ADCS – An ADC Security Layer

Rationale

The Advanced Direct Connect (ADC) protocol is a plain-text communications protocol whose primary applications are chat and file-transfer. The protocol uses a client-server topology to create *hubs* whereby users may connect and interact. The protocol also enables two users on a common hub to establish a client-server connection, primarily for file-transfer. Thus, although commonly referred to as a peer-to-peer protocol, the basic network design is not a peer-to-peer network[1].

There has been concern in the ADC community, for some time, around the fact that the existing protocol is insecure. Specifically, the following issues have been identified:

- Communication is vulnerable to eavesdropping by unknown third-parties.
- The client cannot provide assurance that the hub it connects to is that which it intended to.
- The protocol does not provide a registration method. Password-based registration schemes currently transmit passwords to the server in plain-text in a hub-software-specific manner.
- The hub authentication method is potentially insecure i.e. the hub cannot verify that a connection has not been hijacked[2] in the establishment phase, or initiated by a client impersonating the registered user[3].

These issues come to the fore on the Internet; however, even on a local area network (LAN) these issues may all be present, particularly if a network endpoint is connected to an unsecured wireless network.

Issues and Models

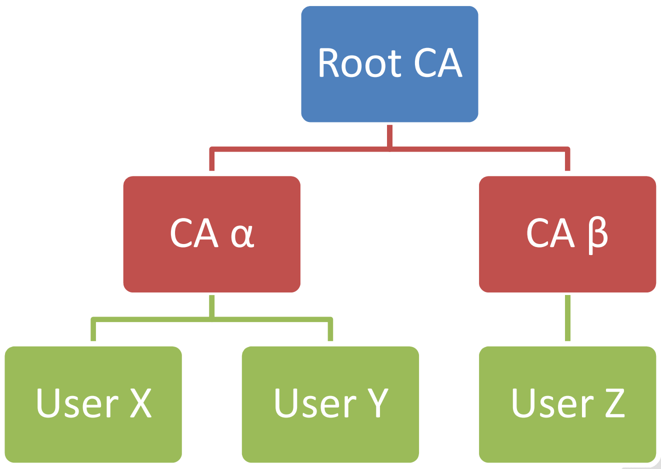
Trust Models

There are two primary trust models in use today: public key infrastructure and web of trust. A trust model is necessary to authenticate other users in a network and validate the credentials they provide.

Public Key Infrastructure (PKI)

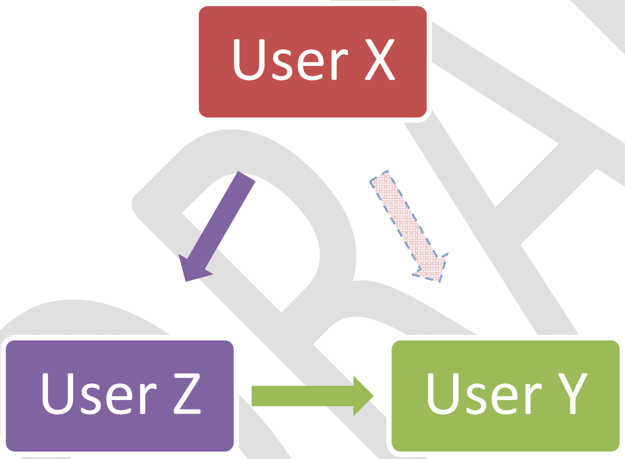
PKI is a hierarchical model, which utilises a chain of trusted third parties, known as certificate authorities (CAs). A Root CA is trusted by all below it in the certificate hierarchy; likewise, a CA is trusted by all below it. In the diagram below, User Z does not directly trust the identity of User X; however, User Z does trust the Root CA via CA β. User X's certificate is signed by CA α, which is in turn signed by the Root CA; thus, due to the shared trust in the hierarchy, User Z will trust User X. Likewise, User X will trust User Y's certificate, because it was signed by the trusted third party, CA α.

