Public key encryption using block ciphers

Chris J. Mitchell

Technical Report
RHUL–MA–2003–6
9 September 2003

Information Security Group
Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Abstract

A method for deriving a public key encryption system from any ‘conventional’ (secret key) block cipher is described. The method is related to, but improves upon, Merkle’s ’puzzle system’.

1 Introduction

In this paper we describe a very simple means of deriving a public key encryption scheme from an arbitrary symmetric block cipher. The idea is based on a simple trade-off between storage and computation. An important aspect of the scheme is that, as the availability of cheap mass-storage grows, so the security of the scheme can be increased.

Note that the scheme described is related, but not identical, to Merkle’s ‘puzzle system’ [2] (see also page 537 of [1]). It has the significant advantage over Merkle’s scheme that the recipient of an encrypted message can decrypt it immediately, whereas in Merkle’s scheme decryption requires an exhaustive search.

2 The scheme

We divide the description of the scheme into five subsections, as follows:

- *Domain parameter selection*,
- *Key generation* — generation of a public/private key pair,
- *Preparing for use of a public key* — computations which a user must perform before use of a public key is possible,
- *Encryption* — using the public key,
- *Decryption* — using the private key.

2.1 Domain parameter selection

Before use of the scheme an n-bit block cipher must be selected, i.e. a block cipher operating on plaintext and ciphertext blocks of n bits for some n. We write $e_K(X)$ for the block cipher encryption of data string X using a key K, which we assume has k bits. We write $d_K(X)$ for the corresponding decryption. Note that if X is longer than the block cipher block length n then in computing $e_K(X)$ we assume that an appropriate mode of operation (e.g. Cipher Block Chaining — see, for example, [1]) is used. Two positive integers must also be selected, namely m and d. Parameter m helps determine the size of the public and private keys, and d helps determine the size of public keys (details below).

2.2 Key generation

A user wishing to generate a key pair must obtain a set of m random keys for the specified block cipher, $K_1, K_2, \ldots, K_m$ say. These constitute the user's private key. The user must also choose a random data string X of dn bits, and then compute $P_i = e_{K_i}(X)$ for every i $(1 \leq i \leq m)$. The sequence of values $(P_1, P_2, \ldots, P_m)$ together with X then constitutes the user's public key. Note that every user should choose a different value for X (e.g. by incorporating a unique identifier into X).

2.3 Preparing for use of a public key (pre-processing)

Suppose user A wishes to send user B an encrypted message, and A knows that B's public key is $((P_1, P_2, \ldots, P_m), X)$. A now generates a sequence of random block cipher keys: $L_1, L_2, \ldots$, and for each such key generates $e_{L_i}(X)$ and compares the result with $(P_1, P_2, \ldots, P_m)$. Eventually a match will be found, i.e. suppose $e_{L_j}(X) = P_i$, for some i and j. Given that d was chosen to be sufficiently large, there is a very high probability that $L_j = K_i$.

User A now stops the search and retains L_j and i, which can be used to encrypt a message to B (as many times as is required). Note that if A retains L_j and i, A can discard the remainder of B's public key.

2.4 Encryption

To encrypt a message M for transmission to B, A simply computes $e_{L_j}(M)$, and sends the result to B, along with the integer i.

2.5 Decryption

On receipt of an encrypted string C, together with the parameter i, B uses key K_i to decrypt it, i.e. B computes $d_{K_i}(C)$.

3 Analysis

3.1 Parameter selection

The parameter d should be chosen so that the probability that two keys will map the dn-bit string X to the same encrypted string is very small. Assuming that the block cipher behaves in a random fashion, this can be achieved by choosing dn to be a few bits longer then k.

The parameter m should be chosen to be as large as is feasible. The larger m is, the more efficient the rest of the scheme is and/or the greater the security level can be.

3.2 Complexity of scheme

The storage complexity of the scheme is simple to compute. Private keys contain m block cipher keys, i.e. they contain a total of mk bits. Public keys contain m encrypted strings, each of dn bits, i.e. public keys contain dmn bits. Of course, once the Pre-processing step described in Section 2.3 is complete, a user need only retain a single block cipher key of k bits instead of the entire public key.

Encryption and decryption are simply block cipher encryption and decryption, and hence will operate much more quickly than most public key ciphers. The major computational complexity of the scheme lies in the Pre-processing step. The expected number of trials to find a key is of the order of $2^k/m$. Hence if $m = 2^s$ then the expected number of trials will be of the order of 2^{k-s}.

Thus, as an example, suppose $k = 64$ and $s = 32$, i.e. $m = 2^{32} \simeq 4 \times 10^9$. Suppose also that $n = 128$. Note that if the key length of the block cipher we wish to use is greater than the desired value of k then we can reduce the effective length by assuming certain key bits are fixed. Hence, to achieve these parameters we could, for example, use AES/Rijndael [3] with 128-bit keys and with half of the key bits fixed.

The private key will thus contain $mk = 2^{35} \simeq 3 \times 10^{10}$ bytes, i.e. around 30 Gigabytes (easily within the storage capability of a modern PC). Of the order of $2^{32} \simeq 4 \times 10^9$ trials will be required to complete the Pre-processing step. Hence, if the entity performing this step can perform 10^5 encryptions per second, then Pre-processing will take of the order of 10 hours. Public keys will contain around 60 Gigabytes, but once pre-processing is complete the public key can be discarded.

These numbers are around the limits of today's computers' capabilities. However, as computing power and available storage continues to grow, these requirements will become less and less onerous. Of course, this example only enables use of 64-bit encryption, and larger keys are likely to be required in the near future to cope with growing cryptanalytic capabilities. Nevertheless, this scheme would appear to become more and more feasible as time progresses, since, to break the scheme, the cryptanalyst requires computational resources equal to the product of the computational and storage resources available to the legitimate user. Thus, the increasing availability of ever cheaper storage increases the margin of safety between the legitimate user and the cryptanalyst.

