Auto-Recoverable Auto-Certifiable Cryptosystems

Adam Young* and Moti Yung**

Abstract. In this paper we solve the open problem known as the "software key escrow" problem. To this end we develop a cryptographic notion of auto-recoverable auto-certifiable cryptosystems. We first present the exact specification of the problem, based on what software key escrow can hope to achieve. Then we develop our new scheme, which is an efficient reduction to a software key escrow system from a certified public key system. Namely, our scheme is as efficient for users to use as a public key infrastructure, it does not require a tamper-resistant hardware (i.e., it can be distributed in software to users), and the scheme is shadow public key resistant (does not allow the users to publish public keys other then the ones certified). The scheme enables the efficient verification of the fact that a given user's private key is escrowed properly.

1 Introduction

We are currently at the point, due to the enormous surge of Internet use, where a large-scale Public Key Infrastructure (PKI) is about to be deployed. But, governments as well as other organizations want decryption keys to be escrowed, e.g. for law-enforcement purposes. Most if not all of the proposed key escrow schemes suffer from at least one form of drawback or another. These include the need for "tamper-resistant" devices (e.g., Clipper and Capstone), added overhead of protocol interaction between users and the escrow authorities, the need for "trusted third parties" to generate cryptographic keys and be active in the user-to-user transactions, and requring changes in protocols which are outside the cryptographic apparatus. In fact, a paper written by a number of researchers, cryptographers, and people who work in security (including pioneers of public-key cryptography, namely Diffie and Rivest) argues that "building the secure infrastructure of the breathtaking scale and complexity that would be required for such a scheme is beyond the experience and current competency of the field..." [K-S]. To this, we agree entirely. Such a result is beyond the current state of the art of the field. What is needed, we believe, is a new approach to Cryptography itself – on this point we seem to diverge from the opinions in the above study (though it is hard to argue with a non-technical policy paper). We think that from a purely research oriented perspective, once a difficult problem is observed, the challenge is to develop new ideas and constructions that solve that problem satisfactorily. This has been an ongoing venue of open cryptographic

* Currently: Columbia University.
** Currently: CertCo LLC.

research for over two decades. Here we attempt to take this step and develop what may be called a new approach. In another paper [FY97], formal arguments are presented explaining why building key escrow on top of a public-key system is a non-trivial task (even when third parties are allowed to be present); this fact motivates the new approach more formally.

We present a new notion which defines a new type of cryptographic system. It attempts to merge the functionality of a typical PKI with the functionality provided by an escrowed system. The solution is motivated by Kleptography (where keys are generated based on other public keys).

Problem Specification

The following are specifications of software escrowed PKI as can be derived from existing documents, discussions in the cryptographic community, and approaches to systems development:

1. **Software implementation:** Each and every system component does not require tamper-proof hardware.
2. **Software distribution:** The software that users employ is public (and hence is easily distributed).
3. **Key self-generation:** Users generate their own private keys independently and efficiently. The private keys (or messages encrypted by these keys) are recoverable by the escrow authorities only.
4. **Escrow authorities minimal intervention:** The escrow authorities act only at the system's set-up, and when key recovery is needed.
5. **PKI-compatible certification process:** To certify a key, a user sends one message requesting certification to the Certification Authority (CA), as in a regular public key infrastructure. This message is created by an efficient procedure, performed independently by the user alone.
6. **Certified keys are recoverable:** A user's public key is certified by a Certification Authority (CA) only if the corresponding private key is verified to be recoverable by the authorities. This verification is conducted solely from the message that forms the request for certification; the verification is successfull if and only if the key is recoverable with very high probability.
7. **PKI-compatible certificates:** A user's key in the certificate should include the same information as in a regular public key.
8. **Universal verifiability of recoverability:** Upon request, a user can present the message that forms the request for certification to any party and this party can verify that the private key is recoverable by the authorities.
9. **Efficient recovery:** The key recovery procedure is efficient (it can preferably be done by distributed parties, e.g., using threshold cryptography [FD89] and verifiable secret sharing which are known 1980's technologies).
10. **PKI-compatible user system:** Our proposed system is as easy for users as a public key cryptosystem, and can be implemented in software. Our solution therefore constitutes a reduction of a PKI with a Certification Authority [Koh78] to an escrowed public key infrastructure with the same configuration. Since our solution can be implemented securely in software, it can be

implemented and distributed in source code form, thus making it as easy to distribute and use as a public key software package (e.g., PGP).

