SSH and PGP Key Convergence

Alistair Crooks
agc@NetBSD.org

Abstract

This paper outlines the uses of PGP and SSH for providing security and privacy for data communications for both data in-flight and at-rest. Some of the issues involved in key management are explored, such as key strengths, distribution of keys, key properties and formats, and relevant attacks on keys such as pre-imaging and collision attacks on key digests are investigated. The differences between the physical formats of PGP and SSH are compared and contrasted, along with their uses of fingerprints, and then key distribution for both PGP and SSH keys are outlined, and benefits and drawbacks compared. NetPGP's ability to convert between PGP and SSH key formats is then presented, and a worked example of distributing an SSH key through netpgp infrastructure is performed. Finally, some innovative uses of key-based authentication are shown.

1. Introduction

1.1 Background

SSH is the predominant method for accessing computing devices across a network. All kinds of devices, from high end servers and network storage devices to handheld devices, phones, routers and switches all use some variant of the SSH protocol [RFC4251] to allow use and administration of the device remotely. As well as machine administration, the BSD projects have used ssh [RFC 4251] and [RFC 4252] as part of their source code control management scheme since the projects were established. The predominant implementation of ssh is the openssh implementation from openssh.org. SSH was first produced in 1995 from the last available BSD-licensed version of ssh, 1.2.12, and has been developed under the auspices of the OpenBSD project and its contributors. The original SSHv1 protocol has been superceded by the SSHv2 protocol, for the additional ciphers like Blowfish and 3DES, and newer HMACs like HMAC-MD5 and HMAC-SHA1, as described in [NESRC2010].

The first non-commercial, patent-free version of PGP (2.6) became freely available in 1994 in the USA [Garfinkel1995]. This version subsequently and very anonymously made its way to Europe and became available for download there. PGP thus predates ssh by about a year.
PGP is mainly used for signing and verification of data, and for securing data in transport and at

rest using encryption and decryption. When signing data, the private key is used to permute a digest of the data, along with the timestamp of the signature creation; this provides a "signature" of the data at that time, by the person with access to the private key. Encryption of data is done with the recipient's public key - the private key is needed to unlock that data.

1.2 Keypairs, RSA and DSA keys

The use of asymmetric keys require a key to be split into two parts, the public and private parts.

Together, a public and private key make up a keypair; this is usually simplified to a "key". Two types of private and public keypair are used; RSA and DSA. When generating an openssh keypair, either RSA or DSA keypairs can be generated. GnuPG, an OpenPGP [RFC 4880] implementation, will generate the type of key that is requested. Until 2009, GnuPG generated DSA keys by default -- since that time, RSA keys have been the default.

Ssh and netpgp [Crooks2009] both use a public key/private key architecture to achieve different goals. The types of these keypairs are either RSA or DSA, in both openssh and PGP.

2 Problems and Issues with Keys

Assumptions about our use of public and private keys abound. One person’s standards of key protection do not necessarily apply to any other person. The following issues are just some that are faced when we use public and private keys to identify ourselves and carry out transactions.

2.1 Key Management

In order to perform operations on computers remotely, it is necessary to have public key credentials in place on the remote machine. This allows users and administrators to login to the remote machine via ssh, and perform operations on there. Some of these operations could use keys to validate information that people have sent us, or to perform actions on behalf of another user; in which case, some assurance that the request was legitimate is necessary.

This used to be a process which was best practice, but over the last ten years this has changed to become a legal and statutory necessity, to comply with legal and security audits, Sarbanes Oxley legislation, HIPPA and Basel II requirements, and other industry legislation and statutory requirements.

To provide this assurance, the keys in use in an enterprise need to be managed in a secure manner. This needs to be accomplished transparently, with a need for audit trails and responsibility logging.

2.2 Key Management Process

Key management plays a large part in enterprise computing these days. SSH public keys have to be distributed to the remote computer in order for users to be able to use the remote computer. Keys which have been compromised, or for users who are no longer a part of the organisation, need to be disabled and removed. At the same time, any enterprise data which has been encrypted by a user must be decrypted and re-encrypted when that user leaves the company.

One alternative to this is to use separate keys for a group of people, which probably raises more problems than it solves. Another alternative, covered later on, is the use of LDAP to disseminate SSH keys.

Yet another alternative is to use a secret-sharing scheme, such as Shamir’s Secret Sharing Scheme [Shamir1979], which is an implementation of a threshold scheme, whereby the secret can be retrieved by a quorum of shares. This scheme is currently being used to distribute the keys to the DNS root servers.

2.3 Key Lifetime

Key lifetime is another issue which relates to key management. The average "key lifetime", before which the key is compromised, the passphrase lost, or the key becomes otherwise irrelevant and a security threat in its own right, varies drastically depending on a number of factors, including the length of the key itself, the length of the passphrase, the security of the enterprise as a whole (nullifying or mitigating the effects of a "trojaned" sshd binary, for example), and other external, unpredictable, factors, such as large-number factoring, or possible advances in quantum computing.

It is probably almost universally true that our efforts in key management barely rise above the mediocre. This trend gets worse as the size of the enterprise grows. Whole configuration management systems have been written to address these problems, and they do a very good job of such things. However, the basic functionality is not always there, and the usual way of revoking an ssh key is to chmod the $HOME/.ssh directory to 0, which is not always effective, or to use `/sbin/nologin` as a login shell for that user, leaving an unmodified ssh public key in place.

This also places an onus on the people who audit the configuration management system to ensure that correct operation takes place in the face of file truncation, and multiple sources of truth for ssh keys and other information.

2.4 Key Replacement

Replacement of keys can be fraught with similar problems, since it is an extension of the previous issues with key management. Either a trusted configuration management system is going to replace keys which may have been compromised, or some other user will have to perform this action on multiple machines, which may themselves have been compromised. If the user performs this action, using a key which has been compromised, there are implications for the new key's integrity, and increased chances of Man-In-The-Middle attacks on the old and new keys.

To illustrate the MiTM attack, any time a key is replaced, its provenance must be checked and double checked to ensure that it is the correct key, since distribution of the new key by a central body implies that the key is approved by that central body. If an attacker can somehow substitute a malicious new key for the replacement new key, the malicious key could be on all of an enterprise’s servers in a short time. This need not be a computer MiTM attack - social engineering could be used to appear at the administration desk in place of the user whose key needs to be replaced.

2.5 Key Protocols

Cryptographic key protocols are used for 3 main purposes:

- to identify a user to a computer
- to authenticate a transaction
- to set up a key

Public key protocols are also difficult to establish [Anderson1995], and the principles established by Anderson and Needham need to be revisited constantly by everyone working in the field of cryptography. Amongst other principles of note here are the ones related to being extremely careful when using RSA keys to perform both signing and encryption, which is particularly relevant when looking at using an SSH key to encrypt data - see later in this paper.

[OpenSSH2010] sets out the protocol for the SSHv2 ssh-agent; this is the same basic protocol as is used in the ssh client public key authentication to authenticate to the server - possession of the private key is authentication, and so, by signing the required data, the ssh client authenticates to the server without transferring passphrases or private keys across the network.

2.6 Key Strength

There has been much debate recently about the optimal strength of keys, especially in light of the attempts to force CAs to use 2048-bit keys after 2013. This optimal strength will depend on the use for the key, since the time taken to verify matters more if this is an operation which takes many times per second, such as SSL Certificate verification, or relatively infrequently, such as a trust signing of a user’s key.

The strength of the key is obviously directly related to its size. Whilst in normal usage, a passphrase is used to gain access to the private key's contents, it is possible, given enough time and resources, to use brute force to calculate the contents of the private key.

This subject is not new - in 2001, Bernstein published a paper detailing a number of methods which could be used to help in factoring RSA 1024-bit keys [Bernstein2001]. In 2002, RSA labs discussed the Number Field Sieve methods that Bernstein used, and found that the estimate of time and resources needed to break 1024-bit keys had not changed due to Bernstein’s work, since the number of operations required by the Number Field Sieve had not been reduced. They further cited a paper by Arjen Lenstra and Eric Verheul [Lenstra2002] which suggested that, by 2009, a machine could be built to use a brute force attack on 1024-bit RSA which would produce the result in a day, and would cost at least $250 million, assuming Moore's law continued to hold, and also that factoring algorithms improved.

Lenstra and Verhaul also predicted:

> "...with 2002 technology, a 768-bit RSA key is comparable to a 72-bit symmetric key in terms machine cost (see also Sec. 4.5 of [Lenstra2002]). The estimated cost with 2002 technology is about $160 million. A 1024-bit RSA key involves about 1000 times as many operations as a 768-bit RSA key with current methods, so is comparable to an 82-bit symmetric key in terms of machine cost (or even higher, if the additional memory cost is considered per [Silverman2000])."

