

Earlier version appeared as MIT Laboratory for Computer Science Technical Report 688, April 1996. Latest version available at <http://www-cse.ucsd.edu/users/mihir>.

Encapsulated Key Escrow

MIHIR BELLARE*

SHAFI GOLDWASSER†

November 1996

Abstract

The main objection to current key-escrow proposals is that they assume complete faith in the authority and its trustees. If the authority does not follow the rules, or is replaced by an un-trustworthy authority tomorrow, it can immediately recover the secret keys of all users, and embark on massive wiretapping.

We introduce a new approach to key escrow called *encapsulated key escrow* (EKE). With this approach it is computationally possible for an authority to wiretap individual users, but computationally prohibitive for the authority to launch *large scale* wiretapping. This is achieved by imposing a time delay between obtaining the escrowed information of a user and actually recovering the secret key. Furthermore, the recoverability is *verifiable* at escrow time. The approach is applicable both for session keys and for public key cryptography.

EKE is a simple general paradigm, applicable across cryptosystems and key distribution protocols, regardless of their type. It solves in one stroke the problem of imposing time delays in key escrow. In particular it yields the first time delayed key escrow system for RSA, and more efficient solutions for Diffie-Hellman than achievable by the previous approach to time delays, namely partial key escrow (PKE).

The idea behind EKE is a new cryptographic tool called a *verifiable cryptographic time capsule* (VCTC). This has broader applications to “sending information into the future.”

*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	3
1.1	Key escrow and objections to it	3
1.2	Partial key escrow and its problems	3
1.3	A new approach: Encapsulated key escrow	5
1.4	Verifiable cryptographic time capsules	6
1.5	Related Work	7
1.6	Road map	7
2	The approach	7
2.1	Cryptographic time capsules	7
2.2	Verifiable cryptographic time capsules and EKE	8
3	EKE for session keys	9
3.1	Cryptographic mechanisms used	9
3.2	Partial escrow of a DES session key	10
3.3	Design issues and goals	11
3.4	Verifiability	12
3.5	Encapsulated key escrow: Protocol EKE1	13
3.6	Split key escrow version: Protocol EKE2	14
4	EKE for public key cryptosystems	15
4.1	EKE: The basic approach	15
4.2	Certified claw generation schemes	16
4.3	Achieving verifiability	17
4.4	An EKE Scheme for the DH Cryptosystem	18
4.5	An EKE scheme for RSA	20
5	General constructions and uses of VCTCs	21
	References	22
A	Verifiable time delayed key escrow: Definition	24
B	Time Capsules: Definitions and Construction	25
C	Claims about protocols	27

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 under some circumstances should be able to gain access to communication and information of selected users.

1.1 Key escrow and objections to it

Several technical solutions have been proposed toward this problem, attempting to balance the privacy of individuals with the needs of law enforcement. 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, from which the authority can immediately can reconstruct the user's secret key. The pioneering proposals of this nature were the Clipper chip (where the government chooses the secret-keys of users and enforces use of a particular hardware and encryption algorithm), and Micali's fair public key cryptosystems [22] (where the user himself can chooses his secret and which can be applied to several public-key encryption algorithms such as RSA and Diffie-Hellman).

The objection to key-escrow, whether in its full or split form, is that 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 (as pointed out by Shamir and Simmons) 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.

1.2 Partial key escrow and its problems

PAST WORK. Shamir proposed *partial key escrow* (PKE) to address this problem [33]. His scheme is for escrow of a DES key. The idea is to escrow all but k_2 bits of the key. (He suggests $k_2 = 48$). 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 k_2 unescrowed bits of the key. This will take at most 2^{k_2} steps. Thus the authority can recover the secret keys of selected individuals, with some effort. But simultaneously recovering keys of a large number of users becomes hard as the total effort involved in key recovery grows with the number of users. Thus, a deterrent to massive, large scale wiretapping is created.

The differential-work-factor proposal of Lotus (described for example in [25]) implements this idea in the session key context. The escrow is performed by simply encrypting $56 - k_2$ bits of the DES session key under the authority public key, and attaching the resulting ciphertext as a LEAF (Law Enforcement Access Field) to the transmission between the users.

Partial key escrow for public key cryptography was investigated by Micali [23] and Bellare and Goldwasser [1]. (Shamir's work [33] and Micali's work [23] were later merged into [24]. We won't discuss the latter separately). These works introduced and addressed the issue of verifiability of the escrow in this context. (Namely the authority can verify that the user has indeed escrowed an appropriate sized piece of the secret key corresponding to her public key). They provided verifiable PKE schemes for the Diffie-Hellman (DH) cryptosystem, and also provided some new factoring based cryptosystems permitting PKE.

Most importantly, however, [1] pointed to some new security issues that arise in this context, in particular the danger of *early recovery attacks*. Roughly, the issue is about *when* the extra 2^{k_2} factor effort must be invested by the authority. If it can be done off-line before obtaining the warrant (consent of $t + 1$ trustees for the split key case) then we say early recovery is possible. This should be avoided since it effectively annuls the time delay. It is shown in [1] that the schemes of [23] suffer from early recovery attacks. The DH scheme they present prevents against these attacks. But no solution for RSA is known.

DRAWBACKS OF PKE. The goal underlying PKE —namely to provide a time delay between obtaining the escrowed information of a user and actually obtaining the secret key— is attractive. But as a means to achieve it, partial key escrow suffers from many problems. Before introducing our new approach, let us detail these problems.

What we consider the main drawback of PKE is that there is no way to apply it in general. The idea of giving away part of the secret key so that the authority must search for the rest works in the context of a DES key. But there is no natural notion of “partitioning” a secret key in general, because the key may be quite structured. In particular this is true for keys of the DH or RSA systems. (For example, in the latter case, the key is the prime factors of some number. How do you give away “part” of this? One must be careful given attacks like [29, 8]). In particular this is the main reason no PKE has been found for the RSA system.

The issue arises also for session keys. Here note DES is not the only system in use. The session key is likely to be a DH key, arising from an authenticated DH key exchange, a popular choice these days. It may also be a sequence of keys for different purposes. In all cases, PKE is in trouble.

Accordingly, PKE proposals like [23, 1] have exploited the structure of the cryptosystem-system to find ways to “break up” the secret key into two parts such that the desired properties can be guaranteed. This requires extreme care. It is hard to find ways to achieve this break up for particular systems, and even when found their quality is unclear. For example the solution for Diffie-Hellman used in [23] relied on the assumption that Shank's algorithm, which given g^a finds a in $|a|/2$ steps, is best possible. The danger of this approach was illustrated when Van Oorschot and Wiener found much better algorithms [34]. (The VPKE scheme of [1] uses the same assumption but in groups of prime order where the attacks of [34] don't apply. Nonetheless, it is a non-standard assumption to be making, and better avoided).

A second issue is that avoiding early recovery appears to be a non-trivial problem. (As illustrated by the attacks on the schemes of [23]). Although a solution for the DH cryptosystem has been found

[1], that solution is tailored to the algebra of the cryptosystem, and doesn't extend to other systems like RSA. It would be better to have a generic way to avoid early recovery attacks which did not rely on specific algebraic assumptions.

Another problem is verifiability for the session key schemes. (As indicated above, verifiability for the public key case was addressed in [23, 1]. But the issue arises for the session key case as well and needs to be addressed in this context). In the known session key schemes (Shamir [33] or differential workfactor cryptography [25]) the user may escrow junk rather than the session key. It would be nice to find some deterrent to this.

The lack of generality in PKE solutions will make it difficult to fix standards. These solutions must change depending on the kind of key being escrowed and the key exchange protocol being used (in the session key case). It would be more practical to have one generic way of achieving time delays for any cryptosystem.

1.3 A new approach: Encapsulated key escrow

We provide a new way to achieve time delays in key escrow, called encapsulated key escrow (EKE). Unlike PKE, we do not try to “partition” the secret key in any way. Instead, we “encapsulate” this key via a “verifiable cryptographic time capsule.”

EKE is a simple, general paradigm, applying across cryptosystems regardless of type or form. It solves in one stroke all the problems associated with PKE, yielding solutions across the board. For a brief discussion of how it works we refer the reader to Section 1.4 and then to the body of the paper, beginning with Section 2. Here we will summarize some of the things it achieves.