[1] StrongDC++ contains a distributed hash table (DHT) implementation which seeks to add a peer-to-peer layer to the network. It utilises UDP packets. See http://www.adcportal.com/wiki/index.php/StrongDC%2B%2B_DHT
[2] Connection hijacking is known as a man-in-the-middle (MITM) attack.
[3] A rogue hub operator may harvest a user's private id and client id for the purposes of connecting to a private hub which the user is a member of.



Web of Trust

A Web of Trust is a distributed trust architecture, which has no reliance on a central authority. The system acts on the concept of *trusted introducers*, i.e. another user which the user trusts to identify a third party. If User X trusts User Z, but does not know User Y, then User Y may become trusted to User X if User Z *introduces* User Y as a known party. This system can be likened to the system in human relationships whereby one person *vouches* for the good standing of another person within the community. A common application of this model is Pretty Good Privacy (PGP).



Comments

As stated previously, an ADC network operates in a client-server topology. User X will trust the hub when it informs it that User Y is a member of the hub. Many hubs also introduce the concept of *operators*, which are trusted managers of the hub, in the absence of the hub owner. Operators have significant powers beyond those of a normal user. ADC networks are thus no strangers to hierarchical trust models.

In this existing trust model, a client trusts the hub in regards to the identity of users. It is a natural extension of this model for the hub to continue this role in a PKI model whereby the hub asserts the identity of other users via the issuance of digital certificates.

The Web of Trust model relates well to that in which people are *invited* to join private networks. However, in an ADC network, the client only maintains constant communication with a hub. This means that a hub should be the party which *vouches* for the identity of all parties connected to the hub.

The hub is free to delegate the power to initiate the issuance of a certificate to its operators; however, although allowing an operator to issue certificates may be beneficial (you can see which operator registered a user), it does introduce additional complexity, and it is simpler for the hub to authorise the operator to prompt the hub to issue a certificate in the hub's name.

Secure and Insecure Clients

The use of secure sessions is widely known to place an extra resource overheard on connecting parties. Although the most resource-intensive ciphers, asymmetric encryption, are only used during the session handshake, symmetric encryption is still more resource-intensive than no encryption. It is therefore important to note that an ADCS-only hub will be constrained to a smaller size than a non-ADCS hub running on the same hub-software.

Hybrid Hubs

A hub which allows both ADCS and non-ADCS connections, a hybrid hub, may hold an acceptable level of security for some situations. The key issues are what is being protected through the use of ADCS from the hub owner's perspective, and what level of security does ADCS imply to end users?

A hub owner may potentially wish to secure all client-hub aspects of the ADC protocol, including:

- Public chat,
- Private chat,
- User information,
- Search requests,
- Authentication, and
- Feature negotiation.

Since an ADC network involves interaction with other clients, and the level of security is only as strong as the weakest link in each interaction, the hub can only certify that certain interactions are secure on a hybrid hub. For an ADCS user, feature negotiation and authentication, with the hub, will be secure. The hub may be able to advise the client that a private chat conversation with another user is secure (i.e. unreadable by third parties), if both users are connected via secure channels; however, the notification to the user is dependent on client support, it is not implicit in the architecture of the hub (a client may also choose to negotiate a secure channel with a client, specifically for a private conversation). Public chat on a hybrid hub does not offer privacy, since insecure plain-text connections are allowed. Likewise, search requests and user information are not private to only the users of the hybrid hub.

Why Not Hybrid Hubs?

In summary, a hybrid hub may reduce the impact of ADCS-induced resource limitations; however, it does not produce a hub which is secure for all communications, and client support is necessary to inform the user appropriately when insecure channels exist on the hub. The usability factor is likely to inhibit such hubs; thus, it is proposed that such a model be disallowed by the protocol, as safely handling the situation places an onus on client developers to inform the user. If a high user count is desirable, then a non-ADCS hub should be used in the absence of more capable computer hardware.

If, however, the hub owner desires a secure hub, then its size should be kept small. This follows the common sense need to keep secrets between a chosen few to reduce the chance of information leakage or infiltration.

