even accept public-key certificates. Since sources and documentation suggest that Active Directory includes support for PKINIT we can only conclude that we were unable to get it working.

MIT Kerberos 5 to MIT Kerberos 5 worked without issue after specifying the certificate to use. The same goes for Heimdal to Heimdal. It should be noted that the same certificate structure was used for both combinations.

However when using the Heimdal client to authenticate using PKINIT to the MIT Kerberos 5 KDC an error occured. Analysis of packets send over the network suggested that the MIT Kerberos 5 KDC found the "Crypto to weak". This was solved by forcing the Heimdal client to use public-key encryption instead of Diffie-Hellman.

Further research remains to analyse the behaviour of Active Directory to PKINIT request from either MIT Kerberos 5 or Heimdal.

4.4 Cross-realm interoperability

When configuring the different implementations to use cross-realm authentication, it can be concluded that all of the implementations, except for Shishi, can work together. Results can be found in table 4.3. Although it looks like

	Active Directory	Heimdal	MIT Kerberos 5	Shishi
Active Directory	✓	✓	✓	✗
Heimdal	✓	✓	✓	✗
MIT Kerberos 5	✓	✓	✓	✗
Shishi	✗	✗	✗	✗

Table 4.3: Cross-realm compatibility

Heimdal, MIT Kerberos 5, and Active Directory work well together, there are some explicit settings required to cooperate with Active Directory. For one, the encryption type must be set on RC4-HMAC[12].

Furthermore, the trust wizard indicates that a realm must be configured with lower case characters, but this must be done with capitals. The DNS records must also be set up correctly. The realm name must have an `A` record towards the IP-address and the `SRV` records must exist.

4.4.1 Active Directory

Windows Active Directory was tested with Heimdal and MIT Kerberos 5. This is done by adding a trust policy in the Windows trust wizard. Both the realms of the MIT Kerberos 5 and the Heimdal implementation are added in a two-way trust relationship. To enforce the RC4 encryption type we entered the following command in the command line:

```
ksetup /SetEncTypeAttre HEIMDAL.OS3.NL RC4
```

[12] `http://social.technet.microsoft.com/wiki/contents/articles/2751.kerberos-interoperability-step-by-step-guide-for-windows-server-2003.aspx`

The following principals were added on the Heimdal KDC:[13]

```
krbtgt/AD.OS3.NL@HEIMDAL.OS3.NL
krbtgt/HEIMDAL.OS3.NL@AD.OS3.NL
```

After the principals are added, other encryption types that will not work with Active Directory, need to be removed. This is done by issuing the following commands:[13]

```
kadmin del_enctype krbtgt/AD.OS3.NL@HEIMDAL.OS3.NL aes256-cts-hmac-sha1-96
kadmin del_enctype krbtgt/AD.OS3.NL@HEIMDAL.OS3.NL des3-cbc-sha1
kadmin del_enctype krbtgt/HEIMDAL.OS3.NL@AD.OS3.NL aes256-cts-hmac-sha1-96
kadmin del_enctype krbtgt/HEIMDAL.OS3.NL@AD.OS3.NL des3-cbc-sha1
```

After this, Windows needs to be rebooted and it works. The reason why AES encryption did not function could not be determined.

[13] This is also what needs to be done for MIT Kerberos 5, replacing the HEIMDAL with MIT.

5 Dynamic cross-realm authentication

When automatic cross realm authentication has to be implemented, one will need several aspects to get to the result. One thing that is necessary is a way to identify and find the realms used by client and server. This can be solved by using the existing DNS solution as mentioned in section 4.1.2. Furthermore, a solution must be found for establishing a key for cryptography between the KDCs.

Ideally one would like to have a public key infrastructure. PKIs come in several flavours, but can be reduced to three classes: a regular public key infrastructure, a third party trusted public key infrastructure or a public key infrastructure using DANE[13].

Next we will discuss several options and methods we consider to be important when creating a dynamic cross-realm authentication procedure.