EKE applies to either session key or public key escrow. In the latter, it applies to any system, whether RSA, DH or other. In particular, it yields the first time delayed key escrow systems for RSA.

EKE yields efficient solutions. In particular the EKE scheme for DH is a factor of 15 times more efficient than the VPKE DH scheme of [1].

The EKE approach makes no assumptions about the structure of the key being escrowed. This could be a DES key or a DH key or an RSA key or a sequence of different keys: they are all treated the same. The provision of a time delay does not come from any attribute of the user secret key. It comes externally, from the process of encapsulation. And encapsulation applies to any piece of data, in particular any user key.

The main security property needed in time delayed escrow is that the authority cannot recover the key faster than the known time delay. We saw that in the PKE solutions this relied on non-standard algebraic assumptions, some of which have failed. With the EKE approach, the time delay relies on no strange algebraic assumptions. Instead standard assumptions, independent of the cryptosystem, suffice to guarantee the time delay. For example, these assumptions might pertain to DES.

One has a choice of encapsulation methods. For a given setting, we might prefer a different type of encapsulation, providing different kinds of guarantees. We thus have greater flexibility and control of the time delays.

The EKE approach automatically ensures that early recovery attacks are not possible. That is, it provides delayed recovery, surmounting the main technical difficulty in the design of time delayed escrow schemes. In particular we get the first delayed recovery scheme for RSA.

The EKE approach provides verifiability. In the public key case, the trustees can verify at escrow time that indeed they hold in their possession information that would later enable them to recover the right secret key with the right amount of effort. In the session key case, we suggest a new kind of verifiability, namely verifiability by receiver. The latter checks that the LEAF is

properly built.

Thus EKE provides a guaranteed security against massive wiretapping.

The physical security of the escrowed information, in any key escrow system is highly important. In the EKE proposal, the secret-key itself is never escrowed. Rather a time capsule containing the secret-key is escrowed. Thus, even if an adversary can break into 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.

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 Verifiable cryptographic time capsules

CTCs, VCTCs AND EKE. 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 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.

Conceptually, EKE consists of putting the secret key of the user into a verifiable cryptographic time capsule (VCTC). In the implementation, the secret key, or some function thereof, is put into a CTC, and the VCTC is a protocol between the person encapsulating information and those that should eventually recover it.

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 split 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.

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.

SENDING INFORMATION INTO FUTURE. May [21] 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

We have already discussed partial key escrow [33, 23, 1, 24].

In [28], 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 [30] proposes the concept and two implementations of time-lock puzzles, any of which can be used as a cryptographic time capsule. [30] 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

We begin by discussing the approach at an informal level in Section 2. In Section 3 we show how to do encapsulated key escrow for session keys. In Section 4 we show how to do encapsulated key escrow for public key cryptography. In Section 5 we discuss the general notion of verifiable cryptographic time capsules that underlies encapsulated key escrow.

The appendices contain the more formal parts of the paper. Definitions for verifiable time delayed key escrow are in Appendix A and definitions for cryptographic time capsules are in Appendix B.

2 The approach

Let us give a high level sketch of the verifiable cryptographic time capsule and encapsulation approach to key escrow. We begin with (plain) time capsules.

2.1 Cryptographic time capsules

Roughly a cryptographic 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. (That is, this much time suffices and is necessary).

We will let k_2 be a security parameter governing the time delay. The time delay is 2^{k_2} steps when the security parameter is k_2 .

The simplest and suggested way to construct such capsules is via DES or other block ciphers as we now describe. For concreteness we suggest the reader think in these terms while reading the paper.

ENCAPSULATION. For this example say $k_2 = 40$, meaning we want a recovery time of 2^{40} . We define a procedure *Encap* that takes a 56-bit DES key K and a data string x to produce a *time capsule* $TC = Encap_K(x)$. It works like this. Encrypt x using DES in CBC mode with key K . Let C be the resulting ciphertext. Let K_1 be the first 16 bits of K . Let $c_0 = DES_K(0^{64})$ and let $c_1 = DES_K(1^{64})$. The capsule we output is $TC = (C, c_0, c_1, K_1)$.

Now suppose x is the information to be encapsulated. We pick a random DES key K and compute $TC = Encap_K(x)$. This is our time capsule containing x . When we use this, x will be the secret key of the user, or some function thereof.

DECAPSULATION. To “open” the capsule, the opener will perform an exhaustive search for the remaining 40 bits of K . More precisely, procedure *Decap* takes $TC = (C, c_0, c_1, K_1)$ and does the following—

```

For each 40-bit string  $K_2$  do
  Set  $K = K_1 . K_2$ 
  If  $\text{DES}_K(0^{64}) = c_0$  then
    If  $\text{DES}_K(1^{64}) = c_1$  then
      DES-CBC decrypt  $C$  under  $K$  and output the result.

```

This procedure takes about 2^{39} steps on the average and 2^{40} in the worst case. The auxiliary points c_0, c_1 are provided in order to enable the opener to identify the right key K when it is found. (Providing only c_0 would still have left a reasonable probability of a “false match” so we provide c_1 as well).

PROPERTIES AND DISCUSSION. The assumption is that the exhaustive search for K_2 is the best way to open the time capsule to recover its contents.

An important feature of this simple seeming scheme is that the security is independent of the data x —it comes from the key K . Note a new key K is chosen for *each* different piece of data: we never re-use a key.

Obviously this can be generalized to other values of the security parameter k_2 than $k_2 = 40$. The time capsule may also be implemented in other ways, via other block ciphers, or even algebraic functions.

FORMALISMS. An obvious issue here is that CPU speeds vary widely, so what are we measuring when we talk about the “time” to open the capsule? 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 the above case, a DES operation would be a basic step.

In Appendix B we provide a more formal treatment of time capsules. We provide formal definitions of security as well as discuss other ways of building them. However for the purpose of the body of this paper it will ease understanding to just think in terms of the concrete example given above. We will refer to the functions *Encap* and *Decap* in what follows.

2.2 Verifiable cryptographic time capsules and EKE

As above, one user (lets call her the encapsulator) can seal some information x into a cryptographic time capsule and provide the capsule to another user. In our context, x will be the secret key of the user, or some function thereof.

However, the party receiving such a capsule has some concerns. He is in essence trusting the 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*. If the information in the capsule is supposed to be the secret key corresponding to some public key, it must be verified that it is indeed this, and not some junk. The second is *time verification*. The parties may have agreed that the capsule can be opened in 2^{k_2} steps. However the encapsulator may set the timer on the capsule too high, some infeasible amount rather than the agreed upon delay.

We want to construct a “verifiable cryptographic time capsule” (VCTC). Here 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.

In the light of the above, there is a very simple and general way to look at encapsulated key escrow. Essentially, what we will do is put the secret key s into a verifiable cryptographic time capsule.

Let us focus on the case of split key escrow for public key cryptography, which is technically the most interesting. The VCTC will be constructed as a protocol between the encapsulator and the n trustees. In this protocol, a time capsule containing some information related to the user secret key will be shared between the trustees, and a cut and choose protocol will then be used to convince them that, were they able to open this capsule, they could recover the secret key within the specified time delay.

Notice that since the time capsule is shared, until a threshold of the trustees get together, no information about it is available. This explains why early recovery is not possible: until enough trustees get together, they have *no* information about the time capsule. So there is no “pre-processing” that would help them open it quicker. Notice that the key is never directly shared or manipulated. Rather we deal with an encapsulation of some function of the key.

The paradigm for building appropriate VCTCs is quite general. We will describe it first for session key escrow and then for public key escrow.

3 EKE for session keys

The setup is that there are two users, A and B . We assume that A has B 's public key pk_B and B holds the matching secret key sk_B . When the parties want to communicate, they engage in some (authenticated) key exchange protocol which results in their sharing a session key s . Typically s is the key for some cipher, like DES. Data is then encrypted using s . The session key is discarded once the session is over.

We want mechanisms to escrow the session key in each session. This should be accomplished by adding a LEAF (Law Enforcement Access Field) flow to the parties communication. An authority can wiretap and use the LEAF to discover the session key and hence decrypt the communications.

There are many reasons for which session key escrow is more attractive than escrow of the long-lived keys based on which the session keys are derived. For example, it limits the time period of law enforcement access to data: the LEAF only gives them access to the one session, not to past or future ones. Also, it is easier and cheaper to achieve.

