BRICS RS-03-22 Damgård & Jurik: Scalable Key-Escrow

Scalable Key-Escrow

Ivan B. Damgård
Mads J. Jurik

BRICS Report Series RS-03-22
ISSN 0909-0878 May 2003

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/03/22/`

Scalable Key-Escrow

Ivan Damgård and Mads Jurik

Aarhus University, Dept. of Computer Science, BRICS*

Abstract. We propose a cryptosystem that has an inherent key escrow mechanism. This leads us to propose a session based public verifiable key escrow system that greatly improves the amount of key material the escrow servers has to keep in order to decrypt an encryption. In our scheme the servers will only have a single secret sharing, as opposed to a single key from every escrowed player. This is done while still having the properties: 1) public verifiable: the user proves to everyone that the encryption can indeed be escrowed, and 2) no secret leakage: no matter how many decryptions a law enforcement agency is presented, it will gain no more information on the users private key, than it couldn't have calculated itself.

Keywords: verifiable, partial, key-escrow, early recovery, cryptosystem.

1 Introduction

1.1 Background

During the last decade there have been a large growth in communication over the Internet. There have also been an increased focus on privacy and sending messages encrypted. This however poses a problem for law enforcement agencies (LEAs) which have relied on their ability to make wiretaps and get search warrants to solve crimes. With encrypted communication and storage both these advantages disappear and the law enforcement agencies are unable to monitor communication.

This has lead to several Key Escrow proposals in where the persons communicating will reveal their keys (or part of these) to the LEAs. This enables the LEA to decrypt messages, but poses the problem that it can also encrypt messages that it is not supposed. Two ways have been proposed to combat this problem: 1) partial key escrow where the LEA only receives part of the key and has

* Basic Research in Computer Science (`www.brics.dk`),
funded by the Danish National Research Foundation.

to do an exhaustive search to find the rest, and 2) introducing a Key Escrow Agency (KEA) that handles the secret keys and helps the LEAs in the case of a lawful request. Partial key escrow relies on the assumption that the LEA cannot do a massive decryption of messages because of the work involved in find the last portion of the key. This however doesn't prevent a LEA to pick a few number of persons that it is not entitled to monitor and decrypt their communication. In both cases there is the problem that if the escrowed keys are reused the LEA agency will be able to keep monitoring the communication after the duration of their permission have expired. However, this can be fixed by only escrowing session keys which are typically short lived.

In [4], Shamir proposed a scheme with partial key escrow. The idea behind partial key escrow is that the user escrows (reveals) for instance 8 bit of a DES key to the Law Enforcement Agency (LEA). If the LEA at some point want to wiretap the user they will have to do an exhaustive search on the last 48 bits. This means that recovering a single key is cumbersome, but definitely possible, whereas recovering a lot of keys at the same time is hard due to the work load.

Concurrently to this result Micali proposed a similar idea of partial key escrow for public keys in [5]. The schemes [4] and [5] were later merged into a joint paper [6].

In [7] Bellare and Goldwasser proposed a verifiable partial key escrow scheme that makes it possible for the receiver to check that the sender has escrowed the correct bits and not some random garbage. This would have made it impossible for the LEA to recover the key and might implicate an innocent receiver. They also address problems from [5] with early recovery, which means that the LEA is able to do the computation before receiving the key escrow information and thus get the key quickly upon receiving the escrow information.

Mao introduced a scheme in [8] where escrowed values could be publicly verified. This has the advantage that the escrowing authorities, the senders, and other interested parties can verify that encryptions are indeed subject to escrow. This is especially useful in settings where persons can be subject to fines/jail time for communicating without escrow.

A scheme proposed by Shaoquan and Yufeng [13] used a different setup in which the LEA doesn't hold the escrow values. Instead a Key Escrow Authority (KEA) holds shares of the escrowed values and discloses these to the LEA upon request. This setup is more like the existing power structures, where the different LEAs are independent from the judiciary system. The usual way to get a search warrant, is that the LEA presents its case to a judge, which then makes the decision whether to allow the search or not. It seems logical that the same should hold for electronic information, in which case the KEA should be a separate unit under e.g. the department of justice.

