"Peer to Peer" Public Key Infrastructure

David Irvine, email: david.irvine@maidsafe.net (LifeStuff: David). Reviewers: Alison Shaw, email: alison.shaw@maidsafe.net (LifeStuff: Alison); Dan Schmidt Valle, email: dan.schmidt@maidsafe.net (LifeStuff: Dan)

maidsafe.net limited (registered in Scotland Sc 297540)

September, 2010 (Revised August 2011) ***Abstract*—We present a system of validation that utilises asymmetric encryption to create a Public Key Infrastructure (PKI) which requires no servers or centralised control. This system provides a mathematically secure method of validation that can be employed in any modern network, especially distributed networks and overlay networks such as Distributed Hash Tables (DHTs). We also consider the system's ability to accommodate user-selected names.**

***Index Terms*—security, freedom, privacy, DHT, encryption**

CONTENTS

I. INTRODUCTION

VALIDATION of identity is essential for any communicating parties that wish to have confidence in the identity of the other party and the information that they communicate: the absence of an effective identity verification system may allow other entities to intercept and alter communications, or masquerade as one of the parties, without either party being aware of the problem.

Public key infrastructures (PKIs) aim to facilitate authentication of the source and integrity of a piece of information. Checking the integrity of a message uses public key cryptography and digital signing, of which section II-A provides a brief reminder. Hence a crucial responsibility of a PKI is to provide a means of verification of an identity and its association with the key-pair used for public key cryptography.

There are two main methods of identity validation provided by the PKIs of today. The first method relies on the trustworthiness and security of a central authority to provide certification of identities, so any compromise of this central authority impacts our ability to verify identities and the information they communicate. The second method, referred to as a web of trust, provides validation of identity based on how trustworthy an entity's peers perceive it to be. A web of trust is vulnerable to attacks where many untrustworthy entities rate each other highly then surround a good identity and intercept or hide information. Thus both of these systems are open to considerable abuse and neither is able to provide a secure system allowing the free creation of identifiable nodes.

The system presented in the paper relies on neither a central authority nor a web of trust, instead enabling entities to validate identities only with the use of mathematical operations. The vital distinguishing features of the system that allow it to operate in this way are as follows:

- An entity is equipped with two key-pairs, the first pair allocated to the usual digital signing of messages and the second pair designated to identity verification procedures.
- An entity's identity is derived directly from these key-pairs.

Such a system, which avoids the pitfalls of the usual central authentication or web of trust, has been the subject of many a wish list[1] for some time now.

In general, our system works in conjunction with a key addressable storage system, which sections III-A, III-B and III-C explain. Section III-D illustrates the system's potential for use in the absence of such storage, and section IV explores the system's potential in the context of DHTs. Section V builds on the case where we have storage, exploring the system's ability to cater for user-chosen names. Section VI explores

some ways of altering and extending the system to provide further functionality.

II. Background concepts and notation

We provide a brief overview of some core concepts used in this paper. Throughout this paper we will use the following notation, which is explained in more detail in this section:

- In the context of asymmetric encryption, we will use Kpub to denote a public key and Kpriv to denote the corresponding private key. See section II-A for more information.
- We will repeatedly make use of a hash function; we will denote the hash of a piece of data M by $\mathsf{Hash}(\mathsf{M})$. See section II-B for more information.
- The digital signature of a piece of data M using a private key Kpriv will be denoted by $\mathsf{Sig}_{\mathsf{Kpriv}}(\mathsf{M})$. See section II-C for more information.

A. Asymmetric public key encryption

This paper makes use of public key cryptography, a system of encryption which does not require passwords or keys to be distributed but instead allows the publication of a *public key*, denoted by Kpub. This key can be thought of as the encryption key, where any data encrypted by this key can only[1] be decrypted by the holder of the corresponding *private key*, denoted by Kpriv. This private key can hence be considered as the key for 'unlocking' the data. The reverse procedure is also valid: data can be decrypted using the private key then encrypted using the public key to recover the original data. Since everyone has access to the public key this may seem strange and useless, but we will see in section II-C that such key-pairs can be used to excellent effect.