5.1 Key sharing

Dynamic configuration for cross-realm authentication requires a means of establishing trust and secure connection between verified endpoints. As such symmetric key algorithms become less attractive due to the need for key management and distribution. Instead public-key cryptography can be used to facilitate secure communications and authentication of realms.

This research describes two methods for facilitating the trust delegation of keys. The first method uses a centralised hierarchical system and the second utilises the existing DNS architecture to allow greater freedom for realms to specify and share certificate information.

5.1.1 PKI

The first method in which keys can be shared is to establish a central authority. In public key cryptography, as facilitated using X.509 certificates, this can be accomplished by using a single certificate authority or to provide a certificate store with the various implementations as is done with modern browsers.

Regardless of the method chosen eventually each realm should be able to provide a certificate which validates against trust anchors. This certificate could be retrieved at the time a KDC is requested for a cross-realm ticket.

5.1.2 DANE

The second method we describe is to use DNS-based Authentication of Named Entities (DANE) to facilitate a mechanism to prove certificate authenticity for a given realm. As DANE uses DNS records it is possible to delegate the trust to DNSSEC. DNSSEC can be used to prove the validity of the certificate a realm is expected to use. This makes it possible for a more varied group of certificate authorities for realms. Including organisational CAs or self-signed certificates in addition to all the normal trust anchors which might be available in a certificate store.

The benefit of DANE is that it leverages an existing framework and does not require a new organisation to maintain large indexes. A KDC simply returns the certificate it wishes to use which can be validated using DANE.

Certificates can be requested by a KDC at the time a cross-realm ticket is requested by a client. DANE should be used to validate the record supplied by the remote KDC.

It should be noted that even when a PKI infrastructure is used DANE may still offer added security by limiting what PKI should be trusted for a particular realm.

5.2 Key revocation

An important consideration when using public-key cryptography for dynamically authenticating remote KDCs for cross-realm operations is the possibility for the authentication to change. Keys can expire, change, or be compromised as such it is possible that a remote KDC uses a different certificate during operations. When this happens, the trust should be reevaluated between KDCs.

This also means that mechanisms must exist before it can be utilised. If certificates are provided by a remote CA then these may be mentioned on certificate revocation lists. This means certificate revocation lists should be checked. DANE may be used to provide further details and constraints by the realm owner on what to trust.

5.3 Trust policies

Cross-realm authentication requires a level of trust. However this trust should only be extended by trusting the authenticity of a user. As such any method for cross-realm authentication should not impose minimum or maximum trust to the client from the KDC.

This means that any application using a KDC which accepts cross-realm clients to work on a minimum-trust principle. We suggest that the trust that should be given be contained within the application domain. Dynamic cross-realm authentication can be used to ensure that a client is a certain user in a certain realm however it should not imply anything beyond that. If any trust policy should be maintained it is not to trust anything.

If an application incorrectly imposes limits and only checks the principal name instead of the realm name as provided this constitutes a breach of security. As such it may be required to use separate KDCs for specific tasks unless the application can be configured properly.

5.4 Time synchronisation

If a globally cross-realm system would function with random participants it is important that all hosts adhere to strict timing. Due to Kerberos imposing time constraints on tickets and cryptography it is imperative that any and all KDCs participating in cross-realm operations be on the same time. This means that

when using an internal time server to synchronise all systems it is imperative that the internal time server be synchronous with external sources.

6 Conclusion

We set out to find the state of cross-realm authentication between various implementations. In the course of this research we found that many of the implementations work together fine, for the most part. Some small quirks exist in the various implementations but for the simple usage scenario each implementation is compatible with others. Cross-realm authentication is possible but only in manually configured trusts. Various peculiarities have been found regarding the ciphers supported and the search paths used by Kerberos 5.

In summary although there are differences in the implementations, they all implement the correct Kerberos 5 protocol. Due to this every implementation can communicate with one another. Most conflicts that occur, happen because of other variables like for example encryption settings.