3.3 Attacks on the scheme

We now briefly examine the ways in which a cryptanalyst E might attempt to break the scheme. Note that we assume throughout that the block cipher is resistant to all attacks other than exhaustive key search. Note also that, since every user employs a different value X, precomputation attacks on multiple public keys are not feasible.

We first consider the simplest attack model, where E has observed one or more messages encrypted by a single user, and hence E will know i and $e_{K_i}(M)$ for one or more messages M. Knowledge of i simply identifies which of the block cipher keys this user has chosen to employ. Hence this is only of use if E has already discovered this key. If E performs 2^t trial encryptions of X, the probability that E will be able to deduce this key is 2^{t-k}. Knowledge of $e_{K_i}(M)$ will not be of any use to E unless E can choose M to be one for which E has a precomputed table of values $e_K(M)$ for a variety of keys K. In such a case, if the table has 2^t entries, then the probability that E will successfully find the key is 2^{t-k}.

Hence, if we reconsider the example from Section 3.2, we see that, if the precomputed table has 2^t entries, then the probability of finding the key is 2^{t-64}. We next consider how large t might be. If we suppose that the cryptanalyst has 1000 times the computing resources of the genuine user, then the cryptanalyst will be able to perform 10^8 encryptions per second, i.e. around 10^{13} encryptions per day. Thus 3 months work will yield a table containing around 2^{50} entries (of course, the storage and sorting costs for a table containing perhaps 10 million Gigabytes will be non-negligible)! This will still only give the attacker a probability of 2^{-14} of discovering the desired key.

We next examine a more complex attack model where E has observed messages encrypted by a number of users (say 2^u users). We suppose that E has been fortunate enough (perhaps by design) to have observed the same message M encrypted by each of the 2^u users. Suppose moreover that E has a precomputed table of values $e_K(M)$ for a variety of keys K. If the table has 2^t entries, then the probability that E will successfully find the key is of the order of 2^{t+u-k}.

Revisiting our example, and supposing the existence of a table of the same size as discussed above, then the probability of finding one user's key is of the order of 2^{u-14}. Hence, if E has observed M encrypted by around 1000 users (i.e. $u \simeq 10$) then the probability of successfully finding one key is still much less than 0.5, although it is now becoming significant. This attack scenario serves as a measure of the overall security of the scheme, assuming a chosen plaintext attack launched against a large number of users.

To avoid attacks of this latter type, it would seem prudent to arrange matters so that obtaining the same message M encrypted using a variety of different keys is made infeasible. This can be achieved by using application-specific mechanisms. Two obvious examples of possible approaches are as follows. Firstly, as is common practice when using public key encryption, the public key can be used solely for encrypting a session key, where session keys are chosen at random by the encryptor. Secondly, the encryptor can always arrange to randomise encrypted strings, e.g. by using a random IV when encrypting using CBC mode (see, for example, page 230 of [1]).

Finally note that these attacks are simply the same as attacks against conventional uses of block ciphers, when one assumes known or chosen plaintext attacks. What weaknesses we observe for our example case apply to all block ciphers with 64-bit keys, which is one reason why 128-bit keys are now becoming the norm. Of course, as storage becomes cheaper, the system described here can be used with significantly longer keys (and correspondingly larger public key tables), increasing the margin of security.

3.4 Choosing parameters revisited

Based on the above analysis, the size of public keys (i.e. $mdn = 2^s mn$) should be chosen to be as large as storage allows. Having chosen s, the approximate number of trials to be performed by a legitimate public key user, 2^t say, should be chosen to be as large as user computing power allows. The key length k should then be defined as $k = s + t$.

The complexity for the attacker will be of the order of 2^k. Hence, if $k - t$, the difference in computing power between the cryptanalyst and the legitimate user, stays roughly constant, then the security level of the system will grow as the feasible choice for s grows. I.e. the security level of the system will grow in line with the growth in available data storage.

3.5 Reducing the size of public key certificates

We now consider one simple way of improving the efficiency of the scheme. As currently presented, the scheme involves rather large public keys which, if distributed within public key certificates, will mean that the certificates are very large. To avoid this, a hash-code computed over the complete public key (i.e. the set of 2^s ciphertexts) can be certified instead.

When a user wishes to process a public key, the complete set of 2^s ciphertexts can be obtained from some public source, which need not be trusted. Before use, the set of 2^s values can then be verified by computing the hash of this set, and verifying the hash-code using the public key certificate. With this simple enhancement, the scheme now has almost all the characteristics of a 'conventional' public key encryption scheme. The remaining major disadvantages would appear to be the preprocessing time, and question marks over whether current technology allows the scheme to be used with large enough block cipher key lengths to be considered completely secure.

4 Summary and conclusions

We have presented a conceptually very simple scheme for building a public key encryption scheme out of a symmetric cipher. The scheme relies on a memory/storage trade-off, and the 'margin of security' (in terms of the difference in computations to be performed by the cryptanalyst and the legitimate user) is proportional to the size of the private key. Hence, as memory becomes ever cheaper, the security margin of the system can be increased in absolute terms, regardless of improvements in computing power (assuming that the ratio of cryptanalyst computing power to legitimate user computing power remains constant).

References

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997.

[2] R.C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, **21**:294–299, 1978.

[3] National Institute of Standards and Technology (NIST), Gaithersburg, MD. *Federal Information Processing Standards Publication 197 (FIPS PUB 197): Specification for the Advanced Encryption Standard (AES)*, November 2001.