Secret-Key Agreement without Public-Key Cryptography (Extended Abstract)

Tom Leighton[1,2] and Silvio Micali[2]

[1] Mathematics Department and
[2] Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract. In this paper, we describe novel approaches to secret-key agreement. Our schemes are not based on public-key cryptography nor number theory. They are extremely efficient implemented in software or make use of very simple unexpensive hardware. Our technology is particularly well-suited for use in cryptographic scenarios like those of the Clipper Chip, the recent encryption proposal put forward by the Clinton Administration.

1 Introduction

1.1 The Problem of Secret-Key Agreement

Private-key cryptosystems are the most common type of cryptosystems; indeed, they are also refered to as "conventional systems." Their goal is to allow two parties A and B, who have agreed on a common and secret key K_{AB}, to exchange private messages via a network whose communication lines are easy to tap by an adversary. If properly designed, conventional cryptosystems are extremely fast, and believed to be very secure in practice. They are also very attractive from a theoretical point of view. In fact, provably-secure conventional cryptosystems can be build based on a very mild complexity assumption: the existence of one-way functions. (Roughly, these are functions that are easy to evaluate, but for which finding pre-images is hard.) Among so many advantages, these systems have a major drawback: agreeing beforehand on a common secret key with every one with which we wish to talk in private is not trivial. Certainly, meeting in a secure physical location is not a practical approach to obtain such an agreement. It is the goal of this paper to forward new, secure, and practical approaches to secret-key agreement. Prior to duscussing our ideas, let us briefly review the main approaches that have been considered so far.

1.2 Prior Approaches

Most of the protocols currently being used for key agreement are either classified or company-confidential. In the public domain, the most popular key-agreement protocols fall into two wide categories: (1) those based on public-key cryptography, and practically implemented with number theory (e.g., the Diffie-Hellman [4] and the RSA algorithms), and (2) those based on symmetric-key generation and a trusted agent, and practically implemented with some form of polynomial or integer arithmetic (e.g., those of Blom [2] and Blundo, De Santis, Herzberg, Kutten, Vaccaro, and Yung [3]). Unfortunately, these approaches are somewaht wanting both with respect to security and efficiency.

DRAWBACKS OF THE PUBLIC-KEY APPROACH. Although very elegant, the Diffie-Hellman and RSA algorithms require that number theoretic problems such as factoring or discrete-log be computationally intractable. In particular, the RSA scheme would become insecure if someone discovered a much improved algorithm for factoring large integers, and the Diffie-Hellman algorithm would suffer a similar fate if an improved algorithm were found for computing discrete logs. If either scheme is used to select the session keys for all government traffic, then all this traffic would be decipherable to anyone who found improved algorithms for these problems. (Although it might be the case that these problems are truly intractable, it would be nice if there were a key agreement protocol that was secure even if there are algorithmic advances made in number theory in future decades.)

Most likely, similar drawbacks will be suffered by any other proposed solution based on the framework of public-key cryptography. This framework is very "distributed" in nature; namely, every user A *individually chooses* a pair of matching encryption and decryption keys (E_A, D_A), publicizes E_D and keeps secret D_A. Any message encrypted via E_A can be easily (and, hopefully, solely) decrypted via the corresponding key D_A. Since the E_A is made public, any one can send A a private message, because any one can encrypt via key E_A. In this setting, it is conceptually very easy for two parties A and B to agree on a common key K_{AB}. For instance, B may individually choose K_{AB} at random and, since E_A is public, send K_{AB} to A encrypted via E_A. This key is common because A can decrypt it thanks to his private knowledge of D_A. For K_{AB} to be secret for everyone else, however, several conditions must be met; in particular, D_A must not be easily computable from E_A. Indeed, public-key cryptography requires very strong complexity assumptions: the existence of *one-way trap-door predicates* or that of *one-way trap-door functions*. (The latter, roughly, are functions that not only are easy to evaluate and hard to invert, but *also* possess an associated secret whose knowledge allows one to easily invert them.) Such assumptions appear to be much more demanding than existence of a one-way function. Indeed, while the existence of one-way functions is widely believed (and plenty of candidates are available), trap-door one-way functions may not exist or may be very hard to find (indeed, only a handful of them have been proposed without being immediately dismissed). In sum, therefore, while one-way functions are sufficient to communicate securely in a conventional cryptosystem once a common secret key has been established, the process of establishing such a key based on public-key cryptography appears to require a much stronger type of assumption, thereby creating a weaker link in the overall security.

In addition to the above concerns about security, and perhaps of more immediate importance, key agreement protocols based on public-key cryptography and number theory tend to be very expensive to implement. Indeed, the cost of building hardware that can quickly perform modular exponentiation is far greater than the cost of building encryption devices based on one-way functions. Indeed, when providing encryption devices to a country of the size of the United States, the cost of the key-agreement hardware is far from being insignificant.

SECURITY DRAWBACKS OF THE TRUSTED-AGENT APPROACH. In this approach there are three main parameters: N, the total number of users in the system, k the length of a common secret key, and B, a bound on the number of (collaborating) malicious users. The algorithms of [2] and [3] make use of polynomial and integer arithmetic for implementing a *Symmetric Key-Generation System.* This consists of computing (from a single, secret, system value K) a set of n secret values, $K_1, \ldots, K_n$, which in turn yield n^2 quantities, K_{ij}, satisfying the following properties. For each i and j the quantity K_{ij} can be computed easily either on inputs j and K_i, or on inputs i and K_j. Moreover, given any B individual values, $K_{a_1}, \ldots, K_{a_B}$, one has no information about the quantity K_{ij} whenever $i, j \neq a_i, \ldots, a_B$.

A symmetric key-generation system can be effectively used by trusted agent to enable the users to provide an elegant solution to the secret-key agreement problem. The trusted agent (after choosing the system master secret) simply computes n secret values $K_1, \ldots, K_n$ and assigns to user i value K_i as his individual secret key. The common key between two users i and j will then be K_{ij}, which can be easily computed by either one of the two users. This key is secret in that no coalition of less than B users, no matter how much computation they perform, can infer the common of two other users. While this is a very attractive property, relying on a trusted agent for assigning the proper, individual secret keys is in itself from a security point of view, a serious drawback.

From an efficiency point of view, while the algorithms of [2] and [3] require small individual keys (i.e., $\Theta(Nk)$ bits per user), they do require a fair amount of algebraic computation in oreder to compute common secret keys from individual keys. Even if this computation could be sped up in a hardware implementation, the cost of this circuitry may not be trivial with respect to that required for conventionally encrypting messages via a one-way function, after common secret keys have been established. Moreoever, for both security and efficiency reasons, conventional encryption schemes are not algebraic; thus the "algebraic hardware" necessary for their secret-key agreement represents pure additional cost, since it will have very little to share with the "encrypting hardware."

1.3 Our Contribution

In this paper, we advocate two new approaches to secret-key exchange, none of which is based on public-key cryptography or polynomial/integer arithmetic.

In the first we introduce a new class of symmetric key-generation systems requiring longer individual keys than [2] or [3], but guaranteeing a that computing common secret keys is absolutely trivial. To enhance the security of these algorithms further we recommend using them in conjunction with special hardware and a set of moderatly trusted agents. This whole approach is described in Section 2.

The second approach requires a simple interaction with moderatly trusted agents, and can be impleneted in software with great efficiency and security (and, of course, wit great savings). This approach is described in Section 3.

Both approaches are information-theoretically secure, though they can be more convenently impemented if *ordinary* one-way functions are used. (In any case, therefore, they will be immune to attacks based on advances in number theory.)

