To appear in *Proceedings of the Fourth Annual Conference on Computer and Communications Security*, ACM, 1997. Earlier version was Technical Report CS95-447, Department of Computer Science and Engineering, University of California at San Diego, October 1995.

Verifiable Partial Key Escrow

Mihir Bellare[*] Shafi Goldwasser[†]

November 1996

Abstract

One of the main objections to existing proposals for key escrow is that the individual's privacy relies on too high a level of trust in the law enforcement agencies. In particular, even if the government is trustworthy today, it may be replaced by an un-trustworthy government tomorrow which could immediately and suddenly recover the secret keys of all users.

"Partial key escrow" was suggested to address this concern, in the context of DES keys. Only some part of a user key is escrowed, so that the authority must make a computational effort to find the rest. We extend this idea and provide schemes to perform partial key escrow in a *verifiable* manner in a public-key encryption setting.

We uncover some subtle issues which must be addressed for any partial key escrow scheme to be secure, the most important of which is the danger of *early recovery*. We show that other proposals for verifiable partial key escrow suffer from the early recovery problem, and thus do not in fact offer an advantage over standard key-escrow schemes. Our verifiable partial key escrow scheme for the Diffie-Hellman cryptosystem does not suffer from early recovery.

Political debate will not make the user versus law-enforcement conflict on privacy vanish. Today we are seeing corporations, pushed by their business needs, ready to accept some form of key escrow. The realistic and urgent question is to find the form which guarantees the most privacy. Our schemes are candidates.

[*]Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: `mihir@cs.ucsd.edu`.

[†] MIT Laboratory of Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-mail: `shafi@theory.lcs.mit.edu`.

Contents

1 Introduction

Over the past few years there has been much debate about how to balance the needs of individuals with the needs of law enforcement in the domain of private communication. On the one hand, means for private communication are essential to enable electronic business and commerce. On the other hand widespread use of strong encryption means that in case of suspected criminal activity, the government's wiretapping abilities are curtailed.

1.1 Solutions that have been considered

Several solutions have been proposed. One is "weak cryptography", where the government forces users to use a cryptosystem which the government can break (for example by allowing only small key sizes). Naturally, this is not popular.

Then there are various forms of "key escrow." The simplest is "full key escrow", where the user is expected to provide his entire secret key to the government. This too is obviously not popular.

More serious consideration is being given to "split key escrow" [27, 41]. Here the user's private key is split into n pieces, and the pieces are "escrowed" with separate trusted agents. The pieces obey the secret sharing property [9, 37]– any k of them can be used to re-construct the original private key, but no t pieces yield any information about it. (The parameters k, t and n can be chosen as desired. In the most popular implementations $k = t + 1$.) It is expected that the trustees will release their pieces only upon presentation of an appropriate court order. In Micali's "fair public key cryptosystems" [27], the user can himself choose his private key —or choose it in conjunction with the government as in [24]— but the government trustees can verify that they indeed received proper pieces of the escrowed keys. In the Clipper chip approach, which applies to private key cryptography, a trusted agency chooses the key and puts it in tamper-proof hardware [41].

Several objections, both on the social and on the technical fronts, have been raised to these approaches. Also many enhancements and variations, and some alternatives, have been considered. We refer the reader in particular to [15, 16, 10, 8, 17, 24, 22, 26, 30, 42, 5]. The summary from all this work is that no existing proposal seem to be acceptable to all parties involved, and new technical solutions are sought after.

1.2 The climate today

Political debate will not make the user versus law-enforcement conflict vanish. Even though some would prefer to not have any form of key escrow, the pragmatic view is that reaching some sort of compromise is necessary.

As the loss to business from this issue increases, corporations are becoming more and more open to the idea of accepting *some* form of key escrow. The bottleneck today is in the lack of a solution that corporations and individuals find protective enough of their privacy.

Thus the technical question is to find a solution as palatable as possible to both sides. It should guarantee enough privacy for the individual that people would go along with it, and yet be acceptable to law enforcement too.

The question is an urgent one. If corporations deem key escrow necessary for business, it will happen, and quickly. (Indeed, it is already happening). If we do nothing, we are in danger of seeing a form of escrow emerge which guarantees less privacy to the individual than we would like or is possible. We should have the goal of finding a solution which guarantees the maximum possible privacy.

This work provides a new possibility, called "verifiable partial key escrow", which addresses some of the main concerns that people have been raising. It applies to any form of key escrow, in particular to both full and split, although our emphasis will be on the latter.

1.3 The concern

The major concern voiced by opponents of key escrow is that the individual's privacy relies on trusting the government and law enforcement agencies to follow the rules. In particular, Shamir [38] raises the concern that even if the government is trustworthy today it may be replaced by an un-trustworthy government in the future. This government may, for reasons of its own, suddenly start un-escrowing keys on a vast scale. (Under full key escrow it can do this directly. Under split key escrow, the concern is that it could obtain court orders in bulk, or even directly control a large enough subset of the trustees.) Now, suddenly, it can wiretap everyone's conversations.

In other words, tomorrow's government may not feel bound by the contracts made by today's government. A society which today obligingly escrows its keys may tomorrow see a political swing as a result of which its collective privacy is compromised.

The obvious fix to this problem is to choose new keys when the new government comes into place. This unfortunately is not workable, as once a system and the rules governing it are in place it would be complex and time consuming to replace it, especially as the new government would have no interest in changing the keys.

1.4 Partial key escrow

THE IDEA. Shamir's idea [38] to mitigate the effects of such a sudden political change is "partial escrow." His scheme is in a context where the private key is a 56-bit DES key S. The user would not escrow his entire private key S. He would escrow only the first 8 bits of this key. Let us call this escrowed part x. Now, even if the trustees have x, they still need to work for 2^{48} time to find S. (They have to do an exhaustive search for the remaining 48 bits of S). Of course this is not much of a deterrent in finding a *particular* key, since 2^{48} steps is not infeasible. But the idea is that it becomes hard to suddenly and simultaneously uncover a *lot* of keys— a large number of 2^{48} time computations can add up quickly.

GENERALIZATION. Although Shamir's idea pertains only to DES keys, it can be generalized as follows. If S is not a DES key, the idea is to escrow some information, call it x, with the property that having x it is possible to recover S, but not immediately— one has to work some amount of time, say 2^{48} steps. (How exactly to do this if S is not a DES key may not be clear. For example, if S is the factorization of some public number, how can we specify x with this property? This is one of the problems we will have to deal with. However, for simplicity, just think for now that any "key" can be "broken" into a "large" component x and a "small" component a such that finding a takes 2^{48} steps.)

ON POSSIBLY WIDER APPLICABILITY. Although Shamir's original suggestion pertains only to the individual versus government issue, we suggest there are other applications. Key escrow may be useful in the confines of a single, large organization, which has very much the same kinds of concerns as a government. Employees want privacy, but to protect business interests, the top managers would like some balance enabling them to recover the key of an employee they suspect of anti-organizational actions. Key escrow seems a natural solution, with the trustees now being some collection of vice presidents, and the users being employees. The possibility of a hostile takeover, or sudden change in policy, in a corporation, is, perhaps more likely than a sudden change in government, at least in democratic countries, and partial key escrow may be quite attractive in this setting.

1.5 New settings and issues

We take Shamir's idea further in several directions. First, we raise a number of issues, some of them subtle, whose correct technical resolution we argue is crucial to having an effective partial key escrow system, and examine proposals for partial key-escrow in this light. Second, we provide concrete partial key escrow schemes for popular existing cryptosystems.

THE SETTING. For simplicity we focus on split key escrow for public key cryptography. (But our ideas do apply also to full key escrow and private key cryptography.) The user will be choosing public key P and matching private key S.[1] We now discuss two issues, verifiability, and early versus delayed recovery. The discussion here is high level; for a full understanding and examples, we direct the reader to the body of the paper.

