Managed Authentication

David Irvine, email: david.irvine@maidsafe.net, LifeStuff: David

maidsafe.net limited (registered in Scotland Sc 297540)

September, 2010 ***Abstract*—Today all known access mechanisms that grant access to distributed or shared services requires a server or authoritative control in some form. This presents many issues, including security, trust and privacy to name only a few. This paper presents a system of authentication that abolishes the requirements for any user-name or password containers as lists or similar. It also negates the necessity for any server based systems as a login entity for people to connect with prior to gaining access to a system for any reason. In addition this paper allows management of user acess in a distributed network.**

***Index Terms*—security, freedom, privacy, authentication**

CONTENTS

I. INTRODUCTION

AUTHENTICATION allows access to a system at a certain level or privilege. This is described in *Self Authentication* [1] with reference to an identity created, managed and maintained in seclusion, or in another way of thinking, anonymously. In many other situations this is not beneficial; control over access and the ability to manage very large groups is required. Examples include corporate and military networks.

A. Conventions Used

There is scope for confusion when using the term "key", as sometimes it refers to a cryptographic key, and at other times it is in respect to the key of a DHT "key, value" pair. In order to avoid confusion, cryptographic private and public keys will be referred to as $\mathsf{K_{priv}}$ and $\mathsf{K_{pub}}$ respectively, and DHT keys simply as keys.

- Node $\equiv$ a network resource which is a process, sometimes referred to as a vault in other papers. This is the computer program that maintains the network and on its own is not very special. It is in collaboration that this Node becomes part of a very complex, sophisticated and efficient network.
- $\mathsf{H} \equiv$ Hash function such as SHA, MD5, etc.
- $\mathsf{PBKDF2_{[Passphrase][Salt][IterCount]}} \equiv$ Password-Based Key Derivation Function or similar
- $\mathsf{XXX_{priv}}$, $\mathsf{XXX_{pub}} \equiv$ Private and public keys respectively of cryptographic key pair named XXX
- $\mathsf{AsymEnc_{[K_{pub}]}(Data)} \equiv$ Asymmetrically encrypt Data using $\mathsf{K_{pub}}$
- $\mathsf{AsymDec_{[K_{priv}]}(Data)} \equiv$ Asymmetrically decrypt Data using $\mathsf{K_{priv}}$
- $\mathsf{Sig_{[K_{priv}]}(Data)} \equiv$ Create asymmetric signature of Data using $\mathsf{K_{priv}}$
- $+ \equiv$ Concatenation
- STORE$\equiv$ Network or other key addressable storage system
- $\mathsf{PutV_{[Key]}(Value)} \equiv$ Store Value under Key on STORE. This value is signed.
- $\mathsf{GetV_{[Key]}} \equiv$ Retrieve Value under Key on STORE. This value is signed.
- $\mathsf{DelV_{[Key]}(Value)} \equiv$ Delete Value under Key on STORE. This value is signed.

II. OUTLINE

A. Creation of an Account

Here we will assume a User who is a member of an organisation named "Org" which has an ID on the system named $\mathsf{ID_{Org}}$ and a Manager with administator privileges. There are two inputs required by User: user-name U, and

password W. A salt S is also derived (in a repeatable way) from U and $\mathsf{ID}_{\mathsf{Org}}$. Manager can recreate this ID at any time, but requires that User cannot alter the user-name.

To generate a unique identifier, we hash the concatenation of the user-name and the salt, $\mathsf{H}(\mathsf{U}+\mathsf{S})$.

PBKDF2 is used here to strengthen any password keys used. This is required as user-selected passwords are inevitably weak and the user may not know the user-name is also used as a password in the system. Account specifies session data, like user details or an index of references to further data. This packet is located through an Access Packet holding a random number Rnd#.

It is important to note that all packets in this process are signed by two seperate IDs called $\mathsf{MID}_{\mathsf{User}}$ and $\mathsf{TMID}_{\mathsf{User}}$ which are created as described in [3]. These packets are in turn signed by Manager who keeps a copy of the IDs. In this way Manager can find the user's ID packets and delete them after deleting the data pointed to by the $\mathsf{TMID}_{\mathsf{User}}$ to revoke access to the system.

