Multi-grade cryptography

Ronald L. Rivest
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

email: rivest@theory.lcs.mit.edu

April 29, 2001

1 Introduction

The recent development of powerful cryptographic techniques has caused a number of public policy issues to become significant. The three most prominent issues are (1) export control, (2) key escrow, and (3) anonymity. These represent tradeoffs between different societal objectives: (1) commerce/national-security and (2,3) privacy/law-enforcement. Some discussion of cryptographic policy issues can be found in Hoffman[3].

While the rapid development of cryptography is likely to make attempts to control it ultimately futile, it is interesting to see what technical possibilities there are for supporting compromise positions in each such trade-off.

This paper presents a new paradigm for secret-key cryptography that may be useful for export control (and perhaps also for law-enforcement). We call it *multi-grade cryptography* because it makes a single cryptosystem appear to present multiple levels of security. It is not based on key-escrow, but rather on computational complexity.

2 The key-size debate

We begin with the classic attempt to forge a compromise for exportable cryptography: key-size limitations. One begins with a standard fixed-key-length cryptographic system (like DES) or a variable-key-size cryptographic system (like RC2, RC4, or RC5), and then hard-wires a key-size limitation into the export version of cryptographic systems. A typical limitation would be to restrict the key size to 40 bits.

Forty bits of key is not very much; the recently publicized successful attack on the export version of Netscape by the Frenchman Damien Doligez demonstrated just how easy it is to search 2^{40} keys.

The question is then often formulated in the following terms: If 40 bits is too short, what is the "right" key-length for exported cryptographic systems? Here the "right" key-size would satsify both national security and commercial requirements.

It seems extraordinarily difficult to answer this question in a manner that satisfies both national-security and commercial requirements. On the one hand, large multinational corporations legitimately require high security in their international communications. On the other hand, intelligence agencies must process so much data that even modest increases in the average key-length used by their adversaries make collecting plaintext extremely expensive.

I think it is important to note the common confounding of the notions of breaking one ciphertext and that of breaking many. Admittedly, a large intelligence agency such as the NSA with many large computers could presumably search some exceptionally large key-spaces (say, well in excess of 64 bits) if all of its resources were devoted to a single challenge ciphertext. However, its bread-and-butter business is collecting large amounts of data, and so having to search a (say) 64-bit keyspace for each of a million messages received each day would be too expensive even for its generous budget. The distinction between an intelligence agency's "peak" capability and its "average" capability is an important one to address in any attempt to resolve the "key-size question."

The problem with the basic approach to export control by limiting key-sizes is that commercial interests want very large key-spaces, so that "breaking-in" is hard to do (near the peak capability of an intelligence agency), while intelligence agencies want modest key-spaces, so that the average time to decrypt a ciphertext is not too large.

3 Multi-grade cryptography

We propose to modify the basic approach by designing cryptosystems that have a dual character. Our cryptosystems are designed so that breaking the first key has one difficulty level (hard) while finding later keys has another difficulty level (easier). No matter which key the adversary attacks first, it will be quite difficult for him to figure out that key. But once he has broken his first key, he learns something about how the keys are created, and subsequent keys will be much easier for him to break.

We might, for example, want a "68/48" multi-grade cryptosystem, where finding the first key requires a searching a set of size 2^{68}, but where finding subsequent keys only require a searching a set of size only 2^{48}.

A commercial user might be happy with such a system, since he knows that there is a very high initial fence around his communications; no-one (not even a well-funded intelligence agency) can easily break in. A large investment is required to get the first break.

On the other hand, intelligence agencies might also be happy with our approach, since the large key-space only has to be searched once to "get in," after which the later problems are easy. On the average, the time to break a ciphertext will be satisfactorily small.

Whereas previous approaches have tended to ignore the distinction between breaking a single instance of the user's cryptosystem and breaking many such instances, we capitalize on this difference. The first instance will be hard, no matter which instance is chosen. Later instances will be easier.

The system requires that users choose keys in a structured manner. While the user may initially choose any full-size key, once he has chosen that key he is constrained as to how he can choose his other keys. The keys are "related" in a pre-specified manner. On an individual basis, however, each key offers the strength of a full-size key.

Multi-grade cryptography is closely related to Shamir's "partial key escrow" proposal[6], except that we are not proposing any escrow capability as part of the proposal, but are depending on computational complexity instead. For export purposes, computational complexity seems simpler to arrange than an acceptable international escrow system. Extensions of Shamir's idea are studied by Micali[5] and Bellare and Goldwasser[1]. Our proposal also bears some vague similarity to the CDMF proposal of IBM [4], and to Lotus' recent announcement.