We are interested in “time delayed” session key escrow. The idea is that even after obtaining the LEAF, an authority must invest some feasible but non-negligible computational effort to find the session key. First we need to recall some basic cryptographic mechanisms.

3.1 Cryptographic mechanisms used

Let \mathcal{E}_X denote encryption under a public key of party X . (Parties could be A or B or a trustee T). This function takes the public encryption key of party X and a plaintext to produce a ciphertext. We recall that to be secure, the encryption function must be probabilistic [15]. It takes plaintext M and coins r to produce a ciphertext $C = \mathcal{E}_X(M; r)$. Sometimes we make r explicit like this. Other times we write $C \leftarrow \mathcal{E}_X(M)$, which is shorthand for picking a random r and letting $C = \mathcal{E}_X(M; r)$. We allow the function to take multiple plaintexts as arguments, so that sometimes we may have $C \leftarrow \mathcal{E}_X(M_1, \dots, M_m; r)$ or $C \leftarrow \mathcal{E}_X(M_1, \dots, M_m)$. This means the vector M_1, \dots, M_m is encrypted to produce a single ciphertext.

\mathcal{D}_X will represent the corresponding decryption algorithm. It takes the secret key sk_X of party X and a ciphertext C to recover the corresponding plaintext.

Public key encryption can be implemented using RSA. It is important that the scheme be secure. A suggested RSA based implementation is the OAEP scheme of [3].

We assume the encryption function can handle plaintexts of arbitrary length. (Typically this is implemented under the covers in two phases: RSA is applied to encrypt a symmetric key, and the data is encrypted under the symmetric key using a block cipher. These details won't concern us.)

3.2 Partial escrow of a DES session key

Several schemes for partial escrow of a DES key have been proposed. Let us fix for discussion the following. (See later discussion for sources and comparison).

DES PKE SCHEME. Let s be the 56-bit DES session key, chosen by A . A now performs the following steps—

- (1) Let s_0 be the first 16 bits of s and s_1 the rest.
- (2) $C_T \leftarrow \mathcal{E}_T(s_0)$; $C_B \leftarrow \mathcal{E}_B(s)$
- (3) $c_0 \leftarrow \text{DES}_s(0^{64})$; $c_1 \leftarrow \text{DES}_s(1^{64})$; $\text{LEAF} \leftarrow (C_T, c_0, c_1)$
- (4) Send LEAF , C_B to B .

Upon receipt, B applies \mathcal{D}_B to C_B to recover the session key s . Now the parties use s to DES encrypt any data they send each other during this session. (For example in CBC mode).

RECOVERY BY AUTHORITY. When the government T wants to obtain access to the data communicated between A and B , it wiretaps the line between them. It picks up the transmission LEAF ; C_B going from A to B . Let C_T, c_0, c_1 be the components of LEAF . Now T decrypts C_T by applying \mathcal{D}_T , to obtain the first 16 bits s_0 of s . To find the remaining 40 bits s_1 it performs an exhaustive search:

```
For each 40-bit string  $z$  do  
  If  $\text{DES}_{s_0.z}(0^{64}) = c_0$  then  
    If  $\text{DES}_{s_0.z}(1^{64}) = c_1$  then  
      Set  $s_1 \leftarrow z$  and  $s \leftarrow s_0.s_1$  and halt
```

Assuming DES behaves like a random cipher, the probability of a false match —meaning finding the wrong s_1 value above— is at most $2^{40} * 2^{-2*64} = 2^{-88}$, which is negligible. So the authority does find the right session key.

What is the cost of the search? In the worst case it searches through all 2^{40} keys, but on the average looks at only 2^{39} keys. The first test is always executed. The expected number of times the second test is executed is only $2^{40} * 2^{-64} = 2^{-24}$. So the overall cost is essentially 2^{39} DES computations on the average.

Having found s the authority has access to all the data communicated in the session. It obtains the ciphertexts via the wiretap and decrypts under s to recover the plaintexts.

The reason for putting c_0, c_1 into the LEAF was to enable the exhaustive search. If we had only c_0 we might get a false match (the authority ends up with the wrong session key) with probability 2^{-24} . This seemed too high, so we added c_1 to reduce the probability of a false match to a negligible amount.

SECURITY AGAINST AN EAVESDROPPER. Notice that the adversary trying to decipher the communications between the parties faces the task of finding the entire 56-bit key s : since he does not have access to the decryption key of T he cannot get s_0 from the LEAF. Thus the communications have normal DES security from the point of view of the adversary.

THE MODEL. Notice that A sends information only to B . There is no direct communication of A with the government or any agency. The “escrow” is provided by the LEAF which is just included in the message to B . To obtain the LEAF the authority will wiretap.

This is the right model for session key escrow. Any proposal which involves direct communication of A with the government will not be a practical solution for session key escrow. The processing costs will be too high for all parties concerned, but especially for the government.

SOURCES AND COMPARISON. The idea of splitting a DES key into two parts s_0, s_1 and escrowing only the first was put forth by [33, 24]. The scheme presented in the latter, however, involved A sending information directly to the authority and receiving some kind of certificate in return. This does not seem like the right model for session key escrow: as discussed above, interaction with a third party in this context is impractical.

The scheme above is essentially the “differential workfactor cryptography” scheme of Lotus Corporation, announced for example in [25]. The above scheme differs in a few details.

WHAT FOLLOWS. Now we raise several issues and concerns with regard to a scheme like the above. Finally we will present new schemes that addresses them and that we suggest replace the above.

3.3 Design issues and goals

The above scheme is very specific to DES keys. We suggest that one should instead have a single, general system that supports escrow of any kind of session key in any kind of setting. We raise and address the following issues.

PORTABILITY ACROSS KEY-EXCHANGE PROTOCOLS. The process of conveying a session key from A to B is rarely as simple as just having A send B an encrypted copy of the key. Typically, session key distribution is accomplished via an authenticated key exchange or key distribution protocol. This protocol will have several flows, possibly involving challenges and authentication, and will yield a session key in some manner. Both the form of the key and the manner the parties’ get it changes from protocol to protocol.

It is possible that the protocol involves A picking the key s and sending B an encryption $C_B \leftarrow \mathcal{E}_B(s)$ of it, with some authentication and extra flows. But nowadays other means are recommended and more popular. Most importantly, s could be the result of an authenticated Diffie-Hellman key exchange. Here s would be $g^{ab} \bmod p$ where a, b are quantities picked by A, B respectively, p is prime, and g is a generator of Z_p^* . (There are many such protocols, differing in details).

We suggest that any session key escrow mechanism should be portable across different agreement protocols. It should say how to add a LEAF to any protocol, without making assumptions about how s is derived or what form it has. The DES PKE protocol does not have such portability.

We take as a goal to achieve portability across different key agreement protocols. Our protocols make no assumptions about how the key was derived. Our protocols also preserve the security of the original key exchange with regard to the adversary.

ARBITRARY KEY HANDLING CAPABILITY. As the above indicates, the session key is not always a 56-bit DES key. Either the form or the size of the key may vary. Here are some other possibilities we should consider:

- (1) Keys for other ciphers: The simplest case that can arise is that the session key may be a key for some other cipher, for example a 128 bit key for some 128-bit key cipher. We must be able to address arbitrary key sizes in some automatic way.
- (2) Structured keys: More problematic is that, as we saw above, the session key may be a Diffie-Hellman key, which is member of the group Z_p^* for some large prime p . Or it may be a RSA

key, consisting of prime factors of some modulus. DES PKE fails to handle these, and moreover they raise difficult technical problems which partial key escrow fails to handle. The problem is that there is no natural notion of “partitioning” such keys into two parts like we partitioned a DES key. Giving away a few bits of the prime factors of a number might endanger the rest to an unknown extent, and there are attacks that can factor a number given partial information about the factors [29, 8]. What is needed is an escrow mechanism that is independent of the key structure.

- (3) More than one key: The session key may be a sequence s_1, s_2, \dots, s_m of keys for different purposes. (The individual keys may be DES keys or other kinds of keys). This is common because we often distribute different encryption keys for the two parties, or keys for authentication in addition to encryption. In such a case, treating $s_1 \dots s_m$ as a single key is a mistake. If we decide to give away all but 40 bits of this sequence, we might give away s_1 entirely. Of course we could do something which is a function of the structure of the keys, but then we lose portability. What we want is to be able to solve the problem without knowing or caring whether the session key is one key or a sequence of keys.

