Appears as MIT Laboratory for Computer Science Technical Report 688, April 1996.

Encapsulated Key Escrow

MIHIR BELLARE* SHAFI GOLDWASSER†

April 1996

Abstract

The widespread need for encryption for private communication and stored information poses a problem when there exists an authority, such as the governement or business employer, who under some predetermined set of circumstances, needs to be able to obtain access to information and communication of selected users.

Key-escrow is the main solution considered to ensure the ability of an authority to wiretap communictaion. The main objection to all current Key-escrow proposals is that they assume complete faith in the authority and its trustees. If the authorities do not follow the rules, or are replaced by an un-trustworthy authority tomorrow, they can immediately recover the secret keys of all users, and embark on massive wiretapping automatically scanning everyone's e-mail and computer files.

We introduce a new approach to key escrow called *verifiable encapsulated key escrow* (VEKE), applicable to any encryption algorithm, which makes it verifiably computationally possible for an authority to only selectively wiretap a small number of individual users, and computationally prohibitive to launch *large scale* wiretapping. This is achieved by imposing a time delay between the obtaining the escrowed information of a user and obtaining the user secret key.

We achieve VEKE by a new cryptographic tool called *verifiable cryptographic time capsules* (VCTC). The capsules are ways of strongly encoding information, which allow an authority to verify that it can obtain the contents of the capsule after (and only after) a specified amount of time delay. When applied to key-escrow, the content of the capsules are secret-keys of users, and the amount of time it takes to open these capsules is a parameter which is set such that it is computationally possible to open a few of them, but computationally hard to open large numbers of them. When several trustees are available, the time capsule is split amongst them via a secret sharing scheme. When trustees pull their pieces together, they can recover the capsule and start computing toward opening it.

VCTC's can be constructed under the general assumption that claw-free trapdoor functions exist. For the purpose of key-escrow for the RSA cryptosystem (and the Diffie and Hellman cryptosystem), we give very efficient implementations of VCTC based on the particular assumption that factoring integers is hard (respectively, the assumption that the discrete logarithm is hard to compute).

Although conceived for the purpose of wiretapping and in the context of key-escrow, VCTC can be used for "sending information into the future" [May] with applications to auctions with closed bids, deferred electronic payments, and the sealing of documents for limited time periods.

---

* 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@watson.ibm.com` ; Web page: `http://www-cse.ucsd.edu/users/mihir`

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

1 Introduction

The need to use encryption to guarantee privacy is spreading as more people use electronic mail, buy products over the Internet, and keep personal records in computer files. Widespread use of strong encryption, poses however, a problem for a government or a business employer who believes that under some circumstances it should be able to gain access to communication and information of selected users. For example, currently it is possible for the various law enforcement agencies to wiretap phone conversations, if it obtains proper court authorizations, a process whose benefit vanishes if the encryption on the wire is encrypted. Several technical solutions have been proposed, attempting to balance the privacy of individuals with the needs of law enforcement.

1.1 Previous Technical Solutions and Objections

One suggestion, called *weak encryption*, is to enforce the use of encryption algorithms which the law enforcement agency can break, for example by requiring that the size of secret keys not exceed 40 bits, a restriction currently imposed by the US government on cryptographic algorithms used for export. Since the resulting encryption is then, however, breakable by anyone, this is not a solution that individuals and business can accept.

The idea currently receiving the most serious attention is *Key Escrow*. The user's secret key is either escrowed with the authority in full, or is split into n pieces and each piece is escrowed with a trustee of the authority so that at least t of the n pieces must be recovered in order to reconstruct the original secret key. The latter is referred to as *split key escrow*. The escrowing can be done either at the time the cryptosystem is set-up or at the time of transmission of secret communications (i.e escrow session keys). In case of authorized wiretapping (and only in such case), the trustees should hand over to the authority their pieces, which can immediately can reconstruct the user's secret key. The pioneering proposals were Micali's fair cryptosystems [Mi1] and the Clipper chip.

The objection to key-escrow is in all key-escrow proposals the individual's privacy relies entirely on trusting the authority and its trustees to follow the rules. If the authority does not follow the rules, then it can immediately recover everyone's secret keys, and embark upon mass wiretapping rapidly scanning everyone's e-mail and files. Furthermore if the authority in question is the government then, even if it is trustworthy today, it may be replaced by an un-trustworthy government tomorrow which could suddenly recover the secret keys of all users and embark on mass wiretapping.

The situation may even be worse with respect to commercial usage, where employees in a company depend on their employer for their livelihood and thus are quite vulnerable. Take a commercial company where e-mail is routinely used by employees for both private and public purposes. Companies may insist on a system where managers may be able to read office e-mail or files of subordinates in special cases (e.g. say when a subordinate sends e-mail to an employee of a rival company, or when an employee leaves the company). However, this is quite a delicate situation. The subordinates should be protected from higher rank employee's continuously searching file-systems and e-mails of subordinates in search for particular character-strings or data pattern, just as they must be protected from continuous wiretapping of their phone conversations by their employers. The danger of serious invasion of privacy via the electronic media within companies does not seem overly "futuristic". Similarly to the situation with government, a change in business management could bring a sudden recovery of employee secret keys and encroachment of their privacy. This problem of abuse of escrow arising from a sudden change in power has been raised by Simmons and Shamir. The following goal emerges: *Design strong encryption schemes for which it computationally possible for the government (or other pre-designated parties) to selectively wiretap individual users under some pre-specified conditions, but computationally impossible for the government to* *obtain mass quantities of secret keys of users and routinely scan through everyones personal files and e-mails.*

To meet this challenge, Adi Shamir proposed the following idea of *partial key escrow* in the context of private key cryptography. Take a private key encryption system such as DES with a long key of size k_1 bits, and escrow all but k_2 bits with the government in a split-key fashionShamir concretely proposes to set To recover the secret key of an individual, the government would need to obtain the escrowed bits of the secret-key, and then exhaustively search for the remaining un-escrowed bits of the key. This will take at most 2^{k_1} steps, which would allow the government to recover the secret-keys of selected individuals, but not suddenly and simultaneously, so it could not embark on massive wiretapping. Thus, effectively, partial key escrow introduces a time-delay for the government before it can recover the secret key.

There are several drawbacks to partial key escrow. First, the government cannot verify that the bits it received are indeed the partial bits of the individuals secret key. Second, when decomposing the key into two distinct parts – the escrowed and unescrowed parts – one should take great care, not to make it possible for the government to embark on finding out the unescrowed bits of a given secret-key, before finding out the escrowed portion of the secret-key – an attack which we refer to as the *early recovery*. (It is described in detail in Appendix A). Indeed some suggested extensions [Mi2] to Shamir's partial key escrow idea addressing verifiability, suffer from this attack as we point out in Section 1.5. Thirdly and most fundamentally, for partial-key escrow to work, it should be the case that the fastest way to recover the unescrowed bits of the secret-key is essentially exhaustive search. However there is no reason to believe this to be true for a generic cryptosystem. The key might be quite structured, and in general it is hard to isolate a set of bits having the property that one is forced to find all of them to recover the key.