1) $\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]} \equiv \mathsf{False}$ (Ensure uniqueness)
2) Generate random number Rnd#
3) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}]}]}(\mathsf{Rnd}\#)\right)$ (Store Access Packet)
4) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#)]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}]}]}(\mathsf{Account})\right)$ (Store Account Packet)

B. Login / Load Session Process

1) $\mathsf{SymDec}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}]}]}\left(\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]}\right) \equiv \mathsf{Rnd}\#$
2) $\mathsf{SymDec}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}]}]}(\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#)]}) \equiv \mathsf{Account}$

For the following operation, Rnd# should be kept locally for the duration of the session.

C. Logout / Save Session Process

1) Generate new random number $\mathsf{Rnd}\#_{\mathsf{new}}$
2) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#_{\mathsf{new}})]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}]}]}(\mathsf{Account})\right)$ (Store new Account Packet)
3) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}]}]}(\mathsf{Rnd}\#_{\mathsf{new}})\right)$ (Update Access Packet)
4) $\mathsf{DelV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#)]}(\mathsf{OldAccount})$ (Delete old Account Packet)

D. Fallback Account Packets

The previous sections outlined the basic system of authentication. However, this can be extended for safety reasons. As with any system, the serialisation and store operation of the account data can fail for any reason, resulting in unreadable data upon retrieval. This would be catastrophic as access to the user's data on the system may be rendered impossible. To reduce any such risk, a fallback copy of an Account Packet (and its Access Packet) is kept, to allow reverting to the previous version in case the current version can't be restored or turns out to have been generated erroneously. Like the main Access Packet, the fallback Access Packet contains an (encrypted) random number, designated $\mathsf{Rnd}\#_{\mathsf{fallback}}$.

1) Updated Account Creation Process:

1) $\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]} \equiv \mathsf{False}$ (Ensure uniqueness of Access Packet)
2) $\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+(\mathsf{S}-1))]} \equiv \mathsf{False}$ (Ensure uniqueness of fallback Access Packet)
3) Generate random number Rnd# and copy $\mathsf{Rnd}\# \rightarrow \mathsf{Rnd}\#_{\mathsf{fallback}}$
4) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}]}]}(\mathsf{Rnd}\#)\right)$
5) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+(\mathsf{S}-1))]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}-1]}]}(\mathsf{Rnd}\#)\right)$
6) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#)]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}]}]}(\mathsf{Account})\right)$

In this case, the random numbers in the Access Packets are the same, thus point to the same (unique) Account Packet. Fallback packets are only kept once the Account Packet is updated.

2) Updated Login Process:

1) if $(\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]} \equiv \mathsf{EncRnd}\#)$
   a) $\mathsf{SymDec}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}]}]}(\mathsf{EncRnd}\#) \equiv \mathsf{Rnd}\#$
   b) $\mathsf{SymDec}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}]}]}\left(\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#)]}\right) \equiv \mathsf{Account}$
2) else (or if previous attempt failed)
   a) $\mathsf{SymDec}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}-1]}]}\left(\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+(\mathsf{S}-1))]}\right) \equiv \mathsf{Rnd}\#_{\mathsf{fallback}}$
   b) $\mathsf{SymDec}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}-1]}]}\left(\mathsf{GetV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#_{\mathsf{fallback}})]}\right) \equiv \mathsf{Account}_{\mathsf{fal}}$

3) Updated Logout / Save Session Process: Designate existing Rnd# as $\mathsf{Rnd}\#_{\mathsf{old}}$ and $\mathsf{Rnd}\#_{\mathsf{fallback}}$ as $\mathsf{Rnd}\#_{\mathsf{fallback\,old}}$

1) Generate new random number $\mathsf{Rnd}\#_{\mathsf{new}}$
2) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S})]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}]}]}(\mathsf{Rnd}\#_{\mathsf{new}})\right)$ (Update Access Packet with new random number)
3) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+(\mathsf{S}-1))]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{U}][\mathsf{S}-1]}]}(\mathsf{Rnd}\#_{\mathsf{old}})\right)$ (Update fallback Access Packet with old random number)
4) $\mathsf{PutV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#_{\mathsf{new}})]}\left(\mathsf{SymEnc}_{[\mathsf{PBKDF2}_{[\mathsf{W}][\mathsf{S}]}]}(\mathsf{Account})\right)$ (Update Account Packet using new random number)
5) $\mathsf{DelV}_{[\mathsf{H}(\mathsf{U}+\mathsf{S}+\mathsf{Rnd}\#_{\mathsf{fallback\,old}})]}(\mathsf{Account}_{\mathsf{fallback\,old}})$ (Delete old Account Packet)

The previous Account Packet remains untouched. Instead the fallback Access Packet is redirected to it and the normal Access Packet points to the new Account Packet. The previous fallback Account Packet is deleted. This is a security measure to hinder slow brute-force attacks on decrypting the Access Packet, which by the time the clear-text random number is obtained would make it obsolete.