Finally, both of our approaches are capable, *if so wanted by society*, of making very secure encryption "compatible with law-enforcement;" that is, in case a court authorizes tapping the communication lines of users suspected of illegal activities (and only in case of these legitimate authorizations), the relevant secret keys can be reconstructed by the Police. This is indeed the major feature of two recent encryption proposals, *Fair cryptosystems*, as put forward by the second author, and *the Clipper Chip* as put forward be the Clinton administration. To illustrate both this additional important point and the cryptographic use of tamper-proof hardware that may enhance the security of our first scheme, let us provide a brief introduction to the government proposal.

1.4 The Clipper Chip Project

In April, 1993, the Clinton Administration announced its intention to develop a cryptographic scheme for widespread use within the government. The scheme is centered around a device known as the *Clipper Chip* which is expected to become standardized for encryption and decryption of telephone, fax, email, and modem traffic. The Clipper Chip does not offer a solution to the secret-key agreement problem; rather, it *assumes* that such a solution exists. Its goal is making conventional cryptosystems compatible with law-enforcement.

The Clipper Chip will be made using a special VLSI process which is designed to prevent reverse engineering. In particular, the conventional encryption and decryption algorithms used on the Clipper Chip will be classified, but the chip itself will not be classified. The Clipper Chip will also contain a protected memory for secret keys. The protected memory is designed to prevent anyone (even the legitimate user of the chip) from gaining access to the keys contained therein.

According to the government press release, each Clipper Chip will be equipped with a unique secret key K_i that is formed by an irreversible process from two pieces of the secret key $K_i^{(1)}$ and $K_i^{(2)}$. The pieces of the secret keys will be held by system-wide trusted agents $\mathcal{T}_1$ and $\mathcal{T}_2$. (Actually, only one of the agents needs to be trusted since $\mathcal{T}_1$ will hold only the first piece of each secret key and $\mathcal{T}_2$ will hold only the second piece.) When two parties wish to communicate using the new system, they first agree on a session key S and they enter this key into their respective Clipper Chips. This key is used by the Clipper Chips as an encryption/decryption key for the message traffic. In other words, once the session key is selected, the Clipper Chips function as a private-key cryptosystem.

There is a major difference between the Clipper Chips and a conventional private-key cryptosystem, however. That is, the Clipper Chips also transmit the session key S being used in encrypted form using the secret key for the chip, thereby allowing the trusted agents to eavesdrop on the conversation. The reason for transmitting the session key in this fashion is so that law enforcement can (upon obtaining the relevant court order) obtain the secret key of the user from the trusted agents and then decrypt the conversation (thereby preserving current wiretapping capabilities) but no other unauthorized person can eavesdrop (thereby providing greater privacy than exists currently for most individuals).

1.5 Our Contribution to the Clipper Chip Project

In the proposed Clipper Chip project, it is assumed that every pair of users has already agreed on a common, secret, (session) key. In practice, therefore, devices that incorporate the Clipper Chip will use a specific key-agreement protocol, or a crucial link would be missing. This, however, has the potential of introducing a host of new difficulties. For instance, if public-key cryptography is used for providing this missing link, the system might become more vulnerable (since it now must rely on stronger —and possibly false— comlexity assumptions) and much more costly.

By contrast, as we shall see, our two schemes can greatly enhance the security and the economicity of the Clipper Chip project. In fact, not only can we guarantee compatibility of law-enforcement with strong encryption, but, within the same framework (without additional costs or loss of security), also the necessary secret-key agreement that was missing in the Administration proposal.

2 A Hardware-Based Approach

For simplicity of exposition, the secret-key agreement of this section is described in two phases. The first phase consists of a special class of symmetric key-generation systems —and thus of a basic scheme for secret-key agreement based on a trusted agent.

In the second phase, we show how to enhance the security of our basic schemes assuming the availability of tamper-proof hardware and the existence of a group of "only moderately trusted" agents rather than a single, totally trusted one.

2.1 The First Basic Scheme

The symmetric key-generation scheme described in this subsection is both very efficient and quite natural. We thus expect that it might have beed already discussed in the literature, but we have been unable to find such reference. We of course be very grateful to anyone who can provide us with such a piece of information.

Recall that in a symmetric key-generation system there are three relevant parameters: N, the total number of possible users in the system; B, an upperbound on the size of a coalition of dishonest users, and k, the number of unpredictable bits contained in each common secret key of two honest users (e.g., $k = 100$). For didactic purposes, however, we shall make use of an auxiliary parameter M, and then show that M should be about $B^3 \ln N$.

COMPUTING INDIVIDUAL KEYS. On input N, B, and k, the trusted agent performs the following steps:

- First, he randomly and secretly selects M, k-bit long, *system secret keys*: $X_1, \ldots, X_M$, where $M = \Theta(B^3 \ln N)$ (the precise constant will be worked out in the final paper).
- Then, he constructs (in $poly(N, M)$ time) a $N \times M$ $0-1$ matrix $A = \{a_{i,n}\}$ with the properties that:
  1) any pair of rows have 0's in at least $F = \Theta(M^{1/3} \ln^{2/3} N)$ common columns, and
  2) any triple of rows all have 0's in at most $G = \Theta(\ln N)$ common columns.
  1. $B < \lceil F/G \rceil$.

  (We will show how to construct such a matrix shortly.)
- He then assignes to player i, where i is a $\log N$-bit identifier, the individual secret key consisting of the vector $(v_{i,1}, \ldots, v_{i,M})$, where $v_{i,n}$ equals the secret system key M_n if $a_{i,n} = 0$, and the empty word otherwise.

COMPUTING COMMON SECRET KEYS. The common secret key of users i and j, $K_{i,j}$, consists of the M-vector whose nth component equals M_n if $a_{i,n} = b_{j,n} = 1$, and the empty word otherwise. Thus, each user trivially computes his common secret key with another user by taking a subset of the system secrets in his possession.

(If so wanted, the size of these common secret keys can be reduced; for instance, by evaluating a one-way hash function on them. Such reductions are not, however, a concern of this paper.)

SECURITY. Assume, for now, that a matrix A as above has been constructed. Then, given the individual secret keys of a set of B users, the common secret key of two users contains at least k random and unpredictable bits. The proof is very simple. By Property 1, above, every common secret key $K_{i,j}$ must contain at least F system secrets. By Property 2, however, the individual secret key of a user other than i or j contains at most G of these secrets. Hence, at least $\lceil F/G \rceil$ other individual secret keys are necessary in order to recover all of the system secrets in a common secret key.

CONSTRUCTING THE MATRIX. Next, let us show that a matrix A satisfying Properties 1 and 2 above exists. We do this by showing that such an A can be constructed with probability > 0 by the following probabilistic algorithm.