Accordingly, further partial key escrow proposals [Mi2, BeGw] have exploited the structure of the crypto-system to find ways to "break up" *sk* into two parts such that the desired properties can be guaranteed. This has several disadvantages. First, it is hard to find ways to achieve this break up for particular systems, and even when found their quality is unclear[1]. Second, there is no general method for achieving time delayed key escrow for a general, given crypto-system; one has to exploit the structure of the given system.

1.2 This work

We provide a new approach, called encapsulated key escrow, which applies to any public or private encryption scheme (with security parameter k_1) and to any given level of time delay (specified by a second security parameter k_2) which the government (or authority) should encounter while trying to reconstruct the secret-key of a user which it wants to wiretap. Our method is verifiable and provably secure against early recovery attacks, and will provably not affect the security of the underlying encryption scheme. This is achieved, by first introducing a new cryptographic primitive called *verifiable cryptographic time capsules* .

Verifiable Cryptographic Time Capsules. Intuitively, a cryptographic time capsule (CTC) is a container into which one can put information, and set a time, so that a computational effort of the set time is required to open it and recover the information in it. In addition our time capsules are verifiable (VCTC) in that it is possible to verify that the capsule can be opened within the claimed computational effort, without giving any information about the contents, or shortening the computational effort required to open this time capsule.

[1]For example the solution for Diffie-Hellman used in [Mi2] relies on a relatively untested cryptographic assumption (that finding the discrete logarithm of an element of Z_p^* which lies in a small subgroup is essentially proportional to the size of the subgroup). Indeed, new attacks [VW] call this assumption into question.

We implement VCTCs as protocols between the person encapsulating information and those that should eventually recover it. In general we show how to implement verifiable cryptographic time capsules based on the general assumption that claw-free permutation exist, and in particular on the assumptions that RSA is hard to invert and discrete log is hard to compute. Moreover, our constructions show how starting from any time capsule to construct a verifiable time capsule.

If the party who should eventually recover the information encapsulated, is not a single entity but consists of n trustees (as in the usage of VCTC in the key escrow context when the authority may be split into n trustees), then we show how to give each trustee a piece of the CTC (not the information encapsulated in it!), so that each trustee individually can verify that they hold a piece of a proper CTC (i.e the properties of VCTC hold with respect to each trustee). Moreover, less than t pieces given to trustees (where t is a parameter of the system), yield no information about the CTC. Only after at least t trustees share their pieces, they can reconstruct the CTC, and start computing, for the pre-set time delay, toward recovering the contents of the CTC.

Verifiable Encapsulated Key Escrow (VEKE). To use VCTC for key-escrow for any public-key encryption algorithm (or private key encryption algorithm for which a commitment to the secret key has been published) whose secret key we denote by sk, we encapsulate sk into a VCTC. The properties of the VCTC guarantee the central features related to the time delayed recovery. In addition we implement some cryptosystem-specific protocols which verify that the information in the VCTC is indeed the secret key sk associated to the publicly known key pk.

We stress that our solution applies to *any* cryptosystem. Typically however we can exploit the properties of a specific cryptosystem to improve the efficiency of the solutions.

Encapsulated key Escrow is an instance of what we call *time delay key escow*(TDKE)[2], an key escrow method which imposes a time delay between the authority obtaining the escrow information from the trustees and recovering the secret key of the cryptosystem. Note, that by definition in a TDKE we require that time delay must take place *after* the escrowed information has been recovered. A *verifiable time delay key escrow* (VTDKE) scheme is one in which the government can verify thet indeed the secret key can be recovered within the claimed time delay. Throughour this paper we interchangably refer to our scheme as VEKE or VTDKE.

1.3 The advantages of VEKE over previous approaches

We summarize the advantages of VEKE as follows:

Generality and flexibility of implementation. Our method applies to any cryptosystem without modifying it and succeeds in escrowing the secret key of this system in a time delayed but verifiable way without changing the structure of the cryptosystem. In contrast, for partial key escrow, the methods depended on the structure of the cryptosystem and new methods have to be devised for each cryptosystem. In particular our method does not depend on any structure or property of the key (although we can exploit this structure sometimes to improve efficiency).

We can use an VCTC, and, in implementing the latter, any basic underlying CTC. This has several advantages. One is that providing time delay is no longer tied to a property of the cryptosystem. Another is the potential available as to choice of methods to provide delays. For a given setting, we might prefer a weak time capsule or a strong one, or adjust some other parameters of the time delay. We thus have greater flexibility and control of the time delays.

Delayed recovery. When the government has n trustees, each of which receives a piece of the CTC whose contents correspond to the secret key, it is information theoretically impossible for the

[2]In an earlier version of this paper we refered to EKE by the general name of TDKE

government to start working on recovering the contents of the CTC before at least t pieces are shared together, as it is impossible to cryptanalyze something on which you have no information. Thus the early recovery attack is impossible.

Verifiability. The trustees can verify at escrow time that (1) indeed they hold in their possession pieces of a time capsule containing the legal secret key of the user encryption algorithm, and (2) that the computational effort to open the time capsule is as claimed.

Provable security against massive wiretapping. Since the secret-key is inserted into a time capsule which is independent of the encryption algorithm used, we are guaranteed that the government will have to work a prespecified amount of work, before it can wiretap. In contrast, in Shamir's partial key escrow enabling wiretapping by the government is achieved by allowing the government access to part of the secret-key of the user. This access may enable the government to recover the remaining unescrowed portion faster than by exhaustive search.

Safe Storage of Escrowed Pieces. The physical security of the escrowed information, in any key escrow system is highly important. In the current proposal, the secret-key itself is never escrowed. Rather the time capsule of the the secret-key is escrowed. Thus, even if an adversary can break into the the data base of escrowed information of a trustee, he can still not embark on large scale wiretapping, as he has to open the escrowed information first.

Export Control. Currently, the government enforces the use of weak cryptography for export purposes. Encryption-software developers find it too expensive to create two versions of their programs – one with strong cryptography for domestic use and one with cryptography that is weak enough for export. The result is that in the United States, developers sell only the weaker cryptography software. Our system allows achieving both levels of security at the same time with the same underlying encryption algorithm, and thus seems especially attractive for export.

1.4 VCTC: Wider Usages

Sending information into future. May [May] points out that it would be extremely useful to be able to send encrypted messages into the future. A few applications for such a mechanism would be sending money into the future (e.g. deferred checks, trust funds, deferred mortgage checks) while protecting it for the time being, writing wills to be opened only after one passes on, sealing documents for a specified time period, and sealing bids for a contract to be opened only if you won the contract.

Our verifiable cryptographic time capsules can be used for exactly this purpose. The property of verifiability is especially attractive in some of these applications domains.

1.5 Related Work

Since Shamir [Sh2] suggested the idea of partial key escrow, it has been investigated by several researchers, including [Mi2, MiSh] and a pre-cursor of this work described in [BeGw].

In [Mi2] verifiable methods for partial-key-escrow are proposed, for the Diffie-Hellman public key cryptosystem and for an extended version of the RSA cryptosystem where the composite modulus has as many prime factors as the number of trustees. We show however that these methods however suffer from the early-recovery problem. Namely, it is possible to recover the unescrowed portion of the key, prior to recovering the escrowed portion of the key. See Appendix A for description of attacks.