E. Creating a User Identity (publically broadcastable identity)

1) Manager creates $\mathsf{USER}_{\mathsf{priv}}$ and $\mathsf{USER}_{\mathsf{pub}}$
2) $\mathsf{H}(\mathsf{USER}_{\mathsf{pub}} + \mathsf{Sig}_{\mathsf{MANAGER}_{\mathsf{priv}}})$ is stored on STORE; this also will be the ID for User.
3) Manager creates a $\mathsf{H}(\mathsf{User@Org} + \mathsf{Sig}_{\mathsf{MANAGER}_{\mathsf{priv}}})$ packet in STORE (this is the contact address of User when connected with Org); again this is revokable by Manager.

It should be emphasised that any communcation from User@Org should require a check of STORE to ensure the ID is still valid; therefore there is no possibility of simply passing the information in a message. This may involve a recursive check on Manager's keys to get to the root Org

packet which is the packet signed by the Org key itself (first user). This is identified as being signed by the $\mathsf{ID}_{\mathsf{Org}}$ packet. This ID will be published in many places and found in STORE as a packet simply called Org and signed by Org (self-signed original packet).

F. Creating User Root Directory

1) Manager creates a parent directory for User as per [6].
2) Manager adds a child directory entry to the parent.
3) This child directory entry is the organisation drive.

In this manner the user has the ability to alter the child directory and add in his or her own files. This is the root drive of User, and personal information can be stored there. Manager though, has access to this as he or she knows the ID and the passkey to it. In addition, as Manager stores and signs the parent directory, User cannot alter this or change passwords.

G. BootStrapping the Network

An initial ID is required to bootstrap the network of ID's in the store. This ID is created as described in [3] as a *selectable ID*[1]. This ID may be countersigned by another party, such as the company who supplies such a system to a customer[2], assuming that supplier is trusted. In the case of a software supplier, then the suppliers ID can be built in as a trusted party.

This ID is then the *root* ID of the network, we shall refer to this as ID_{Net}. The initial user then creates a $\mathsf{ID}_{\mathsf{User@Org}}$. This ID is signed by the private of the ID associated with ID_{Org}.

Fact 1. *All client systems accessing this network will test* $\mathsf{ID}_{\mathsf{User@Org}}$ *is signed by the ID associated with* ID_{Org}*. This is to prevent fraud as any attacker or ID wishing to fraudulently claim they are a valid* $ID_{something@Org}$ *would not have access to the ID associated with* ID_{Org}*.*

Remark 2. This is essential as many nodes or network entities may create a $ID_{something@Org}$ which cannot and should not be prevented, however only one will have the private key required to validate this ID. This ensures mathematical correctness in testing a proposition such as $ID_{something@Org}$ which ensures that the security algorithm is intact and not dependent on obfuscation or obscurity of code or similar.

To check an ID then there is a possibility that the ID who signs the $ID_{something@Org}$ is in fact not ID_{Org} but another $ID_{something@Org}$ (this shoudl be a manager of teh first ID checked). This requires a recursive check until ID_{Org} is reached, otherwise the $ID_{something@Org}$ is fraudulent and not as expected.

On confirming $ID_{something@Org}$ it is assumed that the recipient will maintain the ID associated and possibly sign a validated record and maintain this somehow[3].

[1]This is the major claim in the managed distributed authentication patent
[2]This is the check that MS SAN uses to ensure license payments are made
[3]In MS SAN and MaidSafe this is stored in the data atlas.

H. Banning User from Network

As $\mathsf{ID}_{\mathsf{User}}$ is signed by Manager, then Manager can delete this packet. This renders User's access key useless and all access revoked. To ensure this is absolute, the message type sent cannot be used to verify an ID (by sending public key plus signature in a packet for hashing by the recipient) and instead a STORE lookup for a valid ID has to happen every time.

In addition the user's ability to access any data from the network is revoked with the deletions of the $\mathsf{MID}_{\mathsf{User}}$, $\mathsf{SMID}_{\mathsf{User}}$ and $\mathsf{TMID}_{\mathsf{User}}$ packets.