B. Hash functions

We can give a *hash function* a piece of data and it returns a binary string of fixed length, which we will call the *hash* of the data. The hash of a piece of data can be thought of as a digital fingerprint for several reasons. Firstly, just as a person's fingerprints do not change of their own accord, the hash of a piece of data will never change. Secondly, just as a fingerprint of a person is almost certainly unique, the hash of a piece of data is also almost certainly unique. Thirdly, just as we cannot reconstruct or reveal a person from their fingerprint, we cannot reconstruct or reveal a piece of data from its hash. With both fingerprints and hashes there is the possibility of non-uniquness: if two distinct pieces of data have matching hashes we say they have *collided* and that we have a *hash collision*. The strength of a hash function depends on how low the chance of collisions is and how difficult it is to find pieces of data whose hashes collide. The way in which a hash function works should attempt to produce very different hashes for seemingly similar pieces of data; the binary strings it produces should also be long enough to provide a sufficiently low chance of collisions.

[1]Here we assume the algorithm and its implementation are perfect, which is unlikely to be the case in reality.

Early hash algorithms such as MD4, MD5 and even early SHA allow too many collisions to occur and are thus regarded as broken. Efficient hash functions that produce longer hashes are desirable in cases where a high probability of uniqueness is required. Henceforth we will assume that the hash function we use is strong enough to assign each piece of a data a unique hash[2]. We refer the reader to [6] Ch1. p.33 for more information about hash functions.

C. Digital signatures

Using the ideas of asymmetric encryption (see II-A) and hashing (see II-B), we now outline the process of *digital signing*:

Suppose person A wishes to send a message M to person B. We equip person A with a key-pair consisting of a private key Kpriv and a public key Kpub, the latter of which is publicly available for person B to use. It is in the interests of both parties for person B to be able to ensure that the message M was definitely sent by person A and to detect if it has been changed during its transmission. To facilitate this, person A can hash the message M and then decrypt it using their private key; the result of these operations is called the *digital signature* of the message M. Person A then sends both M and its digital signature to person B. To check that the message M has arrived intact and unchanged, person B does three things: firstly, he encrypts the digital signature using person A's public key to recover the hash of message M; secondly, person B hashes the message he received. Finally, person B compares these two hashes - if they differ, either the message M or its signature has been altered. If the hashes are the same, the message received by person B is cryptographically guaranteed to be the piece of data to which the signature refers, and therefore it is the message M which person A sent. Note that we rely on the hash function to have a very low probability of collisions, otherwise our guarantee is weakened.

Throughout, the signature of a piece of data M using private key Kpriv will be denoted by $\mathsf{Sig}_{\mathsf{Kpriv}}(\mathsf{M})$.

III. Implementation

This validation system requires manipulation of key-pairs, as do all of today's cryptographically secure validation systems. One of the fundamental tenets of this paper is that these key-pairs are themselves used to generate the identity rather than being later tied to an identity (see section III-A). We present two versions of the system: one that provides identity validation by using a key addressable storage system[3] (see section III-C) and one that works without storage and instead lets identities be validated by message-passing (see section III-D). In both versions, from a security perspective it is most thorough to verify the sender's identity for every message, but this may be too slow for some implementations, so it

[2]In practice, producing unique hashes of data is impossible because the only way to avoid collisions is to make the fixed length of the hashes longer than any known piece of data.

[3]A key addressable storage system allows data values to be stored in a location labelled by a key. This key can be looked up to see what its associated value is.

might instead be preferable to check the validity of identities at random, sacrificing some thoroughness for speed.

The system described in section III-C will operate equally well with any kind of key addressable storage system, for example databases or distributed hash tables (DHTs). Although there may be other kinds of storage that our system could be used in conjunction with, the kind of storage used could influence its effectiveness.