Set each entry of A to 0 with probability $p = \Theta\left(\left(\frac{\ln N}{M}\right)^{1/3}\right)$. The probability that Property 1 is not satisfied is then at most

$$\sum_{h=0}^{F-1} \binom{N}{2}\binom{M}{h}(1-p^2)^{M-h}(p^2)^h \le \sum_{h=0}^{F-1} \frac{N^2}{2}\left(\frac{Mep^2}{h}\right)^h e^{-p^2(M-h)}$$

$$\le \sum_{h=0}^{F-1} \frac{N^2}{2}\left(\frac{Me^{1+p^2}p^2}{h}\right)^h e^{-p^2 M}$$

$$\le \frac{FN^2}{2}\left(\frac{Me^{1+p^2}p^2}{F}\right)^F e^{-p^2 M}$$

provided that $F \le Mp^2$. Similarly, the probability that Property 2 is not satisfied is at most

$$\binom{N}{3}\binom{M}{G+1}(p^3)^{G+1} \le \frac{N^3}{6}\left(\frac{Mep^3}{G}\right)^G .$$

Hence, the probability that A fails to satisfy either property is at most

$$\frac{FN^2}{2}\left(\frac{Me^{1+p^2}p^2}{F}\right)^F e^{-p^2 M} + \frac{N^3}{6}\left(\frac{Mep^3}{G}\right)^G . \qquad (1)$$

By setting $p = 2(\frac{\ln N}{M})^{1/3}$, $F = \frac{1}{2}M^{1/3}\ln^{2/3} N$, and $G = 16e\ln N$, we find that if $M \ge 8\ln N$, then this probability is strictly less than 1, as desired. (It is worth noting that substantially better constants can be derived for particular values of N and M with a more careful analysis.)

Finally, let us show that such a matrix A not only exists, but is also easy to compute deterministically. This is so thanks to the *method of conditional probabilities* [1]. In fact, it should be noted that the task of evaluating a simple upper bound on the probability that either Property 1 or Property 2 is violated (as in Equation 1) conditioned on some of the values in A being fixed is easily accomplished in polynomial time.

(Indeed, matrix A needs not to be computed "at once," but can be easily computed row-by-row. This way, the cost of a row can be incurred only when one more user —of a budgeted total of Nusers— joins the system.)

In the case when N is small, even better constructions for A exist. For example, when $N = M^{2/3}$, we can construct an A for which $F = M^{1/3}$ and $G = 1$ by letting each row of A correspond to a plane through an $M^{1/3} \times M^{1/3} \times M^{1/3}$ lattice of points mod$M$. By choosing non-parallel planes for the rows of A, each pair of planes will intersect in a line, and each triple of planes will intersect in exactly one point, thereby achieving the desired bounds for F and G. In this example, it suffices to have $B = M^{1/3} - 1$.

EFFICIENCY. Given what we have said so far, the bit-length of an individual key is about $kB^3 \log N$. An individual key, in fact, consists of a subset of the

$M = \Theta(B^3 \log N)$ system secrets. One must observe, however, that the subset tends to be quite sparse. Indeed, we can construct matrices A enjoying also the following additional property:

0) Every row has at most $O(M^{2/3} \log^{1/3} N)$ 0's.

The number of system secrets entering an individual key is thus about $B^2 \log N$.

Though this is longer than the kB bits of individual key (roughly) needed by an individual key in the Blom' algorithm with the same parameters, in our case the construction of a common secret key is trivial, since it only consists of taking a subset of stored secrets. It is unquestionable that, whether in software or in hardware, this operation is much preferable to computing with B-degree polynomials over fields with size-k elements.

2.2 The Second Basic Scheme.

Also this scheme should have, in our opinion, been discussed in the litterature. Nonetheless, we have not been able to find it.

In this scheme the trusted agent distributes to the users longer individual secret keys. As we shall see in the subsection after next, however, the scheme offers additional advantages if some special hardware is available. This scheme too has a perfectly-secure version, but one-way functions can be used in order to shorten the length of the individual secret keys.

COMPUTING INDIVIDUAL SECRET KEYS. Parameters N, B, k, and M are as in the first scheme. The secod basic scheme has, however, an additional and independent parameter $L \geq 2$.

In the perfectly secure version, the trusted agent computes individual keys as follows.

- He chooses ML, k-bit long, system secrets, $\{X_{i,n}\}$, where i ranges between 1 and M, and n between 0 and $L-1$.
- He assigns to each user in the system an identifier consisting of M random values between 1 and $L-1$, $\alpha = (\alpha_1, \ldots, \alpha_M)$. (In practice, the α_i values can be generated by applying a proper cryptographic function to the "real name" of the user.)
- Then, he assigns to user $(\alpha_1, \ldots, \alpha_M)$ the secret key consisting of the (properly ordered) subset of system secrets $\{X_{i,n} \ : \ i \in [1, M] \ \alpha_i \in [0, L-1]\}$.

COMPUTING COMMON SECRET KEYS. The common secret key of two users with identifiers $\alpha = (\alpha_1, \ldots, \alpha_M)$ and $\beta = (\beta, \ldots, \beta)$ consists of the (properly ordered) set of system values $\{X_{i,max_i}, \ldots, X_{i,L-1-max_i} \ : \ i \in [1, M] \ max_i = max(\alpha_i, \beta_i)$.

(Again, such a key can be substantially reduced in size, if so wanted, but this is not a concern of this paper.)

THE COMPUTATIONALLY-SECURE VERSION. The above scheme requires quite long individual keys. This drawback can be eliminated by using one-way functions. This, of course, will turn the perfect security of the above scheme into computational security, but this is quite tolerable, since the conventional encryption that follows the key-agreement can be at most be computationally secure.

The most straightforward way to use one-way functions to shorten individual key-length in our second basic scheme consists of having the trusted agent set $X_{i,n} = h^n(X_i)$ where X_i is the ith system secret (randomly selected as before) and h is a one-way function. The user identifiers are chosen like in the perfectly-secure version, but the individual key of user $\alpha = (\alpha_1, \ldots, \alpha_M$ now is $(h^{\alpha_1}(X_1), \ldots, h^{\alpha_M}(X_M))$, since user α can reconstruct all other values by evaluating h in the easy direction. This version of our scheme approximates the perfectely-secure one in that the value $X_{i,n}$ is computationally hard to predict from the values $X_{i,m}$ for $m > n$. Nonetheless, if the correct value $X_{i,n}$ is supplied, one can verify its correctness by evaluating h sufficiently many times on it.

In most cases, the above approximation can be deemed sufficient. But, though this lies beyond the scope of this paper, let us mention that a better way to approximate computationally the perfectly-secure, second basic scheme, can be obtained by using a slight variation of the pseudo-random function construction of Goldreich, Goldwasser, and Micali [5]. In particular, one can guarantee that, to someone who is given $X_{i,n+1}$, $X_{i,n}$ is undistinguishable from a random value of the same length. The lenth of an individual key, however, becomes $kM \log L$.

SECURITY. In order for an adversary to compute the common secret key of two users with identifiers $\alpha = (\alpha_1, \ldots, \alpha_M)$ and $\beta = (\beta_1, \ldots, \beta_M)$, he will need to obtain enough individual keys so as to have all system values X_{m,ρ_m} ($Z_m = h^{\rho_m}(X_m)$ in the computationally secure scenario) where $\rho_m \leq max(\alpha_m, \beta_m)$ for $1 \leq m \leq M$. In order for the system to be secure, then, we would like it to be the case that no matter for what set of B users, if the adversary enters in possession of the individual keys of those users, he still does not have enough information to recover a common secret key for any other two users in the system.

To this end, we need to select the identifier $K_i = \{\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,M}\}$ for each user i so that for any $i_1, i_2, \ldots, i_B \in [1, N]$ and any $j_1, j_2 \in [1, N]$ such that $i_r \neq j_s$ for $1 \leq r \leq B$ and $1 \leq s \leq 2$, there exists an integer t ($1 \leq t \leq M$) such that

$$\min_{1 \leq r \leq B} \{\alpha_{i_r,t}\} > \max_{1 \leq s \leq 2} \{\alpha_{j_s,t}\}.$$

In other words, for any pair of users j_1 and j_2 and any set of B users $i_1, i_2, \ldots, i_B$ whose individual keys are learned by the adversary, there needs to be at least one X_t such that none of the B learned keys contains $X_{t,\omega}$ ($h^\omega(X_t)$ in the computationally-secure scenario) for any $\omega \leq \max(\alpha_{j_1,t}, \alpha_{j_2,t})$.