The proposal can be extended in a number of ways to make the parameter choices more elaborate. For example, one can require that the adversary solve not just one, but many hard problems before finding keys begins to get easier for him. One can also have the transition from hard to easy occur suddenly, or gradually.

We note that the hard problem must be re-solved for each user. In other words, there are certain fixed "hidden system parameters" that the user is free to choose initially, but which he may not thereafter change. Finding one or more keys allows one to determine the hidden system parameters, and thus to reduce the search required for later keys. Since another user's keys may be based on different hidden parameters, another large search is required to solve for his parameters, after which only small searches are required for each additional key of that user. Here a "user" might mean a corporation or some other organizational entity within a single security domain, or it might mean a single workstation or piece of cryptographic equipment.

3.1 The simplest case

In the simplest case, the user chooses his first key K at random:

$$K = (K_0, K_1) .$$

Here K_0 is an n_0-bit string and K_1 is an n_1-bit string. Let $n = n_0 + n_1$ denote the total key-length. Subsequent keys are then chosen similarly, *except that they must all agree in their choice of* K_1. The user, once he uses his first key, is irrevocably committed to K_1; all his future keys will agree in their last n_1 bits. We call such a constraint a *multi-grade constraint*. Choosing $n = 68$ and $n_0 = 48$ (so that $n_1 = 20$) gives a "68/48" multi-grade cryptosystem as discussed above.

We call K_0 the "short-term key" (or more accurately, the "short-term key segment"), and we call K_1 the "long-term key" (or long-term key segment). The key K is thus composed of two parts: a short-term part and a long-term part.

We note that K_1 is not a "salt" in the usual sense, since it is not public. It is a fixed hidden parameter chosen by the user.

For an adversary to find any of the user's keys requires a large search of 2^n keys, since he must find both the short-term and long-term parts. However, once the first key of the user is determined by a large brute-force search, it only requires a small search of size 2^{n_0} to find any later key, since only the short-term part needs to be found then.

From a commercial user's point of view, a multi-grade "n/n_0" cryptosystem is clearly an improvement over conventional (single-grade) n_0-bit cryptography. By requiring an adversary to find the long-term key K_1 (at least the first time) in addition to the short-term key K_0, he has established a strong lower bound on how difficult it is for *any* opponent to read his communications.

From an intelligence agency's point of view, our proposal offers a way to reach a compromise on a difficult public policy issue. Choosing two parameters (n and n_0) rather than just one (n_0) permits greater flexibility to satisfy all parties. While the agency would have to perform some large n-bit searches, the bulk of its deciphering work would be n_0-bit searches.

3.2 Enforcing multi-grade constraints

The ability of the cryptographic system to *enforce* a multi-grade constraint is crucial. If a user were able to "get around" the multi-grade constraint and choose an arbitrary full-size (n-bit) key every time he changed keys, then he would defeat the purpose of the multi-grade cryptography. The system must allow the user to pick his first n-bit key arbitrarily, and then it must enforce the condition that his later keys all have the same long-term key segment K_1.

Presumably, systems that do not offer credible enforcement techniques would not be acceptable to intelligence agencies, and will not be approved for export. Manufacturers are thus motivated to provide good methods for dealing with this "multi-grade enforcement problem."

We now discuss several approaches towards handling the enforcement problem. There are probably many other approaches that can be used instead of, or in combination with, the ideas sketched here.

3.2.1 Hardware solutions

Enforcing a multi-grade constraint in a credible manner is perhaps easiest in hardware, where permanent state changes (such as burning a ROM) can be made.

Hardware approaches can also be used for software systems: a hardware "dongle" that performs encryptions and stores the cryptographic key could be used in conjunction with the software. The dongle would enforce the multi-grade constraint.

3.2.2 Software solutions

Software systems are more problematic. What is to keep the user from re-initializing (reloading) his software, and thus "beginning over" at the point where is he allowed to choose a fresh n-bit key?

In addition, software is eminently corruptable, and so a malicious user could modify the software to circumvent any controls or enforcement techniques that have been built in. Our position is that users capable of such sophisticated manipulations would be capable of writing their own software in any case, so that such users are outside the scope of the objectives of export control. Software enforcement techniques can not be perfectly effective, but can limit the capabilities of the cryptographic system for all but the most sophisticated of users.

We suggest two approaches for purely software-based solutions to the enforcement problem. The first is based on specialized distribution techniques, while the second (which we prefer) is based on the use of digital signatures.