All these problems are addressed by our encapsulation technique which is discussed in Section 3.5 below. We move on now to security issues.

3.4 Verifiability

In addition to design issues, we raise some security issues. We suggest the need for *verifiability* and at least an option for *split key escrow*. Here we discuss the former. In Section 3.6 we discuss the latter.

THE ISSUE. In the DES PKE scheme, instead of setting C_T to an encryption (under the trustee key) of the first 16 bits of s , A could set it to the encryption of some junk, or some random 16 bit string unrelated to s . In this case the authority will, after wiretapping, invest 2^{40} steps, only to recover some junk, and it will not be able to decrypt the parties’ communications. (One might suggest this is not a problem since now the authority knows that A has something to hide. But this is not of much help. After all if the authority is wiretapping, it pretty much knows or suspects this already— it wants the communications to confirm this view and gather evidence). Thus it would be better to have some way of making sure that A really constructs a correct LEAF. We call this “verifiability.”

In the session key setting, there are two kinds of verifiability we could consider. The first is verifiability by the recipient, and the second is verifiability by an external agent.

VERIFICATION BY RECEIVER. In the model we are considering, the only flows are between A and B . No authority is directly involved to check the LEAF.

We suggest that the receiver may want to check the LEAF. In communicating with A , receiver B has the right to know whether or not A is obeying the rules. He may feel less inclined to communicate if A is not, because this will compromise B too. Thus, there should be a mechanism via which B can check that the LEAF was properly constructed. No such mechanism exists for the DES PKE scheme.

The goal of verification by receiver is to guarantee something to an honest receiver— there is nothing to prevent the parties from using a bad LEAF if they are in collaboration and have agreed to cheat. But at least honest users have the option of knowing if their partners are obeying the rules.

Another way to put it is that there is some guarantee as long as one of the parties is honest. If they collaborate, obviously all bets are off.

VERIFICATION BY EXTERNAL AGENT. Another possibility is that an external agent who wiretaps can verify that the LEAF is correct without actually obtaining the session key. This provides guarantees even if both A and B are collaborating to try to prevent the authority from gaining access to their data under any conditions.

This kind of verification may be implemented by a “gateway” which verifies each flow going through it. For example an organization might install such a gateway through which all communication between the employees of the organization and the exterior must pass.

Verification by external agent can be enabled in principle. (That is, it is theoretically possible.) But it is costly. We suggest that it is more practical to have verification by receiver alone. That is what our protocols provide.

3.5 Encapsulated key escrow: Protocol EKE1

We now present Protocol EKE1. This protocol is for verifiable, time delayed escrow of an arbitrary session key. (It assumes there is only one trustee, denoted T . This is called “full” key escrow, as opposed to “split” key escrow where there are many trustees. The latter will be discussed later). The session key to be escrowed is s . To achieve portability, we make no assumptions about how s is distributed: we assume that is taken care of by the key exchange protocol so that both parties can be assumed to already be in possession of s . We also make no assumptions about the size or structure of the key s : it may be a DES key, but also may be a Diffie-Hellman or RSA key, or a sequence of keys. The job is to construct a LEAF and a verification tag V . The former is for law enforcement and the latter is to enable the receiver to check that the LEAF is properly constructed. Let us first explain the ideas, and then summarize the protocol.

THE IDEA. The key idea to get time delayed recovery in this setting is encapsulation. We do not try to “partition” s in any way. Instead, the sender picks an encapsulation key K and encapsulates s to get a cryptographic time capsule $TC = Encap_K(s)$. (Recall that in the simplest instance this just means that $TC = (\alpha, c_0, c_1, K_1)$ where α is the DES-CBC encryption of s under K and K_1 is the first 16 bits of K and c_0, c_1 is some auxiliary information to help the authority in its exhaustive search for the missing 40 bits of K). Now, it is this time capsule that is encrypted under the authority public key to yield the LEAF $= C_T$.

To enable verification, we turn the ordinary cryptographic time capsule into a “verifiable cryptographic time capsule.” Recall that this encryption is a probabilistic function. The sender saves the coins r used to produce the ciphertext C_T . These coins, along with the encapsulation key K and the time capsule TC , are provided to the receiver. (They are provided encrypted under the receiver public key. They cannot be provided in the clear because then an eavesdropper can recover the session key). The receiver can check that the time capsule was correctly formed by computing $Encap_K(s)$ and checking that this yields TC . Now the receiver can check that the LEAF was properly formed by re-encrypting TC under T ’s public key, using coins r , and seeing that C_T is indeed the resulting ciphertext.

We now present the protocol emanating from these ideas.

PROTOCOL EKE1. Session key s is assumed to be in possession of both A and B . The sender A performs the following steps:

1. Encapsulation: A chooses a random key K for the encapsulation scheme and encapsulates s via $TC = Encap_K(s)$.
2. Escrow: She picks coins r for the encryption scheme and computes the ciphertext $C_T = \mathcal{E}_T(TC; r)$. She now forms LEAF $= C_T$.
3. Verification tag: She computes $V \leftarrow \mathcal{E}_B(K, TC, r)$.

4. Transmission: She sends LEAF, V to B . (This flow is authenticated in the context of the key exchange if necessary.)

The receiver B verifies the LEAF as follows–

1. He decrypts V to obtain $(K, TC, r) \leftarrow \mathcal{D}_B(V)$.
2. He computes $Encap_K(s)$ and checks this equals TC .
3. He checks that $\mathcal{E}_T(TC; r) = C_T$ where $C_T = \text{LEAF}$.

RECOVERY BY AUTHORITY. When the authority wiretaps it picks up $\text{LEAF} = C_T$ and decrypts it using $\mathcal{D}_T(C_T)$ to get back the time capsule TC . Now, it must “open” this capsule to get the session key. But the capsule has been constructed so that this opening takes a certain prescribed amount of time, say 2^{40} steps. The opening can be done via the *Decap* procedure associated to the time capsule. (Recall that in the simplest case of the DES-CBC time capsule, this is just an exhaustive search over a part of the DES key space). The assumption about the encapsulation is that there is no faster way.

3.6 Split key escrow version: Protocol EKE2

In the setting of the basic DES PKE scheme there is only one trustee T . Yet there has already been much opposition to the idea that a single agent can access user data. This was the reason for the introduction of split-key escrow. Here there are n trustees T_1, \dots, T_n and the user’s data should stay secure even if $t < n$ of them are corrupt. However, any $t + 1$ of them should be able to access the user’s data.

Note that the danger of having only one trustee is mitigated by the presence of the time delay: in this setting, one trustee may not be unreasonable. Still it would be advantageous to implement split-key escrow also in this setting. The LEAF, obtained via the wiretap, would contain some information for each trustee. By pooling their information, and doing an extra 2^{40} steps of work, the trustees would recover the session key s . (This requires cooperation of at least $t + 1$ trustees).

Protocol EKE1 is relatively easily extended to provide split key escrow. Let us briefly describe the idea and then summarize the resulting protocol, which we call EKE2. For simplicity we address the special case $t = n - 1$, because very simple secret sharing can be done in this case. (However it is easy to extend our protocol to arbitrary $t < n$ via standard polynomial based secret sharing.)

THE METHOD. Let \mathcal{E}_{T_i} be the public encryption key of trustee $i = 1, \dots, n$. A encapsulates s as before to get time capsule TC . Now TC is split into n pieces TC_1, \dots, TC_n . (They have the property that all n pieces are needed to reconstruct TC). The i -th piece is encrypted under the public key of the i -th trustee, and the LEAF consists of all n ciphertexts.

To enable verification, as before, the coins used in the encryptions are saved and sent (encrypted) to the recipient, along with the encapsulation key and the pieces comprising the time capsule. Given this information the receiver can check that the n ciphertexts are correctly constructed.

Now let us summarize the protocol EKE2. Notice that when $n = 1$, protocol EKE2 reduces to protocol EKE1.