VERIFIABILITY. The partial escrow scheme will specify how an honest user is to compute, and escrow, some information x knowing which one can recover the user secret key S in 2^{40} time (for the purpose of this abstract we will speak of the concrete number 2^{40} steps, although clearly this number will change as more computing power is available and should be treated as a parameter which stands for what is considered to be a feasible and yet non-negligible amount of time). But there is the concern that the user will cheat, and escrow garbage, or some string x' given which it takes, say, 2^{100} steps, rather than 2^{40} steps, to recover S. Thus we propose to make the process *verifiable.* In our schemes, it will be possible for the trustees to check that the user really chose x correctly, and that they have received shares of this x, so that they are ensured that when they recover x they can get S in at most 2^{40} steps.

We recall that the issue of verifiability for secret sharing was raised by Chor, Goldwasser, Micali and Awerbuch [12], who introduced the notion of VSS (Verifiable Secret Sharing). Verifiability was addressed for standard key escrow by Micali [27]. Now there is now a new element– the partiality of the escrow must be verified too. Namely, it should be possible to check that the secret key can be reconstructed in the claimed amount of time, if the trustees reveal their pieces of the public key.

We remark that the issue of verifiability of partial key escrow for public key cryptosystems was raised, concurrently and independently, by Micali [28].

EARLY VERSUS DELAYED RECOVERY. The main technical difficulty in implementing verifiable partial key escrow is in controlling *when* can the government do the 2^{40} extra computation steps. We note a subtle point here. In some schemes (the obvious ones which come to mind and some proposed solutions) it is possible to perform this work as soon as the user's public key is made available, and, in particular, without knowledge of the escrowed component x. That is, the extra work, thought it exists, can be done, in advance; then when the court order arrives to un-escrow x, the key S is *straightaway* available. We call this *early recovery.* We suggest that although schemes having early recovery do provide some kind of deterrent, they are not much better than standard (not providing partiality) escrow schemes. What we need are schemes which have *delayed recovery.* Here the user is guaranteed that even after the trustees un-escrow x, they must work 2^{40} time to recover S– they cannot "jump-start" the process in any way.

Early recovery is best understood via a concrete example. We refer the reader to Section 2 for some examples which explain the problem, and also for a fuller discussion of why we believe it should be avoided. We stress that schemes having early recovery are much easier to design. But

[1] For simplicity we ignore for the moment the issues of subliminal channels that this raises, since this is easily fixed by having the keys chosen by the user and government together [24].

they should be avoided, because the promised extra deterrent to the government is in effect not present. Thus, in some ways, it is worse than standard escrow, because the individual is promised some extra security which does not exist. Because of this, if a government or corporation proposes to adopt a scheme suffering from early recovery, it is likely to meet with a negative public reaction.

1.6 New schemes

VPKE FOR THE DH CRYPTOSYSTEM. The Diffie-Hellman cryptosystem [18] is one of the most popular and convenient ones, and in particular lends itself quite naturally to standard (verifiable) split key escrow [27]. Thus it is a natural starting point to implement the idea of partiality.

We provide a verifiable, partial key escrow (VPKE) scheme for the Diffie-Hellman cryptosystem. Our scheme, presented in Section 3, has delayed, not early, recovery. Simultaneously achieving verifiability and delayed recovery, while maintaining efficiency, turned out to be more challenging than it might look at first, and required new protocol development.

VPKE FOR FACTORING BASED CRYPTOSYSTEMS. There are many suggestions for public-key encryption schemes based on the difficulty of factoring integers, e.g [36, 11, 7]. It is thus desirable to implement an efficient verifiable partial key escrow protocol, for any factoring based cryptosystem. We do this. Our scheme does not achieve delayed recovery, but has several nice extra features. One is that all communication between the user and the government trustees during the escrow protocol can take place over the public channel, with no need for encryption. This may be good from the export control point of view. Another feature is that there is no sharp threshold of recovery, but rather a gradual release. Very few trustees cannot get the secret key, and lots of them can; in between, as more and more trustees get together, the work to get the key decreases as the number of trustees involved grows. These simple schemes, however, suffer from early recovery. See Section 4.

A NOTE ON EFFICIENCY. The most important element in a practical proposal for verifiable partial key escrow is what is the underlying cryptosystem. Of course the escrow and verification protocols must be as efficient as possible, but the cryptosystem will be used repeatedly, while the verification is only at set-up time and is done once. This is why a highly desired goal is to build schemes in which the underlying cryptosystems are standard ones. We achieve this for Diffie-Hellman. (For RSA we do not because we require the modulus to have many prime factors, making it very big). In addition, we stress that our protocols, while not blindingly fast, are quite efficient.

1.7 Other proposals

Independently, Micali [28] provides a partial escrow scheme for the DH system, and a factoring based partial key escrow scheme. He calls them "guaranteed partial key escrow" (GPKE) schemes. We present early recovery attacks on both his schemes– see Section 2.2 and Section 4.4.

1.8 Related work

After Shamir suggested partial escrow of DES keys [38] several researchers have been inspired to work on it. Our work presented here first appeared as a technical report [3]. As mentioned above, concurrently and independently of this work, Micali has raised the issue of verifiability of partial key escrow for public key cryptosystems, and has proposed two schemes for what he calls "guaranteed partial key escrow," one for the Diffie-Hellman system, and one for factoring based systems [28]. (However he has not considered the issue of early recovery). Subsequently, [38] and [28] were merged into a joint paper [29].

Recently we have developed a different approach, called "encapsulated key escrow" (EKE) [4]. It achieves the same kind of "time delayed recovery" that partial key escrow aims to achieve, meaning that after recovery of the escrowed information there is still a computational effort required to find the user secret key. But the method (EKE) is quite different. EKE yields the first time delayed key escrow scheme for RSA which has delayed recovery, and more efficient solutions for DH than we have here. This work also presents schemes for time delayed escrow of session keys.

Lotus Corporation is using a form of partial key escrow for session keys. They call their approach "differential workfactor cryptography." (It has been described, for example, in [32].) It is essentially the Shamir idea. They will "escrow" 16 bits of the 56 bit DES key S, leaving a 2^{40} workfactor to find the remaining 40 bits. The escrow here is done by having the sender simply encrypt the 16 bit string x under a government public key. This ciphertext C_T is sent to the receiver. (Together with an encryption of S under the receiver public key). When the government wiretaps, it picks up the ciphertext C_T and decrypts it to obtain the 16 bit string x. Now it has to find the other 40 bits of the session key. Note there is no verifiability here.

This is a different problem from the one we look at in the sense that our main concern was escrow of public keys, not session keys. But we can use similar ideas to get solutions for this case as well.

The idea of using processing time to "slow down" some action of an adversary appears also in a work of Dwork and Naor [19] where they propose schemes to combat "junk mail."

2 Issues and attacks

The best way to introduce the issues is by example. Let us recall what is the DH system, and what is the standard fair DH system of [27]. Then we illustrate early recovery by providing an early recovery attack on an existing scheme. In the next section we will present our solution for DH.

2.1 Background

THE DH SYSTEM. Recall that in the Diffie-Hellman cryptosystem-system, a user A will have public key g^{S_A} and secret key S_A, where g is a generator of the group Z_p^* for some large prime p. Secure communication between two users is enabled via the Diffie-Hellman property— another user B, having public key g^{S_B} and private key S_B, shares implicitly with A the Diffie-Hellman key $g^{S_A S_B} \bmod p$, and can use this shared key to send messages privately, or to derive a session key to this end, as desired. This properties make the system very convenient to use, so that it is amongst the foremost choices of public key systems.