Implementation

This specification seeks to address the above concerns through the use of the Transport Layer Security (TLS) protocol v1.2[4]. TLS is used by a number of Internet protocols to establish endpoint authentication and communication confidentiality through the use of cryptography.

Connection Approach

There are two approaches to implementing TLS in application protocols (see Appendix I for further discussion of these approaches):

- Separate ports: two ports are used by the protocol, one for non-secure sessions, and one for secure sessions. The most well known use of this method is HTTP/HTTPS which use ports 80 and 443 respectively. This has a severe limitation in that an existing network cannot upgrade to a secure network on the same port.
- Upward negotiation: this method is known as the 'TLS upgrade' method, and requires only one port. The client and hub negotiate feature support and, when both parties support secure sessions and at least one party requires it, a TLS handshake is commenced. It is used by IMAP, POP3 and SMTP, among others[5].

The approach which is used shall depend on the goals of the hub owner. If the hub owner seeks to migrate an existing hub to a secure protocol, and is not concerned about the potential for the protocol to be blocked by service providers or detected by unknown third-parties, then the upward negotiation method is appropriate as there is no need to obfuscate the existence of the protocol, users will not need to update the address of the hub in their clients, and users of client software which does not support the secure protocol may be informed and refused a connection in a graceful manner. However, if a hub owner wishes to establish a new secure hub, or is concerned about protocol detection and/or blocking, then the secure session should be initiated prior to any protocol-specific messages being transmitted. For a new hub, the hub address should be specified with the 'adcs' URL scheme. For an existing hub which does not use upward negotiation, connection attempts which do not commence with a TLS handshake should be blocked.

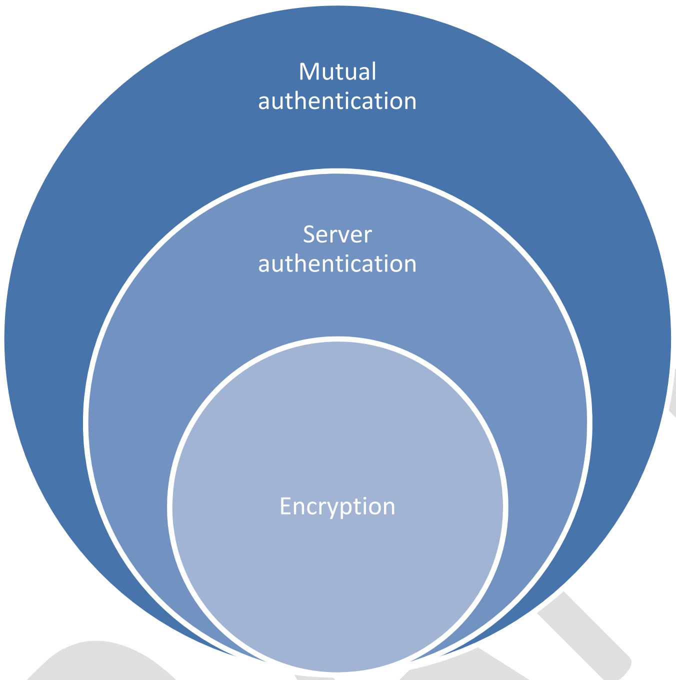
This is summarised in the table below.

Situation	Use upward negotiation	Use ADCS prefix
Existing hub		
Obfuscation unnecessary	✓	✗
Obfuscation necessary	✗	✗
New hub	✗	✓

[4] See http://tools.ietf.org/html/rfc5246 or http://en.wikipedia.org/wiki/Transport_Layer_Security
[5] See http://tools.ietf.org/html/rfc2595 and http://tools.ietf.org/html/rfc3207

Situational Considerations

TLS provides two levels of authentication: server authentication and mutual authentication. The level used will depend on the issues the hub owner is trying to solve. Note that; whilst both levels will result in an encrypted session, mutual authentication provides a higher level of assurance as to the identity of the parties.