Distribution-based methods

In the first approach, enforcement for software systems could be achieved by requiring the purchaser (say, the security officer for an organization) to purchase the software in such a way that the hidden system parameters (such as K_1) are chosen by the purchaser and then embedded into his purchased master copy. This would need to be done so that the seller does not see the parameters, and so that the purchaser can not easily modify them in the purchased software.

Signature-based methods

Our second approach is based on the use of digital signatures. The cryptographic system will not work without a copy of the long-term key (or other hidden parameters) that has been signed by an appropriate authority (such as the manufacturer or the export control office). With digital signatures, the software would be generic: all copies would be identical, making manufacturing and distribution easier. The distributed software would contain a copy of the authority's public key, to be used in verifying these signatures. The software would only function when it is supplied with a copy of the long-term key K_1 that has been signed by the authority. (If no signed value of K_1 is available, the software might warn the user and use a default value of $K_1 = 0$.)

The user would apply to the authority in order to obtain the necessary signed electronic document (containing the long-term key K_1 or other parameters). In some cases the user might trust the manufacturer or distributor to obtain this signed document for him.

This signed version of the long-term key would be installed by the user as part of his configuration and set-up for the cryptographic software.

For the multi-grade scheme to work as desired, the signing authority must sign the long-term key *without knowing what it is.* If the authority needs to see K_1 in order to authorize (sign) it, then we effectively have an escrow system instead of a multi-grade system. There are two standard approaches for enabling the authority to sign K_1 without knowing what it is:

Hash functions: The user supplies the authority with $H(R)$, where H is a suitable hash function (such as MD5 or SHA), and R is a long random number. The authority signs $H(R)$ and returns it to the user. The software is given this signature and R, and checks that both the signature and the hash computation are correct. It then uses the low-order n_1 bits of R as the long-term key K_1.

Blind (RSA) signatures: (as proposed by Chaum[2]). As above with hash functions, but the user also picks a random blinding number b, and gives the authority the value $H(R)b^e \pmod{n}$ to sign, where (n, e) is the authority's public RSA key. The user obtains the authority's signature, $(H(R)b^e)^{1/e}$, and divides it by b to obtain the authority's signature $H(R)^{1/e}$ on the value $H(R)$. The long-term key is then the low-order n_1 bits of R, as before.

The hash function approach makes it *computationally intractable* for the authority to obtain R from $H(R)$. The use of blinding makes *information-theoretically* impossible to determine $H(R)$ from $H(R)b^e$; no amount of computing power helps, since for any proposed value of $H(R)$ there is a corresponding value of b that makes $H(R)b^e$ yield the value given to be signed. With blinding, you can ask your worst enemy to sign something without having the slightest worry that he can figure out what he is being asked to sign.

The blind-signature approach does not require that hashing be used; the value of K_1 could be signed directly instead of signing $H(R)$. However, signatures of $H(R)$ may be more convenient than signatures of K_1 to work with, since the storage of the signature can then be treated separately from the storage of the key K_1, and would require less protection.

A nice feature of the signature approach is that the authority can control the *number* of distinct values of K_1 being used (even though it doesn't know *what* those values are), and can also control *by whom* they are used. For example, the authority could authorize as few as one long-term key per organization, or as many as one long-term key per instance of cryptographic equipment or software.

In a (significant) variation, we note that the authority can authorize *different* values of the length n_1 of the long-term key for different users. For example, the authority might authorize a 20-bit long-term key for a large financial organization, but authorize only a 6-bit long-term key for an individual. The length of the long-term key authorized could depend on any number of factors, such as the length of time it will be in use, the amount of encrypted traffic it will be used for, and the importance or "friendliness" of the user.

To implement such "custom-sized" long-term keys, the authority could sign (in the hash-function approach) pairs of the form $(n_1, H(R))$ instead of just signing $H(R)$. Implementing custom-sized long-term key authorization with blind signatures is more complicated, and seems to require separate public RSA keys for each key-length desired.

In order to support custom-sized long-term keys cleanly, we recommend the hash-function approach for signing long-term key authorization. That is, we recommend using an ordinary signature on the pair $(n_1, H(R))$ rather than using a blind signature.

3.3 Discussion and elaborations

We note that the user is responsible for keeping his long-term key secret. For a "user" that is actually a large organization, this may become difficult without the utilization of tamper-resistant hardware.

It is possible in principle for the user to check that his cryptosystem is working correctly, by verifying that it encrypts test data properly with the key K he has chosen. Thus, the user is assured that he obtains the full benefits of a randomly-chosen n-bit key.

The advantage of multi-grade cryptography increases as the expected number of keys used increases. Roughly, a user who uses d distinct keys could have a hidden parameter of roughly $\lg(d)$ bits without increasing the average workload for an intelligence agency. Thus, a user who uses one million keys might have a 20-bit value for the hidden parameter K_1. A

"68/48" scheme for such a user would be a big improvement over a simple 48-bit key limit.

