

# Securing data backups

Tim Hurman <[kano@kano.org.uk](mailto:kano@kano.org.uk)>

With thanks to:

Isaac Dawson <[isaac.dawson@gmail.com](mailto:isaac.dawson@gmail.com)>

Freaky Clown <[freakyclown@gmail.com](mailto:freakyclown@gmail.com)>

Copyright © 2008 Tim Hurman

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Version: 1.0  
Published: July 23, 2008  
State: Final  
Classification: Public

---

## Table of contents

1. Introduction.....	3
2. Technologies.....	4
2.1. Symmetric encryption.....	4
2.2. Asymmetric encryption.....	4
3. Algorithm selection.....	6
4. Key generation.....	7
4.1. Creating the RSA private key.....	7
4.2. Creating the RSA public key (certificate).....	7
5. System design.....	8
5.1. Encryption system.....	8
5.2. Archiving method.....	9
5.2.1. Archive encryption.....	9
5.2.2. Archive entry encryption.....	9
5.3. Key management.....	10
5.4. Public key infrastructure (PKI).....	10
6. System vulnerabilities.....	11
6.1. Known plain-text attacks.....	11
6.2. Memory resident keys.....	11
6.3. Private key management.....	11
6.4. Public key management.....	11
6.5. Source data.....	12
7. Improvements.....	13
7.1. Known plain-text countermeasures.....	13
7.2. Multiple certificates.....	13
7.3. OpenSSL engine support.....	13
8. Example usage scenarios.....	14
8.1. Separation of backup and restore privileges.....	14
8.2. Data transfers.....	14

## Appendices

i. Resources and further reading.....	16
---------------------------------------	----

---

## **1. Introduction**

In 2007/2008 a number of extremely high profile data losses occurred and in many cases the data on the media was not encrypted or used a mediocre standard of data protection. The result of this was some very public and very embarrassing losses. Furthermore, rumours within the IT security industry was that the major breaches had not been made public and far outweighed the ones already made public.

The overriding factor in these data breaches was that data backups and transfers were not secured to a suitable standard. Furthermore, the algorithms and mechanisms for securing the data have been commonly available since the early 1970s.

This paper will design a strategy for backing up and transferring data that employs strong encryption to prevent the exposure of the data to unauthorised parties. The infrastructure for many of the components discussed already exist within the corporate environment, and the use of which is being promoted by operating systems such as Microsoft Windows 2003/Vista.

---

## 2. Technologies

To fully understand the concepts within this paper, it is necessary to have an understanding of encryption technologies. This section will describe the operation of the encryption technologies employed by the protection solution. This paper is not designed to describe the intricate principals and mathematics behind the technology, but to impart an understanding of encryption as an functional block.

### 2.1. Symmetric encryption

Symmetric encryption is a process of obscuring data using a “single key” system. In this system a single key is used for both the encrypt and decrypt functions.

An approximate analogy for this system is a two-way bridge between an island and the mainland (Figure 1). When the bridge gates are opened using the key, traffic (data) can flow in either direction. In this analogy, the mainland represents the unencrypted data, the island is encrypted data, the bridge is the encryption process and the gate control is the key.