Fortunately, it is not too difficult to find identifiers for the users that satisfy the preceding property. For example, if L is a multiple of B and B is about $(M/e \ln N)^{1/3}$, then this property is likely to hold if each $\alpha_{i,m}$ is chosen uniformly at random in $[0, L-1]$. This is because the probability that

$$\min_{1\le r\le B}\{\alpha_{i_r,t}\} \le \max_{1\le s\le 2}\{\alpha_{j_s,t}\}. \tag{2}$$

for some particular values of $i_1, i_2, \ldots, i_B, j_1, j_2$, and t is at most

$$1-\left(\frac{1}{B}\right)^2\left(1-\frac{1}{B}\right)^B \tag{3}$$

since there is always a $\frac{1}{B^2}$ chance that $\alpha_{j_1,t}, \alpha_{j_2,t} < \frac{L}{B}$ and a $\left(1-\frac{1}{B}\right)^B$ chance that $\alpha_{i_1,t}, \ldots, \alpha_{i_B,t} \ge \frac{L}{B}$. For $B \ge 2$, $(1-\frac{1}{B}) \ge e^{-\frac{1}{B}-\frac{1}{B^2}}$ and thus

$$\begin{aligned}1-\left(\frac{1}{B}\right)^2\left(1-\frac{1}{B}\right)^B &\le 1-\frac{e^{-1-\frac{1}{B}}}{B^2}\\ &\le 1-\frac{e^{-1}(1-\frac{1}{B})}{B^2}\\ &\le 1-\frac{e^{-1}(B-1)}{B^3}\end{aligned}$$

Hence, the probability that Equation 2 holds for all t, $1 \le t \le M$ is at most

$$\left(1-\frac{(B-1)e^{-1}}{B^3}\right)^M \le e^{-(B-1)e^{-1}M/B^3}.$$

Summing this probability over the

$$\binom{N}{B+2} \le \left(\frac{Ne}{B+2}\right)^{B+2}$$

possible choices for $i_1, i_2, \ldots, i_B, j_1, j_2$, we find that the probability that any pair key can be recovered by opening any B other chips is at most

$$\left(\frac{Ne}{B+2}\right)^{B+2} e^{-(B-1)e^{-1}M/B^3} = e^{(B+2)[\ln N+1-\ln(B+2)]-(B-1)e^{-1}M/B^3}$$

which becomes small when B is about $(M/e\ln N)^{1/3}$.

If L is larger than B^2, then the preceding sort of analysis can also be used to show that the probability that any pair key can be recovered by opening B other chips is small when B is about $(2M/\ln N)^{1/3}$. The key difference is that when L is large compared to B^2, the probability in Equation 3 can be improved to

$$1-\frac{(1-o(1))2B!}{(B+2)!} = 1-\frac{2-o(1)}{B^2},$$

which is smaller than $1-\frac{e^{-1}(B-1)}{B^3}$ for large B.

EFFICIENCY. Given what we have said so far, in the computationally secure version, the length of an individual key equals that of the first basic scheme:

$kB^3 \ln N$ bits. The time needed to compute a common secret key from an individual one is however slightly longer, consisting of $M(L-1)$ one-way function evaluations. Since one-way functions are particularly fast to evalaute, however, this operation, implemented in software, may still be negligible with respect to the equivalent one of [2] and [3]. Our common-secret-key computation also is more convenient than that of [2] and [3] with respect to hardware implementations. Not only because the circuitry needed for one-way function evaluation tends to be simpler than that of polynomial arithmetic, but also because the circuitry necessary to evaluate one-way functions needs to be present *any way*, in order to encrypt messages (or session keys) once a common secret key has been computed.

Let us just mention in passing, however, that, like for the first basic scheme, the length of an individual key can be reduced to about $B^2 \log N$, but the initial (omitted) constant can be better than that of the first scheme. (Details will be given in the final paper.)

The real advantage of the second scheme, however, comes in if a special type of hardware is available. This advantage is discussed in section 2.5.

2.3 Optimality of our Basic Schemes

We now show that the schemes for assigning public keys that were just described are all optimal to within a logarithmic factor among those where common secret keys are constructed as "subsets of common secrets." In particular, we will show that no matter how large L is and no matter how public keys are assigned, then there is always a set of $B = \Theta(M^{1/3} \log^{2/3} M)$ chips which, when opened, will lead to the recovery of a pair key for some other pair of users. Because of space limitations, we will only sketch the proof in this extended abstract, and we will not worry about constant factors. (In fact, the constant hidden behind the Θ is quite small.)

Given any set of N public keys $\{K_i | 1 \leq i \leq N\}$ where $K_i = \{\alpha_{i,m} | 1 \leq m \leq M\}$, define an $N \times M$ matrix $A = \{a_{i,j}\}$ as follows:

$$a_{i,j} = \begin{cases} -1 \text{ if } \alpha_{i,j} \text{ is the smallest item in } \{\alpha_{r,j} | 1 \leq r \leq N\}, \\ 0 \text{ if } \alpha_{i,j} \text{ is among the next smallest } \frac{N \log^{1/3} M}{M^{1/3}} \text{ items in } \{\alpha_{r,j} | 1 \leq r \leq N\}, \\ 1 \text{ otherwise.} \end{cases}$$

Ties can be broken arbitrarily. Thus, each column of A will have one -1, $\frac{N \log^{1/3} M}{M^{1/3}}$ 0's, and $N - 1 - \frac{N \log^{1/3} M}{M^{1/3}}$ 1's.

We next use the pigeonhole principle to show that there are two rows u and v in A such that

1) neither row u nor row v contains any -1's, and
2) there are at most $F = \Theta(M^{1/3} \log^{2/3} M)$ columns for which both rows u and v contain a 0.

This is because at most M rows contain a -1, and at most

$$\binom{\frac{N \log^{1/3} M}{M^{1/3}}}{2} \leq \frac{N^2 \log^{2/3} M}{2M^{2/3}}$$

pairs of rows can both contain a 0 in column j for any j. Hence, there exists a pair of rows satisfying Properties 1 and 2 above if

$$\binom{N-M}{2} F > \frac{MN^2 \log^{2/3} M}{2M^{2/3}}. \tag{4}$$

Equation 4 is satisfied for $F = \Theta(M^{1/3} \log^{2/3} M)$ provided that $N \geq 2M$.

We next show how to find $\Theta(M^{1/3} \log^{2/3} M)$ chips which, when opened, can be used to recover the pair key for u and v. The first step towards this goal is to find $\Theta(M^{1/3} \log^{2/3} M)$ rows of A (other than u and v) such that for each column t, one of these rows contains a 0 in column t. We can find such a set of rows because every column of A has $\frac{N \log^{1/3} M}{M^{1/3}}$ 0's and at most 2 of these zeros can be in rows u and v. In particular, there is always a row in A (besides u and v) with at least

$$M \left(\frac{N \log^{1/3} M}{M^{1/3}} - 2 \right) / N = \Theta(M^{2/3} \log^{1/3} M)$$

0's. When we remove this row (call it i_1) and any columns j in A for which $a_{i_1,j} = 0$, there is still a row with $\Omega(M^{2/3} \log^{1/3} M)$ 0's in the remaining columns. Continuing in this fashion, we can easily find a collection of $O(M^{1/3} / \log^{1/3} M)$ rows which collectively have at least one 0 in half of the columns. Continuing further, we can find a collection of $O(M^{1/3} \log^{2/3} M)$ rows which collectively have at least one 0 in each column. By opening the chips corresponding to these $O(M^{1/3} \log^{2/3} M)$ rows, we can then recover the portions of the secret pair key for u and v corresponding to columns t for which $\max\{a_{u,t}, a_{v,t}\} = 1$.