A. Linking identity to key-pairs

We wish to create an identity that is inherently connected to the key-pair we will use for communication, rather than artificially 'tying' an otherwise unrelated identity to our key-pair. A relatively simple way to achieve this uses one key-pair (consisting of Kpriv and Kpub) and defines the identity by $\mathsf{Identity} = \mathsf{Hash}(\mathsf{Kpub})$. This ensures the identity is mathematically linked to a key-pair rather than being later tied to one.

Given the possibility that an identity can be compromised, we have an incentive to introduce a structure that enables us to prove our identity to others (and to prevent other entities from being able to steal or abuse our identity). This is achieved by using key-pairs as follows: the first key-pair (*key-pair 1*, consisting of Kpriv_1 and Kpub_1) will be used for our usual communication purposes, and we introduce a second key-pair (*key-pair 2*, consisting of Kpriv_2 and Kpub_2) whose purpose will be to validate our identity. Throughout, it is understood that the private key Kpriv_2 is never placed in a position where it can be obtained by other parties, so the owner of Kpriv_2 corresponds correctly to the identity. To connect our key-pairs with our identity, we define the identity as follows:

$$\mathsf{Identity} = \mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)) \qquad (1)$$

Note that we have not explicitly defined how '+' works, although its use should be specified and consistent in practice. There are various options for its definition, for example it could represent concatenating the two values involved. Since hashing the value $\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$ produces the identity, we will henceforth refer to this value $\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$ as the *identity precursor*.

We will demonstrate in sections III-C and III-D how this definition of the identity allows it to be validated. Section III-C considers the scenario where we have a key addressable storage system on which we can store information, and section III-D considers the case when we do not have such a storage facility at our disposal.

A drawback of defining the identity as in equation (1) is that the identity may be cumbersome to use in practice, because strong hash functions produce very long hashes. However, in several cases, the system would be a good enough method, for example, in applications such as telephone numbers or bar-codes on products. We will discuss the possibilities for a more user-friendly identity in section V.

B. Addition of an identity revocation method

For identity revocation to operate, we require a key addressable storage system. This storage system should retain all information as immutable unless the signer of the data requests amendment or deletion, the reasons for which we will discuss shortly. The creator of the identity stores the following information on the key addressable storage system:

Key:	Value:
$\mathsf{Identity}$	$\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$

This key and value pair will henceforth be referred to as the *identity packet*, or simply the *packet*. Observe that the value stored is the identity precursor $\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$, so hashing this value produces the identity. The validity of an identity is checked by looking the identity up in the key addressable storage system and checking that the hash of the stored value is equal to the identity. A person wishing to revoke their identity could delete its associated packet on the key addressable storage system, but this means that we can't distinguish between non-existent and revoked identities, so it is preferable to replace the public key in the stored value with a false key (for example, consisting of all 0s) so that the hash of the stored value does not equal the identity and the identity can be recognised as being invalid.

It is necessary that only the holder of Kpriv_2 is able to alter the identity packet, otherwise other parties could invalidate or remove this identity without the holder of Kpriv_2's permission. Far worse, suppose an attacker somehow compromised the identity and it was revoked. If the attacker knew the identity's associated identity precursor, they could edit the identity packet and restore it its previous state, incorrectly presenting the identity as valid.

C. Using and validating an identity using a key addressable storage system

An identity is defined as in equation (1). As in section III-B, the creator of the identity stores an identity packet in a key addressable storage system, with key equal to the identity and value equal to the identity precursor, so the hash of the value equals the key. Consider a message, allegedly from this identity, formatted as shown:

$\mathsf{Identity}$	
$\mathsf{Payload}$	$\mathsf{Sig}_{\mathsf{Kpriv}_1}(\mathsf{Payload})$ (optional)

To validate the identity of the sender, we look up the identity in the key addressable storage system as described in section III-B and check that the hash of the stored value equals the identity.