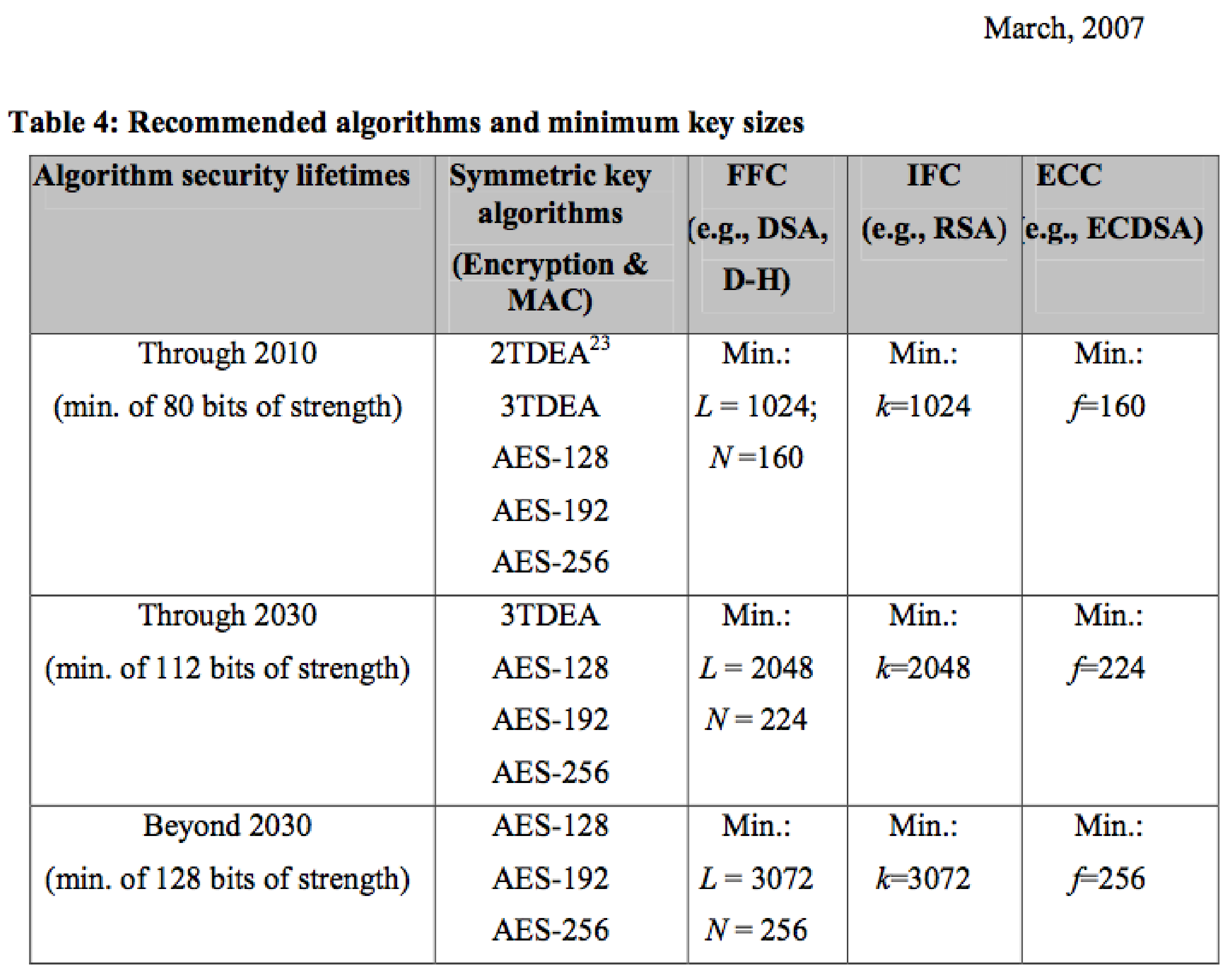
Shamir and Tromer also give a very interesting (and short) discussion, building on Bernstein's Number Field Sieve work, on the cost involved in factoring RSA-1024 in [Shamir2003]

Others, however, were not quite so bullish - in 2000, Robert Silverman gave a much higher estimate in [Silverman2000]:

> "Robert Silverman gives a much higher estimate than Lenstra and Verheul, considering the amount of memory required by current implementations of the Number Field Sieve [8]. He estimates that a $10 million machine, using 2000 computer technology, would take about 3,000,000 years to break a 1024-bit RSA key. This gives a cost-based equivalent of about a 96-bit symmetric key, providing an additional margin of security. Not all researchers accept that memory cost will be an issue, however, and this margin will likely diminish over time as memory costs decrease."

Whilst it is easy to look back in time and criticise previous predictions, it is less easy to give good advice to key sizes.

However, in 2001, NIST provided some guidelines [NIST2001] which were updated in 2007
to provide guidelines in Table 4 as to the minimum size of keys which should be used:



March, 2007

Table 4: Recommended algorithms and minimum key sizes

Algorithm security lifetimes	Symmetric key algorithms (Encryption & MAC)	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
Through 2010 (min. of 80 bits of strength)	2TDEA[23] 3TDEA AES-128 AES-192 AES-256	Min.: $L = 1024$; $N = 160$	Min.: k=1024	Min.: f=160
Through 2030 (min. of 112 bits of strength)	3TDEA AES-128 AES-192 AES-256	Min.: $L = 2048$ $N = 224$	Min.: k=2048	Min.: f=224
Beyond 2030 (min. of 128 bits of strength)	AES-128 AES-192 AES-256	Min.: $L = 3072$ $N = 256$	Min.: k=3072	Min.: f=256

This is especially relevant given the directive to use 2048-bit keys for the DNSsec root-signing keys. For other, normal uses, there seems to be no practical reason for the use of 2048 bit keys beyond that of following NIST guidelines.

Transitioning from 1024-bit keys to 2048-bit keys means a little inconvenience to the user in that they have to generate a new 2048-bit key, and then propagate that key to all the users of its public part. In doubling the size of the key, though, vendors such as Citrix and F5 are reporting a decrease in performance of 5 times, when measured with Transactions Per Second, with the decrease in performance due primarily to heavy use of the key during the handshaking process. [Macvittie2010]

2.7 Key Digest Algorithms and Related Attacks

For digital signatures, a digest of the data to be signed is calculated. This value is then used, along with key information, time, and random text, as the input to the digital signature.

The digital signature's integrity thus depends on the ability of the digest to produce values which cannot be pre-imaged (in other words, finding a different message that has the same digest value). Pre-imaging attacks would need to be used, since, to be useful, the attack would need to be able to change the message to something beneficial to the attacker, rather than being a merely random collision.

Pre-imaging can take two forms:

1. given a hash h, find a message m that satisfies the equation: h = hash(m)
2. given a fixed message m1, find a different message m2 such that hash(m2) = hash(m1)

whilst a collision attack is merely finding two messages which satisfy the equation

hash(m1) = hash(m2).

Wikipedia [Wikipedia-Preimage-2010] cites that all currently-known or almost-practical attacks on MD5 and SHA-1 are collision attacks; whilst this is technically correct, the 2009 attack on MD5 is actually a pre-imaging attack This attack is only theoretical, with a computational complexity of $2_{123.4}$ for full preimage attacks.

RFC 4270, written by Paul Hoffman and Bruce Schneier, gives an informational overview on Attacks on Cryptographic Hashes in Internet Protocols in 2005 [RFC4270]. Their summary of the situation in 2005 was that:

- Both MD5 and SHA-1 have newly found attacks against them, the attacks against MD5 being much more severe than the attacks against SHA-1.
- The attacks against MD5 are practical on any modern computer.
- The attacks against SHA-1 are not feasible with today's computers, but will be if the attacks are improved or Moore's Law continues to make computing power cheaper.

Their conclusions are also interesting, if schizophrenic - Schneier believes that a migration to SHA256 should take place to mitigate the problems with other, more established digest algorithms, whilst Hoffman argues that a move to SHA-256 leads to incompatibility, and that it would be more pragmatic to move to SHA1 in the near term.

2.8 Key Ciphers

A cipher is the algorithm used to perform encryption and decryption of data. There are two types of cipher: block and stream ciphers. Block ciphers have been the subject of attacks in the past.

2.9 Enterprise-wide Deployment

An enterprise may have multiple sources of truth for the user base -- the records kept by Human Resources, those kept by Payroll, and the IT departments records are three that come to mind.

In the same way that telephone directories are snapshots of the organisation at one point in time, a list of valid users and metadata relating to them is needed to provide input to the configuration management systems which provide this data to remote machines.

In the spirit of "Quis custodiet ipsos custodes?", or "Who guards the guards themselves?", the configuration management machines themselves need to be assured to be accessible only by a much smaller section of the userbase. Any changes to the data which is pushed out to the remote machines need to be validated. Remote machines need some way of ensuring that the data they are pulling has not been subject to change, or been manipulated in any way. They need to be able to ensure that the remote machine from which they are receiving data, either by push or pull, is the correct one, and has not been subject to a MITM attack.

3. Key Convergence

Ssh and PGP keys are held, however, in different formats. This paper examines some aspects of ssh and pgp key convergence, describes some of the situations and facilitators for using keys interchangeably, and looks at issues, drawbacks and benefits of using combined technology for both PGP and ssh implementations.

3.1 Key File formats

3.1.1 OpenPGP Key formats

In practical terms, PGP keys are held as collection of OpenPGP packets; many different types of packets are specified in RFC 4880. A later packet may modify values from a previous packet. The whole packet stream is read in as part of the keyring. Two keyrings are used -- pubring.gpg and secring.gpg -- to hold public and secret key packets. The secring.gpg is only needed if a secret key is to be used -- if a signature or decryption is to take place.

3.1.2 SSH Key formats

The format of the SSH key is laid out in [RFC4716], which also comments on the minor differences between PEM and the previous OpenPGP IETF standard (RFC2440). SSH keys are base64-encoded. Furthermore RFC 4716 mandates use of MD5 as a fingerprint mechanism for compliant SSH keys and key exchange.

> "The fingerprint of a public key consists of the output of the MD5 message-digest algorithm [RFC1321]. The input to the algorithm is the public key data as specified by [RFC4253]. (This is the same data that is base64 encoded to form the body of the public key file.)"

Since the SSH fingerprint is used in a number of ways, such as the calculation of whether a machine is being used as a "Man in the Middle Attack", some care should be used when evaluating the integrity of the various checks on host SSH keys, since a successful second pre-imaging attack is all that is needed to masquerade as a targetted machine.

