# "Peer to Peer" Public Key Infrastructure

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

maidsafe.net limited (registered in Scotland Sc 297540)

September, 2010 ***Abstract*—This paper presents a system of validation that utilises asymmetric encryption to create a Public Key Infrastructure (PKI) in a manner that requires no servers or centralised control. This method provides an extremely coherent and mathematically secure method of validation that can be employed in any modern network or system, but is very well suited to distributed networks and in particular overlay networks such as Distributed Hash Tables (DHTs).**

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

CONTENTS



LIST OF TABLES



## I. INTRODUCTION

VALIDATION is a requirement for any connecting parties who are to exchange information to ensure uniqueness, proof of identity and generally any situation where the information requires verification or authentication. This paper presents a system that employs many of the techniques used today, such as asymmetric encryption.

Another significant differentiator in the system presented, is the ability for such a system to create multiple key-pairs and restrict a key's purpose to identification only. Such a key can only be used to validate the identity of a particular node or entity, but can take limited or no further actions aside from that. All other (significant) actions are the responsibility of a non-identification key.

This type of system has been the subject of many a wish list[1] for some time now.

### A. The Issues Addressed by this Paper

The issue with today's PKI networks is the balance between trusting a chain of control or trusting known entities in a so called "web of trust system". The former has to exist in a manner to be secure at the root and more importantly trusted. The latter is open to abuse should enough untrustworthy entities manage to rate each other as trusted in a manner synonymous with a Sybil attack on such networks where groups surround good entities and effectively intercept or hide information.

Both of these systems are very much open to abuse and neither are able to provide a secure system to allow freedom of the creation of identifiable nodes.

### B. Conventions Used

This paper does not require all the operations listed below, but lists these for example implementations, which are outlined later.

Hash = Hash function such as SHA, MD5, etc.

Symm = Symmetric encryption such as AES, 3DES, etc.

Asym = Asymmetric encryption such as RSA, ECC, etc.

PBKDF2 = Password-Based Key Derivation Function or similar

Kpriv = private key (used to decrypt public-key-encrypted data or to sign data)

Kpub = public key (used to encrypt data or to validate signatures)

netput[K]/[V] = put a value (V) on the network with a key (K).

netget[K] = get value from network using the key (K).

### C. Asymmetric public key encryption

This paper makes use of public key cryptography. This is a system of encryption that does not require passwords or keys to be passed around, rather it allows the publication of a public key 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 Kpriv. Therefore the private key can be considered as the key for unlocking the data. It should be noted however, that the opposite is also true, in that data encrypted with the private key can be decrypted by the public key. This seems strange and useless (as everyone has access to the public key), but is used to excellent effect as described shortly.

### D. Cryptographically secure hash

A hash function can be thought of as a digital fingerprint. Just as a fingerprint of a person is supposed to be unique, then a digital hash function is also supposedly unique. We have all heard of two people with identical fingerprints (but perhaps have never met any!) and in the digital world it can be possible to get two pieces of data with the same hash result. This is referred to as a collision and reduces the security of the hash algorithm. The more secure the algorithm, then the likelihood of a collision is reduced. It is very similar to taking more points of reference on an actual fingerprint to reduce collisions in that area of science also. This is an area where both systems share a similarity in the increasing complexity of measurement and recording of data points of reference.

In cryptographically secure hashing, the data is analysed and a fixed length key called the hash of the data is produced. Again similarly to human fingerprinting a hash cannot reveal data just as a fingerprint cannot reveal a person (i.e. you cannot recreate the person from the print) and you cannot recreate the data from the hash.

Early hash algorithms such as MD4, MD5 and even early SHA are considered broken, in the sense that they simply allow too many collisions to occur. Hence larger descriptors (keylengths) and more efficient algorithms are almost always required. [2]

### E. Return to asymmetric encryption to produce digital signatures

In I-C we completed the section with a bit of a laugh at the thought of using a private key to encrypt data that the public key could decrypt. This is, as promised, used to great effect with the addition of hashing as described in I-D.

If we now take a piece of data and hash that data, to produce a fixed length key and encrypt that hash with our private key, anyone can decrypt the encrypted hash and then hash the data themselves to confirm it matches the hash that was decrypted. This confirms the piece of data passed to you is in fact cryptographically guaranteed to be the piece of data the signature refers to, otherwise the decrypted hash would be different.

This is digital signing (albeit with some detail missing) at its simplest. This is as detailed an understanding as is required to manage this paper.