11. **PKI-compatible software/architecture layers:** From an infrastructure and systems integration perspective, our specifications differentiate between various independent layers. The first layer consists of the escrow authorities, who act only at the time the system is established and who act only when a private key needs to be recovered. The later action is performed without interfacing with users. The second layer is the public-key infrastructure, where users and CA's generate certified keys whose corresponding private keys are private to the users. The third layer is the use of the certified public keys within communication and storage applications. In our system the third layer is related to the second layer as in a regular public key infrastructure.
12. **Communication Protocol Compatibility:** The solution should not change headers and messages outside the PKI protocols; e.g., communicating parties use existing communication protocols.
13. **Compliance assurance:** In [FY95, FY97] Frankel and Yung note that an escrow encryption scheme can always be bypassed (hardware or software). This is due to under-encrypting, over-encrypting, etc. Thus, governments cannot hope to solve misbehaviors in general. What is important is the definition given in [FY95] which says that "as long as the parties employ the mechanisms provided for confidentiality by the system, the escrow capability should be enabled". In our system, the CA ties the certification of keys to assure that secret keys are escrowed. We feel that this is a proper choice of control since in order to bypass the system, users will have to use another system or use an unauthorized modification of the system. Also, not performing the 'assurance of escrow' by the CA at the infrastructure level may cause problems in system design, as pointed out in [FY97].
14. **Security:** The public key is as secure as a key in a PKI against all parties (in our implementation we need an additional assumption but only for arguing security against the CA, whereas we are able to reduce the security of the key to a known assumption, otherwise).
15. **Shadow-public-key resistance:** Another aspect of security is that the system should not contain a subliminal channel that enables "shadow public-key" distribution (a notion due to Kilian and Leighton [KL95]). Such a property is hard to prove, but at the very least we can require that what is published in the key escrow system is the same information as what is published in a regular public key system.

Finally, we list another three requirements. (1) One is that of a relative low cost. In particular the user should have no additional cost when compared with a PKI user, the CA may have some additional cost (e.g., a moderate increase in memory and processing is tolerable, though this memory may be maintained at the escrow authorities as well in our solution) and the only real additional cost is in managing and operating the escrow authorities (which is a required cost). A typical cost of the authorities and the needs for their security should be compatible with the corresponding cost and needs of a (perhaps distributed)

CA. (2) Another property which may be required is that rather than opening keys of users, the authorities open session keys which are encrypted under the public keys of users. The session key is openable regardless of which of the two users in the session has been authorized as a target for key escrow. The notion of granularity in taking a key out of escrow was dealt with in [DDFY, FY95] and by Lenstra, Winkler, and Yacobi [LWY95]. This property is typically a function of the key of the escrow authority (but can be always achieved if the authorities, rather than a large number of users, are implemented in tamper-proof hardware). (3) Last, but not least, in a PKI setting the system's trust is the CA system; one would like that in an escrow PKI setting the trust will remain with the CA, and taking keys off escrow should be done only with the CA's collaboration.

Our System – General Structure:

Our system is an efficient mathematical reduction to a software key escrow system from a public key infrastructure (using the same protocol messaging availability, but employing different crypto-algorithms). The new notion combines, for the first time, public key cryptography with zero-knowledge proofs as part of key generation. In our implementation of the suggested notion, we describe an algorithm that when used produces an ElGamal [ElG85] public/private key pair, an escrowed encryption of the private key, and a proof that the escrow authorities can recover the private key. We call the proof, in addition to the 'encryption' of the private key, a **certificate of recoverability**. This certificate is publicly verifiable and assures that the private key is escrowed properly. In short, we describe how to construct a string which constitutes an implicit encryption of the private key x under the escrow authorities key and a non-interactive zero-knowledge (NIZK) proof that allows a prover to prove to a verifier that her private key x in $y = g^x \ mod \ p$ is the same as in the implicit encryption. Hence, a Certification Authority (CA) can insist that a user who submits her own public key for publication also submits the certificate just described. Having done so, the CA can be certain that x is escrowed properly, without ever learning x itself. We emphasize that the proofs and encryptions employed are efficient and do not contain encryptions of circuits and general proofs which employ such constructions (which are typically plausibility results rather than actual systems). Once the keys are certified by the CA, their use within the system is as in a regular PKI based on ElGamal/Diffie-Hellman keys. Key recovery is an efficient procedure between the CA and the escrow authorities, who are otherwise non-active.

For security the primary cryptographic assumption that is made in our system is that the Diffie-Hellman (DH) problem is hard. This assumption is used for security against adversaries. For the user to be secure from the CA, we require a new cryptographic assumption, which is similar to DH and combines the DH problem and the discrete logarithm problem, which we assume are hard. Note that the DH assumption is already required because the ElGamal PKCS is secure iff the DH problem is hard. Due to the non-interactive nature of the certificates of recoverability, we also require a random oracle cryptographic hash assumption (for SHA1) for the validity of the proofs within the certificate.

Given that our system complies with the specifications above, we feel that it constitutes a solution to "software key escrow". This solution is not substantialy more expensive to the user than a regular public key system, and is as easily deployable as a regular PKI as well. Thus, the cost incurred by technology providers and manufacturers should not increase if escrow is required. The only real added cost is the one associated with the escrow authorities operation (which is smoothly interfaced with the rest of the system). This added cost is a must, and is comparable with the cost of a distributed CA.

Remark: social/political aspects. In this paper we do not deal with the social and political implications of key escrow. It seems reasonable to adopt a solution like ours in organizations that need to escrow keys cheaply, so as to insure the internal integrity of operations performed in the organization.

Remark: implementation. We implemented our system using C on a Pentium 166 MHz machine using a key size of 1024 bits; it was compiled using gcc on Linux. We used the multiprecision library CryptoLib [LMS] from AT&T.

