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 $\mathsf{K}_{\mathsf{priv}}$ and $\mathsf{K}_{\mathsf{pub}}$ 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.
- $\mathsf{PBKDF2}_{[\mathsf{Passphrase}][\mathsf{Salt}][\mathsf{IterCount}]} \equiv$ Password-Based Key Derivation Function or similar
- $\mathsf{XXX}_{\mathsf{priv}}$, $\mathsf{XXX}_{\mathsf{pub}} \equiv$ Private and public keys respectively of cryptographic key pair named XXX
- $\mathsf{AsymEnc}_{[\mathsf{K}_{\mathsf{pub}}]}(\mathsf{Data}) \equiv$ Asymmetrically encrypt Data using $\mathsf{K}_{\mathsf{pub}}$
- $\mathsf{AsymDec}_{[\mathsf{K}_{\mathsf{priv}}]}(\mathsf{Data}) \equiv$ Asymmetrically decrypt Data using $\mathsf{K}_{\mathsf{priv}}$
- $\mathsf{Sig}_{[\mathsf{K}_{\mathsf{priv}}]}(\mathsf{Data}) \equiv$ Create asymmetric signature of Data using $\mathsf{K}_{\mathsf{priv}}$
- $+ \equiv$ Concatenation
- STORE$\equiv$ Network or other key addressable storage system
- $\mathsf{PutV}_{[\mathsf{Key}]}(\mathsf{Value}) \equiv$ Store Value under Key on STORE. This value is signed.
- $\mathsf{GetV}_{[\mathsf{Key}]} \equiv$ Retrieve Value under Key on STORE. This value is signed.
- $\mathsf{DelV}_{[\mathsf{Key}]}(\mathsf{Value}) \equiv$ 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 $\mathsf{ID}_{\mathsf{Org}}$ 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], but uses the hash of the $\mathsf{ID}_{\mathsf{User}}$ from II-A as the passkey.
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 parent 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. This is confirmed by the Manager's system periodically; failure to access this would render User's ID locked and access to the network removed.

G. Further Enhancements

1) Time Based Obfuscation: In the previous sections a system of self 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.

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

III. Conclusions

T

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.