**Algorithm 1** Hash(Public key) == ID

**Algorithm 2** Hash(Public key + Signature) == ID

## II. Implementation

### A. Methodology

This validation system requires, as all cryptographically secure validations systems do today, manipulation of key pairs. These key pairs are themselves used to generate the identity rather than being later tied to an identity and this is one of the fundamental tenets of this paper. The system this paper proposes will operate equally well with databases, DHTs or any key addressable storage system. In some cases this system will work outwith such key addressable systems, although in a reduced fashion.

### B. Simple Configuration

In a relatively simple method, identities are created as follows: $\mathsf{Hash}(\mathsf{Kpub}_1) = \mathsf{Identity}$. This strictly ensures the identity is mathematically linked to an identity rather than being later tied to one.

### C. Addition of a key revocation method

In another small step we can introduce the ability for a revocation system. this is done as follows: $\mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{SignatureKpriv}_2) = \mathsf{Identity}$ . For key revocation to operate, the identity packets should be stored in a way that retains all information as immutable, unless the signer of the data requests amendment or deletion. In this case deletion is not recommended and replacement of the public key with a false key (all 0s for instance) would be identified quickly as a revoked key.

Alone this would not seem to make sense as we generally wish to have a chosen identity and then secure this with a validation system. Although in some cases the system would be a good enough method, systems such as telephone numbers or bar-codes on product etc. may be facilitated by the system in this simple state.

### D. Passing ID between parties

Whereas the ID packet may be stored in any key addressable database or network as previously stated, there does exist a mathematically secure mechanism for doing so. This is possible as we have introduced a secure cryptographic hashing mechanism which allows us to ensure that it is unlikely that we can produce two pieces of data that will hash collide.

With this in mind and the fact that the ID packet described in II-C is the hash of the content of the packet, in this case a

[1] Here we assume the perfect algorithm and implementation of such, this is unlikely to exist in perfection.

[2] you can see where the problem exists as the logical conclusion is a key longer than any known peice of uncompressable data, to ensure no collisions

Table I

| $Kpub_1$ | $Sig_{Kpriv_2}$ | $Kpub_2$ | $Sig_{Kpriv_2}$ | Payload |
|---|---|---|---|---|

public key and an (optional) signature, which we now know is another encrypted hash. Consider a message formatted as shown in Table 1.

In this scenario the message is received and is allegedly from an ID. The first 2 fields represent the ID packet that would be stored in the key addressable storage mechanism described earlier. Here though we can simply hash the first two fields and confirm the hash matches the ID we think the message is from. If the message is also signed then this is confirmation of the message's validity. If there is no signature, a valid response would be to to take the public key in field 1, encrypt a message or challenge back to the sender, or indeed simply encrypt the reply the sender requested from you using this key.

It is therefore possible to validate an identity even without a key addressable network as long as all parties send the header fields required to identify themselves in the first place. However, the addition of key addressable storage allows better key revocation in cases where $Kpriv_1$may be compromised (e.g. if it is stored on accessible media such as a hard drive). The most efficient schemes would allow checking key validity on occasion and this is possible in two ways as described below.

### *E. Key validity checks*

We can easily tell if a key is valid cryptographically using either of the schemes previously described. However, there are situations where a cryptographically valid key is not in fact valid from our perspective and this is when, for whatever reason, a key has been revoked. In such cases the system requires to be cryptographically secure in the revocation scheme being used and also has to ensure key validity is maintained and revoked keys no longer get used.

In the scheme described in II-D validity is checked but currency of the key is not; a problem since it will remain cryptographically valid forever! Unless a check is introduced into the system to tell if a key is currently valid as well as cryptographically valid, then there is room for abuse. Therefore a revocation scheme that can be bypassed would be next to useless.

There are two methods in this paper of ensuring a key is currently valid and these are as follows:

*1) Check validity by looking up key:* The validity of a key can be confirmed by forcing a lookup of the key in the key addressable storage medium. A mid way measure may be that keys passed in messages are tested for validity at random. Either way would remove invalid credentials in a balance between speed and efficiency, with the forced lookup being the most efficient from a security perspective but perhaps too slow for some implementations.

*2) Check validity by message passing:* This section is somewhat involved and we need to step back a little and look again at the structure of the key pairs in the double checked message as shown in Table 1. It is notable in this case that key pair 2 must be valid as they sign key pair 1, therefore it follows if key pair 2 is invalid then key pair 1 is automatically invalidated[3].