As an adjunct to the SSH fingerprint, RFC 4255 provides a way, using a DNS Resource Record, to check the SSH host key fingerprints received from remote sites.

3.2 Key fields

There are a number of parts of a key's external representation (what the user sees) which are different between SSH and PGP. There are also a number of similarities. This section examines the details.

3.2.1 Digest Algorithms

As well as the areas for concern mentioned earlier in this paper, it should be noted that DSA uses SHA-1 as a message digest algorithm, and that SSH uses MD5 for its fingerprint and public key information. Netpgp can use any of its supported digest algorithms (MD5, SHA-1, SHA-256, SHA-512, and others), so it can inter-operate with SSHv2 keys, for example, as long as the correct digest algorithm is used.

3.2.2 Fingerprints

The examples shown in section 5.2 and later show the use of the MD5 algorithm with SSH keys in netpgp in order to give the same results as the various SSH tools.

3.2.3 Duration and Expiry

Whilst PGP keys have a duration field as part of the infrastructure, there is no such field in the SSH key.

3.2.4 Trust Signatures

PGP keys have trust signatures built into them, giving credence to the key's provenance. Because of the implication of trust built into the signing of a key (including verification of the user's identity), we tend to trust keys which have a larger web of trust. This trust is, in practice, fairly ad-hoc - one person's standards for checking identity differ from the next person's. However, it's usually fair to say that some trust, even that of an implied trust by means of a trust signature on a public key, is better than none.

3.2.5 Key Uses

PGP keys can have a usage associated with them - whether the key can be used to encrypted data for storage purposes, for example. There can also be an indication of whether this key is part of a key used in a secret sharing scheme, for example. Although these uses are not enforced, and are not widely known or used, they exist, and individual organisations could take advantage of them and plan accordingly. There is no such feature in an SSH key.

4 Key Distribution and Transmission

Various practical ways can be used to distribute public keys to remote machines.

Please note that the trust engendered in such keys is a separate issue which will be covered later.

4.1 Email

It's possible to distribute a public key to an administrator at a remote site by email. There are also various challenges to using this method - email is such an easily-spoofed mechanism that some other form of trust is needed. Some means of assuring the provenance of the key being transferred is necessary. A digital signature of the key would be best - but some means of validating the trusting signature is required.

4.2 HKP

It's possible to use the HKP protocol to obtain the public key part of a PGP key. HKP is the HTTP Key protocol, and uses HTTP to transport the public key (and also an index and verbose index of the same key).

The HKP servers in existence today, such as SKS and PKS, are relatively easy to set up, and do not require a privileged port (the default is 11371), which means that the barriers to running an HKP server within an enterprise are lowered. This is probably best viewed as a two-edged sword -- whilst anyone can run their own HKP server, it is easier to spoof a server's existence.

There are some drawbacks to this method as well, though, since the public key may not have been placed there by the owner of the key. Cross trust in the form of digital signatures are essential in this method. PGP keys are not tied in any way to one particular identity - anyone can claim that they are billg@microsoft.com to the public HKP servers.

In addition, there is no easy way to revoke a key which has been uploaded to one of the public HKP servers. The result is that the public HKP servers serve up a huge number of keys, some of them valid, some not, some blatant attempts at misappropriating some online identities.

4.3 LDAP

LDAP can be used to distribute ssh keys - patches were posted in 2006 to enable sshd to take the public key information from an ldap server [Mens2006]

This method requires there to be a local user of that name on the remote machine, and will simply take the ssh public key information from the LDAP server. This option does require a properly set up LDAP server and infrastructure, obviously, which may not be an option in some enterprises. In some ways, the public key provision is simplified by this interaction with LDAP; in other ways, old issues are now substituted with new challenges.

4.4 USB keys or portable media

Keys can be transmitted to the remote machine administrators through removable media such as CD or DVDs, USB memory stick, or PDA or phone-based Micro SD card. Again, the problem of provenance of keys transmitted in this way is a major challenge to this method.

Even a calligraphic signature will not help this method, since it is unlikely that the recipient would recognise handwriting, let alone a signature of this sort.

4.5 Key-signing party

Whilst these key-signing parties address the trust issue, it is not always possible to extend trust to the correct people in advance. Having said that, any move to proliferate trust in digital keys is to be applauded. At the same time, a trust signature on a key is simply an action at a particular point in time. The signatory may have no way of knowing if the signed key has subsequently been compromised, or otherwise had its secret key divulged.

4.6 Other methods

Several other methods for transferring the keys, such as taking a digital photograph of government id, such as a passport, and then digitally signing it before emailing to the remote party is another low-tech option which can help to get keys to the remote machine.

5 Practical Key Convergence

Since SSH keys and PGP keys both contain the same base data i.e. the primes which make up the keys, unifying the keys for SSH and PGP is an intriguing prospect. The ability for the keys to converge, and for PGP keys to be used to login remotely, for SSH keys to be used to sign and encrypt data at rest or in a stream, and for SSH keys to be distributed by means of PGP key transfer is appealing for any number of reasons. In particular, the ability to have only one passphrase to remember, and for that to be used all the time, lead to usability improvements straight away.

This section investigates some of the issues and outcomes from implementing SSH and PGP key convergence in netpgp.

5.1 Uses for Keys

At the present time, PGP keys have a number of uses:

- authentication
  - machine
  - user
- signing/verification
  - machine
  - user
- encryption/decryption
- trusting others keys
- revocation

SSH keys, meanwhile, are much more focussed:

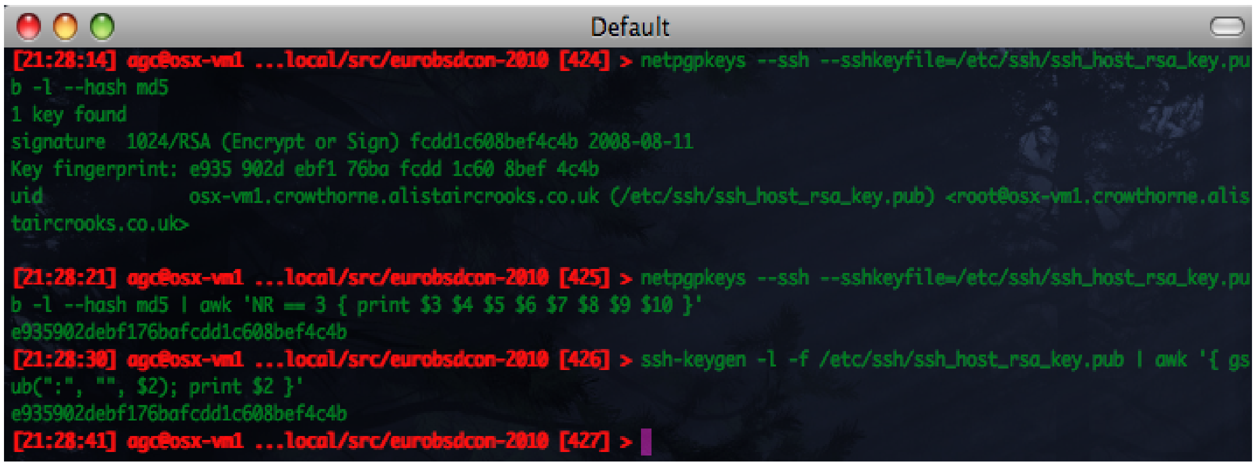
- authentication when logging into a remote machine

Netpgp has been developed to recognise RSA keys generated by ssh-keygen for use in ssh, and to be able to use them as well as OpenPGP keys. There are some interesting implications of this work.

To illustrate this, the rest of this section is devoted to a worked example of deploying an SSH key remotely using a netpgp infrastructure. We will use RSA keys exclusively for the rest of this paper. These examples are taken from the netpgp infrastructure which is provided with NetBSD-current, in `src/crypto/external/bsd/netpgp`