All that remains is to open the $F = O(M^{1/3} \log^{2/3} M)$ chips corresponding to the rows ω for which $a_{\omega,t} = -1$ where t is one of the $F = O(M^{1/3} \log^{2/3} M)$ columns for which $a_{u,t} = a_{v,t} = 0$. These chips contain the remaining secrets necessary to reproduce the pair key for users u and v. Hence B can be at most $O(M^{1/3} \log^{2/3} M)$ if the system is to be secure.

2.4 Enhancing the Security of the Basic Scheme.

Multiple trustees. Let us now modify the basic schemes described above so that they no longer need such a strong assumption as the existence of a trusted agent selecting and distributing individual secret keys to the users. At the simplest level, as envisaged in [6] and in the Clipper Chip scenario, this modification consists of replacing the trusted agent with a group of agents that are moderatly trusted. That is, trusted collectively, but not necessarily individually.

For instance, there may be a set of t independent trustees, each one of which acts as the trusted agent of Sections 2.1 and 2.2. Thus, denoting by $K^n_{i,j}$ the secret common key between users i and j relative to the nth trustee, the *overall* secret common key between i and j, $K_{i,j}$, will be the XOR of the $K^n_{i,j}$'s. It is easily seen that this does not effect the security and the all the relevant properties of the trusted agent scenario. In addition, if society so wants, this way of operating also allows one to make cryptography compatible with law enforcement. In fact, the two trusted agents can give, when presented with a court order, the two individual secret keys of a suspected user i, K^1_i and K^2_i, to the Police. Thus the Police will be allowed, in case of a court order, to understand any message sent or received by i, because it can easily compute all possible common secret keys between i and other users.

SECURE CHIPS. A second modification that may greatly enhance the security of our basic schemes depends on the availability of secure chips. Recall that these are chips whose memory (or parts of it) cannot be read from the outside, and that cannot be tampered with without destroying their content. Given such chips, each of the trustees can then store the individual key of a user into a secure chip. Once a chip has all the required individual keys (i.e., t if there are t trustees), is given to its proper user. We further recommend that he should not see any of the common secret keys enabling him to communicate privately with other users. In fact, we recommend that the only way for one to use his own chip consists of entering the chip a message and a recipient identity and that the only action taken in response by the chip consists of (1) internally (and thus secretly) compute the right common secret key, and then (2) outputting the encryption of that message with that key. (In particular, if the input to the chip is not of the proper form, the chip will take no action, but it will never output part of a secret key, nor will it answer in any way to any "input request" about a secret key.)

This step makes it harder for a coalition of malicious users to compute the common secret key of two honest users. In fact, malicious users no longer know their own individual keys. Even if one assumes that, with a lot of effort and with a substantial investment in sophisticated equipment, one may succeed in reading the content of at most A chips, by selecting the individual keys with parameter $B > A$, one can guarantee that no common secret key between two uncompromised chips can be computed.

The inability of computing common secret keys does not, however, quite coincide with the inability of eavesdropping. Indeed, Yuliang Zheng [8] has observed that, if $B = 0$, then there may exist a *single* chip ω whose individual secret key allows one to compute the common secret key, $K_{\alpha,\beta}$, of two other chips α and β, and that, if this is the case, chip ω can be used (without opening it) in order to eavesdrop communications between α and β. (He also shows, however, that by having the common secret key of two users also depend from their identifiers makes eavesdropping impossible even in that special case. It would be nice to know, by the way, where are such special cases exist, or to have a formal proof that no other sich a case exist.)

ENHANCING THE SECURITY OF PRIOR KEY-DISTRIBUTION SCHEMES. These enhancements can also be used to improve the security of the algebraic schemes of Blom [2] and Blundo et al. [3]. Namely, assume that there are two (for simplicity) trustees, $\mathcal{T}_1$ and $\mathcal{T}_2$, at least one of which is honest. Then, each trustee $\mathcal{T}_i$ can produce up to n secret values K_i^x's, and store K_i^x in the protected memory of chip of user i as his secret individual key, without revealing its value to him. Then, the common secret key of two users i and j can be easily computed (without interaction) by either of their tamper-proof chips as a fixed combination of K_{ij}^1 (i.e., their common secret key as if $\mathcal{T}_1$ were the only trusted agent) and K_{ij}^2 (i.e., their common secret key as if $\mathcal{T}_2$ were the only trusted agent. This way, gathering together more than B secret individual keys is harder (because all such keys are stored in protected hardware and are not known to the users) and, at the same time, one no longer relies on a single trusted agent. In addition, also these so modified schemes may make, if so wanted, encryption and law enforcement compatible.

These security-enhanced schemes, however, will be less efficient and more expensive than our enhanced ones.

2.5 Partially-Openable Chips

So far we have assumed that an adversary who succeeds in opening the protected memory of a chip, succeeds in reading all of this memory. It may be more realistic, however, to assume that, by tampering with a properly protected chip, an adversary can obtain at most a few bits from the protected memory of each chip before he destroys the rest of the bits. In this case, our second basic scheme may, by using a large parameter L, make the adversary's work dramatically difficult. This is so because learning —say— 5 bits of $h^n(X_m)$ for different values of n will be of little help to the adversary. Thus the ability of opening random chips no longer is very useful: the adversary must open many chips with exactly the same portion of the secret key if he wants to obtain "useful information." But, when L is large, even the simple step of *getting hold* of a large number of chips with such identifiers (and thus such secret keys) may be overwhelmingly hard. In fact, in the cases of the first basic scheme and the second one with $L = 2$, the adversary needs to partially open about $B(B^2 \log N)k/5$ chips (if at most 5 bits can be recovered from such an opened chip) in order to compute the individual keys of B chips. By contrast, in the second basic scheme, when the parameters L increases (and all other parameters remain fixed), the number of chips that need to be opened tends to $LB(B^2 \log N)k/5$. It then becomes clear that even for reasonable values of L, in most applications there will not be that many chips in the system. Thus no adversary has a realistic chance of computing the common secret key of two uncompromised chips even if he gets hold and partially opens all the other ones. This is even more true if one assumes (like it may be realistic to do) that before succeeding in partially opening a chip the adversary is expected to destroy a few chips in the process. In fact, many more than $LB(B^2 \log N)k/5$ chips are needed in this case.

Additional protection can be obtained if chips are initialized with users identifiers that are chosen as an unpredictable function of the names of their users.

Security Hierarchies The previous scheme can be easily modified for use in a scenario where there are various gradations of security. For example, assume that the users are categorized into S security levels $1, 2, \ldots, S$, where level 1 is the highest level of security, and level S is the lowest level of security. Then, we can use the same scheme as before except that the public key for a user at security level q is selected so that α_m is a random integer in the range $[1 + (q-1)L, qL]$ for $1 \leq m \leq qM/S$.

The modification increases the time to compute a pair key by a factor of at most S, but it has the nice feature that conversations between users will always take place at the highest (i.e., most secure) common level of security. In particular, in order for an adversary to recover the pair key for a conversation between users at security level q or better, the adversary will need to open at least B chips of security level q or better where B is as defined above (with M decreased by S). Since there are likely to be fewer chips at the higher security level, and since they are likely to be guarded more closely, it will be much more difficult for an adversary to obtain such chips, and he will have to open more of them before being able to recover a pair key (since we can replace N by the number of users with that level of security, which is smaller).