In [14], Damgård and Jurik proposed a public key cryptosystem, that used a combination of the Paillier cryptosystem [9] in a generalized form [11] and the El-Gamal cryptosystem [2]. This scheme has an El-Gamal like encryption and decryption, but a part of the encryption is a normal Paillier encryption. This allows someone with knowledge of the factorization or some derived information to decrypt all messages encrypted under any key made in the El-Gamal variant.

1.2 Our Contribution

We introduce a new cryptosystem that has two kind of secret keys. First there are several normal keys as introduced in [14]. Secondly there is a global master key that is able to decrypt any message encrypted with the normal keys. This is done without revealing any information on the specific private key.

This can be used to make key escrow by having the global master key shared between the escrow servers in a threshold fashion. Note that since there is just one master key the escrow servers don't have to keep a secret sharing of the secret keys of all the different users, as opposed to all other schemes to date.

The setup used in this paper is a setup similar to [13], since we have: 1) some users sending encrypted messages to each other, 2) a Key Escrow Agency (KEA) holding the escrow key(s), and 3) the Law Enforcement Agencies (LEAs) being agencies like FBI, CIA, county sheriff department, etc.

For a LEA to decrypt a message it will ask the KEA servers to provide a decryption value. The KEA generates the random value used in the encryption and sends it privately to the LEA. The LEA can then remove the random part of then encryption and decrypt the resulting value.

The system has the added advantage over existing protocols, that the users don't have to perform an expensive key escrow protocol with the KEA (or LEA) when setting up the system. The KEA simply generates some global parameters, and all the users generate a key pair in this global setup. This allows the KEA to "decrypt" without even knowing the public key of the user.

2 Preliminaries

2.1 Model

The model consists of 3 kinds of players: 1) the users, 2) the Key Escrow Agency (KEA) servers and 3) the Law Enforcement Agencies (LEAs).

The adversary model is 2-sided. The players want to try and cheat the LEA so they're not able to decrypt their messages, and the LEAs try to decrypt messages they're not supposed to. The KEA works as a buffer between the 2 by providing decryptions to the LEA when it gets a valid request, and refuse when the LEA is trying to cheat.

The users are assumed to be able to mount several attacks, namely: 1) flooding: the user floods the channels with a lot of illegal encryptions and one legal encryption to make the LEA waste a lot of resources, trying to find the single legal encryption, 2) collude with some of the KEA servers to make the LEA unable to decrypt, and 3) decrypt messages of other users by colluding with some KEA servers.

The LEAs are unable to break the semantic security of the cryptosystem and corrupting up to t KEAs won't help because the secret is shared among these. We'll assume an even more powerful LEA adversary that can control up to t KEA servers and a number of users (informants) and has three attacks: 1) find the decryption of a ciphertext, 2) find the private key of a user, and 3) find the secret shared between the KEA servers. The third attack is the most dangerous attack, since it will enable the LEA to decrypt all messages, without help from the KEA servers at all.

To make the threshold version we'll assume, that the KEAs have a bulletin board they can access to make decryptions. This is used for distributing their decryption values during decryption.

2.2 Assumptions

Since we use the cryptosystem of [14] and we need the semantic security, the same 2 assumptions apply to this scheme:

Conjecture 1 (The Decisional Composite Residuosity Assumption). Let $\mathcal{A}$ be any probabilistic polynomial time algorithm, and assume $\mathcal{A}$ gets n, x as input. Here $n = pq$ is an admissible RSA modulus of length k bits, and x is either random in $\mathbb{Z}_{n^2}^*$ or it is a random n'th power in $\mathbb{Z}_{n^2}^*$. $\mathcal{A}$ outputs a bit b. Let $p_0(\mathcal{A}, k)$ be the probability that $b = 1$ if x is random in $\mathbb{Z}_{n^2}^*$, and $p_1(A, k)$ the probability that $b = 1$ if x is a random n'th power. Then $|p_0(\mathcal{A}, k) - p_1(\mathcal{A}, k)|$ is negligible in k.