THE FAIR DH SYSTEM. In Micali's fair Diffie-Hellman cryptosystem-system [27], the user, having published g^S, escrows S by simply sharing it via a simple discrete log based verifiable secret sharing (VSS) scheme. (This VSS scheme is based on ideas of Feldman and Pedersen [21, 33]. The VSS schemes of [27, 21, 33] are attractive because they are efficient and non-interactive.) The verifiability is in the fact that the trustees are able to check that they really have pieces of S, meaning that if later they try to recover the shared secret, they will really get the secret key S of the user, and not some garbage. (A feature of this VSS we stress is that it requires that g raised to the power the secret, here S, be published. We will see the relevance later.) Of course, there is yet no partiality in the key escrow; that is the problem we propose to address.

2.2 Early recovery for DH GPKE

GPKE FOR DH. Micali's suggestion for a "guaranteed partial key escrow" (GPKE) scheme for DH is like this. (This scheme is from [28]. The same scheme appears in [29]). The public key has the form $P = g^{x+a}$ where x is long but a is only, say, 80 bits. Now one can escrow x as before, via the VSS of [27, 21, 33]. (This requires providing the trustees with g^x. It is important to note that this means that $g^a = P/g^x$ also automatically becomes known.) Then, the user provides a ZK proof that a is indeed only 80 bits long. (The ZK proofs are cleverly done by working over an appropriate subgroup of Z_p^*. But these details will not concern us.) By Shank's baby-step giant-step method, 2^{40} computation steps are enough to get a given g^a, and no faster algorithm is known. Thus, the suggestion is that partiality is achieved because it takes 2^{40} steps to recover a from g^a.

A WEAKNESS OF THE ABOVE SCHEME. As mentioned above, the details of the ZK proof in a scheme such as the above do not concern us. The weakness we pin-point is based only on the fact that g^a is made available to the trustees right from the beginning. We suggest that this defeats the purpose of partial key escrow. The reason is that a trustee can work, for time 2^{40}, and recover a, at *any* time, and in particular *before the trustees receive a court order enabling them to recover* x. Then, when the trustees do get the court order to recover x, they *straightaway* have the *full* secret key $x + a$. That is, after the court order is issued, there is no extra work to be done at all. We call this the problem of early recovery.

In contrast, our view of partial key escrow is that even after the trustees know x, some work must remain to recover the secret key. That is, we suggest one should have delayed recovery, and that this is crucial. Indeed, partial key escrow with early recovery is not much better than standard key escrow, and in some ways worse, because there is a "promise" of extra security to the individual which is in fact not upheld.

2.3 The early recovery issue

Let us again emphasize the issue of timing. Certainly, a scheme such as the above provides an extra (ie. over and above normal escrow) deterrent to the trustees: they have to work an extra 2^{40} steps for each key they want to recover. The issue is *when* they can do this. In the above scheme, any trustee can do it as soon as the verification protocol ends and the public key is published, and in particular without the court order enabling recovery of x.

One might suggest that the timing is not important since, to recover, say, hundreds of millions of keys could still take a while. But nobody wants to recover hundreds of millions of keys. A bad government is likely to have in mind some set of "important" users, say 10,000 of them, whose keys it wants. Before it comes to power, it can, via a single compromised trustee, compute the a values for these users, and store them. Upon coming to power, court orders to uncover the x values for these users are issued at once, and now, by tapping into the stored a values, the full secret keys of these users come at once into the bad government's hands.

One might say that a trustee will not compute a beforehand, since, after all, he is trusted. But *all* trustees are not trusted— that is after all the entire point of split key escrow. Thus it is reasonable to assume one trustee is in cahoots with the opposition and will pre-compute a table of users and their a values.

SURMOUNTING EARLY RECOVERY. One obvious suggestion is to periodically update the public file; the user would pick a new 80 bit value a', publish $g^{a'}$, update his secret key to $S' = x + a'$ and his public key to $g^{S'}$. But the updating would have to be quite frequent, putting a huge burden on the key management center, and defeating the idea of public key cryptography. So this does not sound like a reasonable solution.

Thus we wish to design a verifiable, partial key escrow scheme, for the DH system, which "truly" avoids early recovery. It should be "direct' and as efficient as possible.

A little thought shows that this will not be too easy. Let us see what kinds of technical problems arise. Let us take the same starting point – the public key is $P = g^{x+a}$ with x large and a small. The key principle is this: it should take 2^{40} steps to recover a after x is revealed, but, before that, recovering a should be intractable.

We want to escrow x. But we run straightaway into a problem. The VSS schemes of [27, 21, 33] require that we publish (ie. make available to the trustees, who need to do the verification) the quantity g^x. But then a trustee can recover $g^a = P/g^x$, and early recovery is possible. We conclude that we cannot give g^x to any trustee, let alone g^a. However, this is quite a constraint. Not just for using VSS. Note that even if we could somehow escrow x without revealing g^x (and hence g^a), we still have to show that a is small. A natural "cut and choose" ZK proof would work by publishing g^a, and then having some challenge response protocol. But we cannot use this.

The problem we face is akin to that of "dictionary attacks." What we would like to do is have a scheme in which it is not possible for the trustees to verify a "guess" as to the value of a.

In Section 3 we show how we can get around these problems and get a verifiable, partial key escrow scheme with delayed (as opposed to early) recovery, for the Diffie-Hellman cryptosystem-system.

2.4 Discussion and other issues

SMALL EXPONENT ATTACK ON GPKE. We note another attack which applies to the particular choice of parameters made in Micali's GPKE scheme discussed above. Namely, the assumption was that the baby-step giant-step was the best algorithm to solve the discrete logarithm problem with short exponents. However, Wiener and Van Oorschot [40] have recently presented much better algorithms, and these will quickly find the exponent a above in much less than 2^{40} steps. However, one can compensate by increasing the size of a. Hence we don't consider this a major attack.

Note that the small exponent attacks of [40] work over Z_p^*, which is the group used in [28]. One way to avoid these attacks is to work in groups of prime order. (The order of Z_p^* is $p-1$ which is not prime.) However, this can't be done for the GPKE of [28] because the ZK proofs require that the scheme is over Z_p^* with $p-1 = mq$ for prime q and some integer m. In contrast, we work over groups of prime order, and thus avoid the small exponent attacks.

SUBLIMINAL CHANNELS AND OTHER SUCH ISSUES. Kilian and Leighton [24] pointed out that if the user in a key-escrow system chooses his secret key himself, then the public key may present a subliminal channel via which the user may communicate with another user. They suggest how this could be fixed. Since their solution is quite general it will apply here too, and so for technical simplicity we will not concern ourselves with this issue in what follows.

A NOTE ON TIMING. Throughout this paper, we are concerned with specifying problems that take a certain amount of time τ to solve. It should be possible to solve them in this much time, but not in less. We want parameterized problems, in which we can set $\tau = 2^l$ for a security parameter l which thereby controls the "degree of partiality."

An obvious issue here is that CPU speeds vary widely, so what are we measuring? We will not be too formal, but roughly we try to count the number of "basic steps," what these being depending on the problem. For example, in many cases, we view modular multiplication as a basic step.

A more subtle issue, raised by Rivest, Shamir and Wagner in [35], is parallelism. If a problem takes time τ to solve, it may be possible to solve it in $\tau/2$ time with two processors, etc. Here we view parallelism as a resource too, and don't differentiate between processors and time. Thus when we say a problem takes time τ, we don't rule out that this could be solved by p processors each taking time τ/p. Our assumption is that in estimating the resources of the trustees, we must also estimate the number of processors they have available, and raise our security parameter accordingly. For suggestions for problems in which parallelism won't yield speedups, see [35].