In an earlier work [BeGw] we proposed a verifiable partial key-escrow method for the Diffie and Hellman public-key escrow system which is provably secure against early recovery and guarantee that recovering the unescrowed portion of the key can take place only after the escrowed portion of the key has been recovered. We believe, however, that our VEKE approach is the correct one, since as we discussed above, the partial key escrow paradigm scales down the security of the users encryption algorithm in unpredictable ways which do not seem amenable to general theoretical treatment. In particular, VCTCs yield the first solution for the RSA encryption scheme that achieves delayed recovery, and an alternative solution for DH encryption.

In [Ri2] , Rivest proposes several ideas of how to incorporate into one encryption algorithm multiple levels of security (which can lead to another generalization of partial key escrow) as follows. For each encryption algorithm, there are several secret keys, such that obtaining one would enable you to compute the other with a certain amount of work, and possessing all of them would enable you to decode. In all his proposals, the secret keys depend on each other and no verifiability is provided. In the key escrow context, this yields similar problems to those discussed for partial key escrow.

The work of Rivest, Shamir, and Wagner [RSW] proposes the concept and two implementations of time-lock puzzles, any of which can be used as a cryptographic time capsule. [RSW] do not address verifiability, but as our construction of VCTC takes any time-capsule as a starting point and makes it verifiable, it can be used to make any of the time-lock puzzles which they propose, into verifiable time-lock puzzles.

1.6 Road map

Definitions for verifiable time delayed key escrow are in Section 2 and definitions for cryptographic time capsules are in Section 3. We then sketch the how we build verifiable cryptographic time capsules in Section 4 and the resulting approach to VTDKE in Section 5. We provide the detailed construction of a VTDKE for the DH system in Section 6, for RSA is in Section 7, and a general construction for any cryptosystem is in Section 8. Early recovery attacks are described in Appendix A.

2 Verifiable time delayed key escrow: Definition

We provide here an informal definition of the notion of verifiable time delayed key escrow.

The parties involved are a user U and n trustees $\text{Trustee}_1, \ldots, \text{Trustee}_n$, with $n \geq 1$. We want to set up for the user a public key pk so that U has the matching secret key, but this key is escrowed in a time delayed and verifiable way.

For full generality, the keys are of some arbitrary, agreed upon cryptosystem. (For example, RSA or Diffie-Hellman.) This system is specified by a triple (G, E, D) of algorithms called the generating, encrypting and decrypting algorithms, the first two of which are probabilistic, and all are polynomial time. The generator G takes a security parameter 1^k, written in unary, and produces a pair (pk, sk) of matching public and secret keys for the user. Another user, given pk, can encrypt a message M via $M' = E_{pk}(M)$, and the user can decrypt this via $M = D_{sk}(M')$. Security is in the usual sense of probabilistic encryption [GM].

The VTDKE system has several parameters. There is the number $t < n$ of trustees that are not trusted. There are two security parameters, k_1 and k_2. The first governs the size of keys (pk, sk), and hence the security of the underlying cryptosystem. The second governs the time for delayed recovery, and that time is denoted $\text{Delay}(k_2)$. (In general, the time delay could be different for different users. This function denotes the chosen time delay for our particular user.)

The VTDKE scheme for cryptosystem (G, E, D) is specified by an escrow protocol $\mathcal{P}$ and a recovery algorithm *Recovery*. The first step is to execute protocol $\mathcal{P}$. It involves all $n+1$ parties. At the end of this, U will have a pair (pk, sk) of the cryptosystem, and Trustee$_i$ holds some information which includes pk and usually something more, and which we denote by I_i and will discuss in the sequel. For any subset T of the trustees we let I_T be the set of all I_i such that Trustee$_i$ is in T. Trustees are modeled as polynomial time algorithms in discussing security. We now list the properties that are required to hold.

Correct distribution of keys. The key pair (pk, sk) is correctly distributed as though it were obtained by running the generator G. This is true even if up to t trustees deviate from the prescribed protocol $\mathcal{P}$.

Security of the cryptosystem. Let T be any set of at most t trustees. Then it is infeasible for them to find sk. More formally, consider a set of at most t trustees, allowed to behave arbitrarily during the execution of protocol $\mathcal{P}$, and later perform arbitrary polynomial time computation based on their joint view. Their success in finding sk is negligible.

This means that that encryption under key pk is secure, and users can use their keys without fear that a number of trustees smaller than the threshold can recover the encrypted data.

Recovery with some effort. After the escrow, it is possible for $t+1$ trustees to recover sk, as long as they invest some computational effort. More precisely, let T be any set of at least $t+1$ trustees. Say that following a successful execution of $\mathcal{P}$ they pool their received information to get I_T. Then applying the recovery algorithm *Recovery* on input I_T yields sk, and the recovery algorithm is guaranteed to run in time at most $\mathrm{Delay}(k_2)$.

No recovery without effort. Informally, the $\mathrm{Delay}(k_2)$ time effort is necessary: $t+1$ or more trustees who pool their information can't reduce below $\mathrm{Delay}(k_2)$ the time to recover sk. It is important, however, to say this more precisely and explain exactly what it means.

Let B be any set of at most t trustees. Allow them to misbehave during the execution of $\mathcal{P}$. Furthermore, after the escrow is completed, allow them to pool their information and perform arbitrary polynomial time computation. Now, *after* this effort, let them join forces with at least one other trustee. Then this set of at least $t+1$ trustees must invest at least $\mathrm{Delay}(k_2)$ time to find out anything useful about sk.

Note that early recovery is by definition ruled out: a pre-computation based on the public information, or even the information of any t trustees, will not help reduce the eventual recovery time below $\mathrm{Delay}(k_2)$. Furthermore, investing time in recovering keys of other users in the system will also not reduce below the threshold the time to recover the key of our particular user, because the creation and recovery of other keys can be "simulated" by the group B of bad trustees. We also stress that it isn't enough to prevent recovery of sk; we are asking for something stronger, namely that even partial information about it won't leak.

We have not said exactly what are the thresholds, namely to what extent sk remains hidden as the invested time approaches $\mathrm{Delay}(k_2)$. This depends on the scheme.

Verifiability. At the end of the escrow protocol $\mathcal{P}$ the parties are assured that the above properties are true. The trustees are assured that (with some effort) they can recover a string, and this string is really the secret key matching the user's public key. (Otherwise the user might just escrow some garbage, unconnected with his public key.) Furthermore, they are assured that the recovery process won't take *more* than $\mathrm{Delay}(k_2)$ time once $t+1$ of them have pooled their information. On the other hand, the user is assured of the distribution of the keys, the security of the cryptosystem and that there is no recovery without effort. In particular the user is assured the trustees would need time *at least* $\mathrm{Delay}(k_2)$ to recover sk, and that early recovery is impossible.

Ensuring this multi-faceted verifiability is one of the important technical difficulties in the protocols.

SUBTLETIES AND ASSUMPTIONS. The above requirements are a simplification in several ways. Let's discuss one.

The goal of enabling the trustees to recover *sk* is to enable them to decrypt information sent to the user. As long as the users in question use the cryptosystem in the prescribed way, *sk* indeed enables decryption. But they may not. In particular, as pointed out by [KiLe], it may be possible to set up "subliminal channels." However, [KiLe] also show some simple and quite general ways in which this can be avoided so that we may effectively assume the user does use the prescribed system. (The idea is that the choice of keys is not left to the user alone, but is done jointly by both parties in the protocol.) So we stick to the simpler setting of [Mi1]. A full and formal definition of time delayed key escrow would follow and extend the definition of [KiLe] to the time delayed setting.