3 A Software-Based Approach

Let us now describe a scheme for exchanging keys that does not rely on any protected hardware at all. The scheme is very simple and, in its basic version, only relies on a trusted agent. (We shall describe how to implement the scheme with multiple trustees later.)

Again, it seems to us that this very scheme should have been considered before, but we have not been able to find it in the literature. Our software-based approach shares, however, some similarities with that of Needham and Schroeder, and it is actually useful to to contrast the two of them.

3.1 The Needham-Schroeder Approach Versus Ours

THE NEEDHAM-SCHROEDER PARADIGM. Outside the public-key cryptography framework, the most popular approaches to secret-key agreement follow a paradigm put forward by Needham and Schroeder [7], whose essential features are summarized below.

A trusted agent T assigns to each user i in the system an individual secret key K_i. This key is thus a common secret between the user and T. Whenever a user i wishes to talk in private to another user j, i sends T a message (in the clear) specifying his own identity and that of the recipient, j. The trusted agent answers this request by selecting a new session key S and sending i a global message, encrypted with key K_i, consisting of two values: S and E, where E is an encryption of S with key K_j. User i, after recovering S and E, sends j two values: his intended message encrypted with S, and E. User j, thanks to his knowledge of K_j, recovers S from E, and then recovers the message.

(After this basic step, i and j also engage in another protocol to check that indeed S is a common key to both of them. We have decided to ignore these additional protocols in this paper.)

CRITIQUE OF THE NEEDHAM-SCHROEDER PARADIGM. Many objections can be and have been moved to the above paradigm for secret-key exchange; in particular:

1. *It requires that the trusted agent be continuously available.*
   Indeed, if the communication link between a user and the trusted agent is down, then that user is deprived of the possibility of communicate privately.
2. *It exposes arbitrarily many cleartext-ciphertext pairs.* Indeed, whenever user i requests to speak to user j he will receive, for free, a session key S and its encryption (with j's secret key).
3. *It requires encryption to provide authentication.*
   Assume that the encryption between i and T consists of exclusive-oring messages with a one-time pseudo-random pad generated on input K_i. Also assume that it is an enemy (instead of T) who send i the global message, and that he chooses it to consist of two random values: α and β. Then, i will compute the decryptions, S and R, of, respectively, α and β. Of course, it is very unlikely that R will be an encryption of S with key K_j. Thus a message sent by i to j encrypted with S will not be understood by j. But i has no way to realize this without engaging in an additional protocol with j. Thus, even if a good encryption scheme (such as exclusive-oring with a pseudo-random pad) is used in the Needham-Schroeder approach, an additional interactive protocol between users becomes more of a necessity than an option. This is a pity, because now their approach stops from being very simple, and is also made less efficient. Moreover, such additional protocols need to be secure, something that it is not trivial to achieve —at least if one wishes to keep them very simple.
   (Indeed, as we have already mentioned, such an additional protocol was recommended in the original paper, though it did not quite achieve its intended goal. Things, in fact, are not that simple. Not only should i and j verify that they share a common key, but they also better verify that this key is the one chosen by T for this particular session —a task further complicated by the fact that an enemy may impersonate both T and j in order to fool i.)
   Do such complications arise because xoring with a pseudo-random pad is not a secure encryption algorithm? The answer is no: xoring with a pseudo-random pad is a very secure encryption scheme. But the Needham-Schroeder paradigm derives its attractive simplicity from the assumption that encryption schemes can guarantee both privacy and authentication. When T sends i the two values S and E, it would be desirable both that (a) only i will

understand them, and that (*b*) user i can be guaranteed that it is T who is sending them. Now, encryption schemes are traditionally designed so as to guarantee property (*a*), but not property (*b*). Indeed, it should be realized that the problems just discussed, which are evident in the case of one-time-pad encryption, may become harder to see, but not necessarily disappear, if more complex block-ciphers are used. Property (*b*) can instead be guaranteed —say— by digital signature schemes, but requiring their explicit use would deprive the Needham-Schroeder approach of its attractive simplicity, and would substantially increase its efficiency and the cost of its hardware implementations. (In fact, each user should then be given the additional circuitry necessary for digital signature verification.)

ADVANTAGES OF OUR APPROACH. Even before relaxing the assumption of a trusted agent, our approach offers significant advantages.
First, it is very simple, efficient, and easy to implement. In particular, it does not require that i and j engage in additional protocols for guaranteeing i that he and j share the same secret key.
Second, the security of our scheme does not depend from complicated assumptions (such as the existence of —hopefully efficient— encryption schemes that possess additional and little-understood properties); rather, our scheme solely requires that *ordinary* one-way functions exist.
Finally, for talking in private to an user j, user i needs to access the trusted party only the first time that such a conversation is desired; the responsibility of choosing session keys is the users'; and no undue stress (such as revealing an arbitrary number of cleartext-ciphertext pairs for free) is imposed on the secret encryption keys.

3.2 Our Software-Based, Trusted-Agent Scheme.

To analize this scheme we only consider two parameters: N, the total number of users, and k the length of the common secret keys. (As we shall see, in fact, a big advantage of this approach is that an adversary cannot eavesdrop conversations between two honest users no matter how many other users he may compromise.)

The high-level mechanics of the scheme are as follows. The trusted agent, T, gives to each user i two secret individual keys: an *exchange key* K_i and an *authentication key* K'_i. (Using two keys considerably simplifies the description and the analysis of the scheme.) In addition, upon request, T can quckly compute and make available a *pair key* $\mathcal{P}_{i,j}$ for each pair of users i and j. To send an encrypted message to user j, i uses K_i and the pair key $\mathcal{P}_{i,j}$. To decrypt the resulting ciphertext, j only uses his own individual key K_j. The scheme possesses two important properties:

1. Without knowledge of K_i or K_j, the pair key $\mathcal{P}_{i,j}$ (and any other pair key as well) is useless for deciphering the encrypted communications between i and j; and
2. Learning the individual keys of any number of users does not help in eavesdropping the conversations between any other two users.

The first property guarantees that there is no need to protect the pair keys, something that greatly accounts for the simplicity of our scheme. Indeed, though this would be quite inefficient, all N^2 of them could be made public, in which case there would be no need to call the trusted party.

The second property implies that, differently from the hardware-based approach of Section 2, there is nothing to be gained by preventing a user from knowing his own key. Of course, each user should not loose or divulge his individual key (since the secrecy of his communications depend on it), but the scheme is such that compromising a user's individual key is only useful to understand the communications relative to that user alone. This greatly simplifies the logistic requirements of our scheme, and allows it to be implemented with a very unexpensive apparatus.

PSEUDO-RANDOM FUNCTIONS. Also the present secret-key agreement has a perfectly-secure version, but given that it is both rather inefficient and easily derivable from the computationally secure one, it is only the latter version that we shall describe below.

A critical building block for our computationally-secure version is the notion of a pseudo-random function generator, as defined by Goldreich, Goldwasser, and Micali [5]. Roughly said, such a generator is a easy to compute function $h(\cdot,\cdot)$ mapping pairs of —say— k-bit strings into k-bit strings. When the first argument is fixed to a randomly and secretly chosen value K, then the resulting single-argument pseudo-random function $f_K(x) = h(K,x)$ passesses all polynomial-time statistical tests for functions. That is, any $poly(k)$-time observer who asks for and receives function values at inputs of his choice (but does not know the initial value K) cannot distinguish the case when his questions are consistently answered by means of the specially-constructed pseudo-random function $h(K,\cdot)$, or by means of a function $f(\cdot)$ randomly selected among all possible ones mapping k-bit strings to k-bit strings. We stress that only the random initial value K needs to be kept secret; the program for h is instead assumed to be public knowledge. The authors of [5] also show that (good) pseudo-random *function* generators can be constructed given any (good) pseudo-random *number* generator. (Thus pseudo-random functions in their sense exist if a and only if *ordinary* one-way functions exist.) The cost of one evaluation of such a pseudo-random function f_K equals that of generating $2k^2$ pseudo-random bits (with a k-bit seed), and is thus essentially negligible.