3 VPKE for the DH System

3.1 Overview and preliminaries

MAIN IDEAS. As above, we will let the public key have the form $P = g^{x+a}$ where x is large and a is small. The user keeps the secret key x, a. Now we need to escrow x and also verify the structure of the key P. We recall we can publish neither g^x nor g^a. Instead, the user *commits* to them *information-theoretically.* (The trustees get no information about x, a, even if they are computationally unbounded, while the user is committed as long as he cannot compute discrete logarithms.) We can no longer use the VSS schemes of [21, 33] to escrow x, but, luckily, Pedersen, in a later paper [34], had a (different) scheme which achieved information theoretic secure VSS, and is just what we need. (See Appendix B). Finally, we prove that a is small based on its commitment. (The protocol we use, due to [13, 14], is perfectly witness indistinguishable, in the sense of [1, 20], which is all we need. See Appendix A). All this is done quite efficiently.

We let l be the parameter such that 2^l is the desired delay time in the partial recovery. (A suggested range of choices is 40 to 48). Also recall that n is the total number of trustees of which at most t may be bad. Example choices are $n = 3$ and $t = 2$ or $n = 5$ and $t = 3$.

If a warrant is issued, any subset of $t+1$ trustees can recover x by running the recovery procedure of the VSS. From x they can compute g^x, and hence $g^a = P/g^x$. We choose a to be $2l$ bits long, so that a can now be recovered in about 2^l steps via Shanks baby-step giant-step method.

In order to implement the information theoretic secure bit commitments, we will be working over a group G of prime order q. To be concrete, we take G to be a order q sub-group of Z_p^*, where $p = 2q + 1$ is prime, so that operations are in fact in Z_p^*.

The correctness of the scheme relies on two assumptions. The first is the usual assumption on the intractability of the discrete logarithm problem in large enough groups. The second is that given g^a, the best algorithm to recover a takes about $|a|/2$ steps. (Note that the baby-step giant-step algorithm, which achieves a in about $|a|/2$ steps, is the best known algorithm in groups of prime order. The small exponent attacks of [40], which do better in other groups, don't apply here.)

INFORMATION THEORETIC COMMITMENT BASED ON DISCRETE LOG. Let G be the group of prime order q. We let g be a generator of the group G, and h an element of $G - \{1\}$ such that $\log_g(h)$ is unknown to the committer. (Note h is also a generator of G.) The committer can commit to any $z \in Z_q$. He does so by picking $v \in Z_q$ at random and letting the commitment be $Z = g^z h^v$. This value is uniformly distributed over G for each fixed value of z, so that the receiver gets no information about z from the committal. (This commitment scheme is used in [34] and extends schemes used in several other places.)

To decommit a value Z, the committer must provide z, v such that $Z = g^z h^v$. To cheat he must be able to find values $z_1 \neq z_2$ and values v_1, v_2, Z such that $Z = g^{z_1} h^{v_1} = g^{z_2} h^{v_2}$. But the committer cannot cheat in this manner if he doesn't know $\log_g(h)$. For completeness let us see why. Indeed, it is easy to see that the above implies $g^{z_1 - z_2} = h^{v_2 - v_1}$. But since q is prime, the element $v_2 - v_1$ (it is non-zero because $z_1 \neq z_2$ implies $v_1 \neq v_2$) has an inverse β in Z_q, and raising both sides to the power β we get $\log_g(h) = \beta(z_1 - z_2)$. (We stress that this argument requires that the order of the group is prime, which is why we chose to work in a group of prime order.)

System wide constants—

n — Number of trustees (say $n = 5$)
t — Assumed bound on number of bad trustees (say $t = 3$)
$p = 2q + 1$ — Prime of length at least 512
$G \subseteq Z_p^*$ — Group of prime order q
$g \in G$ — Random generator of G
$h \in G$ — Random generator of G with $\log_g(h)$ unknown

User keys and their components—

$S = x + a \bmod q$ — Private key of User
$P = g^{x+a} \in G$ — Public key of User
$a = a_0 2^0 + a_1 2^1 + \cdots + a_{2l-1} 2^{2l-1}$ $(a_i \in \{0, 1\})$ — Short ($2l$ bits) component of S
$x \in Z_q$ — Long component of S, escrowed with trustees

Protocol—

1. Commitments: The user chooses $u \in Z_q$ and $u_0, \ldots, u_{2l-1} \in Z_q$ at random. He lets $X = g^x h^u$ and $A_i = g^{a_i} h^{u_i}$ for $i = 0, \ldots, 2l - 1$. He sends $X, A_0, \ldots, A_{2l-1}$ to the trustees.
2. Proof that commitments define the key: He also lets $w = u + u_0 2^0 + u_1 2^1 + \ldots + u_{2l-1} 2^{2l-1} \bmod q$ and sends w to the trustees. Each trustee checks that $P \cdot h^w = X \cdot A_0^{2^0} \cdot A_1^{2^1} \cdots A_{2l-1}^{2^{2l-1}}$.
3. Escrow of large component: The user shares x via Pedersen's information theoretically secure VSS [34], based on the initial commitment X. (See Appendix B).
4. Proof of smallness of small component: The user proves that each A_i is a committal to either 0 or 1, as follows. For each trustee T and each $i = 0, \ldots, 2l - 1$, the user (playing the role of prover) and T (playing the role of verifier) execute protocol KOD of Figure 2 on common input $A = A_i$. (See Appendix A).

Figure 1: The VPKE Protocol for DH.

For any user not knowing $\log_g(h)$, we will, informally, use the notation $D(Z)$ to refer to the "de-committal that the user knows"; namely the value z for which the user also knows v such that $Z = h^z g^v$. We stress that this "notation" is not formally well defined since for any z there exists a v such that $Z = h^z g^v$, but intuitively it is the z that the user "knows", and it is useful to be able to talk about it.

As stated, this scheme can be used to commit to any value $z \in Z_q$. Part of our protocol will use it only to commit to bits $z \in Z_2$. In such a case, we will have to verify that the value committed to is a bit, and not something bigger.

What follows. In Section 3.2 will describe the protocol, and accompany the description with motivations and informal discussions of correctness. In Appendix A we then look closely at the proofs of knowledge component.

3.2 Description of the protocol

System wide constants. Common to all users are parameters p, g, h. For concreteness, let the number p is a prime of length at least 512 bits. We choose $p = 2q + 1$ where q is also prime. We can identify, by standard means, a sub-group G of Z_p^* of prime order q, and work over G. (In particular, membership in G is testable, and operations in the following are over G.) Since G has prime order it is cyclic, and moreover any non-trivial member of G is a generator of G. We let $g \in G$ be a randomly chosen generator of G. Finally, h is a random member of $G - \{1\}$ (hence in particular also a generator of G) such that nobody knows $\log_g(h)$, the discrete logarithm of h with respect to g.

Notice that $p - 1$ has a large prime factor, namely q, so the discrete logarithm problem in Z_p^* is hard, as is the discrete log problem in G.

These system wide constants can be chosen a priori by some center and published, as follows. The center chooses p, g with the required properties, and then computes h via hashing of all these quantities under some suitable public one-way hash function such as SHA.[2] We note that we do not have to trust the center: all necessary properties of the numbers can be checked by any user. (The primality of p and q can be tested. To check that g is a generator it is enough to check that $g \in G$ and then, since G has prime order, to check that $g \neq 1$. And finally test that h is indeed appropriately derived from p, g via hashing.) Also note the center is not active; these choices made, it can disappear.

Choice of keys. The user chooses a random number $x \in Z_q$. He also chooses $2l$ random bits $a_0, a_1, \ldots, a_{2l-1} \in Z_2$, and lets $a = a_0 2^0 + a_1 2^1 + \cdots + a_{2l-1} 2^{2l-1}$. He now sets $S = x + a \bmod q$. This will be his private key. The public key is $P = g^S$. (Here x is the large component of the secret key, and will be escrowed, while a is a random $2l$ bit number which is the small component of the secret key, and will be hidden information-theoretically from the trustees until they recover x, at which point a will be recoverable in 2^l steps but not less.)

The protocol must verifiably escrow x and also convince the trustees that a can be recovered in about 2^l steps if x is known, all this without revealing any polynomial time predicate on a. The steps of the protocol we now discuss are summarized in Figure 1.