3 Time Capsules: Definitions and Construction

We build verifiable time capsules, our main primitive, on top of basic time capsules. Here we provide definitions and sketch a few constructions for the latter.

Roughly a time capsule is a container into which one can put information, and set a time, so that a computational effort of the set time is required to recover the information. The definition is by necessity "concrete:" since we want to talk about specific amounts of time we eschew asymptotics. We point to various parameters that measure the quality of time capsules. For example, the tightness of the recovery threshold: in "weak" capsules, as more effort is made, the recovery probability increases; in "strong" ones, until something approaching the set time is invested, on has no particular advantage in recovering the encapsulated information.

Definition 3.1 An encapsulation scheme, or cryptographic time capsule (CTC) is a pair $(\mathit{Encap}, \mathit{Decap})$ of deterministic functions, the first being polynomial time computable. Associated parameters are a key length $\mathit{Kl}(\cdot)$, an input length $\mathit{Il}(\cdot)$ and an output length $\mathit{Ol}(\cdot)$, all integer valued functions of the security parameter k. The *encapsulation function* Encap takes 1^k, a $\mathit{Kl}(k)$ bit *key* K, chosen at random, and a $\mathit{Il}(k)$-bit string (the information or plaintext) s to be encapsulated, and produces a string $C = \mathit{Encap}(1^k, K, s)$ which we call the *capsule*. Its length is $\mathit{Ol}(k)$. Given 1^k and a capsule C, the *decapsulation function* $\mathit{Decap}(1^k, C)$ returns the information s.

Ideally we want that de-capsulation is unique: if $C = \mathit{Encap}(1^k, K, s)$ for some K then $\mathit{Decap}(1^k, C) = s$. However, we ask less. Namely that it is computationally infeasible to find distinct pairs (K, s) and (K', s') for which $\mathit{Encap}(1^k, K, s) = \mathit{Encap}(1^k, K', s)$. (Computationally infeasible means there is no poly(k) time algorithm for the task.)

An encapsulation scheme is a little like an encryption scheme: think of *Encap* as the encryption algorithm and *Decap* as the decryption algorithm. But notice some differences. The "decryption" algorithm doesn't take the key K, and needn't be polynomial time computable, and the encryption algorithm is not randomized. We'll see that one way to think of it is as a sort of "one-time" encryption scheme which can encrypt just one message under a given key and has very particular security levels.

We now need to define security. It is important to measure computation time precisely, so we fix some RAM model of computation in which to do this, like one used in an algorithms textbook. The security of the capsule is measured by two parameters, $T(\cdot)$ and $S(\cdot,\cdot)$. The first, called the *decapsulation time*, is the running time of *Decap*. (That is, $\mathit{Decap}(1^k, C)$ runs for $T(k)$ steps when

C is the encapsulation of some $Il(k)$ bit string s.) Typically it will be *exponential* in k. The second parameter S measures the success in "breaking" the capsule as a function of the time invested by an adversary. It is a function of k and the running time of the adversary, and needs more elaboration.

The most straightforward definition of breaking a capsule C would be obtaining the information s. But an effective capsule ought to meet a stronger requirement. Namely, partial information about s should also be hidden, just like in an encryption scheme. Our approach to formalizing this follows [GM]. A capsule-cracker is an algorithm A that takes $1^k, C$ and a pair m_0, m_1 of plaintexts, and outputs a bit. Let $P_A^i(1^k, m_0, m_1)$ be the probability that $A(1^k, C, m_0, m_1) = 1$ when $C = \mathit{Encap}(1^k, K, m_i)$ for a randomly chosen, $Kl(k)$ bit key K ($i = 0, 1$). The *advantage* of A, denoted $\mathsf{Adv}_A(1^k)$, is the maximum of $|P_A^1(1^k, m_0, m_1) - P_A^0(1^k, m_0, m_1)|$ over all pairs m_0, m_1 of $Il(k)$ bit strings. We denote by $\mathrm{Time}_A(k)$ the maximum, over all $Il(k)$ bit strings m_0, m_1, of the running time of $A(1^k, C, m_0, m_1)$ in the experiments we have just defined.

Definition 3.2 Encapsulation scheme $(\mathit{Encap}, \mathit{Decap})$ is (T, S)-secure if the decapsulation time is T and for all τ it is the case that $\mathsf{Adv}_A(1^k) \le S(k, \mathrm{Time}(k))$ for all capsule-crackers A.

Notice that $S(k, T(k)) = 1$. Barring this we have said nothing about how S behaves. Thus this is a very general definition. The question is to find schemes with nice S functions. What we would like is that $S(k, \tau)$ be very small for $\tau < T(k)$. The quality of the scheme is the extent to which this is true.

This is the most basic version of the definition. One can further classify capsules according to various features. One such is the extent to which parallelism helps reduce the decapsulation time. This is considered in [RSW] whose "time locks" try to ensure not only that a certain amount of time must be invested but also that this be sequential time.

To get a better understanding and also to see what kinds of encapsulation schemes are available, let's look at a few examples.

ENCAPSULATION BY CIPHERS. Let F be a variable key length cipher, like RC5 [Ri1]. A key κ describes the function F_κ used to encrypt and its inverse F_κ^{-1} used to decrypt, and there is some associated input length $l = Il(k)$. We let $Kl(k) = k + l$. The key K of the encapsulation scheme is a pair (κ, r) and we let $\mathit{Encap}(1^k, (\kappa, r), s) = (F_K(s), r, F_K(r))$. The function Decap works by exhaustive search: given $C = (u, r, v)$ it searches through the space of k-bit strings, and for each candidate x computes $F_x(r)$ and checks if it equals v. If so, it outputs $F_x^{-1}(u)$ as the plaintext.

This sort of thing is typically understood by modeling F as a collection of 2^k independent, randomly chosen permutations on l bit strings. Then one can see that decapsulation is unique with high probability. The opening time $T(k)$ is clearly 2^k computations of the underlying cipher. The question is the security. As long as the capsule-cracker has not tried the right key, it has no information about s. So the security is essentially the probability of having got the right key. One can compute that roughly $S(k, \tau) = 2^{-m}$ if τ is 2^{k-m} steps, where a step is a computation of F or F^{-1}.

THE SQUARING PUZZLE. This is a scheme due to [RSW] for constructing "time locks." Its advantage is that parallelism does not seem to speed up opening process. (There is no formal definition or proof of security in [RSW], but the scheme they design is conjectured to be a good time lock.)

4 Verifiable Cryptographic Time Capsules (VCTC)

In the previous section we in essence were trusting the user (or encapsulator) to put a secret s into a capsule and seal it for a specified delay. What's lacking is verifiability. There are two elements of verifiability. The first is *content verification.* The user may put junk into a capsule, instead of the required object. The second is *time verification.* The user may set the timer on the capsule too high, some infeasible amount rather than the agreed upon delay. We want to construct verifiable cryptographic time capsules, in which the party who eventually is to open the time capsule can verify that the encapsulation was done properly, in the sense that both the contents and the timer were correctly set.