Additionally, if the optional signature $\mathsf{Sig}_{\mathsf{Kpriv}_1}(\mathsf{Payload})$ of the payload is included then digital signing, as described in section II-C, enables us to verify that they payload has not been altered in transmission.

D. Using and validating an identity using message passing

In this case we do not use a key addressable storage system and will instead validate the identity of a message sender by passing messages. An identity is defined as in equation (1). Consider a message, allegedly from this identity, formatted as shown:

Identity	
Kpub_1	$\mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$
Kpub_2	$\mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_2)$
Payload	$\mathsf{Sig}_{\mathsf{Kpriv}_1}(\mathsf{Payload})$ (optional)

Upon receiving this message there are several things we can and should check. Is the identity equal to $\mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1))$? We have Kpub_2, so are the signatures of Kpub_1 and Kpub_2 correct? If any of these checks fail then we have cause for concern: perhaps information has merely been communicated incorrectly, but maybe the entity we are communicating with is not trustworthy. In the former case we may not be able to communicate correctly with the sender, e.g. if one of the keys is wrong. In the latter case we do not wish for communication to continue. Either way, correspondence should halt.

If the above checks all pass then, in particular, we know that Kpub_2 is correct and we can use it to encrypt a challenge for the sender and, as they should be the holder of Kpriv_2, they should be able to decrypt our challenge and respond correctly to it. Alternatively we could send them a piece of information and request that they sign it using Kpriv_2, then verify their response using Kpub_2. A correct response from either of these options verifies that the sender has Kpriv_2 and is therefore trustworthy.

If the optional signature of the payload is included, we also have the ability to verify that the payload has been transmitted correctly.

IV. Combined with DHT

All distributed hash table (DHT) networks have a particular requirement in common: to create a network address that is unique. There are various techniques for achieving this, for example Chord[2] uses the hash of the IP/PORT combination and Kademlia[3] uses a random hash.

Using the method outlined in section III-C and storing the identity packet on the network ensures uniqueness of identity and provides a storage mechanism for finding a node's public key. To implement this, we firstly look up $\mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1))$ in the DHT to check for entries with the same key. If there are no such entries, we store the identity precursor value $\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$ at the key $\mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1))$. For completeness, the hash size chosen should be of the same length as the keys used in the network addressing scheme. The system described can be used to send encrypted information to that node or to validate a signature from that node.

If such a system were implemented in a DHT or similar publicly connected system (as web servers are), then the security of the private key Kpriv_1 is paramount. In today's networks this is achieved via brute force techniques such as firewalls, secured hard drives, private key passwords (which make automatic reboots of a server require human intervention) and other custom approaches. In the system presented there is no obvious improvement in the security of Kpriv_1, however, the system is unique in offering the ability to reduce the effectiveness or scope of the private key Kpriv_1.

V. Selectable identities

The method described in section III-C is extremely secure and resilient to a cryptographic attack, but does suffer form one major deficiency: in this system the identity is a long binary string because it is defined as $\mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1))$. Such strings are not easily readable for a human and are certainly too long to be communicated comfortably on paper or verbally. We would like to enable users to choose a desired and more manageable 'chosen name', then secure it to their usual identity (provided the chosen name is not already in use by someone else).

The functionality and security of the existing system revolve around the storage of the identity packet defined in section III-C in a key addressable storage system, so any way of associating a chosen name with the identity should not alter this characteristic.

A way of securing a chosen name to an identity is to store the following key/value pairs on the key addressable storage system:

Key:	Value:
$\mathsf{Hash}(\mathsf{Chosen\ name})$	$\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$
Identity	$\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$

The first row of the table could optionally be stored elsewhere, provided it is immutable to everyone except its creator.

We would communicate using our chosen name rather than using our identity. To revoke our identity we would change the identity packet as described in section III-B. To validate a chosen name and its associated identity we would look up $\mathsf{Hash}(\mathsf{Chosen\ name})$ then use the identity precursor to find the identity packet and validate the identity as usual - if the identity is not valid then neither is its associated chosen name.