The implementations tested that provide the most compatible interface to the Kerberos protocol are Heimdal and MIT Kerberos 5. Active Directory although it adheres to the protocol has several quirks in both the cryptographic support for cross-realm requests and the necessity to map services and hosts onto users for internal administration. Shishi is promising because it has the ability to implement the correct Kerberos specifications without being burdened by legacy but has a long road ahead before hand.

Configuring Kerberos infrastructures is not always as clear as expected when using different implementations. The protocol itself includes limited means to ease configuration issues. Due to the shared secret philosophy this process is difficult to automate. The use of public key cryptography between KDCs could ease the administrative burden and configuration complexity. In the past some attempts have been made to facilitate authentication by means of public key cryptography and public key infrastructures. Via PKINIT clients can authenticate with the KDC via secure public key cryptography. However PKINIT is only specified for client authentication and can't be used by hosts, services or remote realms.

Finally some ideas have been proposed which might aid in the development of a protocol extension enabling dynamic configuration for cross-realm authentication. This has the benefit of lowering the difficulty in configuring internal Kerberos networks and enabling easy authentication between clients and services in different realms. However once the protocol is extended and the implementations include the changes there still remains much to be done. Applications may need to be adjusted to correctly handle new situations in which valid clients may be presented.

7 Future work

Complete Shishi

Shishi is far from complete. It still needs to implement the cross-realm functionality as well as PKINIT and GSSAPI support. Because Shishi is greenfield technology it is possible to correctly implement these features, without the burdens of legacy, making for a promising future.

Better debugging in the implementations

Most of the implementations do not have a lot of feedback during configuration. The output and logging are not verbose at all. Heimdal for example has no debugging information when not configured at compile time. To improve workability between implementations, it would be useful to get some information in log files. This also extends to the error messages incorporated into the protocol.

Improve interoperability between implementations

There are several settings that differ greatly from one another. The encryption ciphers have to be implemented to cooperate with each other. The implementation needs to look at both the salting and the key generation. Furthermore several clients implement a certain leniency to user input which can cause problems later on. It is therefore advisable to be strict with user input and perhaps include these limits and restrictions in the specification.

Use public-key cryptography for cross-realm authentication

The biggest hurdle for dynamic cross-realm authentication is that there is only support for symmetric cryptography for cross-realm tickets. As long as this remains so it is unlikely that cross-realm configuration can be automated. Over the years some attempts have been made for adding public key cryptography for cross-realm authentication and trust delegation. These attempts however have failed to lead to a standard specification that could be implemented.

Another added benefit of adding public key support to cross-realm configurations is that it may lead to an increased ease of use when configuring local realms. Instead of having to update password change procedures and rollovers this could be delegated to the same method used for dynamic configuration.

References

[1] Jonathan T. Trostle, Irina Kosinovsky, and Michael M. Swift. "Implementation of Crossrealm Referral Handling in the MIT Kerberos Client." In: URL: http://pages.cs.wisc.edu/~swift/papers/xrealm.pdf.

[2] I. Cervesato et al. "Specifying Kerberos 5 Cross-realm Authentication". In: *Proceedings of the 2005 Workshop on Issues in the Theory of Security*. WITS '05. Long Beach, California, 2005, pp. 12–26. ISBN: 1-58113-980-2.

[3] Angelos D. Keromytis and Jonathan M. Smith. "Requirements for Scalable Access Control and Security Management Architectures". In: *ACM Trans. Internet Technol.* 7.2 (May 2007). ISSN: 1533-5399. DOI: 10.1145/1239971.1239972. URL: http://doi.acm.org/10.1145/1239971.1239972.

[4] Stefan Miltchev et al. "Decentralized Access Control in Distributed File Systems". In: *ACM Comput. Surv.* 40.3 (Aug. 2008), 10:1–10:30. ISSN: 0360-0300. DOI: 10.1145/1380584.1380588. URL: http://doi.acm.org/10.1145/1380584.1380588.