5.2 Verifying SSH Host Keys

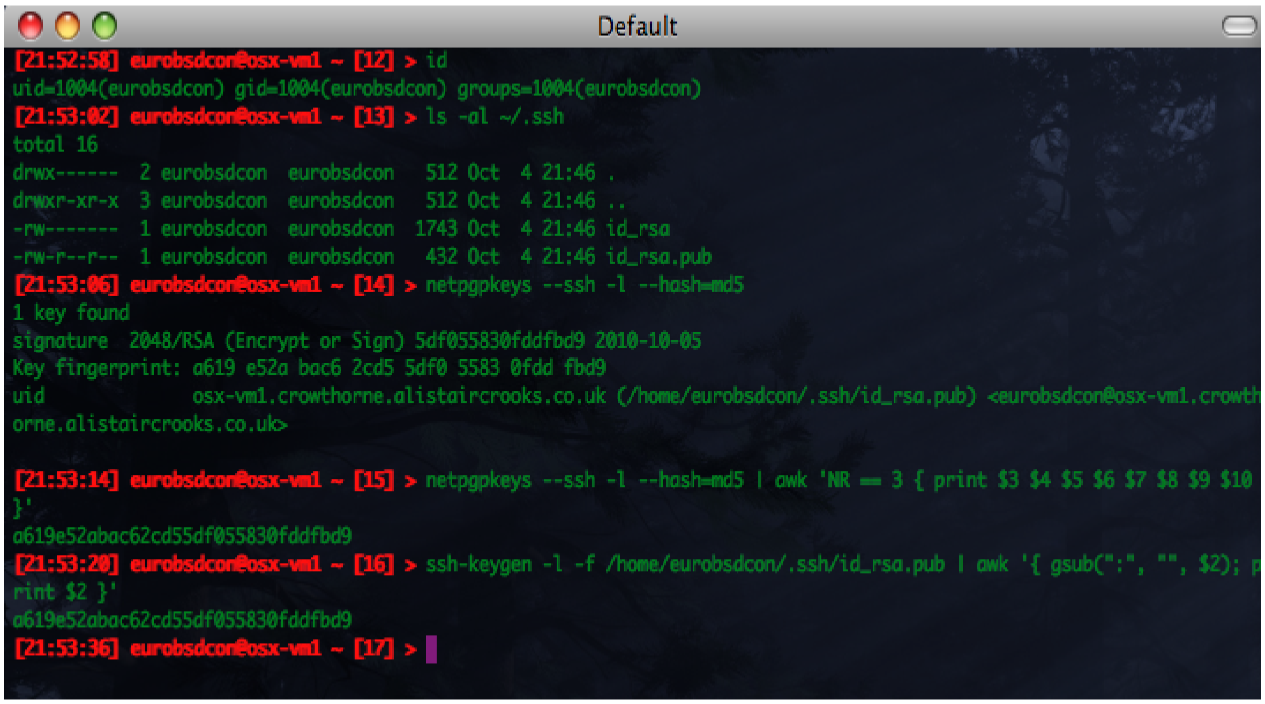
On a practical level, netpgp can be used to list ssh host keys. Since ssh uses an underlying MD5 digest, we specify that on the command line to netpgp, so that it produces the same answers as ssh. By default, netpgp will use an SHA-256 digest. The following example verifies that the answers from ssh-keygen and netpgp are the same:



```
[21:28:14] agc@osx-vm1 ...local/src/eurobsdcon-2010 [424] > netpgpkeys --ssh --sshkeyfile=/etc/ssh/ssh_host_rsa_key.pu
b -l --hash md5
1 key found
signature  1024/RSA (Encrypt or Sign) fcdd1c608bef4c4b 2008-08-11
Key fingerprint: e935 902d ebf1 76ba fcdd 1c60 8bef 4c4b
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/etc/ssh/ssh_host_rsa_key.pub) <root@osx-vm1.crowthorne.alis
taircrooks.co.uk>

[21:28:21] agc@osx-vm1 ...local/src/eurobsdcon-2010 [425] > netpgpkeys --ssh --sshkeyfile=/etc/ssh/ssh_host_rsa_key.pu
b -l --hash md5 | awk 'NR == 3 { print $3 $4 $5 $6 $7 $8 $9 $10 }'
e935902debf176bafcdd1c608bef4c4b
[21:28:30] agc@osx-vm1 ...local/src/eurobsdcon-2010 [426] > ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub | awk '{ gs
ub(":", "", $2); print $2 }'
e935902debf176bafcdd1c608bef4c4b
[21:28:41] agc@osx-vm1 ...local/src/eurobsdcon-2010 [427] >
```

5.3 Verifying SSH User Keys

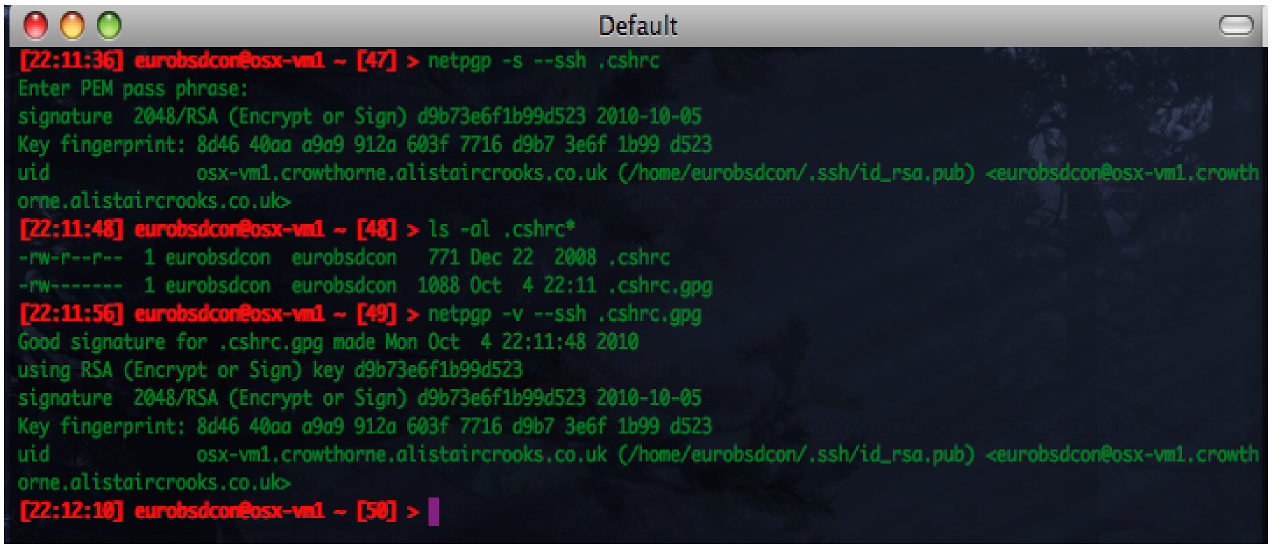
In the same way that host ssh keys can be listed and verified, netpgp can be used to verify a user's ssh key. In this example, a new user called **eurobsdcon** has been added, and ssh-keygen(1) has been used to generate a passphrase-protected ssh key.



```
[21:52:58] eurobsdcon@osx-vm1 ~ [12] > id
uid=1004(eurobsdcon) gid=1004(eurobsdcon) groups=1004(eurobsdcon)
[21:53:02] eurobsdcon@osx-vm1 ~ [13] > ls -al ~/.ssh
total 16
drwx------  2 eurobsdcon  eurobsdcon   512 Oct  4 21:46 .
drwxr-xr-x  3 eurobsdcon  eurobsdcon   512 Oct  4 21:46 ..
-rw-------  1 eurobsdcon  eurobsdcon  1743 Oct  4 21:46 id_rsa
-rw-r--r--  1 eurobsdcon  eurobsdcon   432 Oct  4 21:46 id_rsa.pub
[21:53:06] eurobsdcon@osx-vm1 ~ [14] > netpgpkeys --ssh -l --hash=md5
1 key found
signature  2048/RSA (Encrypt or Sign) 5df055830fddfbd9 2010-10-05
Key fingerprint: a619 e52a bac6 2cd5 5df0 5583 0fdd fbd9
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowth
orne.alistaircrooks.co.uk>

[21:53:14] eurobsdcon@osx-vm1 ~ [15] > netpgpkeys --ssh -l --hash=md5 | awk 'NR == 3 { print $3 $4 $5 $6 $7 $8 $9 $10
}'
a619e52abac62cd55df055830fddfbd9
[21:53:20] eurobsdcon@osx-vm1 ~ [16] > ssh-keygen -l -f /home/eurobsdcon/.ssh/id_rsa.pub | awk '{ gsub(":", "", $2); p
rint $2 }'
a619e52abac62cd55df055830fddfbd9
[21:53:36] eurobsdcon@osx-vm1 ~ [17] >
```

5.4 Using SSH Keys to Sign Data

Ssh keys can be used to sign data in a file. The following example shows a user's ssh key being used to sign and verify a file.

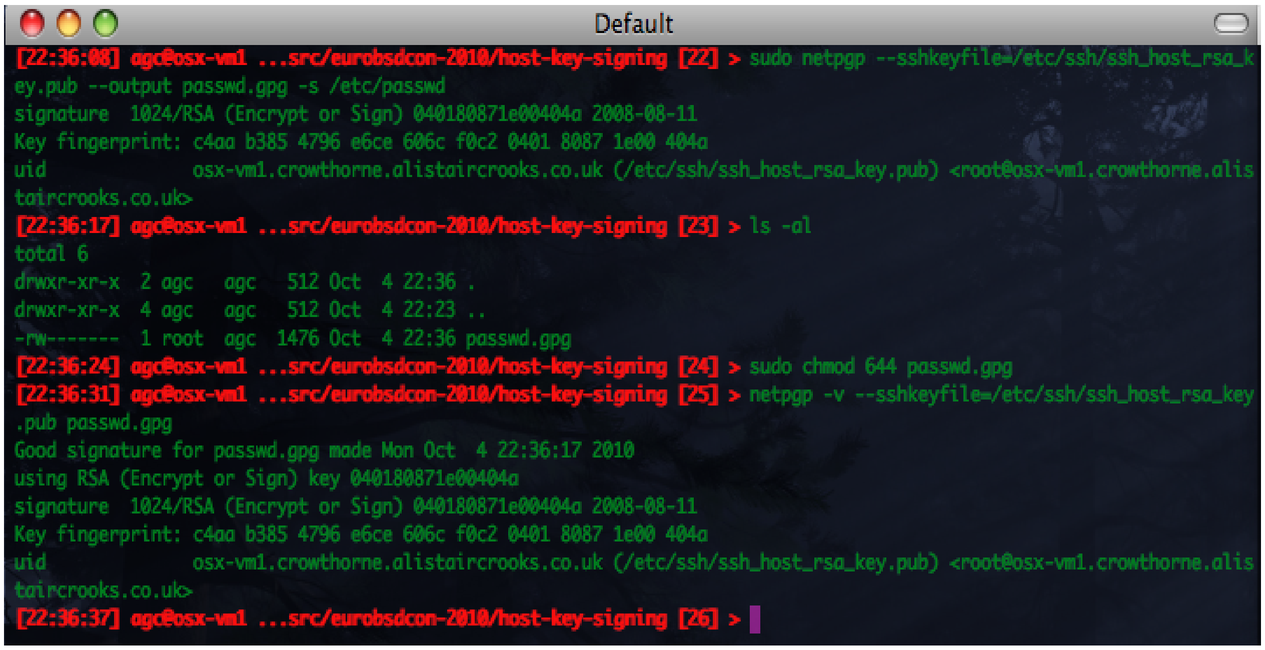


```
[22:11:36] eurobsdcon@osx-vm1 ~ [47] > netpgp -s --ssh .cshrc
Enter PEM pass phrase:
signature  2048/RSA (Encrypt or Sign) d9b73e6f1b99d523 2010-10-05
Key fingerprint: 8d46 40aa a9a9 912a 603f 7716 d9b7 3e6f 1b99 d523
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowth
orne.alistaircrooks.co.uk>
[22:11:48] eurobsdcon@osx-vm1 ~ [48] > ls -al .cshrc*
-rw-r--r--  1 eurobsdcon  eurobsdcon   771 Dec 22  2008 .cshrc
-rw-------  1 eurobsdcon  eurobsdcon  1088 Oct  4 22:11 .cshrc.gpg
[22:11:56] eurobsdcon@osx-vm1 ~ [49] > netpgp -v --ssh .cshrc.gpg
Good signature for .cshrc.gpg made Mon Oct  4 22:11:48 2010
using RSA (Encrypt or Sign) key d9b73e6f1b99d523
signature  2048/RSA (Encrypt or Sign) d9b73e6f1b99d523 2010-10-05
Key fingerprint: 8d46 40aa a9a9 912a 603f 7716 d9b7 3e6f 1b99 d523
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowth
orne.alistaircrooks.co.uk>
[22:12:10] eurobsdcon@osx-vm1 ~ [50] >
```