PROTOCOL EKE2. Session key s is assumed to be in possession of both A and B . The sender A performs the following steps:

1. Encapsulation: A chooses a random key K for the encapsulation scheme and encapsulates s via $TC = Encap_K(s)$.
2. Splitting: A picks $TC_1, \dots, TC_n \in \{0, 1\}^{|TC|}$ at random subject to the constraint that $TC_1 \oplus \dots \oplus TC_n = TC$.

3. Escrow: She picks coins r_i for the encryption scheme and computes the ciphertext $C_i = \mathcal{E}_{T_i}(TC_i; r_i)$, for $i = 1, \dots, n$. She now forms LEAF = (C_1, \dots, C_n) .
4. Verification tag: She computes $V \leftarrow \mathcal{E}_B(K, TC_1, \dots, TC_n, r_1, \dots, r_n)$.
5. Transmission: She sends LEAF, V to B . (This flow is authenticated in the context of the key exchange if necessary.)

The receiver B verifies the LEAF as follows–

1. He decrypts V to obtain $(K, TC_1, \dots, TC_n, r_1, \dots, r_n) \leftarrow \mathcal{D}_B(V)$.
2. He computes $Encap_K(s)$ and checks this equals $TC_1 \oplus \dots \oplus TC_n$.
3. For $i = 1, \dots, n$ he checks that $\mathcal{E}_{T_i}(TC_i; r_i) = C_i$ where LEAF = (C_1, \dots, C_n) .

RECOVERY BY AUTHORITY. Upon wiretap the n ciphertexts are obtained, and C_{T_i} is taken to the i -th trustee, who can decrypt it to obtain TC_i . If the n trustees now cooperate, they can now obtain TC by adding their pieces. Now to get the session key they must open the time capsule as before, which requires running for the prescribed amount of time.

EARLY RECOVERY. The issue of early versus delayed recovery in the context of time delayed escrow was pin-pointed in [1]. This issue is about *when* the extra 2^{k^2} factor effort must be invested by the authority. If it can be done off-line before obtaining the warrant (consent of $t + 1$ trustees for the split key case) then we say early recovery is possible. This should be avoided since it effectively annuls the time delay.

In both our schemes above, the authority has no information before the wiretap. So it cannot make any off-line pre-processing effort, and must invest 2^{k^2} steps after the wiretap. So these schemes don't suffer from early recovery.

The issue goes deeper for the split key scheme EKE2. Assume $n > 1$. We might consider what happens if the authority wiretaps before getting the consent of $t + 1$ trustees. Even in this case, pre-processing will not help. Until $t + 1$ trustees yield their pieces, the authority cannot begin the 2^{k^2} factor effort to find the key. Thus delayed recovery is present in a strong sense.

4 EKE for public key cryptosystems

An alternative to escrowing session keys is for each user to escrow the secret key sk corresponding to his or her public key pk . The latter would be certified by the trustee(s) and available, along with its certificate, on a public server. When the authority wants to wiretap it obtains a warrant to recover s . Let's call this "public key escrow" as opposed to "session key escrow."

Public key escrow has been discussed in several places, most notably by Micali [22] who advocates having verifiability in this context. The relative merits of session key escrow and public key escrow have also been much discussed and we won't enter into it there. Suffice to say that there is motivation for both, and it is thus worth having mechanisms for both.

4.1 EKE: The basic approach

The first cut is like this. The user puts the secret key sk of the cryptosystem into a cryptographic time capsule TC . Namely she picks a random key K for the encapsulation procedure and sets $TC \leftarrow Encap_K(sk)$. If there is only one trustee, she gives TC to him. It takes him the allotted delay time to open the capsule and get sk .

In the split key case, TC is shared amongst the trustees via any standard secret sharing scheme. At recovery time, $t + 1$ trustees pool their shares to get TC . They can now open TC to get s , which takes the allotted delay time.

Notice that this is independent of the structure of sk . We do not care whether it is an RSA key or some other kind of key. The time delay is from the capsule, independently of sk .

Notice also that early recovery is at once avoided. (The issue arises only in the split key case). A group of at most t trustees does not even have the capsule TC . In particular they have no information about sk other than that given by its public counterpart pk . So there is nothing they can pre-compute to enable them to open TC faster, or recover s quicker, until $t + 1$ of them get together and recover TC .

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 time delay.

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. What we will do is construct a "verifiable cryptographic time capsule." Let's now discuss how.

4.2 Certified claw generation schemes

The key to obtaining verifiability for encapsulated key escrow is to exploit an ability to "split" the secret key into certain kinds of "claws." (The terminology is inspired by [17]. The notion is different, being more a form of secret sharing, but related in spirit).

CERTIFIED CLAWS. Let the public key pk be known. The property we want, informally speaking, is that there be an efficient, one out of two verifiable secret sharing mechanism for the matching secret key sk . This mechanism should be able to generate shares of the key sk for which one can verify that if one had both shares one would recover something, and this something would be exactly the matching secret key of the known public key pk . We call this a *certified claw-generation* scheme for the cryptosystem.

Definition 4.1 Formally a certified claw-generation scheme (CCGS) for some cryptosystem is defined by algorithms *Share*, *Verify*, *Reconstruct* satisfying the following correctness and security properties—

1. **Correctness:** *Share* (a probabilistic algorithm) would take pk, sk and generate shares s_0, s_1 together with some public verification information h . Having either s_0 or s_1 (but not both) yields no information about sk , but given both one can easily compute sk . (This is the standard security property of a one out of two threshold secret sharing scheme). Moreover, it will be the case that $Verify(i, s_i, h, pk) = 1$ for $i = 0, 1$.
2. **Security:** Let s_0, s_1, h be some arbitrary strings claimed to be a sharing of the matching secret key sk of pk . Then
 - 2.1. The correctness of each share can be individually checked: given i, s_i, h, P , share s_i is "OK" if $Verify(i, s_i, h, pk) = 1$.
 - 2.2. If this check passes for both $i = 0$ and $i = 1$ then we are guaranteed that given both s_0 and s_1 , one can recover a quantity sk which is indeed the matching secret counterpart of pk . Namely running $Reconstruct(s_0, s_1, h, pk)$ yields sk where sk is the secret key corresponding to pk .

The properties may hold only with high probability rather than always.

We call s_0 the left half of the claw and s_1 the right half.

CCGS FOR DH. To illustrate, here is a certified claw-generation scheme for the Diffie-Hellman cryptosystem. Let G be a group of some order q and let $g \in G$ be a generator of G . The secret key is $s \in Z_q$ and the public key is $P = g^s$. The algorithms are–

- (1) *Share* picks $s_0 \in Z_q$ at random and sets $s_1 = s_0 - s \bmod q$. It sets $h = g^{s_0}$ and outputs s_0, s_1, h .
- (2) If $i = 0$ then *Verify*(i, s_i, h, P) outputs 1 iff $g^{s_i} = h$ and if $i = 1$ it outputs 1 iff $Pg^{s_1} = h$.
- (3) *Reconstruct*(s_0, s_1, h, P) simply outputs $s = s_0 - s_1 \bmod q$.

It is easy to check this works. The main thing to note is that $P = g^s = g^{s_0 - s_1} = hg^{-s_1}$. From this all follows.

CCGS FOR RSA. More interestingly, it is possible to build a simple certified claw-generation scheme for any cryptosystem based on integer factorization, in particular RSA. Here the secret key sk consists of primes p, q , and the public key pk is $N = pq$ (together with possibly some more information, like the public RSA encryption exponent, which we ignore).

We need 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. We ask that N be chosen so that $J_N(-1) = -1$. (This is not much of a restriction and easily done). Now the algorithms are–

- (1) *Share* picks $s_0 \in Z_N^{+1}$ at random and sets $h = s_0^2 \bmod N$. It then computes a random Jacobi Symbol –1 square root $s_1 \in Z_N^{-1}$ of h . (Using p, q !)
- (2) *Verify*(i, s_i, h, P) checks that $s_i^2 \equiv h \bmod N$ and that $J_N(-1) = -1$. It also checks that $J_N(s_i) = +1$ if $i = 0$ and -1 if $i = 1$.
- (3) It is well-known that given two numbers $s_0, s_1 \in Z_N^*$ such that $s_0 \not\equiv \pm s_1 \bmod N$, one can factor N . Algorithm *Reconstruct*(s_0, s_1, h, P) runs this procedure to find p, q .