Server Authentication

In this scenario, only the server is authentication; that is, the client is aware of the server's identity, however the client remains unauthenticated and anonymous. The authentication of the server is performed via the server's digital certificate.

Gaining a digital certificate may pose an issue in ADC networks, because a certificate signed by a certification authority (CA) costs money, which must be borne by the hub owner. Since hub owners are unlikely to find a certificate authority run by the ADC community to be an acceptable substitute, it will in most cases be necessary for a hub owner to self-sign their digital certificate. Thus, hub software should support the use of a certificate signed by a certification authority, but also allow a hub owner to generate a self-signed certificate.

Verification of Self-Signed Server Certificates

A self-signed server certificate raises the issue of how a client can trust that the hub is who they say they are. Since anybody can generate a self-signed certificate, an additional verification method is necessary to ensure the client is connecting to the hub which they believe they are connecting to. The proposed KeyPrint extension for ADC[6] presents a solution to this. When a hub address is given to a user, it is appended with a cryptographic hash of the server's public key, which the client software may then use to verify that the certificate it received from the server does indeed belong to the server. Specifically, the URL contains a query string with a parameter 'kp', which is similar to the 'xt'

[6] KeyPrint extension: http://www.adcportal.com/wiki/index.php/KEYP

parameter of the Magnet URI scheme[7][8], except that the hash is the base32-encoded hash of the server's public key i.e.:

kp=urn:algorithm:base32(hash)

For the case where the hash function is Tiger, this would appear as:

adc://myhub.com:1234/?kp=urn:tiger:USNVXMWXL5MSQHR4ITYJITVFY75RUGIDCBQ3BZQ

The same query string is used for an 'adcs' URL scheme.

Mutual Authentication

In a mutual authentication scenario, all the same facts in server authentication apply; however, the client is also authenticated i.e. the server authenticates the identity of the client. TLS enables three primary methods for client authentication:

- A client certificate,
- TLS-PSK (pre-shared key, not discussed here), or,
- Secure Remote Password (SRP) protocol.

Client Certificates

The use of client certificates presents similar challenges to server certificates. Users are unlikely to pay for a certificate, particularly since it reveals information about their true identity, and many users want trust based around an online identity, disconnected from their physical identity. Thus, one solution is for the hub to issue certificates to clients upon registration, which are signed by the hub.

The method of issuing a certificate is a question of infrastructure, but one option is that when a user in a server-authenticated TLS secure session is registered on the hub, the hub generates a certificate and sends it to the user via a new ADC protocol message (e.g. CRT, for certificate). Since a secure session already exists, the certificate cannot be intercepted, and the only issue is that the user being registered may not have truly been the intended user. Another possible method is for a web site to send the generated certificate to an email address, which the user must then install in their client software; though, of course, email is unencrypted, and thus the certificate would be vulnerable to capture by an opportunistic eavesdropper.

An alternative is for the client to self-sign its certificate. However, this is seen as much less ideal, and therefore an alternative is proposed: the use of the SRP protocol.

Secure Remote Password (SRP) Protocol

SRP allows a user to authenticate themselves to the server without the use of a trusted third-party, such as a certificate authority (CA), whilst being resistant to the dictionary attacks that many challenge-response mechanisms are vulnerable to[9]. The use of SRP in TLS authentication may be for both client and server authentication[10]; however, in this proposal it is only used for client authentication, with certificates still used for server authentication.