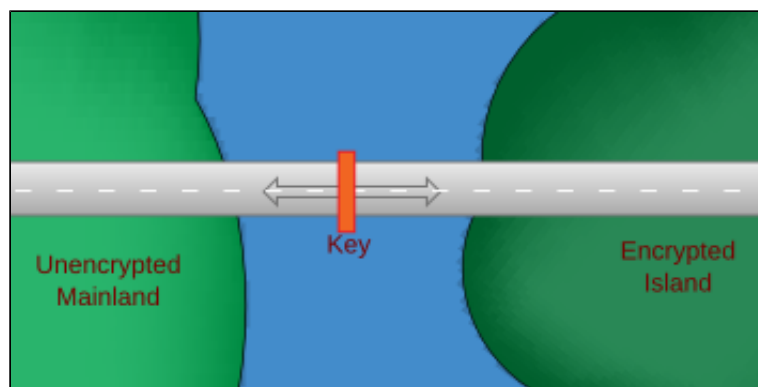


Figure 1 Symmetric encryption analogy

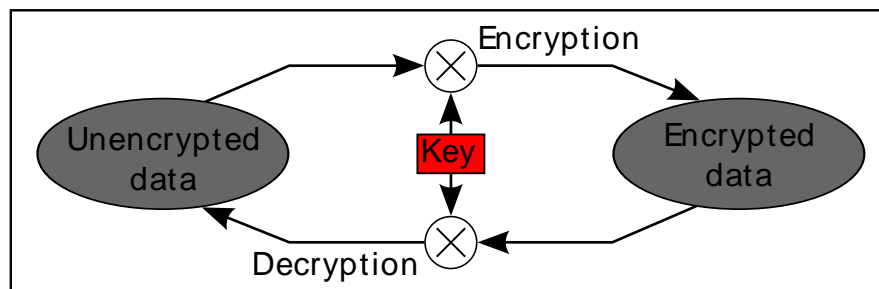


Figure 2 Symmetric encryption system

Symmetric encryption suffers from a number of issues regarding key management, since if the key is exposed, all data that has been encrypted using the key can be easily decrypted. To prevent this, a number of different keys should be used. However, the issue of securely storing and granting access to those keys soon becomes apparent.

A number of symmetric encryption algorithms currently exist, varying in key size, computational speed and complexity. Throughout this paper, the internationally recognised and FIPS approved Advanced Encryption Standard (AES) will be used.

AES operates on blocks of data, mathematically combining a key and data of equal size, to produce an encrypted block (which is the same size as the key/data block). Additionally, AES supports a number of “operating modes”. These modes introduce additional entropy into the system to further resist attacks on the encryption/data. This paper will employ the “Cipher Block Chaining” (CBC) mode. This uses not only the key, but also an additional source of randomness, an Initialisation Vector (IV), which is exclusive-ORed with the data before encryption. In CBC, the output from each block encrypt is used as the IV for the next block.

### 2.2. Asymmetric encryption

Asymmetric encryption is a process of obscuring data using a “dual key” system. This system uses two keys, one key encrypts the data, and the other decrypts it, neither key can be used in place of the other. Modifying the

analogy used in Section 2.1, a second bridge is introduced with a second set of gates. Each bridge is now a one-way road either to, or from the island.

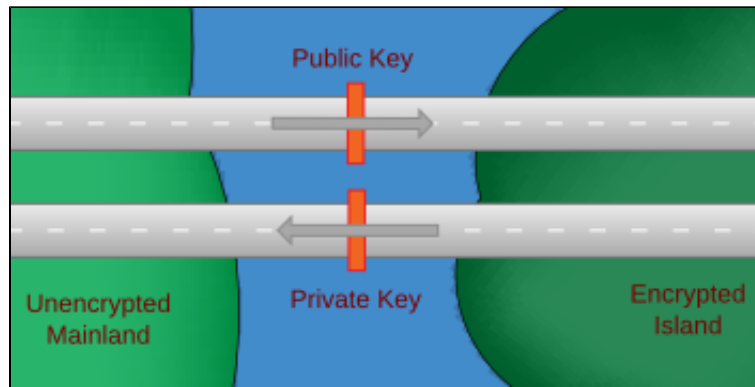


Figure 3 Asymmetric encryption analogy

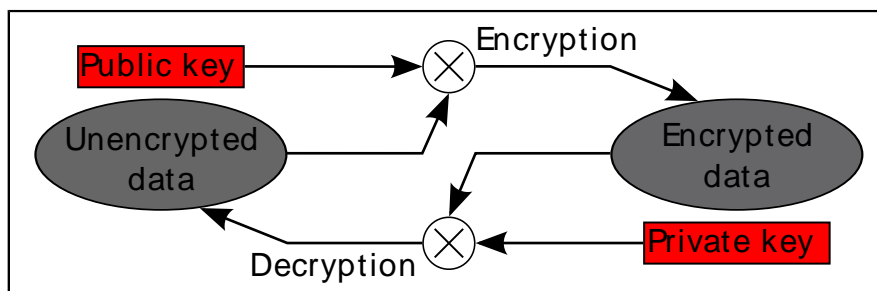


Figure 4 Asymmetric encryption system

When this system is used for cryptography, one key is denoted as “public” and the other “private”. Data encrypted with the public key can only be decrypted with the private key, and visa versa. It should be noted that encrypting data with the private key is usually termed “signing” and is used by systems to verify the integrity of data.

The private key should be password protected and stored securely while the public key is distributed. This allows data to be freely encrypted using the public key, however only the holder of the private key may perform the decrypt function to view the original data. The combination of a public and private key is called a “keypair”. The public key is usually wrapped in a “certificate”, most often in the X.509 format.