Using a user's ssh key in this way to sign data, it is possible to converge on a single key which could be used across an enterprise for both remote access to machines, and for data verification. In the same way that a signature was used to assure the integrity and provenance of the data, encryption could have been used to maintain security and secrecy for the data.

5.5 Using SSH Host Keys to Sign Data

In the same way that ssh user keys can be used to sign data, ssh host keys can be used to sign and verify the data.

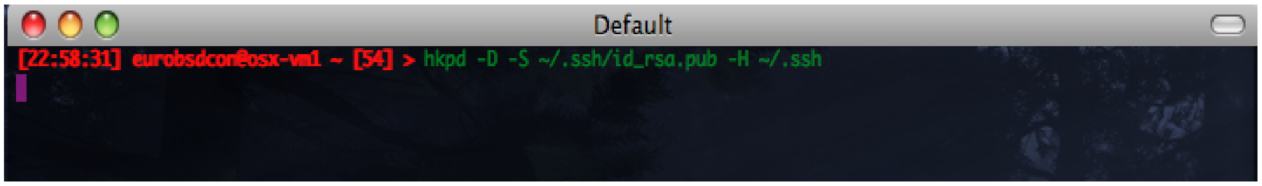


```
[22:36:08] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [22] > sudo netpgp --sshkeyfile=/etc/ssh/ssh_host_rsa_k
ey.pub --output passwd.gpg -s /etc/passwd
signature  1024/RSA (Encrypt or Sign) 040180871e00404a 2008-08-11
Key fingerprint: c4aa b385 4796 e6ce 606c f0c2 0401 8087 1e00 404a
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/etc/ssh/ssh_host_rsa_key.pub) <root@osx-vm1.crowthorne.alis
taircrooks.co.uk>
[22:36:17] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [23] > ls -al
total 6
drwxr-xr-x  2 agc   agc   512 Oct  4 22:36 .
drwxr-xr-x  4 agc   agc   512 Oct  4 22:23 ..
-rw-------  1 root  agc  1476 Oct  4 22:36 passwd.gpg
[22:36:24] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [24] > sudo chmod 644 passwd.gpg
[22:36:31] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [25] > netpgp -v --sshkeyfile=/etc/ssh/ssh_host_rsa_key
.pub passwd.gpg
Good signature for passwd.gpg made Mon Oct  4 22:36:17 2010
using RSA (Encrypt or Sign) key 040180871e00404a
signature  1024/RSA (Encrypt or Sign) 040180871e00404a 2008-08-11
Key fingerprint: c4aa b385 4796 e6ce 606c f0c2 0401 8087 1e00 404a
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/etc/ssh/ssh_host_rsa_key.pub) <root@osx-vm1.crowthorne.alis
taircrooks.co.uk>
[22:36:37] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [26] >
```

The `/etc/passwd` file was chosen as a typical file that might need to be verified on a regular basis. This operation needed to be performed with root privileges since the private host ssh key is needed to sign the data, and access to the signature file was given to other users so that the verification step could be run by an ordinary user. If the signed `passwd` file was being sent to a remote machine for verification, for example, this drop in privilege would happen automatically. It is also possible to encrypt the data as well as, or instead of, signing it, so that the contents would be safe even if transmission of the data was leaked.

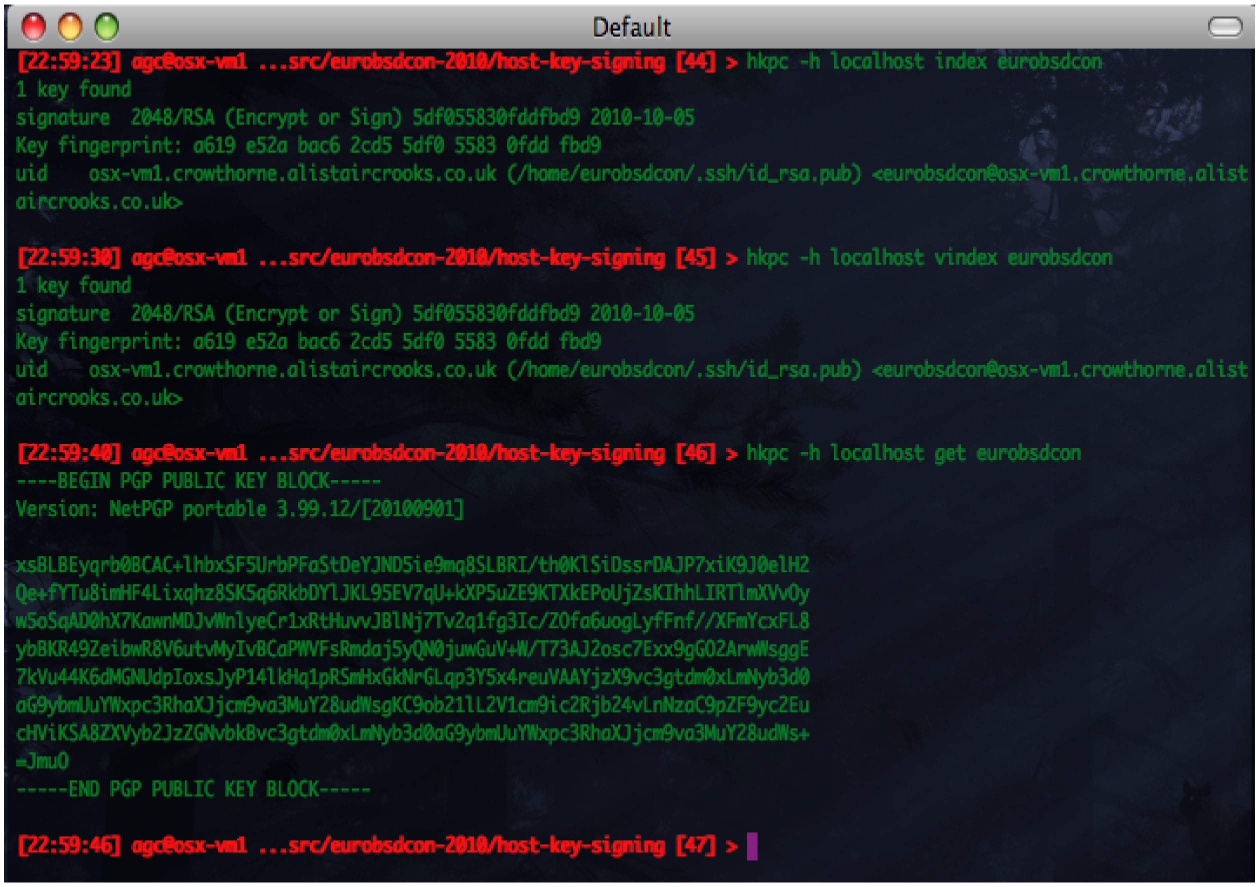
5.6 HKPD and SSH Keys

Netpgp includes hkpd(1), which is used to serve public keys using the (HTTP Key Protocol) HKP protocol. This is the same protocol used by the public PGP key servers, such as pgp.mit.edu. Hkpd can be used to serve ssh keys:



```
[22:58:31] eurobsdcon@osx-vm1 ~ [54] > hkpd -D -S ~/.ssh/id_rsa.pub -H ~/.ssh
```

and in a different window, the information can be accessed by other users:

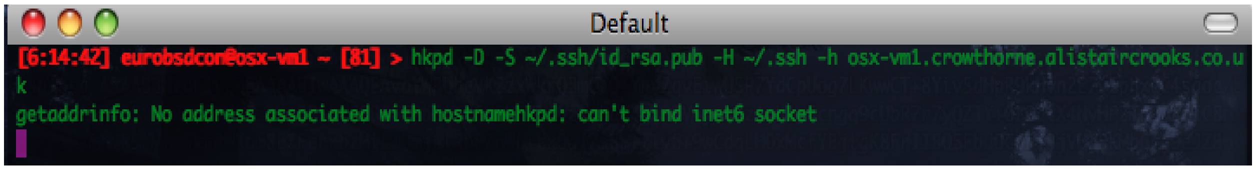


```
[22:59:23] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [44] > hkpc -h localhost index eurobsdcon
1 key found
signature  2048/RSA (Encrypt or Sign) 5df055830fddfbd9 2010-10-05
Key fingerprint: a619 e52a bac6 2cd5 5df0 5583 0fdd fbd9
uid    osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowthorne.alist
aircrooks.co.uk>

[22:59:30] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [45] > hkpc -h localhost vindex eurobsdcon
1 key found
signature  2048/RSA (Encrypt or Sign) 5df055830fddfbd9 2010-10-05
Key fingerprint: a619 e52a bac6 2cd5 5df0 5583 0fdd fbd9
uid    osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowthorne.alist
aircrooks.co.uk>

[22:59:40] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [46] > hkpc -h localhost get eurobsdcon
----BEGIN PGP PUBLIC KEY BLOCK-----
Version: NetPGP portable 3.99.12/[20100901]

xsBLBEyqrb0BCAC+lhbxSF5UrbPFaStDeYJND5ie9mq8SLBRI/th0KlSiDssrDAJP7xiK9J0elH2
Qe+fYTu8imHF4Lixqhz8SK5q6RkbDYlJKL95EV7qU+kXP5uZE9KTXkEPoUjZsKIhhLIRTlmXVvOy
w5oSqAD0hX7KawnMDJvWnlyeCr1xRtHuvvJBlNj7Tv2q1fg3Ic/ZOfa6uogLyfFnf//XFmYcxFL8
ybBKR49ZeibwR8V6utvMyIvBCaPWVFsRmdaj5yQN0juwGuV+W/T73AJ2osc7Exx9gGO2ArwWsggE
7kVu44K6dMGNUdpIoxsJyP14lkHq1pRSmHxGkNrGLqp3Y5x4reuVAAYjzX9vc3gtdm0xLmNyb3d0
aG9ybmUuYWxpc3RhaXJjcm9va3MuY28udWsgKC9ob21lL2V1cm9ic2Rjb24vLnNzaC9pZF9yc2Eu
cHViKSA8ZXVyb2JzZGNvbkBvc3gtdm0xLmNyb3d0aG9ybmUuYWxpc3RhaXJjcm9va3MuY28udWs+
=JmuO
-----END PGP PUBLIC KEY BLOCK-----

[22:59:46] agc@osx-vm1 ...src/eurobsdcon-2010/host-key-signing [47] >
```

HKP can be used to serve host ssh public keys as well as user ssh keys.

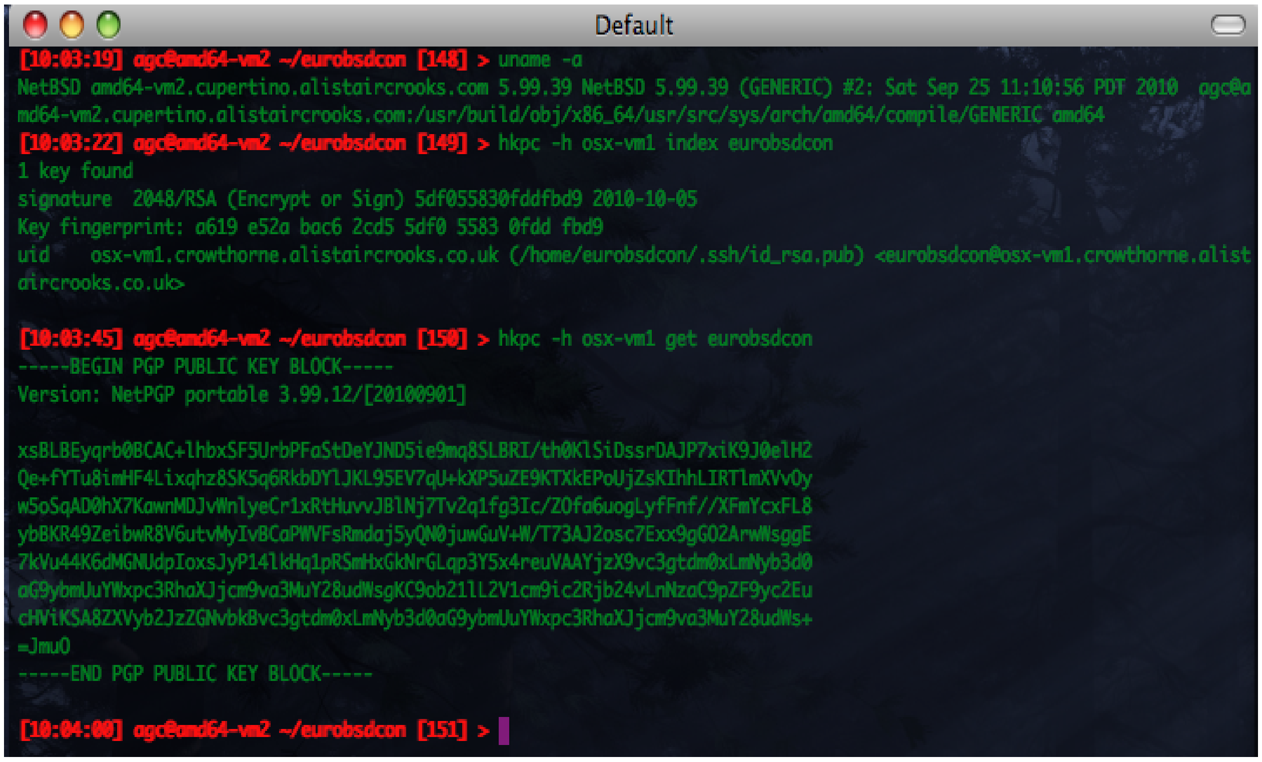
5.7 SSH User keys by Remote HKP

Ssh keys can also be retrieved from remote machines by HKP. Firstly we'll have to stop the hkpd program (which was serving on localhost), and let it serve on a socket which can be connected to by other machines:



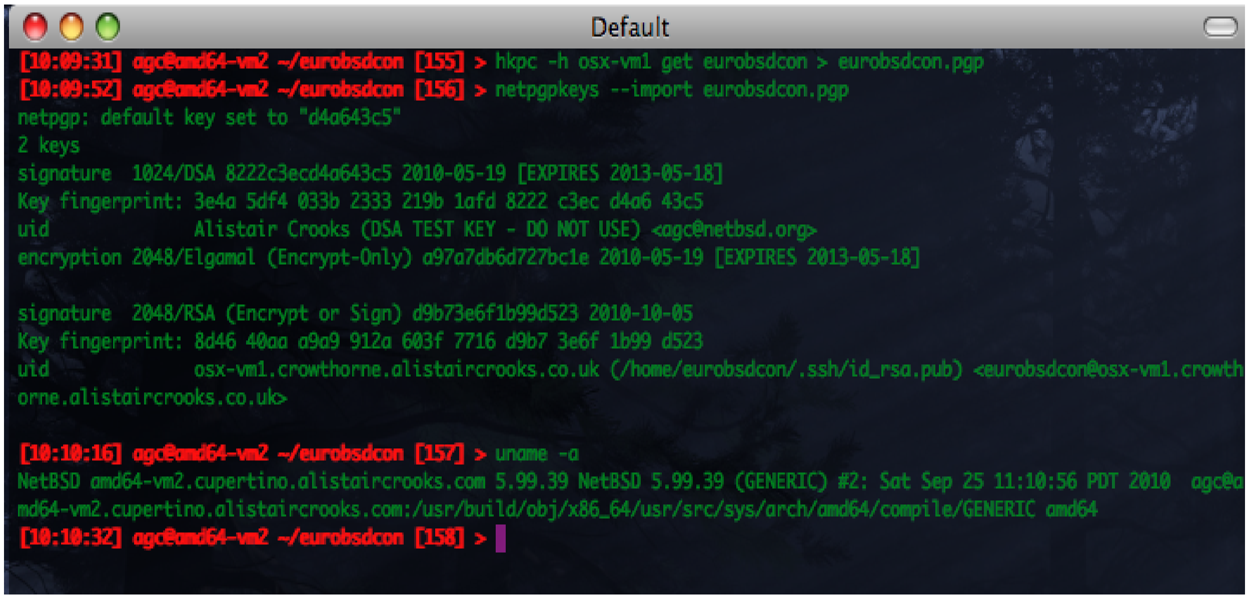
```
[6:14:42] eurobsdcon@osx-vm1 ~ [81] > hkpd -D -S ~/.ssh/id_rsa.pub -H ~/.ssh -h osx-vm1.crowthorne.alistaircrooks.co.u
k
getaddrinfo: No address associated with hostnamehkpd: can't bind inet6 socket
```