Again it is easy to check this works. Note that since $J_N(-1) = -1$ and $J_N(s_0) = +1$ and $J_N(s_1) = -1$ it must be that $s_0 \not\equiv \pm s_1 \bmod N$.

CCGS FOR AN ARBITRARY CRYPTOSYSTEM. Actually it is possible to find a CCGS for an arbitrary cryptosystem. However, this general solution will not be too efficient. Just let s_0 be a random string and let $s_1 = sk \oplus s_0$. Let $h = (h_0, h_1)$ where h_i is a committal to s_i under some bit commitment scheme. For the verifiability one can use ZK proofs to show that the committals are valid.

4.3 Achieving verifiability

Refer to Section 4.1 where we described the basic encapsulation paradigm in the absence of verifiability. To attain verifiability, we don't in fact encapsulate sk . Instead the user applies a CCGS to split the secret key sk into shares s_0, s_1 and public information h . Now, she encapsulates *only one of the shares* to get a time capsule TC . She sends the trustee(s) TC, h . A “cut-and-choose” protocol then follows to convince the trustee(s) of the validity of the sharing and the capsule. For several reasons the protocol will have to be executed several times independently. Now let us look at this in somewhat more detail.

FULL KEY ESCROW CASE. Begin with the case of a single trustee (ie. full time delayed key escrow). Let m be some security parameter, say $m = 100$. The parties repeat the following for $j = 1, \dots, m$, independently and in parallel–

1. Claw generation and encapsulation: The user applies *Share* to sk, pk to get shares $s_{0,j}, s_{1,j}$ and public information h . She picks a key K_j for the encapsulation scheme at random and lets $TC_j \leftarrow Encap_{K_j}(s_{0,j})$. She sends TC_j, h_j to the trustee.
2. Challenge: The trustee picks a random challenge $c_j \in \{0, 1\}$ and sends it to the user.
3. Response and verification: Now, depending on the challenge value–

- 3.1. If $c_j = 0$ the user “opens” the capsule TC_j by providing $K_j, s_{0,j}$. The trustee checks that K_j is of the right length and that $Encap_{K_j}(s_{0,j}) = TC_j$ and $Verify(0, s_{0,j}, h_j, pk) = 1$.
- 3.2. If $c_j = 1$ the user provides $s_{1,j}$. The trustee checks that $Verify(1, s_{1,j}, h_j, pk) = 1$.

This convinces the trustee that for “most” j the capsule TC_j is correctly formed and the sharing is correct, meaning the content of the capsule is really a left claw of the secret key.

Now, what happens at recovery time? Let us say $j \in [m]$ is *good* if TC_j is well-formed, meaning it can be opened in the prescribed time delay and contains the correct left claw $s_{0,j}$ of sk . Let us let $J = \{j \in [m] : c_j = 1\}$. The trustee chooses (at random, say) a value $j \in J$. It has TC_j, h_j from Step 1 of the protocol. In addition, since $j \in J$, it has $s_{1,j}$ from Step 3.2. Now it tries to open TC_j by running *Decap*. (This takes the prescribed 2^{k_2} steps). If it succeeds, it gets $s_{0,j}$. Now it can apply *Reconstruct*($s_{0,j}, s_{1,j}, h_j, pk$) to recover sk .

We expect this to work with high probability since “most” j are good. If it turned out that the j picked by the trustee above is not good, it picks another and tries again. A calculation shows that if $m = 100$, a random j is good with probability 0.98. So 98% of the time the trustee tries only one j value. With a small probability it may try more. The probability of having to try more than two j values is negligible.

We refer to what we constructed above as a “verifiable cryptographic time capsule.” The notion is discussed in more generality in Section 5.

SPLIT KEY ESCROW CASE. The above is easily extended to the split key case where there are n trustees T_1, \dots, T_n of whom at most t may be corrupt. As in Section 3.6 we assume for simplicity that $t = n - 1$. (The ideas extend easily to general threshold schemes).

The change to the above protocol is like this. The quantities $s_{0,j}, s_{1,j}, h_j, K_j, TC_j$ in Step 1 are computed as before, and h_1, \dots, h_m are sent (broadcast) to the trustees. However, each capsule TC_j , rather than being directly sent, is now shared amongst the n trustees, so that each trustee has a piece of it. The challenges are issues as before. In Step 3.1, the user responds as before, but to check the response the trustees first reconstruct TC_j by pooling their shares. (If the reconstruction fails they reject the key). Step 3.2 is unchanged.

Notice that we need only secret sharing, not verifiable secret sharing (VSS). If the user cheats in the sharing, it will be detected in the reconstruction in Step 3.1. Avoiding VSS makes the protocols more efficient.

WHAT FOLLOWS. In the next two sections we will apply this paradigm to the specific cases of the DH and RSA cryptosystems, making use of the CCGSs presented in Section 4.2. We will focus on the split key escrow case since for the public key setting that seems more realistic. We will evaluate the efficiency of the resulting schemes. Finally we will summarize the main properties we achieve by our new methods.

4.4 An EKE Scheme for the DH Cryptosystem

We apply the general encapsulated key escrow paradigm of Section 4.3 in conjunction with the DH CCGS of Section 4.2 to obtain an encapsulated key escrow scheme for the Diffie-Hellman (DH) cryptosystem.

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 EKE for this system.

SETUP AND CHOICE OF KEYS. A (cyclic) group G and generator g of G are fixed and assumed known to all parties. Let q be the order of G . The discrete logarithm problem in G is assumed to be hard. (The most common setting is that $G = Z_p^*$ for a prime p , but other choices are also in use. For example G may be a group of prime order, or even an elliptic curve group). 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.

An encapsulation scheme ($Encap, Decap$) is fixed. The security parameter k_2 is chosen so that the time delay of this scheme is 2^{k_2} . For concreteness, think of the scheme of Section 2.1. Recall there are n trustees T_1, \dots, T_n of whom at most t may be corrupt.

We assume the user is in possession of her secret key and its matching public key, and has already conveyed the latter to the trustees. This does not necessarily mean that the user chose her secret key on her own. In particular, as pointed out by [18], if the user chooses her keys it may be possible to set up “subliminal channels.” To avoid this, the choice of keys may be a result of a prior protocol with the trustees. Our setting is general, making no assumptions about how the keys were derived. This issue is common to all the protocols and will not be discussed again.

PROTOCOL EKE-DH. The user holds her secret key $s \in Z_{p-1}$. Its matching public key $P = g^s \in Z_p^*$ is known to user and trustees. Let m be some security parameter, say $m = 100$. The parties repeat the following for $j = 1, \dots, m$, independently and in parallel–

1. **Claw generation and encapsulation:** User chooses $s_{0,j} \in Z_q$ at random and lets $s_{1,j} = s_{0,j} - s$. She then chooses a random key K_j for the encapsulation scheme and encapsulates $s_{0,j}$ via $TC_j \leftarrow Encap_{K_j}(s_{0,j})$. She lets $h_j = g^{s_{0,j}}$ and sends h_j to the trustees.
2. **Sharing:** User picks $TC_{j,1}, \dots, TC_{j,n} \in \{0, 1\}^{|TC|}$ at random subject to the constraint that $TC_1 \oplus \dots \oplus TC_n = TC$. For $i = 1, \dots, n$ she sends $TC_{j,i}$ to T_i over a private channel.
3. **Challenge:** The trustees pick a random challenge $c_j \in \{0, 1\}$ and send it to the user.
4. **Response and verification:** Now, depending on the challenge value–
 - 4.1. If $c_j = 0$ the user “opens” the capsule TC_j by providing (broadcasting) $K_j, s_{0,j}$. The trustees first pool their shares of TC_j to reconstruct the latter. Now each trustee checks that K_j is of the right length and that $Encap_{K_j}(s_{0,j}) = TC_j$ and $g^{s_{0,j}} = h_j$.
 - 4.2. If $c_j = 1$ the user provides $s_{1,j}$. The trustee checks that $Pg^{s_{1,j}} = h_j$.