The most common asymmetric encryption method in use today is RSA, which will be employed by the system described. Asymmetric encryption, due to its nature, uses far larger key sizes and is significantly slower than symmetric encryption (Section 2.1). More recent asymmetric algorithms such as Elliptic Curve (EC) improve this situation by allowing for the same level of protection using smaller key sizes.

---

### 3. Algorithm selection

The table (Table 1) shows the selected parameters for each encryption type. Although the asymmetric key size is stated, it should be noted that these are external to the system and therefore cannot be verified. It would be possible to use a weak asymmetric key which would compromise the system integrity.

*Table 1 Selected encryption algorithms*

<b>Method</b>	<b>Algorithm</b>	<b>Key size</b>	<b>Mode</b>
Symmetric	AES	256	CBC
Asymmetric	RSA	2048	-

---

## 4. Key generation

This section details the creation of asymmetric keys for testing. It is not recommended that these be used in a production/live situation. Instead, a publicly verifiable key should be used to maintain system integrity, especially when transferring data between two entities. For a standard backup solution, a self-signed key may be used.

Before encrypting data using another entity's public key, the key should be validated by checking both the key's "subject", its fingerprint and its path to a trusted root certificate. Further details on this can be found in Section 6.4.

Since the reference implementation (*Attachment: streamcrypt-0.1.tar.bz2*) uses OpenSSL library to perform the encryption, the OpenSSL command line utility will be used to create the keys.

### 4.1. Creating the RSA private key

The first step in creating the keypair is to generate a private key of suitable size. The following command achieves this. The key itself will also be encrypted to prevent unauthorised access. A password will be required to access the key. The private key should be placed in a keystore for further protection.

```
bash$ openssl genrsa -aes256 -out test.key 2048
Generating RSA private key, 2048 bit long modulus
.....++
+
.....
.....+++
e is 65537 (0x10001)
Enter pass phrase for test.key:T4abzI6=zw
Verifying - Enter pass phrase for test.key:T4abzI6=zw
```

A strong password should be chosen and entered twice when prompted, do not use the example password shown above. A weak password may compromise the key and therefore the security of the entire system.

The file test.key contains the private key which itself is encrypted.

### 4.2. Creating the RSA public key (certificate)

The second step in creating the keypair is to generate the public key/certificate. The certificate also contains details of the owner and sometimes various permissions and attributes that the owner has.

```
bash$ openssl req -new -x509 -key test.key -out test.cer
Enter pass phrase for test.key:T4abzI6=zw
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:UK
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:.
Common Name (eg, YOUR name) []:My Name
Email Address []:root@example.org
```

The file test.cer contains the encrypted public key/certificate.

## 5. System design

### 5.1. Encryption system

The goal of this exercise is to encrypt the data on the backup/transport media and store the keys securely. Symmetric encryption allows us to quickly deposit the data on the backup/transport media but leaves key management issues such as:

- How/where to securely store the key
- Different keys depending on the use/different keys for each daily backup
- How to change the key if it is exposed
- Securely informing a third party of the key

Asymmetric encryption resolves the key management issue as the public key can be used to encrypt, but not decrypt, the data. The data may only be decrypted using the private key which should be securely stored. However, asymmetric encryption is computationally more intensive due to the significantly larger key sizes, and is therefore much slower. It is therefore not viable to use asymmetric encryption to protect large volumes of data.

By drawing from the properties of both encryption technologies, it is possible to design a system that eliminates drawbacks of each individual technology. The system operates as follows.

1. Create a random key for the symmetric encryption process. This key will be generated at run-time and should, depending on the quality of the random number generator, be different every time the process is used. A further random number will be generated to become the IV.
2. Encrypt the symmetric encryption key and IV using the asymmetric public key. The result of this will then be written to the media.
3. Encrypt the data using symmetric encryption and append it to the encrypted key+IV.

This system is called “envelope encryption” and is employed by a number of existing technologies including PGP and the S/MIME protocol. The OpenSSL library even has a high level API function to perform the process (EVP\_Seal\* functions). The reference implementation (*Attachment: streamcrypt-0.1.tar.bz2*) however performs the steps individually to demonstrate the process. Additionally, it would be possible to use utilities like GnuPG instead of the reference implementation to achieve the same result.