Commitments. The user begins by committing to both components of the keys using the information theoretically secure commitment scheme. The large part x is committed to as a block– namely the user chooses $u \in Z_q$ and random and sets $X = g^x h^u$. The small part is committed to bit by bit– namely for each $i = 0, \ldots, 2l - 1$, the user picks $u_i \in Z_q$ at random and sets $A_i = g^{a_i} h^{u_i}$. The values $X, A_0, \ldots, A_{2l-1}$ are broadcast to the trustees.

Proof that commitments define the key. Next the user proves that the committed values do indeed define his secret key $x + a$. To understand this step, we note that if the user is honest then it must be that $x = D(X)$ and $a_i = D(A_i)$, $i = 0, \ldots, 2l - 1$. Thus it must be that

$$\begin{aligned} P &= g^{x + a_0 2^0 + a_1 2^1 + \cdots + a_{2l-1} 2^{2l-1}} \\ &= (X/h^u) \cdot \prod_{i=0}^{2l-1} (A_i/h^{u_i})^{2^i} . \end{aligned}$$

So we have the user provide $w = u + u_0 2^0 + u_1 2^1 + \ldots + u_{2l-1} 2^{2l-1}$ to each trustee. Each trustee now checks that $X \cdot A_0^{2^0} \cdot A_1^{2^1} \cdots A_{2l-1}^{2^{2l-1}} = P \cdot h^w$. (It still remains to convince them that a is small.)

Escrow of large component. The large component x is verifiably secret shared using Pedersen's information theoretic secure scheme [34]. The starting value is the commitment X to x.

Since this VSS scheme is a central component in our protocol, we have described it in Appendix B.

[2] Here are some details. The center will first appropriately extend SHA to yield a function H with a 512 bit output. Then it will successively try the values $H(p, g, 1), H(p, g, 2), \ldots$ until it finds one which is in $G - \{1\}$. This is our h. The value of j for which $H(p, g, j) = h$ will also be published.

An important feature of the scheme is that g^x is not revealed. We also note that this scheme is as efficient as those of [21, 33] and in particular is also non-interactive.

PROOF OF SMALLNESS OF SMALL COMPONENT. The last step of the protocol is to convince the trustees that a is only $2l$ bits long. For this, it suffices to convince the trustees that each A_i is a committal to a bit, either 0 or 1. (Our concern is that the user would cheat and make A_i of the form $g^z h^v$ for some large $z \in Z_q$.) This last task is accomplished by a proof of knowledge described in Appendix A. Figure 2, shown there, represents the proof of knowledge protocol. This protocol (called KOD, for "knowledge of decommittal") will be run with the user in the role of prover and a trustee in the role of verifier. This protocol is not zero-knowledge, but is "perfectly witness indistinguishable," a property which suffices for the correctness of the overall VPKE protocol.

KEY CERTIFICATION. Finally, if all checks pass, the trustees each individually sign the public key P, and it is placed in the public file. This key can then be used for public key cryptography by using the Diffie-Hellman system in the usual way.

RECOVERY. Now, suppose a warrant issued to $t+1$ trustees to recover the user key. The trustees need to recover S. They first pool their shares to reconstruct x. This is possible by the property of the VSS. Now note that $g^a = g^{S-x} = P/g^x$, and since the trustees have x, and of course P, they recover g^a. But a is $2l$ bits long. By Shank's baby-step giant-step method, a can be recovered in 2^l steps. (This is the best known time in groups of prime order. The small-exponent methods of [40] don't apply here.)

3.3 Properties of the scheme

CORRECTNESS. Let us try to give a very informal idea of why the protocol is a verifiable, partial key escrow. First, consider X. The properties of VSS guarantee that after the escrow based on X, the trustees are convinced that there is a unique, well defined de-committal which they can later recover. We can denote it $D(X)$. Now, consider $A_0, \ldots, A_{2l-1}$. For each $i = 1, \ldots, 2l-1$, the proof of knowledge of the underlying de-committal of A_i guarantees that the user, if he knows any de-committal of A_i, knows one which is a bit, not some large string. We can denote this bit by $D(A_i)$.

The fact that X and $A_0, \ldots, A_{2l-1}$ are really committals means that the user knows only one way to de-commit them, given that he doesn't know $\log_g(h)$. This in turn means that the proof that the commitments define the key can be believed— see the above calculations. So now the trustees know that the key is $D(X) + D(A_0)2^0 + D(A_1)2^1 + \cdots + D(A_{2l-1})2^{2l-1}$. Finally, the proofs of knowledge have convinced the trustees that each A_i hides only a bit so they know a is only $2l$ bits long.

Why is early recovery of a impossible? Because a is information theoretically hidden.[3] More precisely, for any of the 2^{2l} values of a, the view of any collection of t trustees would be the same. So they have no idea what a could be. (Of course if $t+1$ trustees get together they can recover x, and hence know g^a.)

EFFICIENCY. We make rough evaluations of the computational cost of executing the protocol, for the user and each trustee.

Take the user first. Step 1 has $2l+1$ commitments, meaning $4l+2$ exponentiations. The dominant cost in Step 3 is t exponentiations, and n public key encryptions (see Appendix B). Step 4

[3] For the purpose of this analysis we are assuming that, in the VSS, perfectly private channels are available between the players. In practice we must use encryption in the VSS. But the security of the encryption scheme then says that the loss of security relative to the ideal case is negligible. See Appendix B for more information.

involves three exponentiations per execution of KOD (see Appendix A) for a total of $3 * 2ln = 6ln$ exponentiations. Let's say we choose an efficient encryption scheme, like RSA with small exponent (3). We can then neglect the cost of the encryptions. The main cost is then roughly $6ln + 4l + t + 2$ exponentiations. If, say, $n = 5$ and $t = 3$ and $l = 48$ this works out to 1637 exponentiations. This is not cheap, but remember these protocols are not executed often. It can be tolerable for the job at hand.

Now take a trustee. Step 2 can be done in one exponentiation and at most $2l^2 + 3l + 1$ multiplications. For $l = 48$ and assuming arithmatic is modulo a 1024 bit prime, this can be estimated as less than the cost of 5 exponentiations, so lets say a total of 6 exponentiations. The dominant cost in Step 3 is $t + 3$ exponentiations to verify the VSS and one decryption (see Appendix B). Let's say the latter is one exponentiation, so the total is $t+4$ exponentiations. Step 4 involves four exponentiations per execution of KOD (see Appendix A) for a total of $4 * 2l = 8l$ exponentiations. Thus the main cost is roughly $8l + t + 10$ exponentiations. If, say, $n = 5$ and $t = 3$ and $l = 48$ this works out to 397 exponentiations. Note this is much less than the user cost.

In addition, note there will be some broadcasts. Broadcast is used both in our protocol and the underlying VSS.

Efficiency improvements. The whole protocol can be made non-interactive. (That is, the user sends a single message, on receipt of which the trustees decide whether or not to certify the key.) The only part that currently is interactive is the proof of knowledge. The interaction can be eliminated by specifying the challenges via a hash of the committals and other information. (See Appendix A for a discussion. Also see [6] for discussions of the random oracle setting in which this can be modeled, definitions for this setting, and discussions of the meaningfulness of instantiating random oracles via hash functions.)

Notice that the bulk of the user cost above is coming from the fact that he must run $2ln$ executions of KOD. Even though KOD itself is relatively cheap, these executions add up. To improve the efficiency, it would be nice to find a protocol which manages to prove that a is small without our bit-by-bit approach. Another possibility is a protocol which proves something *simultaneously* to all trustees.

4 VPKE based on Factoring

In this section we describe ways for users to partially escrow the factors of a given composite number. The composite number can be used as a public key in the RSA system.