For content verification, such contents have to be well-defined. This depends on the setting in which the time capsule is used. For example, in the context of key escrow for a public-key cryptosystem, if sk is the supposed content of the time capsule, pk defines this information uniquely and one can discuss verifying the contents of the time capsule. Alongside we have a general way to provide time verification, namely to ensure that the capsule can be opened within $delay(k_2)$, without in the process revealing the information or decreasing the effort that would eventually be required to open it. We now proceed to describe the paradigm we use to achieve this.

THE PARADIGM AT A HIGH LEVEL. Let s be the information which should be recovered with $delay(k_2)$, and ps be a public commitment to s. Let m be a security parameter. For example, let $ps = f(s)$ where $f(s) = g^s \bmod p$ where p is a prime and g is a generator for Z_p^* and $1 \leq s \leq (p-1)$. The general plan to achieve verifiability is as follows.

- First, find a way to split s into pairs (s_0, s_1), and a corresponding public value h such that (1) given a pair (s_o, s_1), s is easily reconstructed, and (2) Given s_i, ps and h, it is easy to verify that s_i is a half of a valid pair. (For example, for the above f, choose at random $1 \leq s_0, s_1 \leq (p-1)$ such that $s = s_0 + s_1 \bmod (p-1)$, and let $h = (f(s_0), f(s_1))$.)

- Second, A (who has the secret s), chooses m pairs (s_0^j, s_1^j) and h^j as above for $j = 1, ..., m$, encapsulate s_1^j of each pair, letting $C_j = Encap(1^k, K_j, s_1^j)$, where K_j is a randomly chosen k_2 bit key, and sends C_j, h^j to B (the party who should eventually recover the secret s).

- Third, to ensure verifiability, the following protocol is carried out between A and B:

  1. B chooses a random subset $J \subset [1, ..., m]$ and sends J to A.
  2. A sends to B, for every $j \in J$, s_1^j, K_j and for every j not in J, s_0^j. B accepts unless for some j s_i^j is not a half of a valid pair, or if for any $j \in J$, $C_j \neq Encap(1^k, K_j, s_1^j)$. In case $j \in J$ he also checks that K_j is of the appropriate length, which is what tells him that the timer was properly set.

- This convinces B that for most $j \notin J$, the capsule C_j is correct; namely, it contains s_1^j encapsulated under a key of the appropriate length. B can now invest the necessary time to break one of these capsules and get s_1^j; he already has s_0^j, and from that he would get s. (Technical point: B might need to work on a couple of capsules C_j for $j \notin J$, but the probability he needs to look at more than one is only about 0.02.)

SPLITTING THE SECRET. Here we have considered a setting in which there is only one receiver B. In the split key escrow setting, B is a collection of trustees. Rather than providing information to B, information must be shared amongst the trustees via secret sharing. This yields a threshold VCTC which $t+1$ out of n players can open but not less. See the following sections.

REMOVING INTERACTION. As written the above paradigm and the following protocols require interaction between A and the receiver(s). However, the whole protocol can be made non-interactive. (That is, the user sends a single message, after on receipt of which the receiver decides whether or not to accept the capsule.) This is done by specifying the challenges via a hash of the committals and other information. This is a heuristic transformation, but one that seems secure in practice. It is typically justified under the assumption that the hash function behaves like a random oracle. We refer the reader to [BeRo] for discussions of the random oracle setting in which this can be modeled, definitions of zero-knowledge in this setting, and discussions of the meaningfulness of instantiating random oracles via hash functions.

5 VCTC and VEKE

Basic encapsulated key escrow. The basic paradigm is like this. The user puts the secret key *sk* of the cryptosystem into a time capsule C. Now he shares C amongst the trustees, via a t out of n threshold secret sharing scheme. (For example, Shamir's scheme.) At recovery time, $t+1$ trustees will pool their shares to get C, and then open this capsule, which takes the allotted delay time.

The approach works even for just one trustee, and in both the private and public key settings.

Notice that early recovery is at once avoided. A group of at most t trustees does not even have the capsule C. In particular they have no information about *sk* other than that given by pk. So there is nothing they can pre-compute to enable them to open C faster, or recover *sk* quicker, after $t+1$ of them get together and recover C.

However there is something lacking in this approach, namely verifiability. We are trusting the user to put *sk* into a capsule and share it. He might put rubbish into the capsule, or share incorrectly. He might also cheat by setting the timer on the capsule too high, some infeasible amount rather than the agreed upon $\mathrm{Delay}(k_2)$.

There are settings in which one might decide to trust the user to use pre-scribed software, or have protected hardware for the basic time delayed escrow task, and then the above suffices. Most often, however, one imagines verifiability is needed. Providing this is the main technical challenge.

We cannot open the capsule to verify its contents because we don't want to reveal the contents. Let's now discuss our approach to verifiability, and then some discuss why some plausible alternatives are ruled out.

Achieving verifiability. The idea is to use the procedure described in Section 4 with the secret s being the secret key *sk*. Thus, a way must be found to make "certifiable claws." We split *sk* into two pieces s_0, s_1 to which is also associated some public object h. The properties are that (1) Given s_0, s_1 one can find *sk*; (2) Given one of s_0 or s_1 but not both, it is infeasible to find *sk*; (3) Given s_i and h, pk it is possible to be assured that s_i is half of a valid claw. Now, instead of encapsulating the secret key, we encapsulate one of the claws. The resulting capsule, and the other claw, are shared amongst the trustees, and then a cut and choose is used to ensure that all this was properly done. This builds a VCTC for the key *sk*.

We apply this paradigm with the Diffie-Hellman system and the RSA system, using in each case specific, efficient ways to make certifiable claws that depend on the algebraic structure of the cryptosystem in question. We then propose ways to make certifiable claws for any cryptosystem and get a general solution. Before heading into the technical part, though, let's discuss some of the considerations underlying this approach and ruling out others.

Efficiency considerations. The task is to verifiably escrow the given key *sk* of the given cryptosystem. (Example, the prime factors of the modulus in the RSA cryptosystem.) That is, the job is not to show the existence of a VTDKE for some crypto-system, but rather for the given one. We do *not* allow a modification of the cryptosystem. This is crucial because certain cryptosystems like RSA and Diffie-Hellman are already deployed and part of the infrastructure, and we want to escrow the corresponding keys. In particular, there should be no loss of efficiency to the underlying usage of the crypto-system as the results of escrow: Encryption and decryption times should not be impacted.

We stress this because simple solutions, based on generic cut and choose, come to mind if one is willing to allow a "composite" cryptosystem with many keys. But we don't permit such a system.

Since our solutions will in fact work for any crypto-system, we achieve the above.

The VTDKE protocols must be as efficient as possible. (Yet this is less important than keeping the crypto-system intact, because the VTDKE protocols is run only once to set up the keys, but the crypto-system is used every day by users across the net.) Our protocols meet this goal quite well.

We could treat the statement that the content of capsule C is the secret key as an NP statement and providing a zero-knowledge proof using the general tools for proving NP statements. However this will result in an extremely inefficient protocol. (This is not so much due to the inefficiency of the actual protocols, where work on efficiency has been bringing lots of improvements. Rather the issue is that translating the NP statement to a SAT formula via Cook's theorem incurs an enormous blowup so the ZK protocols is applied to a very large instance.) So we want a direct solution.