Assuming the asymmetric private key is secure, the data is also secure, since only the holder of this key (or a copy of it) may decrypt the archive to obtain the original unencrypted data. Anyone who has access to, or compromises the machine on which this system is run, may only gain access to the asymmetric public key, and is therefore only able to encrypt data.

One possible attack is to compromise the machine and replace the public key with one that the attacker has control of. This can be mitigated by using a publicly verifiable certificate and ensuring it is valid before use.

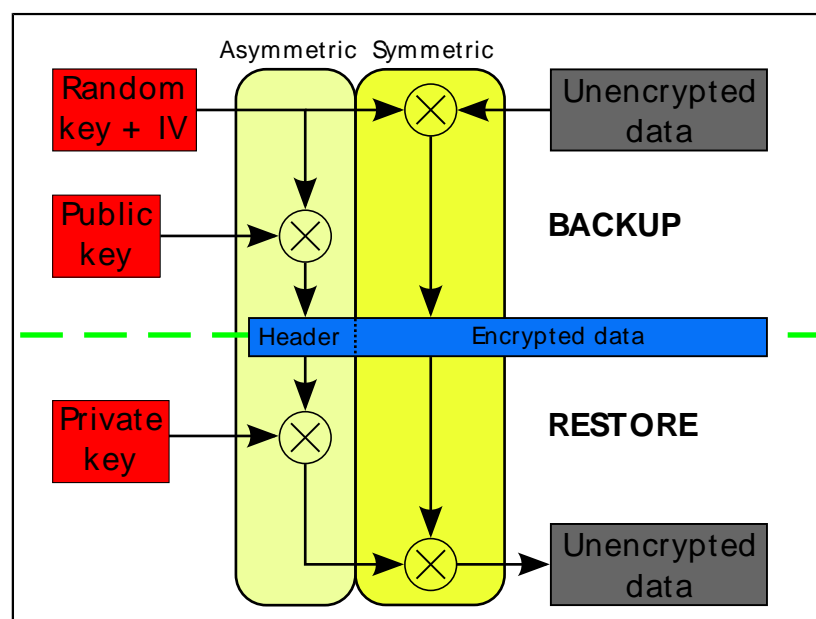


Figure 5 System architecture

## 5.2. Archiving method

The final design decision to be made is how to store the encrypted data. The two archiving strategies available, are “archive encryption” and “archive entry encryption”.

### 5.2.1. Archive encryption

Archive encryption performs the encryption process on all data, including the archive header and contents, before being written to the backup device. On UNIX type systems this is very simple to archive as data can be piped through the encryption process before being written to the media.

This system, like all encrypted filesystems, is vulnerable to a “known plain-text” attack. This is because the attacker is able to assume what the first few bytes of the data is, for example a Zip or tar archive header. Knowing this, an attacker is able to try every possible permutation of the symmetric key to see which results in the correct decryption output. Once the attacker has this key, all data on the backup can be accessed. Due to the key generation method, only the media being attacked would be vulnerable. However, the number of permutations available using the selected symmetric encryption algorithm is extremely large and would take a significant amount of effort to brute-force the key.

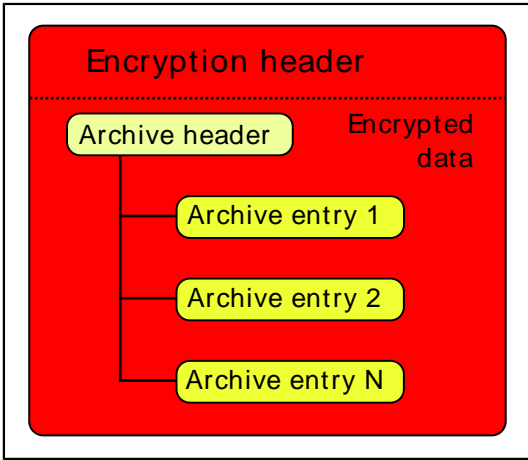


Figure 6 Archive encryption

### 5.2.2. Archive entry encryption

Archive entry encryption requires a modification to the archiving software. In this case, each file written to the archive would have a different symmetric encryption key, each encrypted with the asymmetric encryption cipher.

The drawback with this system is that the archive table of contents can easily be viewed. Using this, an attacker can guess the first few bytes of any particular data file and perform the same “known plain-text” as before. In this instance however, if the symmetric key is broken, only the data contents that particular file can be accessed.