**Algorithm 3** if (Hash(Public key + Signature) == Unique) then Hash(Public key + Signature) == ID == Network address

Taking this into account we can clearly see the chain of validity in this case. The chain being a set of two key pairs.

In the case of message passing, the message should contain both ID packets as per the table. The validity of key 2 is then tested to confirm key 1 is valid. This does make sense in the design so far where key 2 is never available on a disk or anywhere that can be placed in a position of mistrust. To do so we simply encrypt a message back using ID2's public key and the sender should be able to answer and/or sign a response. This does though assume the request being made is a request that has come from an entity that is at a higher level than the key1 holding node alone. So this mechanism is only valid for higher level messages and not the lower level messages or actions that are restricted to the node level entity.

So in this implementation, the message passing checking mechanism is only effective with a certain message type and therefore in a closed network or system where message types are in fact known and/or able to be identified.

The maidsafe[4] network can provide such a system as it depends on a node that can be identified and behave, building rank which a user (who has $Kpriv_2$) can use as the owner of $Kpriv_1$thereby getting the benefits of any quid pro quo relationship with a node and the person's rank on the system. In this case the person can easily revoke and recreate the key, transferring any rank across to the new key.

### *F. Combined with DHT*

All DHT networks have a particular requirement in common: to create a network address that is unique. There are various techniques for achieving this (e.g. chord[2] uses the hash of the IP/PORT combination, Kademlia[3] uses a random hash and so on). Using the above method and actually storing the identification packet on the network can provide for uniqueness while simultaneously providing a mechanism for finding a node's public key, which can in turn be used to send encrypted information to that node or to validate a signature from that node.

This is implemented as follows: $\mathsf{netget}[\mathsf{Hash}(Kpub_1)]$to check for uniqueness and if false then the true keys can be stored as follows: $\mathsf{netput}[\mathsf{Hash}(Kpub_1)/[Kpub_1]$ this is the simple case above without revocation, simply for brevity.

The hash size chosen should be of the same length as the network addressing scheme in place for completeness.

[3]This must be true otherwise we are saying key pair 1 is signed by an non existant or invalid key pair 2

## III. Improvements with security of private keys

If such a system were implemented in a DHT or similar publicly connected system (as web servers are), then the issue over the security of the private key is paramount. In today's networks this is achieved via some brute force techniques such as firewalls, secured hard drives, private key passwords (which make automatic reboot of a server require human intervention) and other custom approaches. There is no obvious improvement in this particular situation of this private key, however, the system presented does have flexibility that the others do not and this is in the ability to reduce the effectiveness or scope of the private key, $\mathsf{Kpriv}_1$.

Here, it is assumed that $\mathsf{Kpriv}_2$ is a key that is used to sign the identification packet for $\mathsf{Kpub}_1$ and that $\mathsf{Kpriv}_2$ is not located on the machine or node as $\mathsf{Kpriv}_1$ has to be. The intention now would be to ensure that the node can only identify itself, but be limited in its possible actions. In such a case, the node acts for some client or person, and this person takes action on the network as that node, perhaps as the node is a node that succumbs to some kind of ranking mechanism which allows the person using this system to operate in a particular sphere of reference as laid out by the rank or effectiveness of the node. In such cases the person would usually maintain the identity of the node and sign/decrypt messages with $\mathsf{Kpriv}_1$ acting as the ID that $\mathsf{Hash}(\mathsf{Kpub}_1 + \mathsf{SignatureKpriv}_2)$ provides.

A helpful addition in this case is to consider the signature key of this identifier packet. This can be identified by the person quite easily as shown in II-D.

## IV. Selectable ID's

A PKI implementation as described in this paper is extremely secure and resilient to a cryptographic attack, but does suffer form one major deficiency that most readers may spot. In these systems the node ID is a long name (in this case a 512 bit or 64 byte sequence) because the keys are derived via a hashing mechanism. This is not easily readable for a human and certainly too long to communicate comfortably on paper or verbally. An implementation of human-selectable ID is required, so that a user can give the network a desired ID, and it can be assigned and secured to that user (assuming it is available).

This completely flies in the face of the discussion so far, but all is not lost; it is possible.

### A. Option 1 (the crude version)

In this case we create a key-pair and store an ID packet, but instead of storing it at the hash of the content (that being the public key plus a signature) we store it at $H(chosen\ name)$. We then simply communicate the chosen name to friends who can query the store for the public key.