The schemes described have several novel and useful features. One is that all communication between the user and the government trustees during the escrow protocol can take place over the public channel, with no need for encryption. This may be good from the export control point of view. Another feature, present in the second scheme, is that there is no sharp threshold of recovery, but rather a gradual release. Very few trustees cannot get the secret key, and lots of them can; in between, as more and more trustees get together, the work to get the key decreases as the number of trustees involved grows. Unfortunately these schemes don't achieve delayed recovery.

4.1 A threshold scheme

As before n is the number of trustees. (Note that n could be 1.) Let k_1, k_2 be a pair of security parameters. The first parameter, k_1, denotes the size of numbers recommended for use in a cryptosystem such as RSA which is based on the difficulty of integer factorization. The second parameter, k_2, denotes size of integers which would take a moderate amount of time to factor.[4] The user chooses n different numbers $M_1, ..., M_n$ (one per trustee) each product of two large (length k_1) primes. This is done in a way which ensures that $J_{M_i}(-1) = 1$. (The Jacobi symbol of -1 mod M_i is 1. There are many well-known ways of making sure this is true.) He also chooses a number N_2 which is the product of two small (length k_2) primes. The user makes $M_1, \ldots, M_n$ and N_2 public. Let N_1 be the product $M_1 \cdots M_n$ and let the composite number N, of the user, be the product $N_1 N_2$. This modulus is the basis for the cryptosystem, and could be used, for example, for RSA. The prime factors are kept secret, and enable decryption. (When encrypting and decrypting modulo N, computations can be performed individually modulo each of the M_i and modulo N_2. The results can then be combined via Chinese remainders. This will be much more efficient than working directly modulo N, and is the reason we made all the numbers $M_1, \ldots, M_n$ public.)

Now we describe how to do the verifiable partial escrow. Let the trustees be $T_1, \ldots, T_n$. For each $i = 1, \ldots, n$, the user and trustee T_i engage in the following protocol–

(1) The trustee checks that indeed $N = N_1 N_2$, and that the size of N_2 is $2k_2$. He also checks that $J_{M_i}(-1) = 1$.

(2) The trustee chooses at random a $y \in Z_{M_i}^*$ with Jacobi symbol -1, sets $z = y^2 \bmod M_i$, and sends z to the user. The user computes a Jacobi symbol $+1$ square root w of z modulo M_i (which he can do because he knows the prime factors of N_1) and sends w to the trustee. Upon receipt of w, the trustee checks that $w^2 = z \bmod M_i$. He then computes $\gcd(y - w, M_i)$ and thereby factors M_i. These factors are his partial share.[5]

In case the government receives legal authorization to wiretap and wants to recover the factors of N it does the following. First, it obtains from trustee T_i the factorization of M_i, and thereby has the factorization of N_1. Then it works to factor N_2, which is feasible as the size of the factors is k_2. The security is in the fact that $t < n$ trustees cannot factor N_1. However, since N_2 can be factored at any time, we do have early recovery.

Note that the above scheme besides providing partiality, requires no private communication between trustees and user.

4.2 Gradual disclosure version

The following scheme, in addition to providing partial recovery and non need for private channels between trustee and government, has a novel feature in comparison with known schemes. Namely there is a "gradual" increase in recovery probability as more and more trustees get together. Thus it is not a threshold scheme in the traditional sense of having some sharp threshold; rather as more and more trustees get together, the work factor they must put in to get the full key steadily decreases.

As above, the user's modulus N will be the product of two numbers N_1 and N_2, where N_2 is the product of two small (size k_2) primes. However $N_1 = p_1 \cdot p_2 \cdots p_l$ is now the product of l primes, each of length k_1 bits. It is chosen so that $J_{N_1}(-1) = 1$. Both N_1 and N_2 are given to the trustees. Again, for each $i = 1, \ldots, n$, the the user and trustee T_i engage in a protocol–

[4] This can be parameterized further by introducing functions $small(k)$, $large(k)$ where $small(k)$ denotes the number of wiretaps per year, and $large(k)$ denotes the total number of individuals. Then we require that the time required to factor $small(k)$ number of integers of size k_2 should be moderate, whereas the time required to factor $large(k)$ number of integers of size k_2 should be infeasible. For appropriate choices for k_1 and k_2 given the performance of factoring algorithms today see the article of Odlyzko [31].

[5] Since the result of this is just to transfer the factorization of M_i to T_i one might ask why we do it this way, as opposed to just sending the factors of M_i to T_i. The reason is that to do the latter we need a private channel from the user to T_i, and we want to avoid encryption.

(1) The trustee checks that indeed $N = N_1N_2$, and that the size of N_2 is $2k_2$. He also checks that $J_{N_1}(-1) = 1$.

(2) The trustee chooses at random a $y \in Z^*_{N_1}$ with Jacobi symbol -1, sets $z = y^2 \bmod N_1$, and sends z to the user. The user computes a Jacobi symbol +1 square root w of z modulo N_1, and sends w to the trustee. Upon receipt of w, the trustee checks that $w^2 = z \bmod N_1$. He then computes $\gcd(y - w, N_1)$ and thereby obtains a split S_i of N_1. This split is his partial share.

Note, that even if only the trustee (or similarly, if only the user) acted according to the protocol, then the trustee will have in its hands a random split of N_1 (among all possible splits of N_1 into integers a, b such that $N_1 = ab$.)

When the government wants to wiretap it first factors N_2. Then it gives the court order to the trustees who return their splits of N_1. Since each share S_i represents a random split of N_1, the probability that there exists a pair of primes p_u, p_v $(u \neq v)$ such that for all S_i both p_u, p_v divide S_i (or alternatively both divide N_1/S_i), is less than $l^2/2^n$. (Eg. to achieve probability of recovery greater than 1/2 set the number of prime factors l to be say 3, and the number of trustees to be $n = 5$.) The factorization of N_1 is obtained by combining the n splits of N_1 by repeated greatest common divisor (gcd) computations.

Second, even if t of the trustees cooperate to try to factor N_1 by telling each other their shares, before court authorization is obtained, then there will be at least one pair of distinct primes p_u, p_v whose product remains unsplit in all the shares of the t trustees, as long as $l > 2^t$.

Suppose we want to make the probability of recovery in the above scheme $1 - 2^{-60}$. If $l = 3$, we will need $n = 65$. That is, the scheme needs a lot of trustees. Here is a way to increase the recovery probability without increasing the number of trustees. Have the user simply repeat the exact same protocol above several times with the same trustee, and the share of the trustee now becomes the set of shares obtained from every execution of the above protocol. If the number of repetitions is m, then the probability of complete recovery is greater than $1 - l^2/2^{mn}$. On the other hand we must also have $l > 2^{tm} + 1$ to guarantee security against t trustees.

Finally, the entire procedure of disclosing the partial key N_1 is done over the public network. (For authenticity it is expected that everybody knows a public key of each trustee under which trustees sign the messages they send. But we require no additional public key cryptography, and in particular no encryption. This is good for export control.) As long as the trustees verify their computation as specified above (even in the presence of bad t trustees when $l > 2^t$), the partial disclosure of keys has been fully verified.

4.3 Early recovery

A drawback of the schemes in this section is that early recovery is possible. Namely, as soon as N_1 and N_2 are published, the government can start working on factoring N_2, a task which takes non-negligible time but is feasible. Thus, in the interim before the government ever gets the escrowed shares of the factorization of N_1, it can spend all its time factoring N_2 for all users in existence. This, as we discussed in the introduction and Section 2, is not desirable.

Next we note that other existing schemes have the same problem.

4.4 Early recovery for factoring based GPKE

We note that [28] discusses a "guaranteed partial key escrow" factoring based scheme as well. This scheme is very similar to the first scheme above, but, in fact, more efficient. But the scheme requires the use of private channels between the government and the individuals.

We note that the GPKE scheme of [28] suffers from early recovery. The reason is interesting so let us discuss it.