This method has significant speed advantages in random access situations (even those that read from sequential media) as whole sections of the archive may be skipped to access the required entry directly, rather than having to decrypt the entire archive.

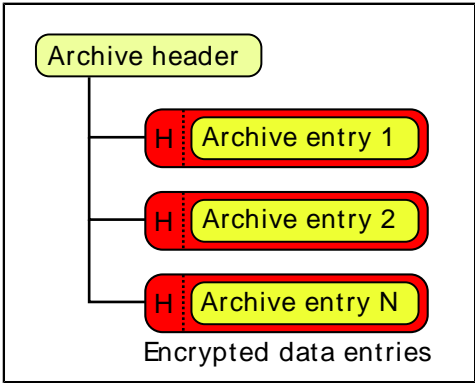


Figure 7 Archive entry encryption

---

For clarity and simplicity, archive encryption will be implemented. This also allows for the system and reference implementation to be separate to any archiving software.

### **5.3. Key management**

Using the system designed above, it can be seen that the primary goal has been achieved. If media containing the encrypted data is lost, it is unlikely (depending upon the attacker's capabilities) that the data would be exposed.

The system vastly simplifies any key management processes since security is only mandated for the private key. Furthermore, there is no requirement to grant any third party access to the private key. The symmetric key is stored with the encrypted archive and the user is not required to interact with it.

The system design also allows for backups to be protected if the private key is somehow compromised. It is a simple process to read the header, extract the symmetric key and re-encrypt the header using a new public key.

If the symmetric key is exposed, it would be required to re-encrypt the entire archive with a new random key. However, it would be far more valuable for an attacker to try to determine the asymmetric private key since this would allow access to all archives.

### **5.4. Public key infrastructure (PKI)**

While not strictly related to the system design, it is worth noting that large scale use of this system would benefit from a PKI deployment. Fortunately Microsoft Windows 2003 and on have an integrated certificate authority and PKI with auto-enrollment (Appendix i). There are also several solutions for UNIX/Linux environments including OpenSSL.

In general PKI solutions tie in a number of other features and aim towards single sign on (SSO) environments using LDAP/Kerberos/certificate authentication and authorisation. Depending on the required features, PKI solutions may require a significant overhead or administrative load, a common problem limiting the acceptance and deployment of certificates.

However, in Windows environments there is a growing requirement for signed documents. This is being pushed by the fact that signed documents have more rights than unsigned ones, particularly for macro access. While this requirement may have grown from the distrust of document formats, particularly on the Windows platform where viruses and malware are prevalent, it has started to mandate a PKI in the enterprise environment.

---

## 6. System vulnerabilities

### 6.1. Known plain-text attacks

As mentioned previously, the system suffers from a known plain-text vulnerability. This is where the attacker knows the few bytes of the unencrypted data and is able to try every permutation of the symmetric key until the correct one is found.

Firstly, this would only expose one archive since the header does not suffer from this an attack. However, once the symmetric key is known, a number of attacks on the private key are possible.

Secondly, since the number of key permutations in the selected encryption algorithm is significant ( $2^{256}$  values) it is unlikely that the key would be found with any useful timeframe.

It is possible to design countermeasures to mitigate this vulnerability, though not eliminate it. One example would be to prefix the unencrypted data with random data of an arbitrary length. This would force an attacker to examine not just the first encrypted block but a number of consecutive blocks as well. It is not recommended that the encrypted data be prefixed by random data since the attacker would still be able to guess that the unencrypted data started on a block alignment, since padding with an arbitrary size of random data is pragmatically complex. Furthermore, depending on the properties of the random number generator, it may be able possible to statistically differentiate the random data from encrypted data.

### 6.2. Memory resident keys

While performing the symmetric encryption or decryption process, the symmetric key is required and is therefore held in memory. While this is also true for the asymmetric decryption process, the private key is only unprotected for a brief period before being destroyed.

A user with suitable privileges may be able to obtain a copy of the process' virtual memory which would contain the symmetric key. This is an inevitable fact of the system's operation. Some operating systems have methods for protecting the virtual memory of processes which would in turn protect the symmetric key. Refer to the GNU Privacy Guard source (secmem.c) for further details.

### 6.3. Private key management

While not strictly a vulnerability of the system, it is recommended that the private key not be backed up in the encrypted archive. This would give an attacker who managed to gain access to the symmetric key, access to the private key as well. This could compromise the security all archives and backups.