Conjecture 2 (The Decisional Diffie-Hellman). Let $\mathcal{A}$ be any probabilistic polynomial time algorithm, and assume $\mathcal{A}$ gets $(n, g, g^a \bmod n, g^b \bmod n, y)$ as input. Here $n = pq$ is an admissible RSA modulus of length k bits and g is an element of Q_n, the group of squares in $\mathbb{Z}_n^*$. The values a and b are chosen uniformly random in $\mathbb{Z}_{\phi(n)/4}$ and the value y is either random in Q_n or satisfies $y = g^{ab} \bmod n$. $\mathcal{A}$ outputs a bit b. Let $p_0(\mathcal{A}, k)$ be the probability that $b = 1$ if y is random in Q_n, and $p_1(\mathcal{A}, k)$ the probability that $b = 1$ if $y = g^{ab} \bmod n$. Then $|p_0(\mathcal{A}, k) - p_1(\mathcal{A}, k)|$ is negligible in k.

Some of the protocols also rely on the Strong RSA assumption, namely the key escrow exponentiation, and the distributed setup protocol so for the security of these we need:

Conjecture 3 (The Strong RSA Assumption). Let $\mathcal{A}$ be any probabilistic polynomial time algorithm, and assume $\mathcal{A}$ gets (n, g) as input. Here $n = pq$ is an admissible RSA modulus of length k bits and g is an element of a subgroup G of $\mathbb{Z}_n^*$, which is chosen efficiently in polynomial time. Then the adversary $\mathcal{A}$ has to output 2 values y and e such that $y^e = g \bmod n$. The chance that $\mathcal{A}$ outputs 2 such values should be negligible in k.

3 A Simple Key Escrow System

The system in this section is the proof friendly variant from [14], where the setup phase has been taken over by the escrow authority. The system here has two extra phases compared to the system from [14], namely the key escrow and the escrowed decryption phase. The system only have one KEA server that provides the escrow value to the LEA directly without proof of correct behavior.

Since the order of the generated groups are unknown we pick exponents from the group $\mathbb{Z}_N$. For a more detailed explanation of how to chose a suitable N the reader is referred to [14]. We also use a function for computing discrete logs with base $(n + 1)$, which is referred to as $dLog_s()$. For the algorithm to compute this function the reader is referred to [11].

In section 4, the simple system is extended to have several KEAs and verification between the KEA servers and the LEA. The problems of users trying to flood LEA are addressed in section 5 by adding proofs of legal encryption to the encryption step. In most cases $s = 1$ will be sufficient for the key escrow scenario, but for completeness we will describe it with the general s, and in section 7 we'll address some of the efficiency issues arising from using larger s values.

Global Setup (KEA):

1. Pick 2 primes p, q of size $k/2$ bits each, where k is the security parameter. They should also satisfy that $p = 2p' + 1$ and $q = 2q' + 1$ for primes p', q' (i.e. p and q are safe primes).
2. Set $n = pq$ and $\tau = p'q'$.
3. Pick $g \in Q_n$, the group of squares.
4. Release the parameters: (n, g).
5. Store the escrow key: $d = n^{-1} \bmod \tau$.

Key Generation (user i):

1. Pick $\alpha_i \in \mathbb{Z}_N$.
2. Set $h_i = g^{\alpha_i} \bmod n$.
3. Release the public key: h_i.
4. Store the secret key: α_i.

Encryption (to user i):

To encrypt message $m \in \mathbb{Z}_{n^s}$, choose $r \in \mathbb{Z}_N$ and $b_0, b_1 \in \{0,1\}$:

$$\begin{aligned} E_s(m, r, b_0, b_1) &= (G, H) \\ &= ((-1)^{b_0} g^r \bmod n, \\ &\quad (-1)^{b_1}(h_i^r \bmod n)^{n^s}(n+1)^m \bmod n^{s+1} \end{aligned}$$

Decryption (user i):

To decrypt (G, H):

$$m = dLog_s((G^{\alpha_i} \bmod n)^{-2n^s} H^2 \bmod n^{s+1})/2 \bmod n^s$$

Key Escrow (KEA):

Given (G, H):