If a person's identity is compromised, they can revoke it then create a new identity and put the new identity packet on the key addressable storage system. They can then alter the value addressed by the key $\mathsf{Hash}(\mathsf{Chosen\ name})$ to correspond to this new identity, so they are able to continue communication using their chosen name.

VI. Adaptations

This section presents some ways in which the system described could be adapted to offer greater functionality. Both ideas rely on a more complex key addressable storage system than previously required, hence these options are not necessarily attainable with more basic forms of the system.

A. Sharing chosen names

Section V presents an extention of the current system that enables an entity to communicate using a more manageable 'chosen name'. However, this requires an entity's chosen name to be unique to them, potentially making it difficult to choose a name that is simultaneously short, easy to remember and unique. This section provides a way for multiple entities to share a chosen name and be distinguished by additional information, e.g. geographical location or date of birth. For the sake of example we will suppose that entities with the same chosen name will use their geographical location to

distinguish themselves; in practice it may be desirable to use several types of information (e.g. use both location and date of birth) or allow entities to supplement their chosen name with information of an unrestricted nature.

Given an entity x, we denote the keys of their first key-pair by $\mathsf{Kpub}_{(x,1)}$ and $\mathsf{Kpriv}_{(x,1)}$, and analagously $\mathsf{Kpub}_{(x,2)}$ and $\mathsf{Kpriv}_{(x,2)}$ denote the keys of their second key-pair. For brevity, the identity associated with x is written as ID_x and is defined by $\mathsf{ID}_x = \mathsf{Hash}(\mathsf{Kpub}_{(x,1)} + \mathsf{Sig}_{\mathsf{Kpriv}_{(x,2)}}(\mathsf{Kpub}_{(x,1)}))$. This is the same as in section III-A, except the notation that allows us to distinguish between entities. The location of the entity x will be denoted by $\mathsf{location}_x$.

To allow entities a, b, c, etc. to share the name Chosen name, they each store their signed identity plus location on the key addressable storage system as follows:

Key:
$\mathsf{Hash}(\text{Chosen name})$
Values:
$\mathsf{ID}_a + \mathsf{location}_a + \mathsf{Sig}_{\mathsf{Kpriv}_{(a,1)}}(\mathsf{ID}_a + \mathsf{location}_a)$
$\mathsf{ID}_b + \mathsf{location}_b + \mathsf{Sig}_{\mathsf{Kpriv}_{(b,1)}}(\mathsf{ID}_b + \mathsf{location}_b)$
$\mathsf{ID}_c + \mathsf{location}_c + \mathsf{Sig}_{\mathsf{Kpriv}_{(c,1)}}(\mathsf{ID}_c + \mathsf{location}_c)$
etc.

Each entity also stores their usual identity packet, for example entity a would store the following:

Key:	Value:
ID_a	$\mathsf{Kpub}_{(a,1)} + \mathsf{Sig}_{\mathsf{Kpriv}_{(a,2)}}(\mathsf{Kpub}_{(a,1)})$

Suppose we want to find and verify a's identity. We ask for their chosen name and location, then look $\mathsf{Hash}(\text{Chosen name})$ up in the key addressable storage system. We have demanded that different entities sharing a chosen name cannot have the same location, so we can use $\mathsf{location}_a$ to find ID_a. We use ID_a to find their identity packet, which we check the validity of as usual. The packet contains $\mathsf{Kpub}_{(a,1)}$ which we can use to verify the signature of the value stored at $\mathsf{Hash}(\text{Chosen name})$.

The reliability of this system depends on the following property of the key addressable storage system: only the entity who signed a value can change or remove it. The key addressable storage systems discussed previously have not had multiple entities storing values under the same key and would not necessarily cater for the ideas presented in this section.