In the scheme of [28], the modulus published has the form $N = p_1 \cdots p_n q_1 q_2$, with the p_i being large (512 bits) and the q_i being small (150 bits). Trustee i is given p_i over a private channel. (Each trustee reveals a few bits of his prime so that they know that they all got different primes.) In the recovery phases, the trustees pool their shares and have $p_1 \cdots p_n$. Now, dividing N by this they get $q_1 q_2$. Now they must factor $q_1 q_2$. This is expected to take a "reasonable but not too large" amount of time, due to the sizes of q_1, q_2.

However, we recall that the elliptic curve based factoring algorithms have the property of finding the smallest prime factor of a given number in time proportional to the size of that factor [25]. So just given N, anyone can quickly find q_1, q_2. This is an early recovery attack.

How well this attack performs depends of course on the size k_2 of q_1 and q_2. We see a tradeoff. The trustees, in the recovery phase, would run the Number Field Sieve (NFS) algorithm to get q_1, q_2. The early recovery attacker is running the slower elliptic curve method. Thus the early recovery attack will take a little longer than what the trustees would invest. As long as it is feasible however, it is still a very damaging attack, because we find q_1, q_2 using only off-line computation, at any time after the escrow has been done.

Now, we might try to choose k_2 large enough that the elliptic curve method becomes "infeasible" but small enough that NFS stays "feasible." In this narrow gap, one might say that, at least in practice, the early recovery fails. But the gap is small, and it is not clear we can actually find numbers for which it can be exploited: what will happen is that the trustee computation will get really quite large even if "feasible." Furthermore, there is no proof (assuming only, say, that factoring is hard) that there are no other early recovery attacks. For example, if someone were to develop a NFS variant which had a running time equal to NFS but measured as a function of the smallest prime factor, then the early recovery attack would take exactly the same time as the trustees are supposed to invest. To assume that no such variant of NFS exists is a very strong assumption, too much to have the security of a scheme depend upon.

4.5 Better schemes

As we saw, one drawback of the above schemes is early recovery. Another is that the numbers defining the cryptosystem are large, making public key operations slow.

Both these problems are addressed in our new work [4]. The scheme we present there enables a user to escrow the factors of a given RSA modulus, say one product of two primes. That is, unlike the above schemes, we will not have to use large modulii which contain more than two primes. (In other words the underlying cryptosystem can be the standard RSA one. In particular, encryption and decryption will as efficient as standard RSA). The scheme works for a given number of trustees n, unlike the above where n may need to grow for security. Finally, the scheme achieves delayed recovery. It is based on a new paradigm for key escrow via "encapsulation."

References

[1] M. Bellare, S. Micali and R. Ostrovsky. The (True) Complexity of Statistical Zero-Knowledge. *Proceedings of the* 22nd *Annual Symposium on the Theory of Computing*, ACM, 1990.

[2] M. Bellare and O. Goldreich. On defining proofs of knowledge. *Advances in Cryptology – Crypto* 92 *Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.

[3] M. Bellare and S. Goldwasser. Verifiable partial key escrow. Technical Report number CS95-447, Dept of CS and Engineering, UCSD, October 1995.

[4] M. Bellare and S. Goldwasser. Encapsulated key escrow. Available at `http://www-cse.ucsd.edu/users/mihir`. Earlier version was MIT Laboratory for Computer Science Technical Report 688, April 1996. Also presented at Eurocrypt 96 rump session, May 1996.

[5] M. Bellare and R. Rivest. Translucent cryptography– An alternative to key escrow and its implementation via fractional oblivious transfer. MIT Laboratory for Computer Science Technical Memo. 683, February 1996.

[6] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.

[7] M. Bellare and P. Rogaway. Optimal asymmetric encryption. *Advances in Cryptology – Eurocrypt* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1994.

[8] T. Beth, H. Knobloch, M. Otten, G. Simmons, and P. Wichmann. Towards acceptable key escrow systems. *Proceedings of the Second Annual Conference on Computer and Communications Security*, ACM, 1994.

[9] G. Blakley. Safeguarding cryptographic keys. *AFIPS Conference Proceedings*, June 1979.

[10] M. Blaze. Protocol failure in the escrowed encryption standard. *Proceedings of the Second Annual Conference on Computer and Communications Security*, ACM, 1994.

[11] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption that hides all partial information. *Advances in Cryptology – Crypto* 84 *Proceedings*, Lecture Notes in Computer Science Vol. 196, R. Blakely ed., Springer-Verlag, 1984.

[12] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. *Proceedings of the* 27th *Symposium on Foundations of Computer Science*, IEEE, 1986.

[13] R. Cramer. Private communication, January 17, 1996.

[14] R. Cramer, I. Damgård and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[15] D. Denning. To tap or not to tap. *CACM* 1993.

[16] D. Denning and M. Smid. Key escrowing now. *IEEE Communications Magazine,* Sep. 1994.

[17] Y. Desmedt. Securing traceability of ciphertexts: towards a secure software key escrow system. *Advances in Cryptology – Eurocrypt* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 921, L. Guillou and J. Quisquater ed., Springer-Verlag, 1995.

[18] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory* IT-22, pp. 644–654, November 1976.

[19] C. Dwork and M. Naor. Pricing via processing, or how to combat junk mail. *Advances in Cryptology – Crypto* 92 *Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.

[20] U. Feige and A. Shamir. Witness Indistinguishable and Witness Hiding Protocols. *Proceedings of the* 22nd *Annual Symposium on the Theory of Computing*, ACM, 1990.

[21] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. *Proceedings of the* 28th *Symposium on Foundations of Computer Science*, IEEE, 1987.

[22] Y. Frankel and M. Yung. Escrow encryption systems visited: attacks, analysis and designs. *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[23] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. *SIAM J. Comput.* Vol. 18, No. 1, pp. 186–208, February 1989.

[24] J. Kilian and T. Leighton. Fair cryptosystems revisited. *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[25] H. Lenstra. Factoring integers with elliptic curves. *Annals of Math* Vol. 126, pp. 649–673, 1987.

[26] A. Lenstra, P. Winkler and Y. Yacobi. A key escrow system with warrant bounds. *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[27] S. Micali. Fair public key cryptosystems. *Advances in Cryptology – Crypto* 92 *Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.

[28] S. Micali. Guaranteed partial key escrow. MIT Laboratory for Computer Science Technical Memo. 537, September 1995.

[29] S. Micali and A. Shamir. Partial key escrow. Manuscript, February 1996.

[30] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems. *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[31] A. Odlyzko. The future of integer factorization. CryptoBytes, RSA Labs Newsletter, Vol 1, No. 2, pp. 5–12, Summer 1995.

[32] R. Ozzie. Prepared remarks. Delivered at RSA Data Security Conference, San Francisco, January 17, 1996. Available at `http:// www.lotus.com/notesr4/ozzie.htm`.

[33] T. Pedersen. Distributed provers with applications to undeniable signatures. *Advances in Cryptology – Eurocrypt* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 547, D. Davies ed., Springer-Verlag, 1991.

[34] T. Pedersen. Non-interactive and information theoretic secure verifiable secret sharing. *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[35] R. Rivest, A. Shamir and D. Wagner. Time-lock puzzles and timed-release crypto. Manuscript available at `http://--0----8yr.s-lv-tore.rests.mit.edu:80/~rivest`.

[36] R. Rivest, A. Shamir, L. Adleman. A method for Obtaining Digital Signatures and Public Key Cryptosystems. *CACM*, Vol 21, No. 2, pp. 120–126, February 1978.

[37] A. Shamir. How to share a secret. *CACM*, Vol. 22, No. 11, pp 612–613, November 1979.

[38] A. Shamir. Partial key escrow: A new approach to software key escrow. Private communication made at Crypto 95, August 1995. Also presented at Key escrow conference, Washington, D.C., September 15, 1995.