Related Work

Various tamper-resistant hardware solutions have been proposed, like the U.S. government's Clipper chip and Capstone chip. These solutions are undesirable for users since they require special hardware, and since secret unscrutinized algorithms have to be trusted. (See [YY96, YY97a, YY97b] for potential problems with such designs). Also, attacks has been found (e.g., [FY95]).

Fair Public Key Cryptosystems (FPKC) is a public key escrow system that can be implemented in software [Mi92]. The problem with FPKC's is three-fold. First, every user must split his private key and send the shares to the authorities. The authorities must then interact to insure that the key is escrowed. The system thus has more communication overhead than a typical public key system. Second, in FPKC's only the authorities are convinced that the key is escrowed (and the proof is only validated when all escrow authorities are present to perform the verification). There is no separation of layers in FPKC's since users are in direct interaction with the authorities. The third drawback is that FPKC's can be abused via the use of shadow public key cryptosystems, as shown by Kilian and Leighton [KL95]. They also show how to conduct Failsafe Key Escrow (FKE). However, FKE systems still suffer from the first two drawbacks that FPKC's suffer from. FKE can be regarded as a desirable specification and as a plausibility result (implemented very inefficiently).

A "Fraud-Detectable Alternative to Key-Escrow Proposals" based on ElGamal was described in [VT97]. This system, called binding ElGamal, allows users to send encrypted information along with a poly-sized proof that the session key that was used in the encryption can be recovered by the escrow authorities. The primary disadvantage of this alternative is that it does not permit detection of abuse by two conspiring criminals who employ the system. Another disadvantage of this alternative is that it introduces a substantial amount of space overhead per message sent. To see that two conspirators can evade detection by law-enforcement, note that user A can public key encrypt a message using user

B's public key, and then send the resulting ciphertext message using binding ElGamal. In this case, the proof simply serves to show that the escrow authorities can recover this ciphertext, and therefore prevents law-enforcement from being able to monitor the underlying communications. When this abuse is employed, fraud is not detectable, contrary to what is claimed in [VT97]:

"A special property of the binding concept is that abuse of the system is not only difficult but also detectable by any third party (e.g. network or service provider) without harming the privacy of law-abiding users."

The underlying plaintext being sent is hidden from law-enforcement because binding ElGamal in itself does not escrow B's private key (FKE systems could be used, but then binding ElGamal serves no function). The attack is not detectable by law-enforcement because binding ElGamal operates at the session level and user A is free to send anything (even 'random' looking strings) to user B. Note that the attack involves the means that are provided by the system: the public key data-base of the users and the message encryption to law enforcement. It therefore employs the available mechanisms themselves to achieve confidentiality, but does not assure decryption by the escrow authorities, which violates the definition of escrow systems.

It is necessary to carefully articulate the threat model of our proposed system in regards to criminal activity. A situation that is just as bad as shadow public key abuse is the existence of unescrowed public key databases that are open to the public. Hence, for PKI's, the framework in which our system is useful is when the CAs are trusted not to have unescrowed public key databases. CAs are the trustees of PKI and should be trusted to follow procedures. This can be accomplished by and large: A public key database is only useful when it is available to everyone. So, the act of running a 'pirate' unescrowed public key database is detectable by law-enforcement. This contrasts with the binding ElGamal solution in which collaboration by two criminals is not detectable. Once the underlying public-keys are escrowed, the only way to try to dodge escrow is via a protocol change, e.g., double encryption. Yet this constitutes "security by obscurity" and does not secure the confidentiality of criminal communications from the escrow authorities. Such dodgery constitutes using another system (which as explained above is always a threat to escrow [FY95]). One may argue that steganographic techniques could be employed to publish unescrowed keys. Yet, we argue that this reduces to the same problem for large scale systems, since the stegonographic tools must be published, and are thus available to law-enforcement as well.

A highly criticized Trusted Third Party solution [JMW96] requires the key escrow entities to get involved in session key generation, which prevents scalability; it also lacks many of the properties required.

Recently, IBM and TIS announced "key recovery systems" where keys are escrowed at the session level via information that is added to the message header. We note that this violates the separation of layers and requires changes in all communication protocols. In the case that these systems will be used in a global infrastructure, these systems call for extensive changes that are out of the domain that handles cryptographic management and operation and thus they violate the architecture layering requirement.

2 Background and Definitions

Informally, an Auto-Recoverable and Auto-Certifiable cryptosystem is a system that allows a user to generate auto-certifiable keys (keys with a proof of the method of generation) efficiently. The following is a formal definition.

Definition 1. An Auto-Recoverable and Auto-Certifiable Cryptosystem is a triple (GEN,VER,REC) (where REC may be an m-tuple (REC_1,REC_2,...,REC_m) such that:

1. GEN is a publicly known poly-time probabilistic Turing Machine that takes no input and generates the triple (K_1,K_2,P) which is left on the tape as output. Here K_2 is a randomly generated private key and K_1 is the corresponding public key. P is a poly-sized certificate that proves that K_2 is recoverable by the escrow authorities using P.
2. VER is a publicly known poly-time deterministic Turing Machine that takes (K_1,P) on its input tape and returns a boolean value. With very high probability, VER returns true iff P can be used to recover the private key K_2.
3. REC is a deterministic Turing machine with a private input. For a distributed implementation: REC_i, where $1 \leq i \leq m$ is a poly-time deterministic Turing Machine with a private input that takes P as input and returns share i of K_2 on its tape as output, assuming that K_2 was properly escrowed. (Subsets of) the Turing Machines REC_i for $1 \leq i \leq m$ can be used collaboratively to recover K_2.
4. It is intractable to recover K_2 given K_1 and P without REC (or REC_1,...,REC_m).