B. Validity provision

In this section we consider a way of allowing other parties to associate information with an entity - for example, if a person has paid for six months' membership to a website, it would be useful for the website administrators to be able to associate confirmation of this membership with the person, and after six months revoke the information.

For brevity we will write ID to denote the identity. For a party X with private key Kpriv_X and public key Kpub_X we will write info_X to denote the information that they want to associate with an entity.

We store the identity packed as usual:

Key:	Value(s):
ID	$\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$

To add functionality we associate the identity with a 'chosen name' and allow parties A, B, C etc. to add values under the key $\mathsf{Hash}(\text{Chosen name})$ as follows:

Key:
$\mathsf{Hash}(\text{Chosen name})$
Values:
$\mathsf{Kpub}_1 + \mathsf{Sig}_{\mathsf{Kpriv}_2}(\mathsf{Kpub}_1)$
$\mathsf{ID} + \mathsf{info}_A + \mathsf{Sig}_{\mathsf{Kpriv}_A}(\mathsf{ID} + \mathsf{info}_A)$
$\mathsf{ID} + \mathsf{info}_B + \mathsf{Sig}_{\mathsf{Kpriv}_B}(\mathsf{ID} + \mathsf{info}_B)$
$\mathsf{ID} + \mathsf{info}_C + \mathsf{Sig}_{\mathsf{Kpriv}_C}(\mathsf{ID} + \mathsf{info}_C)$
etc.

The chosen name and identity are related as in section V. To check the validity of, for example, info_A, we use Kpub_A to check that it has been signed correctly. The inclusion of the identity in the stored values ensures that the information from party A is tied to this entity and cannot be copied or abused by other entities. However, it may be in the interests of party A to include an expirey date in info_A so that the identity can't replicate the value and abuse it.

The ideas in this section again rely on a key addressable storage system where stored values can only be changed or removed by the entity that has signed them. A necessary disadvantage of this scenario is that the key $\mathsf{Hash}(\text{Chosen name})$ could accumulate values that the identity cannot change or remove.

VII. Conclusions

The system presented in this paper is ground-breaking in its ability to enable the creation of secure PKI networks that don't rely on a central authority or web of trust, instead allowing entities to validate identities autonomously.

Sections III-A, III-B and III-C have together constituted the system in the presence of key addressable storage. Section V has extended this system to allow the basic use of user-chosen names, and adaptations of this extension may have further benefits to offer, as section VI has illustrated.

Section IV has considered some additional possibilities for this form of the system in the context of a DHT; we refer the reader to the MaidSafe paper on distributed hash tables for a more thorough consideration of our system in this situation. It has not been within the scope of this paper to explore the system's abilities when used in conjunction with other specific types of key addressable system.

Sections III-A and III-D have together demonstrated the system's basic abilities in the absence of a key addressable storage system, although this case may also have potential for additional functionality.

Practical examples of possible utilisations of the system include:

- Identification of credit card data linking the identity to a known name in another secure location. People could have a card and a revocation card or, perhaps preferably, directly use key-pairs as described in this paper.
- Single continuous validation systems where a known identity can be used across multiple web sites or on-line systems that require history to operate effectively.

We refer the reader to the MaidSafe Self Authentication paper for further practical applications of the theory presented in this paper.

REFERENCES

[1] As described by Van Jacobson in this link below, August 30, 2006 http://video.Google.com/videoplay?docid=-6972678839686672840

[2] Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan,Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

[3] Petar Maymounkov and David Mazirese Kademlia: A Peer-to-peer Information System Based on the XOR Metric {petar,dm}@cs.nyu.edu http://Kademlia.scs.cs.nyu.edu

[4] David Irvine, maidsafe: A new networking paradigm, david.irvine@maidsafe.net

[5] David Irvine, MaidSafe Distributed File System, david.irvine@maidsafe.net

[6] A. Menezes, P. van Oorschot & S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996, www.cacr.math.uwaterloo.ca/hac

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.