1. Abort if either G or H is malformed (i.e. $\gcd(G, n) \neq 1$, $\gcd(H, n) \neq 1$ or either G or H has Jacobi symbol -1 wrt. n.
2. Compute: $x_s = H \bmod n = \pm(h_i^r)^{n^s} \bmod n$
3. Compute x_0 using s repetitions of:

$$x_{i-1} = (x_i)^d = (\pm(h_i^r)^{n^i})^{n^{-1}} = \pm(h_i^r)^{n^i n^{-1}} = \pm(h_i^r)^{n^{i-1}} \bmod n$$

4. Send x_0 securely to the LEA.

Escrowed Decryption (LEA):

Given (G, H) and x_0, compute:

$$m = dLog((x_0)^{-2n^s} H^2 \bmod n^{s+1})/2 \bmod n^s$$

The work of the KEA server is of the order $O(sk^3)$, where k is the security parameter (size) of the modulus n, and the work of the LEA is $O((sk)^3)$.

Also note that the key generation of the users can be made in a threshold way to create a threshold decryption key. The escrow decryption will still work, even if the cryptosystem is changed slightly to accommodate the threshold version as is done in [14].

The above scheme, however only works in the passive case. In the active case there are a lot of problems. When the LEA cannot decrypt (that is $(x_0)^{-2n^s} H^2 \bmod n^{s+1}$ is not a power of $(n+1)$) there is no way to tell if it is because the user submitted a bad encryption (see 5) or if the KEA gave it a wrong value.

4 Threshold Key Escrow

To make the system threshold, we have to share the decryption value d and set up some verification values. Furthermore, the decryption phase has to provide verification proofs, so that KEAs can't cheat the LEA and ruin the decryption at some point.

The protocol uses a trusted third party (TTP) to setup the protocol, but in section 4.1 we show a brief sketch of how to get rid of this assumption.

Global Setup (TTP):

1. Pick 2 primes p, q, such that $p = 2p' + 1$ and $q = 2q' + 1$ for primes p', q' (ie. p and q are safe primes).
2. Set $n = pq$ and $\tau = p'q'$.
3. Pick $g, v \in Q_n$, the group of squares.
4. Compute: $d = (4\Delta^2 n)^{-1} \bmod \tau$, where $\Delta = w!$ for w KEA servers.
5. Pick random $a_i \in \mathbb{Z}_\tau$ for $i \in \{1, \cdots, t\}$, where $t < w/2$ is the threshold of the system.
6. Set $a_0 = d$ and create the polynomial $f(x) = \sum_{i=0}^{t} a_i x^i \bmod \tau$ (Shamir secret sharing [1]).
7. Release the parameters: (n, g).
8. Send $d_j = f(j)$ to the j'th KEA server.
9. Calculate: $v_j = v^{d_j} \bmod n$, for $j \in \{1, \cdots, w\}$.
10. Release the verification values: $(v, v_1, \cdots, v_w)$.

Key Generation (user i): As above.
Encryption (to user i): As above.
Decryption (user i): As above.
Key Escrow (KEAs):

Given (G, H) we do as above, except the method for calculating x_{i-1} is now a distributed protocol:

1. Given x_i, server j computes: $x_i^j = x_i^{2\Delta d_j}$.
2. Server j makes a proof that:

$$\log_{x_i^{4\Delta}}((x_i^j)^2) = \log_v v_i$$

   which can be done exactly as described in [10].
3. Server j sends x_i^j and the proof to the KEA bulletin board.

4. The servers check the proofs of the submitted values and picks a qualified set S with legal proofs and computes:

$$x_{i-1} = \prod_{j \in S} (x_i^j)^{2\lambda_j^S} \bmod n$$

where λ_j^S is the slightly modified Lagrange coefficient:

$$\lambda_j^S = \Delta \prod_{i \in S \setminus \{j\}} \frac{-i}{j-i}$$

This means that:

$$x_{i-1} = \prod_{j \in S} (x_i)^{4\Delta d_j \lambda_j^S} = (x_i)^{4\Delta^2 f(0)} = (x_i)^{n^{-1} \bmod \tau} \bmod n$$

which is what we want.

The decryption share x_1^j, and the proof is not posted to the bulletin board by server j, but is sent directly to the LEA using a secure authenticated channel.