The following screenshot shows a different machine being used to download the eurobsdcon user's ssh key details and the ssh public key by HKP:


```
[10:03:19] agc@amd64-vm2 ~/eurobsdcon [148] > uname -a
NetBSD amd64-vm2.cupertino.alistaircrooks.com 5.99.39 NetBSD 5.99.39 (GENERIC) #2: Sat Sep 25 11:10:56 PDT 2010  agc@a
md64-vm2.cupertino.alistaircrooks.com:/usr/build/obj/x86_64/usr/src/sys/arch/amd64/compile/GENERIC amd64
[10:03:22] agc@amd64-vm2 ~/eurobsdcon [149] > hkpc -h osx-vm1 index eurobsdcon
1 key found
signature  2048/RSA (Encrypt or Sign) 5df055830fddfbd9 2010-10-05
Key fingerprint: a619 e52a bac6 2cd5 5df0 5583 0fdd fbd9
uid    osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowthorne.alist
aircrooks.co.uk>

[10:03:45] agc@amd64-vm2 ~/eurobsdcon [150] > hkpc -h osx-vm1 get eurobsdcon
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: NetPGP portable 3.99.12/[20100901]

xsBLBEyqrb0BCAC+lhbxSF5UrbPFaStDeYJND5ie9mq8SLBRI/th0KlSiDssrDAJP7xiK9J0elH2
Qe+fYTu8imHF4Lixqhz8SK5q6RkbDYlJKL95EV7qU+kXP5uZE9KTXkEPoUjZsKIhhLIRTlmXVvOy
w5oSqAD0hX7KawnMDJvWnlyeCr1xRtHuvvJBlNj7Tv2q1fg3Ic/ZOfa6uogLyfFnf//XFmYcxFL8
ybBKR49ZeibwR8V6utvMyIvBCaPWVFsRmdaj5yQN0juwGuV+W/T73AJ2osc7Exx9gGO2ArwWsggE
7kVu44K6dMGNUdpIoxsJyP14lkHq1pRSmHxGkNrGLqp3Y5x4reuVAAYjzX9vc3gtdm0xLmNyb3d0
aG9ybmUuYWxpc3RhaXJjcm9va3MuY28udWsgKC9ob21lL2V1cm9ic2Rjb24vLnNzaC9pZF9yc2Eu
cHViKSA8ZXVyb2JzZGNvbkBvc3gtdm0xLmNyb3d0aG9ybmUuYWxpc3RhaXJjcm9va3MuY28udWs+
=JmuO
-----END PGP PUBLIC KEY BLOCK-----

[10:04:00] agc@amd64-vm2 ~/eurobsdcon [151] > 
```

5.8 Importing the SSH Key

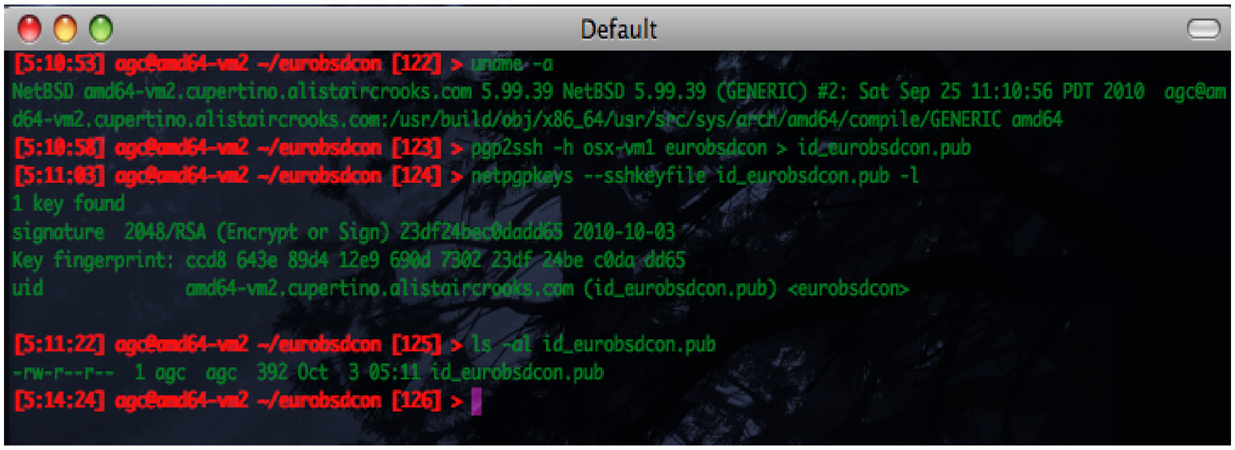
Saving the eurobsdcon user's ssh public key from the previous step, it can be added to any user's PGP public keyring very easily:


```
[10:09:31] agc@amd64-vm2 ~/eurobsdcon [155] > hkpc -h osx-vm1 get eurobsdcon > eurobsdcon.pgp
[10:09:52] agc@amd64-vm2 ~/eurobsdcon [156] > netpgpkeys --import eurobsdcon.pgp
netpgp: default key set to "d4a643c5"
2 keys
signature  1024/DSA 8222c3ecd4a643c5 2010-05-19 [EXPIRES 2013-05-18]
Key fingerprint: 3e4a 5df4 033b 2333 219b 1afd 8222 c3ec d4a6 43c5
uid              Alistair Crooks (DSA TEST KEY - DO NOT USE) <agc@netbsd.org>
encryption 2048/Elgamal (Encrypt-Only) a97a7db6d727bc1e 2010-05-19 [EXPIRES 2013-05-18]

signature  2048/RSA (Encrypt or Sign) d9b73e6f1b99d523 2010-10-05
Key fingerprint: 8d46 40aa a9a9 912a 603f 7716 d9b7 3e6f 1b99 d523
uid              osx-vm1.crowthorne.alistaircrooks.co.uk (/home/eurobsdcon/.ssh/id_rsa.pub) <eurobsdcon@osx-vm1.crowth
orne.alistaircrooks.co.uk>

[10:10:16] agc@amd64-vm2 ~/eurobsdcon [157] > uname -a
NetBSD amd64-vm2.cupertino.alistaircrooks.com 5.99.39 NetBSD 5.99.39 (GENERIC) #2: Sat Sep 25 11:10:56 PDT 2010  agc@a
md64-vm2.cupertino.alistaircrooks.com:/usr/build/obj/x86_64/usr/src/sys/arch/amd64/compile/GENERIC amd64
[10:10:32] agc@amd64-vm2 ~/eurobsdcon [158] > 
```

and the ssh key is able to be used on the remote machine.

5.9 Pgp2ssh and Key Convergence

A separate utility called pgp2ssh was written which speeds up this process, and allows all processing to take place in one utility. It downloads a key via HKP, and converts it to be in ssh format.



```
[5:10:53] agc@amd64-vm2 ~/eurobsdcon [122] > uname -a
NetBSD amd64-vm2.cupertino.alistaircrooks.com 5.99.39 NetBSD 5.99.39 (GENERIC) #2: Sat Sep 25 11:10:56 PDT 2010  agc@am
d64-vm2.cupertino.alistaircrooks.com:/usr/build/obj/x86_64/usr/src/sys/arch/amd64/compile/GENERIC amd64
[5:10:58] agc@amd64-vm2 ~/eurobsdcon [123] > pgp2ssh -h osx-vm1 eurobsdcon > id_eurobsdcon.pub
[5:11:03] agc@amd64-vm2 ~/eurobsdcon [124] > netpgpkeys --sshkeyfile id_eurobsdcon.pub -l
1 key found
signature  2048/RSA (Encrypt or Sign) 23df24bec0dadd65 2010-10-03
Key fingerprint: ccd8 643e 89d4 12e9 690d 7302 23df 24be c0da dd65
uid              amd64-vm2.cupertino.alistaircrooks.com (id_eurobsdcon.pub) <eurobsdcon>

[5:11:22] agc@amd64-vm2 ~/eurobsdcon [125] > ls -al id_eurobsdcon.pub
-rw-r--r--  1 agc  agc  392 Oct  3 05:11 id_eurobsdcon.pub
[5:14:24] agc@amd64-vm2 ~/eurobsdcon [126] > 
```

6 Implications of Key Convergence

The ability to download a key via HKP is useful, but not compelling in itself. Being able to convert between key formats seamlessly is, similarly, an interesting feature, but not immediately striking as being at all practical. Each key type has benefits and drawbacks.

6.1 SSH Keys

SSH keys are simple and efficient. The individual BIGNUM parts of the public key are base64-encoded, and the only other part of the key is the string at the start of the key which denotes the type: “ssh-rsa” and “ssh-dss”. Typically, a file will hold one key. There is no trust information, no expiry, duration or validity information, and no trust signatures are kept as part of the SSH key. The key itself holds no meta-data beyond the type, creation time and the email address of its creator.

6.2 PGP Keys

In contrast, PGP keys themselves suffer from being a number of packets concatenated together. There is no structure to the file, but all of the required information, including trust information to verify the key’s provenance, its validity and time information, what the key can be used for (such as stream data encryption and a part of a shared key), and all of the trust signatures. At key generation time, two keys of the required type are generated: one for signing and one for encryption. The reason for the separate uses for each key are to prevent the RSA Blinding Attack, described in [WIkipedia-BlindSignature2010], and also because DSA is used only for signing, and uses Elgamal for encryption in PGP.

6.3 Signing and Encryption Keys

One of the warnings in Anderson and Needham's 1995 paper was to beware of using RSA keys to sign and decrypt data - due to this multiplicative property of RSA, the same key should never be used for both encryption and signing purposes. We can achieve this in SSH in the same way that it is achieved in PGP, by generating a second key and using that as the signature key.

7 Practical Key Convergence

However, bearing these caveats in mind, we can look at the practical side of key convergence. If the same keys are used for both SSH and PGP, then there are a number of new opportunities that can be used.

7.1 Encrypting Data with SSH User Keys

Due to the nature of the adoption patterns of SSH and PGP utilities, it is much more likely for an SSH public key to be available on a machine. This key, or combination of keys, can now be used to encrypt data on the remote machine, since, for encrypting, all that is necessary is a public key for the intended recipient.

7.2 Verification using SSH User Keys

In practice, PGP verification of data seems to be performed much more often than the signing of the data. This is especially true when the provenance of the data is a desirable property, such as in configuration management systems for static files, or through verification of data retrieved through more dynamic means such as LDAP. By using signed material to ensure the data's integrity and provenance, data no longer needs to be transferred in a tightly controlled way. Signed configuration information can be retrieved from any server, not just trusted or SSL-certfied ones, via LDAP, FTP or HTTP, and verified on the remote machine.

7.3 Verification using SSH Host Keys

A system's host key can also be used as a public key for use in digital signature verification. Consider, for example, the case where a syslog message is received from another machine, in an environment where there has been a number of syslog spoofing attempts. If the syslog message itself is signed with a machine’s host key, the provenance of that message can be readily verified, since the public key is available using the standard ssh tools.