RECOVERY. As described in Section 4.3. Namely the trustees choose (at random, say) a value j such that $c_j = 1$. They have TC_j from Step 1 of the protocol. In addition, since $c_j = 1$, they have $s_{1,j}$ from Step 4.2. Now they try to open TC_j by running $Decap$. (This takes the prescribed 2^{k_2} steps). If it succeeds, they get $s_{0,j}$. Now they can get $s = s_{0,j} - s_{1,j} \bmod q$. Else they try with another j . With high probability only one j is tried.

COST. Let us assess the efficiency of Protocol EKE-DH, both for the user and then for each trustee.

First, the cost for the user. In Step 1 she performs one exponentiation. Encapsulation, done via DES or other block ciphers, has negligible cost in comparison, as does picking some n random strings, which is the cost of the sharing, so we neglect these. Finally there are n public key encryptions (one under the public key of T_i for each $i = 1, \dots, n$) for transferring the shares privately to the trustees. And all this is repeated m times. The total cost is thus m exponentiations plus mn public key encryptions.

Let us say $m = 100$ and $n = 5$. We could encrypt with RSA using a small encryption exponent (3), making the encryption cost negligible: about $2mn$ multiplications, which is about one expo-

mentation with 1024 bit numbers. The total user cost then works out to only 101 exponentiations.

A trustee performs one decryption and two exponentiations per round. Let's put the cost of decryption at an exponentiation, making for a total of $3m$ exponentiations. For $m = 100$ this is 300 exponentiations.

In comparison, the cost of the solution of [1] for comparable parameters was over 1600 exponentiations for the user and about 400 exponentiations for each trustee. We see a *considerable* reduction in the user cost: a factor of about 15!

FURTHER EFFICIENCY IMPROVEMENTS. As written the protocol requires several rounds of interaction. However, in practice interaction can be eliminated by specifying the challenges as a hash of other quantities, a heuristic but seemingly sound approach justifiable in a random oracle model [2]. This is suggested for an implementation. The same applies to the following protocols and we won't mention it again.

4.5 An EKE scheme for RSA

RSA is the most popular choice for a cryptosystem. Encryption under RSA can be done for example as specified in [3]. We present the first time delayed key escrow method for RSA which achieves delayed recovery. This is done by applying the general encapsulated key escrow paradigm of Section 4.3 in conjunction with the CCGS for factoring described in Section 4.2, to obtain an encapsulated key escrow scheme for RSA. Refer to Section 4.2 for notation.

PROTOCOL EKE-RSA. The user's secret key is the pair of primes p, q . The matching public key is the modulus $N = pq$ and possibly other information, such as the RSA encryption modulus, and is known to both user and trustees. It is assumed that $J_N(-1) = -1$. (The trustees check this). The parties repeat the following for $j = 1, \dots, m$, independently and in parallel—

1. **CLAW GENERATION AND ENCAPSULATION:** User chooses $s_{0,j} \in Z_N^{+1}$ at random and lets $h_j = s_{0,j}^2 \bmod N$. She then computes a random Jacobi Symbol -1 square root $s_{1,j} \in Z_N^{-1}$ of h_j . She then chooses a random key K_j for the encapsulation scheme and encapsulates $s_{0,j}$ via $TC_j \leftarrow \text{Encap}_{K_j}(s_{0,j})$. She sends h_j to the trustees.
2. **SHARING:** User picks $TC_{j,1}, \dots, TC_{j,n} \in \{0, 1\}^{|TC_j|}$ at random subject to the constraint that $TC_1 \oplus \dots \oplus TC_n = TC$. For $i = 1, \dots, n$ she sends $TC_{j,i}$ to T_i over a private channel.
3. **CHALLENGE:** The trustees pick a random challenge $c_j \in \{0, 1\}$ and send it to the user.
4. **Response and verification:** Now, depending on the challenge value—
 - 4.1. If $c_j = 0$ the user “opens” the capsule TC_j by providing (broadcasting) $K_j, s_{0,j}$. The trustees first pool their shares of TC_j to reconstruct the latter. Now each trustee checks that K_j is of the right length and that $\text{Encap}_{K_j}(s_{0,j}) = TC_j$ and $J_N(s_{0,j}) = +1$ and $s_{0,j}^2 = h_j \bmod N$.
 - 4.2. If $c_j = 1$ the user provides $s_{1,j}$. The trustee checks that $J_N(s_{1,j}) = -1$ and $s_{1,j}^2 = h_j \bmod N$.

RECOVERY. As described in Section 4.3. Namely the trustees choose (at random, say) a value j such that $c_j = 1$. They have TC_j from Step 1 of the protocol. In addition, since $c_j = 1$, they have $s_{1,j}$ from Step 4.2. Now they try to open TC_j by running *Decap*. (This takes the prescribed 2^{k_2} steps). If it succeeds, they get $s_{0,j}$. Now, since $s_{0,j}^2 = s_{1,j}^2 \bmod N$ and $s_{0,j} \neq \pm s_{1,j} \bmod N$, they can factor N to obtain p, q (cf. the CCGS for factoring in Section 4.2). Else they try with another j . With high probability only one j is tried.

THE FORM OF N . An implicit assumption above is that N is the product of two primes. What if the user tried to cheat by making N a product of more than two primes? There are several answers.

First, as we have indicated above, the choice of N is usually made by a protocol run between trustees and user. We could view it as the job of this protocol to make sure N is of the right form.

Second, it would not matter anyway. What the trustees can do is check that N is of the right length, say 1024 bits. If N is the product of three or more primes then the shortest has length at most 341 bits, and the elliptic curve methods [19] can be used to simply factor N in the recovery phase. Alternatively note that running the above recovery process will result in a split of N into smaller factors and these in turn can be factored. Thus in practice at least it would appear that the user has nothing to gain by making N not of the right form.

Note the restriction that $J_N(-1) = -1$ is not a problem: the trustees just check this by computing $J_N(-1)$.

COST. Let us assess the efficiency of Protocol EKE-RSA, both for the user and then for each trustee.

First, the cost for the user. The dominating cost in Step 1 is taking a square root, which costs roughly the same as an exponentiation. As before we neglect the encapsulation, sharing, and bound the encryption cost by the cost of one exponentiation, assuming small exponent RSA is used. Thus the total cost is roughly $m + 1$ exponentiations, or 101 exponentiations for $m = 100$, just like for EKE-DH.

The trustee gets off cheaper because the CCGS verifications now only involve multiplications. He performs one decryption plus negligible stuff at every round, and putting the cost of decryption at an exponentiation, this is a total of m exponentiations. For $m = 100$ this is 100 exponentiations, cheaper than EKE-DH.

COMPARISON WITH PREVIOUS WORK. Partial key escrow systems for a factoring based cryptosystem were provided in [23, 1], but were not very good.

Firstly, there was no solution for RSA itself. The solutions required one to use a modulus product of many primes, making it much larger than a standard RSA modulus, and making the cryptosystem very inefficient. Thus the first contribution above is to get a system which works for standard RSA.

Second, previous solutions suffered from early recovery [1]. Ours does not.

5 General constructions and uses of VCTCs

Underlying encapsulated key escrow is a more general notion and tool. Namely that of a verifiable cryptographic time capsule discussed in Section 2. This is a way to “send information into the future” (cf. [21, 30]). But in such a way that claims about how far into the future it is being sent, and whether it can be recovered at the claimed time, can be verified without revealing the information.

We already discussed time capsules in Section 2.1. (See Appendix B for a more general and formal treatment). Consider a user who, for some purpose, wishes to put some information s into such a capsule, with a prescribed time delay, say 2^{k_2} steps. She can encapsulate it, putting the information into a capsule via $TC \leftarrow Encap_K(x)$.

However, the party receiving such a capsule has some concerns. In the above we are in essence 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 sender and receiver may have a contract under which certain very precise information (for example the key to a bank account) is to be put in the capsule. But the encapsulator may put junk into the capsule instead of the required object. The second is *time verification*. The parties may have agreed that the capsule can be opened in 2^{k_2} steps. However the encapsulator 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. In this context we saw how to provide content verification. Alongside we had a way to provide time verification, namely to ensure that the capsule can be opened within 2^{k_2} , without in the process revealing the information or decreasing the effort that would eventually be required to open it.

These methods were quite general. What they enable is a way to get verifiable cryptographic time capsules. The method is that described in Section 4.3.

In the light of the above, there is a very simple and general way to look at encapsulated key escrow. Essentially, what we are doing is putting the secret key s into a verifiable cryptographic time capsule.