Next we will define informally the steps taken in a Public Key Infrastructure (PKI) and in an Auto-Recoverable Auto-Certifiable PKI. The following is the structure (protocol) of a Public Key Infrastructure:

1. CA's addresses and parameters are published and distributed.
   (a) Each user generates a public/private key pair, and submits the public key, along with an ID string, to a CA.
   (b) The CA verifies the ID string, certifies the public key (by signing it), and enters the certification in the public key database.
   (c) To send a message, a user queries the CA to obtain the public key of the recipient, and verifies the signature of the CA on the public key.
   (d) The user then encrypts the message with the recipients public key and sends the corresponding ciphertext to the recipient.
   (e) The recipient decrypts the ciphertext with his or her own private key.

The following is an Auto-Recoverable Auto-Certifiable PKI:

1. A set of system parameters are agreed upon. The escrow authorities generate an escrowing public key with corresponding private shares. The public parameters and CA's parameters are distributed (e.g., in software).

(a) Each user generates a public/private key pair, and submits the public key along with an ID string and a certificate of recoverability, to a CA.
(b) Using the escrowing public key, the CA verifies the certificate of recoverability. Provided that this verification holds, and that the ID string is valid, the CA certifies the public key (by signing it), and enters the certification in the public key database.
(c) To send a message, a user queries the CA to obtain the public key of the recipient, and verifies the signature of the CA on the public key.
(d) The user then encrypts the message with the recipients public key and sends the corresponding ciphertext to the recipient.
(e) The recipient decrypts the ciphertext with his or her own private key.

2. If a wire-tap is authorized for a given user, the escrow authorities obtain the certificate of recoverability of that user (from the CA), and recover the key or cleartext under the key.

Note that (a) through (e) above are functionally equivalent in both systems. The only difference is that in the escrow system, the CA is able to verify that the private key is recoverable by the escrow authorities. The only added items in the auto recoverable PKI to what is required for a PKI are set-up extra work in step 1 and step 2. step 2 is necessary by definition and step 1 additional work seems necessary to bind the system to the escrow authorities.

In our system, the i-th escrow authority EA_i knows only REC_i, in addition to what is publicly known. To publish a public key, user U runs GEN() and receives (K_1,K_2,P). U keeps K_2 private and sends the pair (K_1,P) to the CA. The CA then computes VER(K_1,P), and publishes a signed version of K_1 in the database of public keys iff the result is true. Otherwise, U's submission is ignored. In either case the certificate of recoverability is not published. Suppose that U's public key is accepted and K_1 appears in the database of the CA. Given P obtained from the CA, the escrow authorities can recover K_2 as follows. EA_i computes share i of K_2 by running $REC_i(P)$. The authorities then pool their shares and recover K_2.

2.1 Mathematical Preliminaries

Let Z_{2q}^* denote the multiplicative group of canonical elements relatively prime to $2q$ (we used it to call the group and its elements). Here q is a large odd prime. It is straightforward to show that Z_{2q}^* is a cyclic group (it possesses a primitive root). In fact, if s is a primitive root modulo q and if s is odd, then s is also a primitive root modulo $2q$. If s is a primitive root modulo q and s is even, then $s+q$ is a primitive root modulo $2q$. See [Ro93] for details. It can be shown that there exists a generator s for all groups Z_{2q}^*. Thus, there is a probabilistic poly-time algorithm to find a generator of Z_{2q}^*. The following first two simple claims are used to show that if discrete log problem is hard, then the discrete log problem in Z_{2q}^* is hard.

Claim 1 $(s^k \ mod \ 2q) \ mod \ q = s^k \ mod \ q$.

Claim 2 *If* $(s^{k'} \bmod 2q) = s^k \bmod q$, *then* $k' = k$.

Claim 3 *If DH mod q is hard, then DH mod 2q is hard.*

Proof. We will prove this by proving the contrapositive. Suppose we are given a box X that takes A and B and returns $s^{ab} \bmod 2q$ where $A = s^a \bmod 2q$ and $B = s^b \bmod 2q$. We need to show that we can use X to perform DH given $A' = s^{a'} \bmod q$ and $B' = s^{b'} \bmod q$. To do this, we choose $r_1, r_2 \in_R Z^*_{q-1}$ such that A'^{r_1} and B'^{r_2} are odd mod q, provided that s is odd (if s is even, we make sure these two values are even). We then compute $t = X(A'^{r_1} \bmod q, B'^{r_2} \bmod q)$. By Claim 2 it follows that $t = s^{a'b'r_1r_2} \bmod q$. We then output $t^{(r_1r_2)^{-1}} \bmod q$. Note that r_1r_2 has a unique inverse mod $q-1$ since $r_1, r_2 \in Z^*_{q-1}$. Our algorithm thus outputs $s^{a'b'} \bmod q$ as needed. QED.