In practice, one might prefer to use a one-way hash function

$$H : \{0,1\}^{2k} \rightarrow \{0,1\}^k$$

as his pseudo-random function generator. In this case, $h(K,x) = H(K|x)$ will be his chosen pseudo-random function, where the symbol "|" denotes concatenation. Such pseudo-random function generators will be even faster to evaluate.

In any case, in describing our scheme with inputs N and k, we assume that a pseudo-random function generator $h : \{0,1\}^k \times \{0,1\}^k \rightarrow \{0,1\}^k$ has been publically agreed-upon. We also assume that the number of possible users is upperbounded by a small polynomial in k.

COMPUTING INDIVIDUAL KEYS AND PAIR KEYS. The trusted agent randomly and secretly selects two k-bit *master secret keys* K and K'. He then privately gives to user i, as his individual exchange key, the value $K_i = h(K, i)$, and as his individual authentication key the value $K_i' = h(K', i)$. Whenever user i asks him for the pair key of two users i and j, the trusted agent computes and sends him the k-bit value value

$$\mathcal{P}_{i,j} = h(K_j, i) \oplus h(K_i, j)$$

together with the k-bit authentication value

$$\mathcal{A}_{i,j} = h(K_i', h(K_j, i)).$$

COMPUTING COMMON SECRET KEYS. The common secret key used by user i to send a private message to user j is $K_{i,j} = h(K_j, i)$. User i computes this key by retrieving the pair key $\mathcal{P}_{i,j}$ from his personal directory (if he has spoken privately to j in the past) or by asking the trusted agent for it (if it is the first time he wishes to speak privately to j). Then i computes $V = \mathcal{P}_{i,j} \oplus h(K_i, j)$ (which is easy for user i because he knows his individual key and the name of the intended recipient) and checks whether $h(K_i', V) = \mathcal{A}_{ij}$. If so, he then randomly selects a session key S and sends to j an encryption of S with key V.

To read the session key sent by i, j simply computes the common secret key $V = h(K_j, i)$, from the sender's name and his own individual key (no table lookup is needed for receiving) and then decrypts i's ciphertext with V so as to obtain S.

(Notice that the common secret key by which i sends private messages to j does not coincide with that by which j sends messages to i, but this does not cause any problems with respect to their ability of communicating with one another. Also notice that while we guarantee i that

ENCRYPTING MESSAGES. While we recommend the use of session keys, recommendation of a particular conventional encryption scheme is beyond the goals of this paper. Whether the common secret key computation occurs within a secure chip or an ordinary personal computer, we insist that it should not become "known" to the user. That is, in the case of a sender i, the key $K_{i,j}$ should reside within i's computer. User i can access only indirectly, by giving his own computing device a message string m that needs to be encrypted with $K_{i,j}$.

EXTENDING OUR SCHEME. In the scheme above, in order to provide a fairer comparison between our approach and that of Needham and Schroeder, we have addressed, with better results, (after all, technical advances should occur after 15 years) essentially the same security concerns as in the summarized Needham-Schroeder scenario. However, it should be realized (and we will explicitly show it in the final paper) that our techniques easily extend to include much tighter security requirements. For instance, we can accomplish that whenever user i asks the trusted party for the pair key with user j, the trusted agent can check that this request indeed comes from i. As for another example, we can accomplish that once both i and j have computed a common secret (session) key, the recipient of a message encrypted with it can be sure that that particular message was indeed sent by i. Details about this will be given in the final paper. We would like to stress from now, however, that like in the simpler setting below, no authenticating step needs to be interactive (like in a challenge-response mechanism), which makes also these secure protocols simple and efficient.

SECURITY. Let us briefly sketch the security of the above scheme in an itemized manner.

- *Unpredictability of individual keys.* Because the k-bit string K is random and secret, and because $h(\cdot,\cdot)$ is a pseudo-random function generator, the individual key of a user i, $h(K,i)$, is unpredictable to an adversary. In fact, $f_K(\cdot) = h(K,\cdot)$ is poly(k)-time undistinguishable from a truly random, secret function f from k-bit strings to k-bit strings. Thus, $f_K(i)$ is polynomial-time unpredictable because $f(i)$ would be totally unpredictable. (This argument actually shows that individual keys are more than unpredictable, they are undistinguishable from random numbers even when their values are actually revealed —provided that the secret master key is still secret. These are important differences, but we shall not discuss them in detail in this abstract.) Further, because $f(i)$ would be totally umpredictable even given the value of f at any other number of inputs, $K_i = f_K(i)$ remains poly(k)-time unpredictable even if all other individual keys become known.
- *Unpredictability of common secret keys.* When user i wishes to talk to user j in private, he does so by means of the common secret key $h(K_j,i)$. User i can compute this key thanks to the special help given him by the trusted agent (i.e., thanks to the particular way in which the trusted agent chooses the pair key $\mathcal{P}_{i,j}$ based on i's individual key). But $K_{i,j}$ is unpredictable to any number of other users, that is, unpredictable given all possible individual and pair keys in the system except the individual keys of i and j. The unpredictability of $h(K_j,i)$ can be proved in two steps. First, $f_{K_j}(\cdot) = h(K_j,\cdot)$ passes all poly(k)-time statistical tests for functions in the sense of [5] whenever K_i is random and secret. This is not sufficient, however, because key K_j, though secret, is not random, but poly(k)-time undistingushable from a truly random number. In fact, $K_j = f_K(j)$ and thus we are not exactly within the hypothesis of [5]. But given that K is random and secret to all users, we can apply a kind of "transitivity property" for poly(k) undistinguishability. Namely, if $h(K_j,i)$ were predictable, being K_j secret and h a pseudo-random function generator, then this would imply the predictability of $K_j = f_K(j)$. But, since K is random and secret, this would contradict the undistinguishability of f_K from a random function. It should be noticed that the this basic argument for the unpredictability of $h(K_j,i)$ keeps on holding when the adversary also knows additional information, such as the individual keys of users z other than i and j. This is so because (a) for a truly random and secretly selected function $f(\cdot)$, the value of f on input i is un-

predictable no matter for how many other inputs z the value of f becomes known, and (b) the undistinguishability (in poly(k)-time) of $h(K_j, \cdot)$ from such an f. This argument can be extended so as to take in consideration the case in which *all* possible pair keys in the system and their authenticating values are also known to the adversary (e.g., because he has eavesdropped all possible requests for pair keys). Full details will be given in the final paper. In addition, the authenticating value $\mathcal{A}_{i,j}$ provided by the trusted agent does not help an adversary who has not compromised the individual keys of i and j to predict $K_{i,j}$. In fact, even if he knew the value of $K_{i,j}$, the authenticating value would undistinguishable from a truly random number to him. Indeed, $\mathcal{A}_{i,j} = h(K_i^!, K_{i,j})$, and the key $K_i^!$ is secret and undistinguishable from a truly random number. Thus, the function $h(K_i^!, \cdot)$ is poly(k)-time undistinguishable from a function $f(\cdot)$ truly randomly and secretly selected among those mapping k-bit strings to k-bit strings. But because for such a function the value $f(K_{i,j})$ cannot possibly betray $K_{i,j}$, the same is true for $h(K_i^!, K_{i,j})$. (To be precise, for any users $a, b, \ldots$ for which i asks and obtains pair keys, the adversary also sees the authenticating values $h(K_i^!, K_{i,a}), h(K_i^!, K_{i,b}), \ldots$. But these additional values are practically useless. In fact, in the case of a truly random and secretly selected function f, the values $f(K_{i,a}), f(K_{i,b}), \ldots$ would be useless because random and independent of $f(K_{i,j})$, and $h(K_i^!, \cdot)$ is poly(k)-time undistinguishable from such a function.)