References

- [1] M. BELLARE AND S. GOLDWASSER. Verifiable partial key escrow. *Proceedings of the Fourth Annual Conference on Computer and Communications Security*, ACM, 1997. Preliminary version appeared as Technical Report CS95-447, Dept. of CS and Engineering, UCSD, October 1995.
- [2] 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.
- [3] 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.
- [4] G. BLAKLEY. Safeguarding cryptographic keys. *AFIPS Conference Proceedings*, June 1979.
- [5] M. BLAZE. Protocol failure in the escrowed encryption standard. *Proceedings of the Second Annual Conference on Computer and Communications Security*, ACM, 1994.
- [6] 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.
- [7] 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.
- [8] D. COPPERSMITH. Finding a small root of a bivariate integer equation; factoring with high bits known. *Advances in Cryptology – Eurocrypt 96 Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.
- [9] D. DENNING. To tap or not to tap. *CACM* 1993.
- [10] D. DENNING AND M. SMID. Key escrowing now. *IEEE Communications Magazine*, Sep. 1994.
- [11] 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.
- [12] W. DIFFIE AND M. HELLMAN. New directions in cryptography. *IEEE Trans. Info. Theory* IT-22, pp. 644–654, November 1976.

- [FeSh] U. FEIGE AND A. SHAMIR. Witness Indistinguishable and Witness Hiding Protocols. *Proceedings of the 22nd Annual Symposium on the Theory of Computing*, ACM, 1990.
- [13] P. FELDMAN. A practical scheme for non-interactive verifiable secret sharing. *Proceedings of the 28th Symposium on Foundations of Computer Science*, IEEE, 1987.
- [14] Y. FRANKEL AND M. YUNG. Escrow encryption systems revisited: attacks, analysis and designs. *Advances in Cryptology – Crypto 95 Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.
- [15] S. GOLDWASSER AND S. MICALI. Probabilistic encryption. *J. of Computer and System Sciences*, Vol. 28, pp. 270–299, April 1984.
- [16] S. GOLDWASSER, S. MICALI, AND C. RACKOFF. The knowledge complexity of interactive proofs. *SIAM J. Comput.* Vol. 18, No. 1, 186–208, February 1989.
- [17] S. GOLDWASSER, S. MICALI AND R. RIVEST, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing* Vol. 17, No. 2, pp. 281–308, April 1988.
- [18] 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.
- [19] H. LENSTRA. Factoring integers with elliptic curves. *Annals of Math* Vol. 126, pp. 649–673, 1987.
- [20] 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.
- [21] T. MAY. Timed-release crypto. <http://www.hks.net/cpunks/cpunks-0/1460.html>.
- [22] 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.
- [23] S. MICALI. Guaranteed partial key escrow. MIT/LCS TM–537, September 1995.
- [24] S. MICALI AND A. SHAMIR. Partial key escrow. Manuscript, February 1996.
- [25] R. OZZIE. Prepared remarks. Delivered at RSA Data Security Conference, San Francisco, January 17, 1996. Available at <http://www.lotus.com/notesr4/ozzie.htm>.
- [26] 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.
- [27] R. RIVEST. The RC5 encryption algorithm. Manuscript.
- [28] R. RIVEST. Multi-grade cryptography. Manuscript.
- [29] R. RIVEST AND A. SHAMIR. Efficient factoring based on partial information. *Advances in Cryptology – Eurocrypt 85 Proceedings*, Lecture Notes in Computer Science Vol. 219, F. Pichler ed., Springer-Verlag, 1985.
- [30] R. RIVEST, A. SHAMIR AND D. WAGNER. Time-lock puzzles and timed-release crypto. Manuscript available at <http://theory.lcs.mit.edu:80/~rivest>.
- [31] 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.
- [32] A. SHAMIR. How to share a secret. *CACM*, Vol. 22, No. 11, pp 612–613, November 1979.
- [33] 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.
- [34] 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.

A Verifiable time delayed key escrow: Definition

We provide here an informal definition of the notion of verifiable time delayed key escrow. (This is the goal. Encapsulated key escrow is a means to this goal).

The parties involved are a user U and n trustees $\text{Trustee}_1, \dots, \text{Trustee}_n$, 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 [15].

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 $\text{Delay}(k_2)$.

NO RECOVERY WITHOUT EFFORT. Informally, the $\text{Delay}(k_2)$ time effort is necessary: $t + 1$ or more trustees who pool their information can't reduce below $\text{Delay}(k_2)$ 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 $\text{Delay}(k_2)$ 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 $\text{Delay}(k_2)$. 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 $\text{Delay}(k_2)$. 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 $\text{Delay}(k_2)$ 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* $\text{Delay}(k_2)$ 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 [18], it may be possible to set up “subliminal channels.” However, [18] 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 [22]. A full and formal definition of time delayed key escrow would follow and extend the definition of [18] to the time delayed setting.

B 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.

Time capsules were discussed at an intuitive level in Section 2.1. 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, one has no particular advantage in recovering the encapsulated information.

The notation here is more general than that used in the body of the paper. In particular, the $Encap$ and $Decap$ functions take more arguments.

Definition B.1 An encapsulation scheme, or cryptographic time capsule (CTC) is a pair $(Encap, Decap)$ of deterministic functions, the first being polynomial time computable. Associated parameters are

a key length $Kl(\cdot)$, an input length $Il(\cdot)$ and an output length $Ol(\cdot)$, all integer valued functions of the security parameter k . The *encapsulation function* $Encap$ takes 1^k , a $Kl(k)$ bit *key* K , chosen at random, and a $Il(k)$ -bit string (the information or plaintext) s to be encapsulated, and produces a string $C = Encap(1^k, K, s)$ which we call the *capsule*. Its length is $Ol(k)$. Given 1^k and a capsule C , the *decapsulation function* $Decap(1^k, C)$ returns the information s .

Ideally we want that de-capsulation is unique: if $C = Encap(1^k, K, s)$ for some K then $Decap(1^k, C) = s$. However, we ask less. Namely that it is computationally infeasible to find distinct pairs (K, s) and (K', s') for which $Encap(1^k, K, s) = Encap(1^k, K', s')$. (Computationally infeasible means there is no $\text{poly}(k)$ 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, $Decap(1^k, C)$ 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 [15]. 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 $C = Encap(1^k, K, m_i)$ for a randomly chosen, $Kl(k)$ bit key K ($i = 0, 1$). The *advantage* of A , denoted $Adv_A(1^k)$, 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 $\text{Time}_A(k)$ 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 B.2 Encapsulation scheme $(Encap, Decap)$ is (T, S) -secure if the decapsulation time is T and for all τ it is the case that $Adv_A(1^k) \leq S(k, \text{Time}(k))$ 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 [30] 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 [27]. 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 $Encap(1^k, (\kappa, r), s) = (F_K^*(s), r, F_K(r))$ where F_K^* denotes encryption by running F_K in CBC mode. 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 is a generalization of the DES based capsule discussed in Section 2.1.

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 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 [30] 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 [30], but the scheme they design is conjectured to be a good time lock.)

C Claims about protocols

Certain security claims can be made about our protocols. It is not hard to see that the session key escrow protocols EKE1 and EKE2 meet the above definitions. The public key escrow protocols are more difficult.

Let us first take Protocol EKE-DH of Section 4.4. If the m repetitions are done in serial (not in parallel) then the protocol 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 first discuss this claim and then turn to the case of parallel repetitions.

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 TC_j corresponding to an iteration j of the protocol with challenge $c_j = 1$. In this case, to any subset of the trustees of size at most t , the ciphertext TC_j is information-theoretically hidden, by the properties of the secret sharing, and $s_{1,j}$ alone gives them no information about s or $s_{0,j}$. 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 the secret sharing was correct and most capsules do contain a properly encapsulated left claw. The cut and choose also guarantees that most of the claws are proper.

When the rounds are executed in parallel rather than in serial, each round is still ZK, but we can't argue the entire protocol is ZK. We would like to argue it is witness hiding, but there is only one witness, so the usual witness indistinguishability methods [FeSh] fail. It may be possible to give a direct proof but we haven't done so and for the moment make no formal claims about the

proven security of the parallel version.

Similar arguments can be made about Protocol EKE-RSA of Section 4.5.