From Claim 3 it follows that if the DH problem is hard, then the discrete log problem in Z^*_{2q} is hard.

Problem 1: Let $p = 2q + 1$ and let $q = 2r + 1$ where p, q, and r are prime. Find $t^k \bmod 2q$ given $s^k \bmod 2q$ and $g^{t^k} \bmod p$. Here g, s, t, and p are public. g generates Z_p, s generates Z^*_{2q}, and t generates a large subgroup of Z^*_{2q}.

We were unable to prove that Problem 1 is a cryptographicly hard problem via a reduction using a known hard problem. Thus, the difficulty of Problem 1 is a cryptographic assumption in regards to our system. Intuitively it seems that this problem should be hard, since finding t^k given s, t, and $s^k \bmod 2q$ requires solving the DH problem and $g^{t^k} \bmod p$ is a presumed one-way function of $t^k \bmod 2q$. Clearly, Problem 1 is not hard if the discrete-log problem is not hard, or if DH is not hard.

3 Our Scheme

3.1 System Setup

A large prime r is agreed upon s.t. $q = 2r + 1$ is prime and s.t. $p = 2q + 1$ is prime. We have produced such large values efficiently. A generator g is agreed upon s.t. g generates Z_p, and an odd value g_1 is agreed upon s.t. g_1 generates Z^*_{2q}. The values (p,q,r,g,g_1) are made public.

We give one example of organizing the escrow authorities; other settings of threshold schemes or even schemes where users decide on which authorities to bundle together are possible. There are m authorities. Each authority EA_i chooses $z_i \in_R Z_{2r}$. They each compute $Y_i = g_1^{z_i} \bmod 2q$. They then pool their shares Y_i and compute the product $Y = \prod_{i=1}^m Y_i \bmod 2q$. Note that $Y = g_1^z \bmod 2q$, where $z = \sum_{i=1}^m z_i \bmod 2r$. The authorities choose their z_i over again if $(g_1/Y) \bmod 2q$ is not a generator of Z^*_{2q}. Each authority EA_i keeps z_i private. The public key of the authorities is (Y,g_1,$2q$). The corresponding shared private key is z.

3.2 Key Generation

GEN uses "double decker" exponentiation and operates as follows. It chooses a value $k \in_R Z_{2r}$ and computes $C = g_1{}^k \ mod\ 2q$. GEN then solves for the user's private key x in $Y^k x = g_1{}^k \ mod\ 2q$. GEN computes the public key $y = g^x \ mod\ p$. GEN computes a portion of the certificate v to be $g^{Y^{-k}} \ mod\ p$. GEN also computes three NIZK proof transcript P_1, P_2, P_3 (which are generated by the NIZK proof systems $ZKIP_1, ZKIP_2$, and $ZKIP_2$, described below). The certificate P is the 5-tuple (C,v,P_1,P_2,P_3). GEN leaves ((y,g,p),x,P) on the output tape (note that y need not be output by the device since $y = v^C \ mod\ p$). Criterion 1 of definition 1 is therefore met. The user's public key is (y, g, p).

3.3 Public Escrow Verification

VER takes ((y,g,p),P) on its input tape and outputs a boolean value. VER verifies the following things:

1. P_1 is valid, which shows that U knows k in C
2. P_2 is valid, which shows that U knows k in v
3. P_3 is valid, which shows that U knows k in $v^C \ mod\ p$
4. verifies that $y = v^C \ mod\ p$

VER returns true iff all 4 criterion are satisfied. P_1 is essentially the same as the proof described first in [GHY85] for isomorphic functions, but we are specifically operating in Z^*_{2q}. It is easy to show that P_1 is complete, sound, perfect-zero knowledge, and that it constitutes a proof of knowledge. $ZKIP_2$, which is the basis for P_2 and P_3, will be explained in the following section.

3.4 *ZKIP*

In $ZKIP_2$, the prover wishes to interactively prove to a verifier that the prover knows k in $T = g^{s^k} \ mod\ p$. We assume that the verifier does not know $s^k \ mod\ 2q$ (and hence he doesn't know k). The values T, g, s, and p are public. The quantity g generates Z_p. The following three-pass protocol is repeated n times.

1. The prover chooses $e \in_R Z_{2r}$ and sends $I = T^{s^e} \ mod\ p$ to the verifier.
2. The verifier sends $b \in_R Z_2$ to the prover.
3. The prover sends $z = e + bk \ mod\ 2r$ to the verifier.
4. The verifier verifies that $I = (T^{1-b} g^b)^{s^z} \ mod\ p$.

The verifier accepts the proof iff step 4 passes in all n rounds of the protocol. $ZKIP_1$ is a three-pass protocol that uses values (I,b,z) which are very similar to the values (I,b,z) which are used in $ZKIP_2$. It is straightforward to show that this proof is complete and that it constitutes a proof of knowledge; it is also zero-knowledge.

We will now explain how the NIZK proofs in P are constructed. Let $e_{i,j}$ denote the prover's random choice for iteration j of proof P_i. Here $1 \leq i \leq 3$ and $1 \leq j \leq n$.