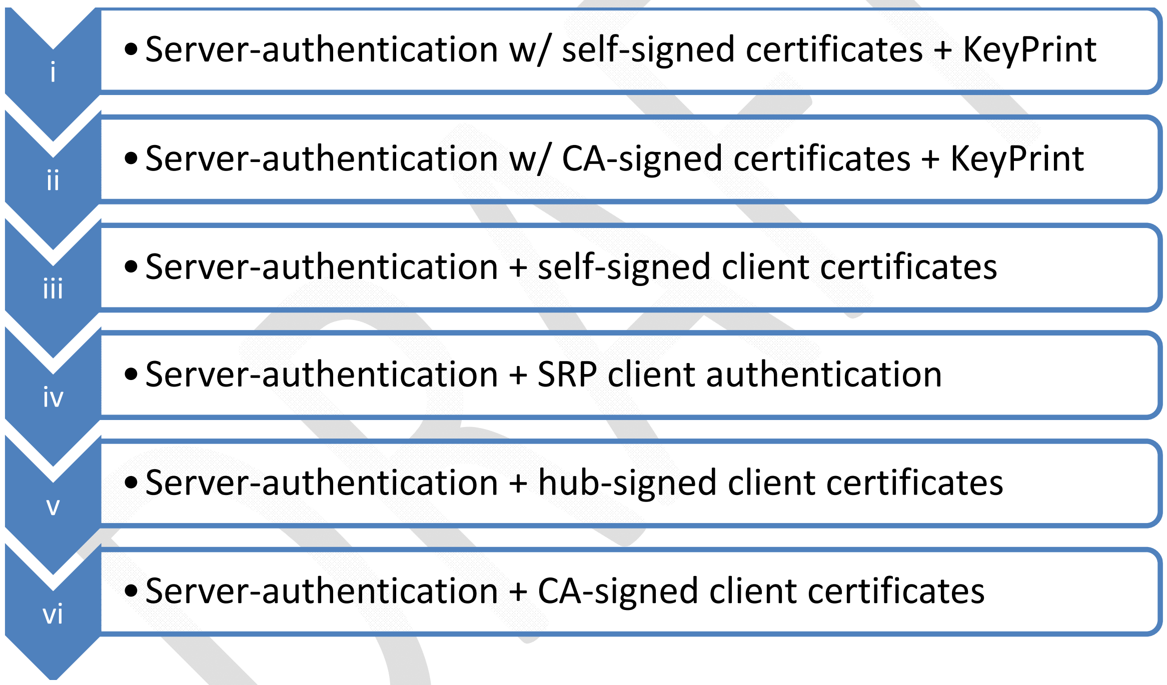
[7] URN, containing hash: http://en.wikipedia.org/wiki/Magnet_URI_scheme#URN.2C_containing_hash_.28xt.29
[8] The Uniform Resource Name (URN) Syntax: http://tools.ietf.org/html/rfc2141
[9] The SRP Authentication and Key Exchange System v3: http://tools.ietf.org/html/rfc2945
[10] Using SRP (v6) for TLS Authentication: http://tools.ietf.org/html/rfc5054

Since SRP uses both a user name and password, there is room for discussion as to whether the user name should be the user's Client ID (allowing them to change their nick name and IP address), or if it should be their nick name (allowing them to change their Client ID and IP address). There are merits to either scheme, and the extension may allow both, with the specific method used to be given to the client during the registration process. It is envisioned that supplying the user name parameter will be invisible to the user, with only the password needing to be entered (if it is not retrieved from a secure store on the user's system). Hub software should limit the rate of authentication attempts from a particular IP address or against a particular user name.

It should be noted that when a secure session has not already been established, the user name is sent in clear text. Thus, it may be desirable to establish a server-authenticated connection, and then renegotiate an SRP-authenticated connection with the handshake now protected by the first connection. This should be agreed upon for compatibility between implementations.

Choosing an Authentication Method

The methods described above may be represented as series of escalating levels of security.


i
• Server-authentication w/ self-signed certificates + KeyPrint
ii
• Server-authentication w/ CA-signed certificates + KeyPrint
iii
• Server-authentication + self-signed client certificates
iv
• Server-authentication + SRP client authentication
v
• Server-authentication + hub-signed client certificates
vi
• Server-authentication + CA-signed client certificates

The recommended methods are (i), (iv) and (v):

- Method (i), server-authentication using self-signed certificates and KeyPrint, should be used for basic secure hubs.
- Method (iv), server-authentication (server certificates and KeyPrint) with SRP client authentication, enables a familiar username/password scheme without the overhead of generating and installing client certificates.
- Method (v), server-authentication with hub-signed client certificates, is for environments where the hub owner places a strong emphasis on trust, though it is more difficult for hub software to implement, and poses certificate-management issues.