In Section 6 we provide the solution for the special case of the DH system, Section 7 we provide it for RSA and then in Section 8 for an arbitrary cryptosystem. All our solutions follow the VEKE paradighm outlined above.

6 A VTDKE Scheme for the DH Crypto-system

THE DH SYSTEM. Recall that in the Diffie-Hellman cryptosystem, a user A will have a 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 and $S_A \in Z_{p-1}$. 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 $g^{S_A S_B}$, 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. We now describe how to achieve VTDKE for this system.

SETUP AND CHOICE OF KEYS. A prime p and a generator g of Z_p^* are assumed fixed and known to all parties. We let k_1 be the length of p; this should be at least 512 bits, preferably 1024. We let $\log_g(\cdot)$ denote the logarithm in base g. (That is, $\log_g(b) = a$ iff $g^a = b$.) These system wide constants can be chosen a priori by some center and published.

A (T, S)-secure encapsulation scheme $(\mathit{Encap}, \mathit{Decap})$ is fixed. The security parameter k_2 is chosen so that the desired delay time for our user in our escrow scheme, i.e. $\mathrm{Delay}(k_2)$, is equal to $T(k_2)$. Letting $\mathit{Kl}(\cdot), \mathit{Il}(\cdot), \mathit{Ol}(\cdot)$ denote the key length, input length and output length parameters of the encapsulation scheme, it is also assumed that $\mathit{Il}(k_2) \geq k_1$.

The user has a secret key s which is uniformly distributed in Z_{p-1}, and $P = g^s$ is the corresponding public key.[3]

THE VTDKE PROTOCOL. The protocol that is now executed is described in Figure 1. It consists of m independent executions of a basic protocol which we now discuss.

[3] Think of these as having been chosen by the user. As we have already indicated, however, this is a simplification: in reality some joint protocol is executed to yield these keys.

Parameters and ingredients— Number of trustees n (say $n = 5$); Assumed bound t on number of bad trustees (say $t = 3$); Prime p and generator g of Z_p^*; Length k_1 of p; Security parameter k_2 governing desired time delay; Encapsulation scheme $(\mathit{Encap}, \mathit{Decap})$ with key length $\mathit{Kl}(\cdot)$, input length $\mathit{Il}(\cdot) \geq k_1$ and output length $\mathit{Ol}(\cdot)$; Field F with $\lg(|F|) \geq \max(\mathit{Ol}(k_2), n)$.

User keys and their components— Secret key $s \in Z_{p-1}$ of user; Public key $P = g^s \in Z_p^*$ of user.

Protocol: Repeat the following protocol, independently, $m = \lceil 100n/(n-t) \rceil$ times—

1. Claw generation and encapsulation: User chooses $s_0 \in Z_{p-1}$ at random and lets $s_1 = s_0 - s$. He then chooses a random $\mathit{Kl}(k_2)$ bit key K for the encapsulation scheme and encapsulates s_1 via $C = \mathit{Encap}(1^{k_2}, K, s_1)$. He lets $h = g^{s_0}$ and sends h to the trustees.
2. Sharing: User sets $f_0 = C$. Next he picks at random $f_1, \ldots, f_t$ from the field F and creates the polynomial
$$f(x) \;=\; f_0 + f_1 x + \cdots + f_t x^t \in F[x] \;. \qquad (1)$$
For $j = 1, \ldots, n$ he sends $f(j)$ to Trustee_j over a private channel. Let β_j denote the value received by Trustee_j.
3. Challenge: The trustees send the user a random bit c.
4. Response: The user responds according to the value of c:
   - If $c = 0$ then the user sends s_0 to each trustee and the trustee checks that $h = g^{s_0}$.
   - If $c = 1$ then the user sends $f_0, \ldots, f_t$ to the trustees. He also sends K and s_1. Trustee_j now has the polynomial $f(x)$ via Equation 1, and checks that $\beta_j = f(j)$. Each trustee also checks that $|K| = \mathit{Kl}(k_2)$ and $Pg^{s_1} = h$ and $\mathit{Encap}(1^{k_2}, K, s_1) = f_0$.

Figure 1: *The VTDKE Protocol for the DH system. See text for details.*

The user first chooses s_0 at random and lets $s_1 = s_0 - x$. He also lets $h = g^{s_0}$. Notice this means that $g^{s_0} = h = Pg^{s_1}$. This makes something we call a *certifiable claw.* If someone has both s_0 and s_1 then they can recover $x = s_0 - s_1$. If someone has only one of s_0 or s_1 then they have no information about x since each of s_0 and s_1, taken alone, is random. But if someone has only one of s_0 or s_1 then can check that they have a half of a valid claw: If s_0, check that $g^{s_0} = h$ and if s_1 check that $Pg^{s_1} = h$. The user sends (broadcasts) h to the trustees.

The user doesn't encapsulate the secret key. Rather, he encapsulates one of the claws (but not the other), and shares the encapsulated claw and its un-encapsulated sibling. Then he "opens" one or the other sharings according to a challenge from the trustees, and the latter verify the correctness of the opened information via the claw properties and the capsule properties.

Specifically, a random $\mathit{Kl}(k_2)$ bit key K is chosen for encapsulation, and the capsule $C = \mathit{Encap}(1^{k_2}K, s_1)$ is computed. The capsule C is *not* provided to the trustees; were we to do this, early recovery would be possible. Instead, C (not s_1!) is shared amongst the trustees via Shamir's secret sharing scheme [Sh1]: C is made the constant term of an otherwise random degree t polynomial $f(x)$, and $f(j)$ is sent privately to Trustee_j. (We work here over a finite field F chosen so that $\lg(|F|)$ exceeds $\max(|C|, n)$.) Now the trustees (jointly) issue a challenge which is a bit c. (Recall we are executing the basic protocol m times. The easiest way to do the challenges is that each trustee is responsible for issuing some number, say $100/(n-t)$, of them.) If $c=0$ then the user reveals s_0 and the trustees check that it is a proper left-half of a claw by checking that $h=g^{s_0}$. If $c=1$ the trustees want to check not only the claw but also that the encapsulation was properly done. The user opens f (opening a polynomial means the user broadcasts all the coefficients and the trustees check their shares, provided above, were correct). This reveals C (the constant term of the polynomial f). He also provides K and s_1. A trustee can check that s_1 was a correct right-half of the claw by checking that $Pg^{s_1}=h$. A trustee can recompute $Encap(1^{k_2}, K, s_1)$ and check that he does indeed get back C, which means the encapsulation was proper. If all m rounds pass the user key is accepted.

In discussing the protocol, we will super-script the quantities by an index l ($1 \leq l \leq m$) which indicates the iteration.

Recovery. Let $L = \{\, l \in [m] \;:\; c^l = 0 \,\}$. For each $l \in L$ the trustees already have the value s_0^l satisfying $g^{s_0^l} = h^l$, because this was revealed in the challenge phase. Now consider the following recovery procedure $\mathsf{Recover}(l)$. The trustees use their shares of f^l to recover f_0^l which is the capsule C^l. They now run the $T(k_2)$ time recovery procedure $Decap(1^{k_2}, C^l)$ to get s_1^l, and output $s_0^l - s_1^l$ as the key s of the user.