Other uses for signing and verification using SSH host keys include the setup of communication tunnels such as IPsec or VPN tunnels. This is much easier if an ssh-like mechanism could be used, or if the first steps were verified by the use of host keys to make sure that a MiTM attack is not underway.

7.4 Using Trust Signatures

As an SSH key is just a straight key, using a PGP key in conjunction with the SSH key to provide trust confirmation of the key itself is a way of building in timed-expiry of the SSH key. Consider, for example, an enterprise which endorses users by signing their public key by an **endorsing user**. The signed key can be made available by HKP or by other means in a PGP keyring.

Not only can an SSH key be verified that it is the same key as the PGP key, but the presence of the endorsing user’s signature gives verification that the SSH key is still in use. If a user leaves the organisation, then the PGP key can have the trusting signature revoked, thereby causing the SSH key to lose its validity.

Regular standard crontab jobs can run which check that any user on a given set of machines has the endorsement on (the PGP version of) the key.

This could be extended to provide a loose form of group membership due to the existence of a number of role endorsements.

Another way of achieving the same goal is to use an HKP server as the source of truth for all SSH keys installed on a machine. Should a user leave the company, the corresponding keys would be revoked, or disappear from, the HKP server, and the accounts could be disabled on all the client machines. The terminated public key could also zero out the existing public key, or have a **terminated user** sign the key, thereby overriding the endorsing user’s signature.

7.5 Decryption using SSH Host Keys

Whilst decryption obviously uses the host’s private keys, these keys are available after sshd(8) has been started, and so can be used for individual decryption of configuration files for a specific machine, for example - although the principles set out by Anderson and Needham state that such information should also have been signed before being encrypted, to avoid the problems suffered by X.509, for example, where the original message was stripped of its signature and re-signed by the attacker, in an attempt to build up bona fides for the attacker’s identity.

7.6 SSH Host Public Key Database

Openssh can be compiled to use TCP wrappers to draw conclusions about connection attempts to sshd from remote machines. Most critics point to the reliance on DNS, and its ability to withstand cache-poisoning attempts, to verify the connection address. By advertising host SSH keys via HKP, it would be possible to verify the connection attempts using verification of the signature.

[RFC4255] already provides an example of using a DNS RR to provide SSH fingerprints to corroborate the key fingerprint. DNS checking can be done in the OpenSSH ssh client. This provides an extra corroboration step, should people consider that DNS information is inaccurate, or has suffered from cache poisoning or other malicious attempts to subvert it.

Using a database of SSH host public keys, and making that available by HKP or other means is a much more scalable approach to the problem of verifying SSH host keys than relying on the MiTM protection in sshd alone. It would also address the problem suffered by Debian after the modification of its openssl package meant that no entropy was being used in its sshd key generation process, just after boot time when sshd was started for the first time. This is also a much more reliable and scalable process than relying on entries in individual users' `known_hosts` files.

7.7 PGP Ubiquity

This section has, until now, looked at the advantages that netpgp technology could bring to the ssh family of utilities. PGP has suffered because its command line interface is cumbersome, which in turn, leaves users dreading any interaction with it. So anything that makes it easier for them to use PGP to protect themselves and improve their security is to be applauded.

The other side of the coin is the benefit that PGP could gain from using public keys which are available on every machine, and most appliances, in the world.

When a new developer joins a project and has to establish both a PGP key and an SSH account, the process is long and fraught with loopholes which could be easy to subvert, not necessarily with either side of the transaction knowing that a compromise had occurred. A single key, covering the signing of government id, as well as being the bootstrap SSH public key has a simplicity and power that is hard to dismiss.

7.8 DNSsec

Although it has yet to be ascertained, there may be opportunities to use SSH keys as part of the ongoing DNSsec deployment, since the need for organisations to sign their zone files with an unusual key at regular, but wide, intervals of roughly a month gives scope for passphrases to be forgotten, and key files to be lost, leaked or compromised.

7.9 Simplified Protocols

Using digitally signed requests does away for the need for more complicated protocols. Appliance manufacturers have long been distributing new firmware with digital signatures to avoid the need for protocols to report a positive or negative acknowledgement of receipt of packets making up the firmware updates. The appliance just reads the update and verifies the signature on the firmware after reception. This continues until the firmware is verified to be good.

7.10 Combined Signing and Encryption

By signing a request, and then encrypting the result to the remote user's key, the transaction is guaranteed to be visible and understandable only by someone with access to the remote user's private key. At the same time, it is guaranteed to have come only from someone with knowledge of the requestor's key. Requests and responses can be built in this fashion to ensure that identities at either end are as reputable as knowledge of the key. This works for both human users and host machines. Although this is nothing new, the distribution of ssh keys makes this much easier to accomplish.

7.11 Just in Time SSH Key Infrastructure

By using different role-based endorsing users to sign PGP versions of keys, it is possible to write a small program which looks at the signatures on a key to determine whether the key, home directory, rc files and passwd entries should be created on a machine. Take, for example, a machine situated on a DMZ, and an endorsing user **DMZendorsed**, and another endorsing user **DEVELendorsed.** Only users whose keys have been endorsed by the **DMZendorsed** user would be allowed to exist on the machines in the DMZ.

8. Related Work

Ssh-agent is a program which will hold private keys, and authenticate a user's identity based on the contents of that key. Ssh-agent works with existing keys, and has no knowledge of PGP keys.

An orthogonal program to PGP and SSH key convergence is the **gpg-agent** program distributed with gnupg2. This is similar to ssh-agent, but will also listen on a named pipe for requests to sign data to allow for authentication to remote programs.

Gould has a Pubkey Access Authentication scheme defined as an RFC [Gould2010]. There is an implementation of this scheme in Python, and one by the author in C. It uses PGP keys as the basis for its authorization, but could equally easily use ssh keys as part of the key convergence work described here.

As already mentioned, Mens implemented a means to retrieve SSH Public keys from LDAP. This could easily be expanded to work with PGP Public keys as well. This means of key distribution is also useful, but orthogonal, to the key convergence work presented here.

9. Conclusions

This paper has compared and contrasted SSH and PGP key formats, presented the idea of SSH and PGP key convergence, and shown that key convergence is possible and useful through a worked example of netpgp being able to sign data using SSH keys, for both user and host keys. It has also shown that ssh key distribution can be effected using PGP key distribution means. Some of the underlying issues involved in key management and use have been investigated, and some underlying principles involved in key management and use have been revisited. Finally, some examples of the benefits and uses of key convergence were suggested.

10. Acknowledgements

Thanks go to Amanda Hockley and Petra Zeidler for their (independent) proof-reading and extremely constructive criticism of this paper.

11 References

Anderson1995	Anderson and Needham, Robustness Principles for Public Key Protocols http://www.su.fee.vutbr.cz/~cvrcek/cards/robustness.ps
Bernstein2001	D.J. Bernstein. Circuits for Integer Factorization: A Proposal. Manuscript, November 2001. http://cr.yp.to/papers.html#nfscircuit.
Crooks2009	Alistair Crooks NetPGP - European BSD Conference, Cambridge, 2009
Garfinkel1995	Simson Garfinkel PGP - Pretty Good Privacy, Simson Garfinkel, OReilly Books, 1995
Gould2010	Oliver Gould - Pubkey Access Authentication Scheme http://www.olix0r.net/PubKeyAccessAuthScheme.txt
Lenstra2002	Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. Journal of Cryptology. http://www.cryptosavvy.com/.
Macvittie2010	The 2048-bit Keys to the Kingdom http://devcentral.f5.com/weblogs/macvittie/archive/2010/09/10/f5-friday-the-2048-bit-keys-to-the-kingdom.aspx
Mens2006	SSH Public keys from LDAP http://blog.fupps.com/2006/03/02/ssh-public-keys-from-ldap/
NERSC 2010	National Energy Research Scientific Computing Center, 2010
NIST2007	NIST. Key Management Guideline - Workshop Document. Draft, October

2001. http://csrc.nist.gov/encryption/kms/key-management-guideline-(workshop).pdf.

NIST2001	Recommendation for Key Management, Part 1 (Revised) http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf
OpenSSH2010	openssh authors `src/crypto/external/bsd/openssh/dist/PROTOCOL.agent`
RFC 4251	Ylonen and Lonvick - The Secure Shell (SSH) Protocol Architecture, January 2006
RFC 4252	Ylonen and Lonvick - The Secure Shell (SSH) Authentication Protocol, January 2006
RFC 4255	Shlyter and Griffin - Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints January 2006
RFC 4270	Hoffman and Schneier - Attacks on Cryptographic Hashes in Internet Protocols - November 2005
RFC 4716	Galbraith and Thayer - The Secure Shell (SSH) Public Key File Format - November 2006
RFC 4880	Callas, Donnerhacke, Finney, Shaw and Thayer - OpenPGP Message Protocol - November 2007
Shamir1979	(1979), "How to share a secret", *Communications of the ACM* **22** (11): 612–613, doi:10.1145/359168.359176
Silverman2000	Robert D. Silverman. A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths. RSA Laboratories Bulletin #13, April 2000.
Shamir2003	Adi Shamir, Eran Tromer, On the cost of factoring RSA-1024, RSA CryptoBytes, vol. 6 no. 2, 10-19, 2003. http://people.csail.mit.edu/tromer/papers/cbtwirl.pdf
Wikipedia-Preimage-2010	Preimage Attacks http://en.wikipedia.org/wiki/Preimage_attack
Wikipedia-BlindingSignature-2010	Blind Signatures http://en.wikipedia.org/wiki/Blind_signature