It is not difficult to invent more elaborate multi-grade cryptographic schemes. We describe a few possibilities here.

One could require that the adversary solve t_1 instances of the "hard" (n-bit) problem before he is able to solve the remaining problems easily. For example, a "$64^{16}/48$" cryptosystem would require the adversary to solve $t_1 = 16$ hard problems before the remaining ones become easy. We note that a $64^{16}/48$ cryptosystem requires the same amount of work from an intelligence agency as a 68/48 system requires before things get easier, but allows him to decode the first 16 keys attacked with less work (only 2^{64} encryptions) per key.

How can this be done?

The basic idea is that K_1 can be determined as a function of K_0:

$$K_1 = f_u(K_0) \ .$$

The function f_u is randomly picked (once and for all time) by the user initially, as a member of public-known class of functions F. Then, every key (K_0, K_1) that the adversary determines (by a large search) gives him one more input-output example of the unknown function f_u. Computational learning theory provides many examples of function classes F that require (more-or-less-)controllable number of input-output examples to identify. For example, f_u could be a degree-$(t_1 - 1)$ polynomial (over some suitable field); then t_1 examples are required to learn f_u. The hidden system parameters are then the coefficients of this polynomial.

A second natural approach is to have more than two parts to the key. A "$70^1/64^{32}/40$" multi-grade cryptosystem would require an adversary to perform at least one 70-bit search, and then at least 32 64-bit searches, before the remaining keys are obtainable with only a 40-bit search each. It is easy to arrange such hierarchies, although it is not yet obvious what the best general approaches and techniques are for doing so. The basic idea is that level of keying involves new hidden parameters that can only solved for once the earlier parameters have been determined.

In another variation on the basic idea, keys might be triples (K_0, K_1, K_2), where K_0 is allowed to change daily, K_1 is allowed to change monthly, and K_2 is allowed to change yearly. The user would be assured that this year's keys are totally independent of last year's keys, while the export agency would know that the user's key changes could only require a large search once a year. Enforcing such a routine is similar to the original enforcement problem, and could be handled by similar means.

The proposal given here works most smoothly when the keys are used for file encryption or for encrypting intra-organization messages. In these cases, sender and receiver are either the same individual or share the same long-term key. When Alice and Bob are in different organizations and wish to communicate securely, care needs to be taken that they don't reveal their long-term keys to each other in establishing a common communication key. One approach is as follows. Let K_A denote a multi-grade key for Alice (containing her long-term key), and let K_B denote a multi-grade key for Bob. Alice and Bob agree on a session key K_S (say by using Diffie-Hellman key exchange), and then exchange values $E_{K_A}(K_S)$ and $E_{K_B}(K_S)$. These values are like "LEAF's" in the Clipper proposal. If an eavesdropper knows either K_A or K_B, he can compute K_S. The encryption algorithm needs to utilize these two values in an essential way, so that if they are accidentally or maliciously modified the ciphertext becomes useless to the recipient.

4 Conclusions

We have proposed a new paradigm for secret-key cryptography: multi-grade cryptography. Multi-grade cryptosystems are like Brazil nuts: hard to get open at first, but then much softer and chewier. Perhaps "crypto with the shells on" (multi-grade crypto) is more acceptable than "weak crypto" (shelled nuts) for export purposes. (I'd appreciate suggestions for better analogies!)

Acknowledgments

I'd like to thank Peter Schweitzer for a stimulating discussion that prompted me to think along these lines.

References

[1] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. Technical Report CS95-447, Dept. of Computer Science and Engineering, U.C. San Diego, October 1995.

[2] David Chaum. Blind signatures for untraceable payments. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proc. CRYPTO 82*, pages 199–203, New York, 1983. Plenum Press.

[3] Lance J. Hoffman, editor. *Building in Big Brother: The Cryptographic Policy Debate*. Springer-Verlag, 1995.

[4] D. B. Johnson, S. M. Matyas, A. V. Le, and J. D. Wilkins. Design of the commercial data masking facility data privacy algorithm. In *First ACM Conference on Computer and Communications Security*, pages 93–96, Fairfax, 1994. ACM.

[5] Silvio Micali. Guaranteed partial key escrow. Technical Report MIT/LCS/TM-537, MIT Laboratory for Computer Science, Sep 1995.

[6] Adi Shamir. Partial key escrow, 1995. Unpublished.

[The latest version of this paper can be obtained from
`http://theory.lcs.mit.edu/~rivest/publications.html`]