However, the above will not work for all $l \in L$, because there may be some chance a cheating user was lucky in the challenge phase. But what we will be guaranteed, from the VTDKE protocol, is that it will work for most $l \in L$. So, the trustees pick a random $l \in L$ and run $\mathsf{Recover}(l)$. If it doesn't work, they pick another random l and try again. They may have to do this a few times, but not too often. Specifically, with the parameters as we have set them, it is possible to show that the probability that the trustees have to run the procedure twice is at most 0.02, and the probability that they have to run it three times is at most 0.006. (So 98% of the time it takes $T(k_2)$ time to recover s; however 2% of the time it may take twice as long, etc.) Thus for all practical purposes, the recovery procedure is run just once, so the trustees compute for essentially $T(k_2)$ time.

Correctness and security. We now briefly argue that the above VTDKE protocol meets the conditions outlined in Section 2, namely—

Theorem 6.1 *The protocol of Section 6 constitutes a verifiable time delayed key escrow scheme for the Diffie-Hellman cryptosystem under the assumption that the discrete logarithm problem is hard.*

Let us now sketch the proof. The correct distribution of keys is true by assumption on the way they are chosen. The main technical claim to see that other requirements are satisfied is that the protocol is zero-knowledge (under the assumption that the discrete logarithm problem is hard). Intuitively, because in a given iteration we provide information only about one half of the claw, and this is a random string, independent of s. (The information about the other half is hidden as long as the trustees cannot compute discrete logarithms.) This implies that s is not revealed, which means the security of the cryptosystem requirement is satisfied. We have already discussed why the recovery with some effort requirement is met. That there can be no recovery without effort, and in particular no early recovery, is what we argue next.

Recall that to recover the trustees need to decapsulate a capsule C corresponding to an iteration of the protocol with challenge $c=0$. In this case, to any subset of the trustees of size at most t, the cipher-text C is information-theoretically hidden, by the properties of the secret sharing, and s_0 alone gives them no information about s or s_1. So they have no a priori advantage in decapsulation. Once they have a capsule, that the delay corresponds to the security of the time capsule itself.

Now we discuss verifiability. The trustees are assured recovery by $t+1$ of them is possible and won't take more than the prescribed time because the cut and choose guarantees them that most un-opened polynomials do have degree t and do contain a properly encapsulated s_1. The cut and choose also guarantees that most of the claws are proper.

IMPLEMENTATION. As written the protocol requires several rounds of interaction. The analysis applies to this protocol. As we noted in Section 4, however, in practice interaction can be eliminated by specifying the challenges as a hash of other quantities, a heuristic but seemingly sound approach justified in a random oracle model [BeRo]. This is suggested for an implementation. The same applies to the following protocols and we won't mention it again.

7 A VTDKE scheme for RSA

Many popular cryptosystems are based on factoring. Specifically, in these systems, the public key of the user is a composite modulus N product of two primes, and the secret key of the user is the prime factorization of N [RSA, BlGo, BeRo]. It is important to be able to accomplish VTDKE for such systems as well.

In the solution for DH provided in Section 6 we seemed to use the properties of the discrete logarithm function quite strongly in the way we generated s_0 and s_1. So it may not be clear a priori how we can get a VTDKE system for RSA. However, there is in fact a very general technique underlying the protocol in Section 6, based on "splitting" the secret key into "certifiable claws." Such a split can be made for RSA too. We now describe this and the resulting protocol and then, in Section 8, generalize this to arbitrary cryptosystems.

First, some notation. Recall that $J_N(\cdot)$ is the Jacobi symbol. We let $Z_N^{+1} \subseteq Z_N^*$ be the Jacobi symbol $+1$ elements, and $Z_N^{-1} \subseteq Z_N^*$ the Jacobi symbol -1 elements. The number N is the product of two primes, p and q.

The idea is to let s_0 and s_1 be square roots, of opposite Jacobi symbol, of a random square h. Indeed, we can see that given both s_0 and s_1 we can recover the prime factors of N. On the other hand, each alone gives no information, and also given s_i one can check that $s_i^2 = h \bmod N$. Then we would use the same cut and choose mechanism as before, each time opening and verifying one of the "claws." The protocol is in Figure 2. In summary:

Theorem 7.1 *The protocol of Figure 2 constitutes a verifiable time delayed key escrow scheme for any cryptosystem in which the public key is a modulus product of two primes and the secret key is the primes, under the assumption that factoring is hard.*

Note the protocol assumes that the given N is really a product of two primes. Guaranteeing this is not the job of our protocol. Rather it should be done by the protocol which is run between user and trustees prior to the beginning of the escrow process to set up keys for the user; as we have mentioned before, such a protocol is necessary anyway to prevent subliminal channels. Several protocols to guarantee that N is a product of two primes exist in the literature, so we won't get into this.

8 A general transformation

We generalize the protocols of Section 6 and Section 7 to work with any cryptosystem for which we can define a way to split the secret key via "certifiable claws." (This notion of claws is weaker than the notion of a claw-free pair of permutations [GMRi].) Certifiable claws can actually be defined for any cryptosystem so the method is general, but the efficiency depends on how we define

Parameters and ingredients— Number of trustees n (say $n = 5$); Assumed bound t on number of bad trustees (say $t = 3$); Security parameter k_1 of cryptosystem; Security parameter k_2 governing desired time delay; Encapsulation scheme (*Encap*, *Decap*) with key length $Kl(\cdot)$, input length $Il(\cdot) \geq k_1$ and output length $Ol(\cdot)$; Field F with $\lg(|F|) \geq \max(Ol(k_2), n)$.

User keys and their components— Secret key p, q of user; Public key $N = pq$ of user, of length k_1. These keys are guaranteed to be of the right form and distribution.

Protocol: Repeat the following protocol, independently, $m = \lceil 100n/(n-t) \rceil$ times—

1. Claw generation and encapsulation: User picks $s_0 \in Z_N^{+1}$ at random and lets $h = s_0^2 \bmod N$. He then computes a random Jacobi Symbol -1 square root $s_1 \in Z_N^{-1}$ of h. He then chooses a random $Kl(k_2)$ bit key K for the encapsulation scheme and encapsulates s_1 via $C = Encap(1^{k_2}, K, s_1)$. He sends h to the trustees.
2. Sharing: User sets $f_0 = C$. Next he picks at random $f_1, \ldots, f_t$ from the field F and creates the polynomial
$$f(x) \;=\; f_0 + f_1 x + \cdots + f_t x^t \in F[x] \; . \tag{2}$$
For $j = 1, \ldots, n$ he sends $f(j)$ to Trustee$_j$ over a private channel. Let β_j denote the value received by Trustee$_j$.
3. Challenge: The trustees send the user a random bit c.
4. Response: The user responds according to the value of c:
   - If $c = 0$ then the user sends s_0 to each trustee and the trustee checks that $s_0^2 = h \bmod N$ and $J_N(s_0) = +1$.
   - If $c = 1$ then the user sends $f_0, \ldots, f_t$ to the trustees. He also sends K and s_1. Trustee$_j$ now has the polynomial $f(x)$ via Equation 2, and checks that $\beta_j = f(j)$. Each trustee also checks that $|K| = Kl(k_2)$ and $Encap(1^{k_2}, K, s_1) = f_0$ and $s_1^2 = h \bmod N$ and $J_N(s_1) = -1$.