Escrowed Decryption (LEA):

The LEA checks the proofs on the bulletin board, and checks the proofs of the shares x_1^j, which were sent to it, and picks a set S with correct proofs. It performs the Lagrange combination as in step 4 of the key escrow to get x_0 and m is computed using x_0 as it was done in section 3.

4.1 Removing the Trusted Third Party

In the protocol above we made use of a trusted third party to setup the global values, the secret sharing of d and the verification values. This can be done in a distributed fashion, so that the KEA servers can perform the setup them self, without affecting the security of the system.

To generate a product of safe primes the technique from [12] can be used. This generates an additive sharing of p and p' and tests for primality test on both (and likewise for q and q'). A secret sharing of the modulus n is then created from the sharing of p and q and then opened. To get a sharing of τ the sharing of p' and q' can be combined in the same way.

To generate the random values a_i the servers can simply chose sufficiently large random numbers (about $k + k_2$ bits, where k_2 is e.g. 160 bits) using the technique for creating the prime candidates p', q'. Then the modulo protocol can be used to create a value in $\{0, \cdots, \tau\}$ which will be statistically close to a uniform value.

The values g and v can be generated by generating two random elements $y, y' \in Z_n^*$, using e.g. commitments. The values can then be set to $g = y^2 \bmod n$ and $v = y'^2 \bmod n$, which are both in Q_n.

Now all values used in the setup phase are either public or secret shared and we can compute the rest using the general computation framework of [12].

5 Encryption Verification

The escrowed decryption cannot distinguish legal encryptions from illegal encryptions, since it computes the randomness used for the second part of the encryption. This means that a malicious sender could generate a lot of encryptions on the form:

$$(G, H) = (r_1, (r_2)^{n^s}(n+1)^{m^*} \bmod n^{s+1})$$

The values r_2, m^* might not be know by the adversary, but 2 such values exists. The decryption by a user will show that it is a illegal encryption and it will be discarded, whereas the escrow decryption will result in the value r_2 being passed to LEA and the message m^* being output. The above encryption cannot be distinguished from a normal encryption (by conjecture 2). So the KEA servers or the LEA servers can be overloaded by sending just 1 correct encryption and a lot of illegal encryptions as above. The receiver will discard all the illegal encryptions and accept the single correct encryption, whereas the LEA will have a lot of plaintexts of which only one is actually received.

To take care of this problem a non-interactive ZK-proof very similar to the one in [14], can be used by the sender to prove that the sent message is really a legal encryption. The proof is shown in section 6. The receiver checks this proof before decrypting, so that it knows it have received an escrowed encryption. If a bad ciphertext is constructed as above the sender will be unable to create a valid proof, which will make the flooding attack impossible. This means that the LEA can discard the encryptions with illegal proofs and thus the sender cannot flood the LEAs.

6 Auxiliary Protocols

Protocol for legal encryption
Input: $n, g, h, c = (G, H)$
Private input for P: $r \in \mathbb{Z}_N$ and $m \in \mathbb{Z}_{n^s}$, such that $c = E_s(m, r, b_0, b_1)$ for some b_0 and b_1.

1. P chooses at random r' in $\{0, ..., 2^{|N|+2k_2}\}$ and $m' \in \mathbb{Z}_{n^s}$, where k_2 is a secondary security parameter (e.g. 160 bits). P sends $c' = (G', H') = E_s(m', r', 0, 0)$ to V.
2. V chooses e, a random k_2 bit number, and sends e to P.
3. P sends $\hat{r} = r' + er$ and $\hat{m} = m' + em \bmod n^s$ to V.
4. V checks that G, H, G', H' are prime to n, have Jacobi symbol 1 and that $E_s(2\hat{m}, 2\hat{r}, 0, 0) = (G'^2 G^{2e} \bmod n, H'^2 H^{2e} \bmod n^{s+1}) = c'^2 c^{2e}$, and accepts if and only if this is the case.

The protocol above can be proven to be sound and complete honest verifier zero-knowledge. This is enough for the correctness proof of encryption, since it will be used in a non-interactive setting using the Fiat-Shamir Heuristic. To get the challenge the sender P uses the hash function $\mathcal{H}$ to calculate the challenge:

$$e = \mathcal{H}(ID_S, ID_R, G, H, G', H')$$

Here ID_S and ID_R is some information identifying respectively the sender and the receiver.