1. $P = (C,v)$
2. The prover chooses values $e_{1,1}$, $e_{1,2}$, ..., $e_{1,n}$, $e_{2,1}$, $e_{2,2}$, ..., $e_{2,n}$,$e_{3,1}$, $e_{3,2}$, ...,$e_{3,n} \in_R Z_{2r}$. Note that the e's must be in Z_{2r}, otherwise information about k may be leaked in step (8). To see this, note that e is needed to blind kb perfectly.
3. The prover computes $I_{1,j} = {g_1}^{e_{1,j}} \ mod\ 2q$, $I_{2,j} = v^{Y^{-e_{2,j}}} \ mod\ p$, and $I_{3,j} = y^{(g_1/Y)^{e_{3,j}}} \ mod\ p$ for $1 \leq j \leq n$
4. The prover includes all the values $I_{i,j}$ in P, where $1 \leq i \leq 3$ and $1 \leq j \leq n$.
5. The prover computes $rnd = H(I_{1,1}, I_{1,2}, ..., I_{1,n}, I_{2,1}, I_{2,2}, ..., I_{2,n}, I_{3,1}, I_{3,2}, ..., I_{3,n})$ where H is a cryptographic one-way function.
6. The prover gets the $3n$ values $b_{i,j}$ for $1 \leq i \leq 3$ and $1 \leq j \leq n$ from the $3n$ least significant bits of rnd. These are the challenge bits. Note that the verifier can calculate these bits given the values for I.
7. The prover computes $z_{i,j} = e_{i,j} + b_{i,j}k$ for $1 \leq i \leq 3$ and $1 \leq j \leq n$.
8. The prover includes the values $z_{i,j}$ in P, where $1 \leq i \leq 3$ and $1 \leq j \leq n$.

The verifier accepts the proof iff all $3n$ checks pass and if $y = v^C \ mod\ p$. This method of making a ZKIP non-interactive is due to Fiat and Shamir [FS86].

3.5 Key Recovery

REC_i recovers share i of the user's private key x as follows. REC_i takes C from P. It then computes share s_i to be $C^{z_i} \ mod\ 2q$, and outputs s_i on its tape. The authorities then pool their shares and each computes $Y^k = \prod_{i=1}^{m} s_i \ mod\ 2q$. From this they can each compute $x = CY^{-k} \ mod\ 2q$, which is the user's private key. Criterion 3 of definition 1 is therefore met.

The escrow authorities can recover the plaintext of users suspected of criminal activity without recovering the user's private key itself. To decrypt the ciphertext (a,b) of user U the escrow authorities proceed as follows:

1. Each of the m escrow authorities i receives C corresponding to U.
2. Escrow authority 1 computes $s_1 = a^{C^{-z_1}} \ mod\ p$.
3. Escrow authority $i+1$ computes $s_{i+1} = {s_i}^{C^{-z_{i+1}}} \ mod\ p$.
4. Escrow authority m decrypts (a,b) by computing $b/({s_m}^C) \ mod\ p$.

Our system allows for multiple CA's to be associated with the escrow authorities. Escrowing across escrow authorities domains (e.g., different countries) can be solved by the users employing the long-lived Diffie-Hellman key as their common key (which is recoverable by either country) or by bilateral agreements.

4 Security of the Trapdoor Values

From Claim 3 it follows that if the DH problem is hard, it is intractable to find z given Y. It also follows that finding x given C is intractable. We would like to show that assuming that the DH problem is hard, it is intractable to find x (and

hence k) given C and v. Note that by setting $s = g_1$ and $t = Y^{-1}$ *mod* $2q$, this is the same as Problem 1. Note that Y^{-1} does not generate the entire group Z_{2q}^* because $-z$ *mod* $2r$ is even. The reason that g^{-z} is not a generator is that we want to insure that g^{1-z} *mod* $2q$ is a generator of this group, so as to optimize the number of possible values for x. Since we were unable to prove that Problem 1 is hard, we assume that it is intractable to find x given C and v.

Claim 4 *If finding x is hard given (C,v), then finding k is hard given (C,v).*

Proof. We will prove this by proving the contrapositive. Suppose it is easy to find k given (C,v). We can then compute x in poly-time by computing $x = (gY^{-1})^k$ *mod* $2q$. QED.

Assuming that Problem 1 is hard, Claim 4 establishes that if the DH problem is hard, then it is hard to find x (and hence k) given (C,v,y).

Claim 5 *If gcd($1 - z$ mod $2r$,$2r$) = 1, then there are $2r$ different possible values for x, all of which are equally likely.*

Proof. We have that $Y^k x = {g_1}^k$ *mod* $2q$. Solving for x, we have that $x = ({g_1}^{1-z})^k$ *mod* $2q$. If $\gcd(1 - z \text{ mod } 2r, 2r) = 1$, it follows that $h = {g_1}^{1-z}$ *mod* $2q$ is a generator of Z_{2q}^*. So, $x = h^k$ *mod* $2q$. Since h is a generator and since GEN chooses k uniformly at random from Z_{2r}, it follows that x can take on any of $2r$ different values, each of which is chosen with equal probability. QED.