Figure 2: *The VTDKE Protocol for RSA.*

the claws, and for specific algebraic systems it is best to search for specific claws that lead to an efficient construction.

Let us now briefly discuss how the generalized protocol goes. We will be quite informal.

Assume that we have a cryptosystem with the following properties. Let *sk* be the secret key and *pk* the public key of the user. Then there is a way to (probabilistically) generate from *sk* a triple (s_0, s_1, h) such that:

- Given *sk* it is easy to generate pairs triples (s_0, s_1, h)
- Either member s_i of the pair, taken individually, reveals no information about *sk*
- Given both members s_0, s_1 of a pair, *sk* is easy to compute.
- Given s_i and h it is easy to check that there is a s_{1-i} such that (s_0, s_1, h) is a valid triple
- It is easy to generate (s_i, h), one member of a pair and the auxiliary information, without even knowing *sk*, so that they are distributed as though they were part of a triple.

We call (s_0, s_1) a claw. For example for the DH system we chose s_0, s_1 so that $s_0 - s_1$ is the secret

Parameters and ingredients— Number of trustees n (say $n = 5$); Assumed bound t on number of bad trustees (say $t = 3$); Security parameter k_1 of cryptosystem; Security parameter k_2 governing desired time delay; Encapsulation scheme $(\mathit{Encap}, \mathit{Decap})$ with key length $\mathit{Kl}(\cdot)$, input length $\mathit{Il}(\cdot) \geq k_1$ and output length $\mathit{Ol}(\cdot)$; Field F with $\lg(|F|) \geq \max(\mathit{Ol}(k_2), n)$.

User keys and their components— Secret key sk of user; Public key pk of user. These keys are guaranteed to be of the right form and distribution.

Protocol: Repeat the following protocol, independently, $m = \lceil 100n/(n-t) \rceil$ times—

1. Claw generation and encapsulation: User splits sk into (s_0, s_1, h) according to the splitting property of the cryptosystem. He then chooses a random $\mathit{Kl}(k_2)$ bit key K for the encapsulation scheme and encapsulates s_1 via $C = \mathit{Encap}(1^{k_2}, K, s_1)$. He sends h to the trustees.
2. Sharing: User sets $f_0 = C$. Next he picks at random $f_1, \ldots, f_t$ from the field F and creates the polynomial
$$f(x) \;=\; f_0 + f_1 x + \cdots + f_t x^t \in F[x]\ . \tag{3}$$
For $j = 1, \ldots, n$ he sends $f(j)$ to Trustee_j over a private channel. Let β_j denote the value received by Trustee_j.
3. Challenge: The trustees send the user a random bit c.
4. Response: The user responds according to the value of c:
   - If $c = 0$ then the user sends s_0 to each trustee and the trustee checks that s_0 is valid with respect to h
   - If $c = 1$ then the user sends $f_0, \ldots, f_t$ to the trustees. He also sends K and s_1. Trustee_j now has the polynomial $f(x)$ via Equation 3, and checks that $\beta_j = f(j)$. Each trustee also checks that $|K| = \mathit{Kl}(k_2)$ and $\mathit{Encap}(1^{k_2}, K, s_1) = f_0$ and s_1 is valid with respect to h.

Figure 3: *The VTDKE Protocol for a general cryptosystem.*

key s and $h = g^{s_0}$. The verifiability is that given s_0 we can check $g^{s_0} = h$ and given s_1 we can check that $Pg^{s_1} = h$ where $P = g^s$ is the public key. Or, for the RSA system, h is a random square, s_0 is a Jacobi symbol $+1$ square root of h, and s_1 is a Jacobi symbol -1 square root of h. The verifiability is that given s_i and h we can check that $s_i^2 = h \bmod N$.

There is one very simple and general way to generate claws: just let s_0 be a random string and let $s_1 = \mathit{sk} \oplus s_0$. Let $h = (h_0, h_1)$ where h_i is a committal to s_i. For the verifiability one can use ZK proofs to show that the committals are valid. This means any cryptosystem can be put in the above framework, but, in the most general case, it may be at a loss in efficiency, since the ZK proofs may be inefficient. Typically we look for better ways to make the split into claws, such as we found for DH and RSA.

The protocol for this general case is similar to the DH and RSA ones and is provided in Figure 3.

Notice we ask that the public and secret key of the user be guaranteed to be of the right form and distribution before our protocol begins. This should be guaranteed by the protocol between user and trustees that is run, prior to our protocol, to choose the keys.

References

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

[BeRo] 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.

[BeRo] 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.

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

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

[BlGo] 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.

[CGMA] 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.

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

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

[Des] 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.

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

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

[FrYu] 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.

[GMW] O. Goldreich, S. Micali and A. Wigderson. Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems. *JACM*, Vol. 38, No. 1, July 1991.

[GM] S. Goldwasser and S. Micali. Probabilistic encryption. *J. of Computer and System Sciences* 28, 270–299, April 1984.

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

[GMRi] S. Goldwasser, S. Micali and R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[KiLe] 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.

[LWY] 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.

[May] T. May. Timed-release crypto. `http://www.hks.net/cpunks/cpunks-0/1460.html`.

[Mi1] 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.

[Mi2] S. Micali. Guaranteed partial key escrow. MIT/LCS TM–537, September 1995.

[MiSh] S. Micali and A. Shamir. Partial key escrow.

[Pe1] T. Pederson. 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.

[Ri1] R. Rivest. The RC5 encryption algorithm. Manuscript.

[Ri2] R. Rivest. Multi-grade cryptography. Manuscript.

[RSW] R. Rivest, A. Shamir and D. Wagner. Time-lock puzzles and timed-release crypto. Manuscript available at `http://theory.lcs.mit.edu:80/~rivest`.

[RSA] R. Rivest, A. Shamir, L. Adleman. Methods for Obtaining Digital Signatures and Public Key Cryptosystems. CACM 1978, vol 21.

[Sh1] A. Shamir. How to share a secret. *CACM*, Vol. 22, No. 11, 1979.

[Sh2] A. Shamir. Private communication made at Crypto 95. August 1995.

[VW] 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. ??, U. Maurer ed., Springer-Verlag, 1996.

A Early recovery attacks

The best way to introduce the issues is by example. We are in the DH system discussed in Section 6. We will recall what is the standard fair DH system of [Mi1]. Then let us look at some existing approaches to partiality [Mi2, MiSh]. This will enable us to illustrate the early recovery problem.

THE FAIR DH SYSTEM. In Micali's fair Diffie-Hellman cryptosystem-system [Mi1], the user, having published g^S, escrows S by simply sharing it via a verifiable secret sharing (VSS) scheme based on techniques of Feldman and Pederson [Fel, Pe1]. 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.

GUARANTEED PARTIAL KEY ESCROW SCHEME. The suggestion of [Mi2, MiSh] is to make the public key have 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 [Fel, Pe1]. (This requires publishing g^x. It is important to note that this means that $g^a = P/g^x$ also automatically becomes known.) Then they provide a ZK proof that a is indeed only 80 bits long. 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 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.

DISCUSSION. 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. In fact there is a technical challenge here.

The VCTC based methods in this paper achieve delayed recovery. A partial escrow scheme extending the above to achieve delayed recovery is also presented in [BeGw], based on the same assumption that Shank's algorithm is the best.