[5] K. Kamada et al. "Design and evaluation of a client-friendly cross-realm framework for Kerberos 5". In: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. July 2008, pp. 541–546. DOI: 10.1109/INDIN.2008.4618160.

[6] C. Neuman et al. *Public Key Cryptography for Cross-Realm Authentication in Kerberos*. PKCROSS draft 08. Internet Engineering Task Force, Nov. 2001. URL: http://tools.ietf.org/html/draft-ietf-cat-kerberos-pk-cross-08.txt.

[7] C. Neuman et al. *The Kerberos Network Authentication Service (V5)*. RFC 4120 (Proposed Standard). Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806. Internet Engineering Task Force, July 2005.

[8] L. Zhu, K. Jaganathan, and S. Hartman. *The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2*. RFC 4121 (Proposed Standard). Updated by RFCs 6112, 6542, 6649. Internet Engineering Task Force, July 2005.

[9] L. Zhu and B. Tung. *Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)*. RFC 4556 (Proposed Standard). Updated by RFC 6112. Internet Engineering Task Force, June 2006. URL: http://www.ietf.org/rfc/rfc4556.txt.

[10] A. Gulbrandsen, P. Vixie, and L. Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. RFC 2782 (Proposed Standard). Updated by RFC 6335. Internet Engineering Task Force, Feb. 2000. URL: http://www.ietf.org/rfc/rfc2782.txt.

[11] L. Hornquist Astrand and T. Yu. *Deprecate DES, RC4-HMAC-EXP, and Other Weak Cryptographic Algorithms in Kerberos*. RFC 6649 (Best Current Practice). Internet Engineering Task Force, July 2012.

[12] G. Hudson. *Camellia Encryption for Kerberos 5*. RFC 6803 (Informational). Internet Engineering Task Force, Nov. 2012. URL: http://www.ietf.org/rfc/rfc6803.txt.

[13] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698 (Proposed Standard). Updated by RFC 7218. Internet Engineering Task Force, Aug. 2012. URL: http://www.ietf.org/rfc/rfc6698.txt.

A PKINIT EKU Extensions

Attached are the extensions for the X.509 certificates used in PKINIT tests. More documentation and directions on how to use these extensions can be found at `http://web.mit.edu/kerberos/krb5-1.12/doc/admin/pkinit.html`. The realm and client principals are set using environment variables.

Listing 1: KDC certificate extension

```
[kdc_cert]
basicConstraints=CA:FALSE
keyUsage=nonRepudiation,digitalSignature,keyEncipherment,keyAgreement
extendedKeyUsage=1.3.6.1.5.2.3.5
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
issuerAltName=issuer:copy
subjectAltName=otherName:1.3.6.1.5.2.2;SEQUENCE:kdc_princ_name

[kdc_princ_name]
realm=EXP:0,GeneralString:${ENV::REALM}
principal_name=EXP:1,SEQUENCE:kdc_principal_seq

[kdc_principal_seq]
name_type=EXP:0,INTEGER:1
name_string=EXP:1,SEQUENCE:kdc_principals

[kdc_principals]
princ1=GeneralString:krbtgt
princ2=GeneralString:${ENV::REALM}
```

Listing 2: Client certificate extension

```
[client_cert]
basicConstraints=CA:FALSE
keyUsage=digitalSignature,keyEncipherment,keyAgreement
extendedKeyUsage=1.3.6.1.5.2.3.4
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
issuerAltName=issuer:copy
subjectAltName=otherName:1.3.6.1.5.2.2;SEQUENCE:princ_name

[princ_name]
realm=EXP:0,GeneralString:${ENV::REALM}
principal_name=EXP:1,SEQUENCE:principal_seq

[principal_seq]
name_type=EXP:0,INTEGER:1
name_string=EXP:1,SEQUENCE:principals
```

```
[principals]
princ1=GeneralString:${ENV::CLIENT}
```