Since z is chosen s.t. $\gcd(1 - z \text{ mod } 2r, 2r) = 1$, it follows from Claim 5 that there are $2r$ different possible values for x, all of which are equally likely. We have therefore shown that the trapdoor values in Y, y, C, and v are secure, under the DH assumption and assuming that Problem 1 is hard. It follows that criterion 4 of definition 1 is met.

One might argue that our proposed system is risky for use because the security of the entire system rests on the difficulty of computing the discrete log of the fixed value Y, the public key of the escrow authorities. Indeed, in the setting of a national PKI, this is a valid concern. One possible approach to this problem is to split this key into, say 200 shares. Each user then chooses 100 of these shares at random, and uses the product of them as the public key of the escrow authorities. This allows the user's public key to be the same size as before, and increases the difficulty of cracking the system by two orders of magnitude. The user should indicate which 100 keys are in use in its message to the CA.

5 Security of the New Cryptosystem

Recall the four verification steps performed by VER. We need to show that nothing is leaked during verifications 1, 2, and 3. We already showed the security of the trapdoor values, so verification 4 alone does not compromise the system. Also, we showed that all three of these steps are perfect ZK proofs, and therefore

they do not leak any knowledge individually. It can be shown that together these ZK proofs do not leak knowledge. So, we can show that our cryptosystem is perfect ZK. It is also easy to show that our proof system is complete.

The remainder of this section is devoted towards proving that our cryptosystem is a sound ZK proof. That is, we wish to prove formally that it meets criterion 2 of definition 1. To do so we need to introduce the following problem instance.

$$\text{(eqn. 1)} \quad Y^{k_3}(g_1 Y^{-1})^{k_1} = {g_1}^{k_2} \ mod\ 2q$$

Problem 2: Find a k_1, k_2, and k_3, s.t. $k_2 \neq k_3$ and eqn. 1 holds, where z is even, $z \neq 1 - r \ mod\ 2r$, $Y = {g_1}^z \ mod\ 2q$, and where z is unknown.

Note that ${g_1}^{1-z} \ mod\ 2q$ is a generator of Z^*_{2q} iff z is even and $z \neq 1-r \ mod\ 2r$. Note also that $k_2 = k_3$ iff $k_1 = k_2 = k_3$.

Claim 6 *Problem 2 is solvable in poly-time iff the Discrete Log problem mod $2q$ is solvable in poly-time.*

Proof. Suppose we know a k_1, k_2, and k_3, s.t. $k_2 \neq k_3$ and eqn. 1 holds. Then from the exponents of eqn. (1) we have that,

$$zk_3 + k_1 - zk_1 = k_2 \ mod\ 2r$$

$$z(k_3 - k_1) = (k_2 - k_1) \ mod\ 2r$$

Since z and $2r$ are even, $(k_2 - k_1)$ must be even. $(k_3 - k_1)$ is either even or odd (and isn't r). If it is odd, then by Lemma 2 in [Du78], only one solution for z exists. It can easily be found. If $(k_3 - k_1)$ is even, then by Lemma 3 in [Du78], there are exactly two solutions for z. We can find the first solution by solving $((k_3 - k_1)/2)z = (k_2 - k_1)/2 \ mod\ r$. The other can easily be found. We then see which solution satisfies $Y = {g_1}^z \ mod\ 2q$. Therefore, if we know a k_1, k_2, and k_3, s.t. $k_2 \neq k_3$ and eqn. (1) holds, we can compute z easily.

Suppose we know the z in $Y = {g_1}^z \ mod\ 2q$. We then choose k_2 and k_3 randomly s.t. $k_2 \neq k_3$. Now consider the following equation where k_1 is unknown,

$$zk_3 + k_1 - zk_1 = k_2 \ mod\ 2r$$

$$k_1(1 - z) = k_2 - zk_3 \ mod\ 2r$$

Since z is even, $(1 - z)$ mod $2r$ is odd. Since $z \neq 1 - r \ mod\ 2r$, it follows that $(1 - z)$ has an inverse, so $k_1 = (k_2 - zk_3)(1 - z)^{-1} \ mod\ 2r$. Therefore, if we can compute the discrete log of $Y \ mod\ 2q$, then we can find a k_1, k_2, and k_3, s.t. $k_2 \neq k_3$ and s.t. eqn. (1) holds. QED.

Claim 7 *Assuming the difficulty of the Discrete Log problem mod $2q$, our cryptosystem is sound (i.e. with very high probability, the key is recoverable if the verification passes).*