### 6.4. Public key management

As mentioned previously, when sending data to other parties, it is important to validate the owner of the certificate. It is also important that either the certificate fingerprint be validated with the owner, or the certificate path be validated.

It is possible to use the OpenSSL command line interface to obtain the certificate owner (subject).

```
bash-3.1$ openssl x509 -in test.cer -noout -subject
subject= /C=UK/CN=My Name/emailAddress=root@example.com
```

Similarly, it is possible to obtain the certificate's fingerprint. The fingerprint value is unique to an individual certificate and should be verified with the owner via a trusted medium, usually a face-to-face meeting or telephone call. Recently, a number of organisations have started to print certificate fingerprints on the reverse of business cards.

```
bash-3.1$ openssl x509 -in test.cer -noout -fingerprint
SHA1 Fingerprint=49:50:AC:C5:76:AB:84:0F:23:A3:C4:AF:7F:77:51:89:ED:0A:5B:4B
```

Finally, to verify a certificate, follow the next example. This requires that the root certificates for the authorities which you trust be installed (Appendix i). The Verisign root certificate package can turned into a useful structure by the attached shell script (*Attachment: certs2cadir.sh*).

```
bash-3.1$ openssl verify -CApath ~/.certs test.cer
test.cer: OK
```

---

It should be noted that when verifying a self signed certificate, such as the one created previously (Section 4.2), OpenSSL will always report an error.

```
bash-3.1$ openssl verify -CApath ~/.certs test.cer
test.cer: /C=UK/CN=My Name/emailAddress=root@example.org
error 18 at 0 depth lookup:self signed certificate
OK
```

## 6.5. Source data

While this system provides very strong protection for the data within it, the original source data is still vulnerable to attack. In some cases this may be unavoidable, for example a backup of an active filesystem. However leaving backups or temporary files such as database exports or archive files unprotected on disk compromises the security of the system. Furthermore journaled filesystems may compound this issue.

When using the reference implementation (*Attachment: streamcrypt-0.1.tar.bz2*) on UNIX/Linux, pipelines (“pipes”) can be used to avoid temporary files. Other features of UNIX/Linux may also be used, such as opening a file then removing it. While an open file handle exists, the original contents are readable however the file cannot be opened by another process.

---

## 7. Improvements

A number of improvements can be made to the reference implementation for both usability and security.

### 7.1. Known plain-text countermeasures

As stated before, the system designed suffers from known plain-text attacks. This is where an attacker knows (or can guess) the unencrypted form of some encrypted data. In this case, the unencrypted data would be the file headers, the same information that MIME software uses to detect the file type. By knowing the unencrypted form of some data, an attacker can try every key permutation until the decrypt operation yields data that is expected.

An example of a countermeasure for this would be to prepend the unencrypted data with an arbitrary sized block of random data. The decrypt process would then step over this data until the archive header was located. The attacker would then not know if the plain-text data was in the first or one thousandth block. This countermeasure is not absolute however, as by decrypting enough data an attacker would be able to find the archive header.

It is not recommended that the random data be prepended to the encrypted data for a number of reasons. Firstly the decryption implementation would be complicated as the process would need to be restarted many times while stepping over the data. Without this, the padding would have to be block aligned, a fact an attacker could use. Secondly, depending on the quality of the random number generator, it may be possible to statistically differentiate the random data from encrypted data.

### 7.2. Multiple certificates

The system designed and implemented only supports a single target (certificate). This may require sharing a single private key between multiple users, which is never recommended.

A simple solution to this problem would be to encrypt the symmetric key with each certificate provided. Each of these encrypted keys could then be placed on the media before the encrypted data (Figure 8). This would allow the use of multiple certificates and therefore prevent the sharing of keys.