Although method (vi) is the most secure, many users may be uncomfortable with the use and cost of CA-signed certificates.

Client-to-Client Connections

The establishment of secure sessions between clients enables peer-to-peer communications, unobservable by other members of the hub, or external parties. The fact that ADC uses a client-server model for client-to-hub and peer-to-peer connections means that similar concepts apply for the implementation of a security model. However, since each client has an established secure session with the hub, this may be used to facilitate authentication.

In a server-authenticated client-server model, the server is believed to be trustworthy by the client. Thus, when the client X wishes to initiate a secure session with client Y, X can assume that the hub will use the secure channels it holds with X and Y to relay this request to the intended recipient, Y. This means that X and Y are both implicitly authenticated to one-another, via the trusted hub.

This potentially means the hub could be used to facilitate the exchange of symmetric keys for message encryption; however, the TLS protocol provides algorithm support negotiation and message authentication, which need not be reinvented by an alternative method. Thus, the proposed method involves each client on a secure hub sharing the KeyPrint of their public key with all hub users via the INF message. A KP parameter is added to the INF message, containing the same 'urn:algorithm:base32(hash)' form as for hub addresses e.g.:

```
        INF KPurn:tiger:USNVXMWXL5MSQHR4ITYJITVFY75RUGIDCBQ3BZQ
```

When a client initiates a TLS connection with another client, each client..

Use TLS-PSK. http://tools.ietf.org/html/rfc4279

Appendix I – Securing a network

Imagine a hub, adc://myhub.com:1234, which has an owner H, and users X, Y and Z. The users X and Y become concerned that the hub is insecure, and lobby H to move to a secure protocol. The secure protocol lobbied by X and Y involves a secure connection being established prior to any protocol-specific communication.

The problem (separate ports)

H agrees to move the hub to a secure protocol; however, since Z was absent from the conversation, H sees some potential issues:

1) To enable Z to still connect to the hub, H must use an additional port (e.g. 4321) for the secure connection. In addition, H must run two instances of the hub software, the legacy hub informing connecting users (i.e. Z) the address of the secure hub, and the new secure hub.
2) Users connecting to the secure hub must run client software which supports the secure protocol. If a user attempts connecting to the secure hub without security-enabled client software, there is no way for the hub to inform the user how to obtain such software, it simply rejects the connection attempt.
3) To indicate to users that it is secure, the secure protocol specifies that the hub address must now use the 'adcs' URL scheme, rather than 'adc'. This means that even if H decides to maintain the hub on port 1234, and exclude Z from the hub (or informs Z of the new address by other means); users X and Y must still change the address of the hub in their client software to adcs://myhub.com:1234.

The alternative (upward negotiation)

H then proposes a different secure protocol to X and Y. This protocol uses the same security mechanism as proposed by X and Y; however, the secure session is established after feature negotiation by the insecure protocol. This means that the hub can remain on the same address and port; however, the hub will now announce to connecting clients that it supports the secure protocol and, if the client supports the protocol too, a secure session may be established.

Note that support for the secure protocol by both the client and hub does not mean that a secure session must be established. The hub may require a secure session, and thus initiate it, or the client may request a secure session, and the hub may initiate it at the client's request. If the client requests a secure session and the hub ignores or denies the request, then the client may choose to terminate the connection if it requires a secure session. If the hub initiates a secure session and the client ignores the initiation, then the hub may terminate the connection.

If a hub enables clients to establish a secure session, it should force all clients to establish a secure session, at threat of session-termination. This is because if both secure and insecure sessions are allowed, information which one client sends securely becomes insecure when sent to an insecure client.

Issues

The alternative method allows an eavesdropper to detect the protocol being used in the connection. A consequence of this is that a network operator (e.g. Internet service provider or local area network administrator) may detect the protocol is in use and block the connection.