- *Requesting pair keys.* In requesting the pair key $\mathcal{P}_{i,j}$, a user i does not need to autheticate himself to the trusted agent. Indeed, there is no need for the trusted agent to ensure that the request of $\mathcal{P}_{i,j}$ comes from i. This is so because, as we have already said, even if all the pair keys were to be made public, no group of malicious users can compute the common secret key of two honest users.
- *Authenticating pair keys.* We have already argued that the authenticating values produced by the trusted agent do not betray the common secret keys. We must now argue that they prevent an adversary z, who cuts off the communication line between i and the trusted agent and tries to impersonate the latter, from finding a false pair key $\mathcal{FP}_{i,j}$ and a false authenticating value $\mathcal{FA}_{i,j}$ such that honest user i accepts as valid

$$FK_{i,j} = \mathcal{FP}_{i,j} \oplus h(K_i, j)$$

as his common secret key with j. What is immediate to argue is that z cannot mislead i into accepting a false secret key $FK_{i,j}$ known to z. This is so because $K_i^!$ is unkown and unpredictable to z and h is a pseudo-random function generator; thus, the function $h(K_i^!, \cdot)$ is undistinguishable from a truly-random secret function, and the necessary authenticating value $h(K_i^!, FV)$ is unpredictable to z. We must also argue, however, that z does not have a realistic chance of authenticating a false common secret key that he does not know. The reason for this is that the pseudo-random functions $h(K_i, \cdot)$ and $h(K_i^!, \cdot)$ not only "behave randomly" (because *each* of K_i and $K_i^!$ is random and secret), but also behave as *independent* functions, because the values K_i and K'_i are independent random values. Details will be given in the final paper.

REMARK. Notice that this is not a public-key approach not only because each user does not choose his own keys, but also because no user has a public key. Pair keys not only are associated to pairs of users and are chosen by an external party, but do not need to be made public for the scheme to work. Rather, the scheme remains secure *even if* they become public.

EFFICIENCY AND OTHER CONSIDERATIONS. The scheme above described is most efficient.

For the users, computing common secret keys from pair keys is quite trivial. Moreover, user i needs to ask the trusted agent for pair key $\mathcal{P}_{i,j}$ only when he wants to talk to user j in private for the first time. In fact, $K_{i,j}$ is then stored by i for future use. It is important to notice that if also its associated authenticating value is stored alongside with it, this pair key needs not to be stored in a protected memory. Indeed, it can be stored outside the user's own computer. Indeed, the user needs to keep secret only two k-bit values: K_i and K'_i. Since no particular precaution needs to be applied to the pair keys and authenticating values, the user can easily store them all. Thus, if the link to the trusted party is down, users still can (very much as in the public-key scenario) talk in private with every other user with which they did so in the past.

Moreover, since our scheme does not depend on any interactive authentication protocols, the initial effort of calling up the trusted party is negligible. Indeed, obtaining the necessary two k-bit values from the trusted party can be handled much as we currently handle a call to 411 (information) in the phone system today. In fact, the whole process can be easily automated—the caller dials in his own identity and that of j and then receive two 10-byte values in response.

Our scheme is also most convenient from the trusted agent point of view. Indeed, computing an individual key consists of a single pseudo-random function evaluation, which is trivial todo whether or not one uses a one-way hash function in practice. Also handling a request for a pair key is trivial. In fact, even if decides to securely store only his k-bit master secret key K, the trusted party can satisfy a pair-key request by making 5 pseudo-random function evaluations and one sum modulo 2.

3.3 The Multiple-Trustee Scenario

Also our software-based scheme can be easily adapted for use with multiple trusted agents, only one of whom needs to be honest. For example, if there are two trustees, we can make 2 copies of the preceding scheme (one for each trustee) thus there will be two individual secret keys, two individual secret authentication keys, and two pair keys for each pair of users. There will be only one common secret key for each pair of user, however, set to be the sum modulo 2 of the two common secret keys relative to each trustee.

Once again, not only does this decrease the amount of trust required, but allows our scheme to make strong encryption compatible with law enforcement.

It should also be realized that, though not needed, secure chips can be useful in this approach too. First, storing individual keys in secure chips cannot but be useful. Second, in the multiple-trustee scenario, each of the trustees can inbed his own k-bit master secret key in a number of secure chips, and then gives these chips to the phone company. Thus, the phone companies need not to be trusted with respect to user privacy (or law enforcement) but only to deliver efficiently pair keys on request in a 411-like manner, something that they are set up to do quite well. In fact, once a pair key internally computed by a secure chip is output, it can be handled without other privacy concerns.

Thus, like our first approach, this one too succeeds in simultaneously accomplishing two tasks: (1) making law enforcement compatible with encryption like in the Clipper Chip scenario, and (2) providing the secret-key agreement missing in the Clipper Chip. Moreover, the present approach succeeds in achieving an additional important goal; namely, (3) *being very economical.* Our second approach, in fact, can be totally and securely implemented in software. Indeed, the only operation required from a user consists in summing two numbers modulo 2 —which, rather than in software, can actually be done by hand.

4 Conclusions

In this paper, we have described two simple schemes for key agreement which offer significant advantages in terms of cost and (potentially) security over traditional number-theoretic schemes such as Diffie-Hellman and RSA. The new schemes are also particularly well-suited for use with the emerging Clipper Chip technology proposed by the Clinton Administration, and with Kerborous.

5 Acknowledgment

Many thanks to Mihir Bellare, Joe Kilian, and Phil Rogaway for their helpful comments.

References

1. N. Alon, P. Erdos, and T. Spencer. The Probabilistic Method. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, NY, 1992.
2. R. Blom. An Optimal Class of Symmetric Key Generation Systems. In *Advances in Cryptology: Proceedings of Eurocrypt 84, Lecture Notes in Computer Science*, vol,. 209, Springer-Verlag, Berlin, 1987, pp. 335–338.
3. C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly Secure Key Distribution for Dynamic Conferences. In *Advances in Cryptology: Proceedings of CRYPTO '92, Lecture Notes in Computer Science*, Springer Verlag, 1992.

4. W. Diffie and M.E. Hellman New Direction in Cryptography. In *IEEE Transaction on Information Theory*, vol. 22, no. 6, December 1976, pp. 644–654.
5. O. Goldreich, S. Goldwasser, and S. Micali How To Construct Random Functions. In *J. of the ACM*, vol. 33, no. 4, October 1986, pp. 792-807.
6. S. Micali. Fair Public-Key Cryptosystems. In *Advances in Cryptology: Proceedings of CRYPTO '92, Lecture Notes in Computer Science*, Springer Verlag, 1992.
7. R. M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. In *Comm. ACM*, vol. 21, no. 12, December 1978, pp. 993-999.
8. Y. Zheng. Personal Communication, September 1993.