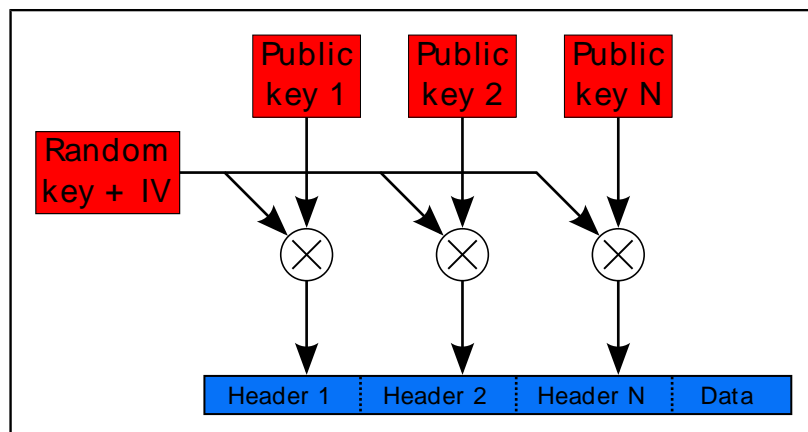


Figure 8 Multiple certificate solution

### 7.3. OpenSSL engine support

OpenSSL supports many cryptographic hardware accelerators (engines). Considering the amount of encryption involved in processing a reasonably sized block of data, it would be preferable to offload this operation to a dedicated co-processor.

## 8. Example usage scenarios

### 8.1. Separation of backup and restore privileges

The immediate application for this system is to separate the backup and restore privileges. The public key can be placed in the keystore for users with the backup permission, and the private key in the keystore of the users with the restore permission. The major benefit of this system is that it allows for automated backups without leaving sensitive keys unprotected.

This model can be further extended to database (and other) backups, especially since archive encryption was used. This allows for the data to be transparently wrapped without disturbing the proprietary structure of the data.

### 8.2. Data transfers

Another area where this system significantly increases security is data transfers, especially where the data is transported via an untrusted medium, either electronic or physical. In this situation, the destination entity would provide the source entity a public key. After the data has been encrypted, only the receiving entity may decrypt the data.

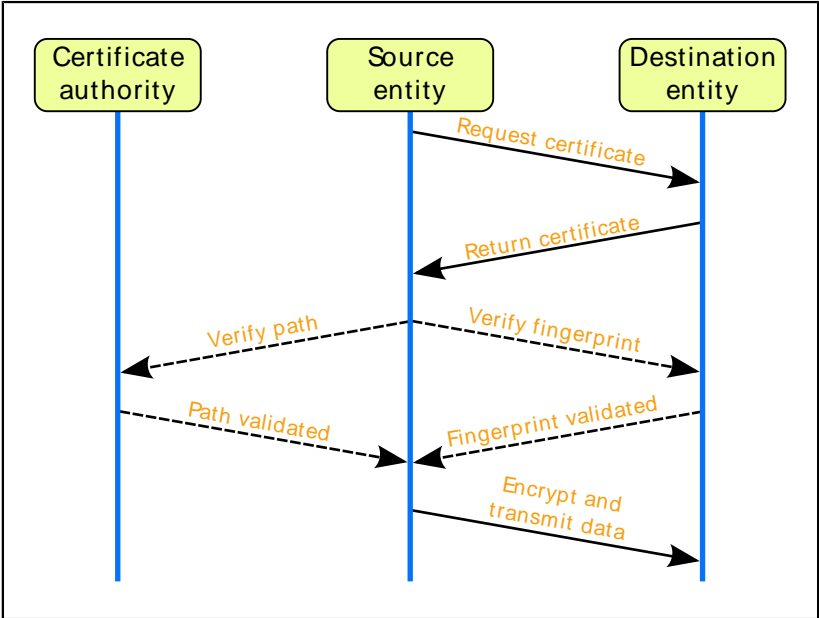


Figure 9 Data transfer mechanism

The significant part of this process is the public key verification. Ensuring that the data is being encrypted to the correct destination entity and not an attacker is paramount to the security.

Using this system, or one similar to it, the recent data losses would have been prevented or at least mitigated.

---

# Appendices

---

## **i. Resources and further reading**

- OpenSSL Library: <http://www.openssl.org>
- Verisign root certificates pack: <http://www.verisign.com/support/roots.html>
- Windows 2003 PKI: <http://www.microsoft.com/windowsserver2003/technologies/pki/default.msp>
- OpenCA PKI: <https://www.openca.org/>
- GNU Privacy Guard: <http://www.gnupg.org/>
- RSA workings: <http://en.wikipedia.org/wiki/RSA>
- AES workings: [http://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](http://en.wikipedia.org/wiki/Advanced_Encryption_Standard)
- CBC overview: [http://en.wikipedia.org/wiki/Cipher\\_block\\_chaining](http://en.wikipedia.org/wiki/Cipher_block_chaining)
- Schneier's Cryptography Classics Library. ISBN: 0470226269