I. N Plus P Key Sharing

As the initial node has an extreme amount of power and no equal to absorb this power should that identity become unstable for whatever reason (such as retiring, leaving or simply becoming hostile) then a mechanism must be put into place to ensure this ID is not stand alone.

The answer is n+p[4] key sharing scheme where a user can select p users to share his key with. If there is ever an issue, then n of these users are all that are required to recreate the key.

III. Hardened security mode

As this system allows access to a distributed file-system such as [6] we can achieve a level of security unknown until now. This file-system can be presented through a File System In Userspace (FUSE) format (or through a system that can reconstitute files in real time and display them, such as the light versions of LifeStuff [tm] and mssan), thereby appearing like a hard disk to the operating system. On this disk can run any system that runs on a normal hard disk (with the exception of extreme fast transaction services[5]). With this being the case, even applications can be installed on the disk as shares the users can access and presented through a menu system as one would normally expect to see on a computer.

If such systems were built that had no external access capabilities (USB, FireWire, Bluetooth etc.) and had a read only file system (which is possible as this system does not require disk access at all) then a extreme level of security could be obtained. All applications managed as well as virtual disk access with no virus capability or no way to write to the hard disk (no trojans) and no requirement for a firewall or intrusion detection system as there is no hard disk.

Such system would find use in military, financial institutes and organisations requiring 100% control and protection of internal data.

A. Multi factor authentication

In some situations there may be a requirement to present more than one method of authentication. As described this

[4]There are many methods for achieving this, Shamir's scheme is as good as any and an implemention of this is availiable in many crypto libraries such as cryptopp.
[5]This is only a current limitation and can be overcome even today with a NOSQL policy

system presented makes use of a User name, PIN and password combination, which may or may not require the PIN to be entered. This does not however provide security of theft of the user entered components, i.e. via a key logger, over the shoulder attack or similar. Another factor is required.

Such a system may be a mobile device such as a phone with a unique serial number (SIM card) that can be registered to the ID in question. Therefore an organisation may provide access to certain ID's for certain privileges, there may also be a requirement for a further check.

In this case a simple solution would be to map the user ID to a telephone number or similar and when access is requested and validated via the encryption keys, a message can be sent to the device. The application at login or access screen will await a code (encrypted with the public key of the ID presented, to prevent man in the middle attacks) to be sent from the organisation to the device. On receipt the application will send this pass-phrase back to the organisation, thereby validating against something the user has (phone) as well as what he knows (user name/password).

IV. Conclusions

This paper presents a mechanism for managed access to organisations networks that may exist completely distributed. From this point forward there is occasion for many additional features to be added to this system. If combined with a DHT [5], distributed databases[6] and a network such as the maidsafe network[4], then this could represent a new paradigm for corporate computing. This would enable remote working, network scalability and security at a level never seen previously.

V. Future Works

A. Time Based Obfuscation

In the previous sections a system of authentication was detailed which is very effective. A potential failure point, however, may be the Access Packets, as those are never altering their location (key) and can pose a target for any attacker who is monitoring data traffic between the user and the system.

To remove this weakness, a predictably altering piece of data can be introduced, such as time (e.g. week number, day of year or similar). In this case in II-B the GetV call would be iterative, starting at the current time slot and decrementing until it returns with a value. Access Packets at "outdated" locations would be deleted on detection, and updates always stored in the current location, resulting in regularly "moving" packets.

References

[1] David Irvine, Self-Authentication, david.irvine@maidsafe.net
[2] David Irvine, Self Encrypting Data, david.irvine@maidsafe.net
[3] David Irvine, "Peer to Peer" Public Key Infrastructure, david.irvine@maidsafe.net
[4] David Irvine, maidsafe: A new networking paradigm , david.irvine@maidsafe.net
[5] David Irvine, Autonomous Network, david.irvine@maidsafe.net
[6] David Irvine, MaidSafe Distributed File System, david.irvine@maidsafe.net

David Irvine is a Scottish Engineer and innovator who has spent the last 12 years researching ways to make computers function in a more efficient manner.

He is an Inventor listed on more than 20 patent submissions and was Designer / Project Manager of one of the World's largest private networks (Saudi Aramco, over $300M). He is an experienced Project Manager and has been involved in start up businesses since 1995 and has provided business consultancy to corporates and SMEs in many sectors.

He has presented technology at Google (Seattle), British Computer Society (Christmas Lecture) and many others.

He has spent many years as a lifeboat Helmsman and is a keen sailor when time permits.