[39] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, Vol. 4, pp. 161–174, 1991.

Common Input: $A \in G$, and $g, h \in G$.

Prover Input: $y \in Z_q$ such that either $A = h^y$ or $A = gh^y$.

Prover's Claim: I know y such that either $A = h^y$ or $A = gh^y$.

Protocol:

1. The prover forms R_1, R_2 according to A as follows–

1.1. If $A = h^y$ then the prover picks $w_1, r_2, c_2 \in Z_q$ at random, and sets $R_1 = h^{w_1}$ and $R_2 = h^{r_2} \cdot (A/g)^{-c_2}$.

1.2. If $A = gh^y$ then the prover picks $r_1, c_1, w_2 \in Z_q$ at random and sets $R_1 = h^{r_1} A^{-c_1}$ and $R_2 = h^{w_2}$.

He sends R_1, R_2 to the verifier.

2. The verifier now issues a random challenge $c \in Z_q$.

3. The prover responds according to A–

3.1. If $A = h^y$ the prover computes $c_1 = c - c_2 \bmod q$ and $r_1 = w_1 + c_1 y \bmod q$.

3.2. If $A = gh^y$ then prover computes $c_2 = c - c_1 \bmod q$ and $r_2 = w_2 + c_2 y \bmod q$.

He sends r_1, r_2, c_1, c_2 to the verifier.

4. The verifier checks that $c_1 + c_2 = c \bmod q$ and $h^{r_1} = R_1 \cdot A^{c_1}$ and $h^{r_2} = R_2 \cdot (A/g)^{c_2}$.

Figure 2: *Protocol* KOD– *Proof of knowledge of de-commitment:* Witness hiding proof that the prover knows y such that $A \in \{h^y, gh^y\}$.

[40] P. Van Oorschot and M. Wiener. On Diffie-Hellman key agreement with short exponents. *Advances in Cryptology – Eurocrypt 96 Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[41] White House press release regarding the Clipper Chip, April 16, 1993. Also Escrowed Encryption Standard (EES), Federal Information Processing Standards Publication (FIPS PUB) 185, 1994.

[42] A. Young and M. Yung. The dark side of "Black-Box" cyrptography or: Should we trust Capstone? *Advances in Cryptology – Crypto 96 Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

A The Proof of knowledge

In the VPKE protocol for DH given in Figure 1, A_i is supposed to be a commitment to a bit, namely the i-th bit of a. Our conviction that the underlying a is really only $2l$ bits long, and not something much bigger, comes from the conviction that A_i hides a bit, and not something much bigger. Our fear is that the user would cheat, and commit instead to a string z, by constructing the commitment as $g^z h^v$ for some $z \in Z_q$. So we prove, in the last step of the VPKE protocol, that each A_i really hides a bit.

This is done via a sub-protocol. We present it here as a protocol between a prover and a verifier. They have common input A and the prover has y (the witness) such that either $A = h^y$ or $A = gh^y$. We regard g, h as fixed and consider the predicate $P_{g,h}(A, y)$ which is true if and only if either $A = h^y$ or $A = gh^y$. Notice that for any A there are exactly two, distinct witnesses, namely $\log_h(A)$ and $\log_h(A/g)$. The prover will prove he knows one of them, without revealing which.

We stress that what we want is a proof of knowledge, not of membership in any language. Indeed, there is no non-trivial language membership problem to consider. The statement $\exists y : P_{g,h}(A, y)$ is true for all A— what we need to do is show the prover *knows* such a y.

Regarding the security, what we want is perfect witness indistinguishability. Namely, the view of a verifier is the same regardless of which witness the prover holds [1, 20]. This is a weaker property than perfect zero-knowledge, but suffices for this application, because it means the verifier cannot tell whether the bit $D(A)$ is 0 or 1. We also want the protocol to have a low "knowledge error" κ in the sense of [2]. This means that without knowing a witness, it is possible for a prover to make the verifier accept a κ fraction of the time, but any excess above κ in the accepting probability translates into witness recovery with probability equal to this excess, via a witness extractor algorithm M which uses the prover as an oracle.

Our earlier protocol. The preliminary version of our work [3] had presented and analyzed a protocol which consisted of repetitions of some three round atomic protocol. We had showed that the atomic protocol was perfect zero-knowledge (more than we needed) and had knowledge error one-half. We know that Parallel repetition lowers the knowledge error [2]. On the other hand, the parallel repetition protocol is witness hiding [20].

Although this works, it is relatively costly, because we need about 100 parallel repetitions, and each involves exponentiations.

The protocol KOD. After the appearance of [3], Cramer [13] informed us that it was possible to get a more efficient solution to our problem, by applying the techniques of Cramer, Damgård and Schoenmakers [14]. The protocol of Figure 2 is the one he communicated to us [13].

This protocol uses only three rounds to get a very low error. (That is, the protocol does not consist of repetitions of some atomic protocol. It *directly* gets low knowledge error.) The prover needs three exponentiations and the verifier needs four. As with our previous protocol, it is perfectly witness indistinguishable.

In terms of the framework of [14], protocol KOD ("knowledge of decommittal") is obtained by applying the transformation of [14] to the Schnorr protocol [39], on the instance $(A, A/g)$, with a 1 out of 2 threshold scheme, to show that you know the discrete logarithm of one of the two inputs without revealing which. For the complete proof that this is correct, refer to [14].

This protocol can be made non-interactive by specifying the verifier's challenge via a hash function applied to the input and previous messages of the prover. If the hash function is assumed random the protocol meets a random oracle model definition of witness indistinguishability given in [6].

B Information-theoretically secure VSS

The VPKE protocol of Figure 1 uses the non-interactive, information theoretically secure VSS of Pedersen [34]. For completeness, we describe that VSS protocol here.

The user wants to share x. He is understood to have already broadcast to the trustees the commitment $X = g^x h^u$, as in Figure 1.

Sharing. To share x the user does the following-

(1) The user chooses $f_1, \ldots, f_t \in Z_q$ at random. He sets $f_0 = x$. He then sets f to the polynomial $f(y) = f_0 + f_1 y + \ldots + f_t y^t$. (Recall t is a bound on the number of bad trustees).

(2) He computes $s_i = f(i)$ for $i = 1, \ldots, n$. (Recall n is the total number of trustees, and we assume $n < q$.)

(3) He chooses $v_1, \ldots, v_t \in Z_q$ at random and constructs the commitments $F_i = g^{f_i} h^{v_i}$ for $i = 1, \ldots, t$. He also sets $F_0 = X$ and $v_0 = u$. He broadcasts $F_1, \ldots, F_t$ to the trustees. (They already have F_0).

(4) He sets v to the polynomial $v(y) = v_0 + v_1 y + \ldots + v_t y^t$. He sets $t_i = v(i)$ for $i = 1, \ldots, n$. He then sends (s_i, t_i) *secretly* to trustee i, for $i = 1, \ldots, n$. (This means (s_i, t_i) is sent encrypted under a public encryption key of trustee i.)

VERIFY. For $i = 1, \ldots, n$, the i-th trustee checks that

$$g^{s_i} h^{t_i} \;=\; \prod_{j=0}^{t} (F_j)^{i^j}$$

If this check succeeds, trustee i accepts the share as valid.

SECURITY. There is a point to clarify about what "information theoretically secure" means. It is taken to mean that even a computaionally unbounded adversary gets no information about the secret. What it is important to emphasize is that this is true of the above protocol under the assumption that the private channel used in Step 4 is *perfectly private*. In practice, as we indicated above, one must use encryption.

This does not mean the notion of information-theoretic security is meaningless. What we can do is analyze our VPKE protocol (which uses the above VSS) under the assumption the channels are perfectly secure. Then, when we use encryption, the security of the latter guarantees that the extra loss of information in the VPKE context is negligible.