Proof. For concreteness, suppose that GEN does not output y. Hence, suppose the verifier reconstructs y by computing $v^C \ mod \ p$. So, we need only consider the first three criterion. We need to show that if $C = {g_1}^k \ mod \ 2q$ is not the encryption of the exponent of $g^x \ mod \ p$, then the prover will fail the proof with very high probability. Assume for the sake of contradiction, that the prover can find such a private key x where C is not the encryption of x, and s.t. the prover is still able to convince the verifier of the validity of the proof. Clearly, if the prover doesn't know a k_2 s.t. $C = {g_1}^{k_2} \ mod \ 2q$, then the prover will fail criterion (1) of VER. Similarly, if the prover doesn't know a k_3 for the value $v = g^{Y^{-k_3}} \ mod \ p$, then the prover will fail criteria (2). So, the prover needs to know k_2 and k_3. To pass criteria (3), the prover must know a k_1 s.t.,

$$g^{(g_1 Y^{-1})^{k_1}} = (g^{Y^{-k_3}})^{{g_1}^{k_2}} \ mod \ p$$

It follows that the prover must know a k_1 s.t.,

$$Y^{k_3}(g_1 Y^{-1})^{k_1} = {g_1}^{k_2} \ mod \ 2q$$

Note that this is eqn. 1. Either $k_2 = k_3$ or $k_2 \neq k_3$. Suppose that $k_2 \neq k_3$. It follows that finding such a k_1 is equivalent to solving Problem 2. But, since we assumed that computing discrete logs mod 2q is hard, it follows from Claim 6 that the prover cannot find such a k_1. So, the prover must choose $k_2 = k_3$. It is easy to see from eqn. 1 that $k_2 = k_3$ iff $k_1 = k_2 = k_3$. But, this implies that C is an encryption of $(g_1 Y^{-1})^{k_1} \ mod \ 2q$. Hence, we have reached a contradiction. QED.

We've shown that if the DH problem is hard, then the Discrete log problem mod $2q$ is hard. Therefore, assuming that the DH problem is hard and that Problem 1 is hard it follows from Claim 7 that our cryptosystem is sound.

Abuse Related Issues: We only publish the user's key as in a regular public key system. In fact, we insisted that the certificate of recoverability not be published. This is to prevent the establishment of shadow public-key.

References

[DB96] D. Denning, D. Branstad. A Taxonomy for Key Escrow Encryption Systems. In volume 39, n. 3 of Communications of the ACM, 1996.

[DDFY] A. De Santis, Y. Desmedt, Y. Frankel, M. Yung. How to Share a Function Securely. In ACM Symp. on Theory of Computing, pages 522–533, 1994.

[DH76] W. Diffie, M. Hellman. New Directions in Cryptography. In volume IT-22, n. 6 of IEEE Transactions on Information Theory, pages 644–654, Nov. 1976.

[Du78] U. Dudley. Elementary Number Theory. 2nd edition, pages 36, 37, 75, 1978. W. H. Freeman and Co.

[ElG85] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In CRYPTO '84, pages 10–18.

[FD89] Y. Frankel, Y. Desmedt. Threshold Cryptosystems. In CRYPTO '89, pages 307-315.

[Fr94] J. B. Fraleigh. A First Course in Abstract Algebra. 5th edition, Theorem 1.9, page 76, 1993. Addison Wesley.

[FS86] A. Fiat, A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In CRYPTO '86, pages 186–194.

[FY95] Y. Frankel, M. Yung. Escrow Encryption Systems Visited: Attacks, Analysis and Designs. In CRYPTO '95, pages 222–235,

[FY97] Y. Frankel, M. Yung. On characterization of Escrow Encryption Schemes. In ICALP '97.

[GHY85] Z. Galil, S. Haber, M. Yung. Symmetric public-key encryption. In CRYPTO '85, pages 128–137.

[GMR85] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof-systems. In ACM Symp. Theory of Computing, 1985.

[JMW96] N. Jefferies, C. Mitchell, M. Walker. A Proposed Architecture for Trusted Third Party Services. In Cryptography: Policy and Algorithms, LNCS 1029.

[K-S] H. Abelson, R. Anderson, S. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P. Neumann, R. Rivest, J. Schiller, B. Schneier. The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption. available at http://www.crypto.com/key_study

[KL95] J. Kilian and F.T. Leighton. Fair Cryptosystems Revisited. In CRYPTO '95, pages 208–221, 1995. Springer-Verlag.

[Koh78] L. Kohnfelder. A Method for Certification. MIT Lab. for Computer Science, Cambridge Mass., May 1978.

[LMS] J. Lacy, D. Mitchell, W. Schell. CryptoLib: Cryptography in Software. AT&T Bell Laboratories, version 1.2.

[LWY95] A. Lenstra, P. Winkler, Y. Yacobi. A Key Escrow System with Warrant Bounds. In CRYPTO '95, pages 197–207, 1995.

[Mi92] S. Micali. Fair Public-Key Cryptosystems. In CRYPTO '92, pages 113–138, 1992. Springer-Verlag.

[Ro93] K. R. Rosen. Elementary Number Theory and its Applications. 3rd edition, Theorem 8.14, page 295, 1993. Addison Wesley.

[VT97] E. Verheul, H. van Tilborg. Binding ElGamal: A Fraud-Detectable Alternative to Key-Escrow Proposals. In Eurocrypt '97, pages 119–133, 1997.

[YY96] A. Young, M. Yung. The Dark Side of Black-Box Cryptography. In CRYPTO '96, pages 89–103,

[YY97a] A. Young, M. Yung. Kleptography: Using Cryptography against Cryptography. In Eurocrypt '97, pages 62–74.

[YY97b] A. Young, M. Yung. The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In CRYPTO '97, pages 264–276.