This would possibly work if the store were secure or perhaps was backed up by another layer that was secure (such as a distributed file-system as described in [5]), however it is not a good idea to design a system based on such inadequacies, so we need to improve on this as it is not completely mathematically verifiable and depends on less secure methods.

### B. Option 2 (slight improvement)

Here we can use the method described in III and we have the initial key-pair as a revocation system. This at least allows us to revoke a key, but does not allow us to protect it in the first place, which is clearly dangerous. If the network suffered an attack or even a Byzantine type failure and the chosen name key was lost, then we may have problems. If the key is in fact replaced, then somebody has stolen our ID. This could be disastrous, particularly if the applications using this did not maintain records of contacts passed along with the original public key, as this would allow an attacker to mimic us and potentially do an incredible amount of damage.

This disaster of course assumes many things that a good implementation should avoid; however it is worth noting the consequences of a value that does not hash to its ID (breaking our tenet). There appears nothing can be done as we cannot create data and know what it will hash to or reverse the process somehow to define a hash to be equal to a human readable and selectable name. We need to take this one step further.

### C. Option 3 (preferred)

In this option we take a very simple step that circumvents all of the above issues and attack vector possibilities.

In this solution we take the simple step of storing the chosen name chunk twice. Once as above, storing it with the key $H(chosen\ name)$ and we also store it in the usual (correct) way which is stored at the key that is the hash of the content, let's call this $K(content)$. This key becomes the Id we're looking for and it is very secure. What we have done is relegate the $H(chosen\ name)$ to a mere mapping effort.

In all communication, we communicate as $K(content)$ except when introducing ourselves. If another person wishes to communicate with us they can check $H(chosen\ name)$ or (and this is important) they can confirm against any other form of mapping, such as published information, or if users create a public share directory (immutable) such as in [5] then the mapping can also be stored there. This would mean the $H(chosen name)$ would eventually become redundant, which is possible, depending on the configuration of the application implementation.

In any case that ID is now restricted in its capabilities, and an attack on it would yield no useful information. An attacker could perhaps try and pass himself off as somebody else, but this would only succeed for new contacts, as all existing contacts communicate with the $K(content)$ ID. This also allows a person who has been hacked the ability to recreate his ID.

In the case of a DHT with ranking implemented, this attack is near impossible. However there are solutions as described here and using immutable locations for such mapping keys is very valid.

### D. Option 4 (multiple Public Names)

As the purpose of the $H(chosenname)$ packet is to provide a mapping of that name to an ID, there is no reason for not having multiple chosen names that are the same. Uniqueness

now is not an issue. It therefore makes sense to replace the content of the $H(chosen\,name)$ packet with a simple ID that represents the actual ID of that name.

As this is no longer unique, a system of including some additional information may be useful, such as town, date of birth or similar. to allow multiple identities share this packet and differentiate themselves then perhaps the easiest thing to include is town and country. This information should be signed by the $K(content)$ ID. This also ensures safety of the information, as all an attacker could do would be remove a mapping or remap an ID. This is functionally equivalent to adding another map. In this case all we need to do is add another value to the STORE. This is possible as we do not limit values stored by a key, which in a DHT is easy. However in a disk based store or poorly implemented database, it could prove more difficult, although such multiple value packets are not required in a secured disk based store. For multiple IDs that are equivalent, then storing the hash of the name plus town or department would prove to be secure.

### *E. Option 5 (validity provision)*

The $K(content)$ ID is the packet whose content (public name and signature) hashes to the ID of the packet and its location in the STORE. In this option another ID can add another value to this packet to provide some validation of that person or ID. The ID is verified by another 'chosen name' on the system and does so by adding a signed value to this key. The values should be the person's chosen name and the chosen name of the signing party in clear text with the signature of the signing party saved. This introduces a validation service and the signing party can revoke this signature, as he has signed it.

We also have double checked the user has not tried to switch names or commit some other fraudulent masquerade by tying the signed chosen name to the ID packet.

## V. CONCLUSIONS

This paper introduces a particular case for "non rooted" yet cryptographically secure PKI networks to be created. It is envisaged that this method will be extended with many more capabilities. Such measures may include:

Identification of credit card data linking the ID to a known name in another secure location. People could have a card and a revocation card or perhaps even better all done in software using a keying approach as described in this paper.

Single continuous validation systems where a known ID can be used across multiple web sites or on-line systems that require history to operate effectively.

## 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

**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.