Note that the above protocol also works if the encryption function E is changed to the more general function:

$$\begin{aligned} E_s(m, r, b_0, b_1) = (G, H) = ((-1)^{b_0} g^r \bmod n, \\ (-1)^{b_1} (h_i^{\beta r} \bmod n)^{n^s} (n+1)^m \bmod n^{s+1} \end{aligned}$$

where β is a fixed value. This is the case for the threshold version of the encryption system in [14], where $\beta = 4w!$, where w were the number of servers in that threshold setting.

7 Improving Performance for $s > 1$

The calculation of x_0 requires s rounds of exponentiation in the previous schemes. In the case where there are many messages using $s > 1$ it might be an advantage to decrease the number of needed exponentiations in the escrow part of the protocol. To do this extra decryption values can be computed:

$$d^s := 4\Delta^2 n^{-s} \bmod \tau$$

this will allow the servers to remove s powers of n in a single exponentiation step. To be able to verify correctness this requires an extra set of verification values which this exponentiation is verified up against.

When there are only a few number of s's that are frequently used these values can be computed in advance together with the verification values. If the KEA servers keep the sharing of τ they can compute new values after the setup phase is done.

If s is arbitrary in general, there are different strategies that can be used to reduce space (number of values kept by the KEAs and size of all verification values), time and communication (number of rounds of exponentiation to compute x_0). In figure 1 are three different approaches given some upper bound s': 1) only d^1 used, 2) use all the $d^1, d^2, \cdots, d^{s'}$, and 3) use only the powers of 2: $d^{2^0}, d^{2^1}, \cdots, d^{2^{log_2(s'/2)}}$.

Scheme	$\sharp d$	$\sharp$ verification values	$\sharp$ exponentiations when $s \leq s'$	$\sharp$ exponentiations when $s > s'$
Only 1	1	$w+1$	s	s
All	s'	$s'w+1$	1	s/s'
Logarithmic	$log_2(s')$	$log_2(s')w+1$	$\sim log_2(s)/2$	$\sim s/s' + log_2(s')$

Fig. 1. Different values when using upper bound s' to setup system, assuming s' is a power of 2

8 Security of the System

Theorem 1. *The system defined in section 3 is semantically secure.*

Proof. The proof follow directly from [14], since the only thing changed is the addition of decryption key, which does not affect the security of the system except for the KEAs that have a new trapdoor.

Now we can define some lemmas about the combined cryptosystem as defined in 4 with the user proof shown in 5:

Lemma 1. *Senders cannot flood any LEAs.*

Proof. This follows from the correctness of the proof of correct encryption. Since the sender has to create a correct proof it will be unable to fool a LEA into decryption bad messages.

Lemma 2. *Users cannot prevent decryption when controlling less than $w/2$ servers.*

Proof. The exponentiation of the KEA servers uses proof of correct behavior, which means that the user cannot inject bad values without being noticed with all but a negligible chance.

The secret d is shared between w servers with a threshold of $t < w/2$. If the user controls less than $w/2$ servers there will be at least $t + 1$ honest servers left which is enough to perform the exponentiation. This means that the servers will be able to finish the protocol and give the correct value to the LEA.

Lemma 3. *A user cannot decrypt messages from other users when t or less KEA servers are helping*

Proof. This follows directly from the semantic security of the cryptosystem and the fact that t or less KEAs have no information on the shared secret.

Lemma 4. *The LEA cannot decrypt messages encrypted by users, without getting the decryption value from KEA.*

Proof. This is the same as for lemma 3, except that LEAs can ask for getting messages decrypted. There are 2 reasons for granting such a decryption, namely either if sender or if receiver is considered suspicious. Now if the original sender/receiver pair is not considered suspicious the LEA will have to create a related ciphertext where either the sender or receiver is a suspicious person.

However, if the sender or receiver is changed the input to the hash function is changed and another challenge will be used for the proof of a legal encryption. This means that the message cannot be changed to look like it is to/from some suspicious person.

Lemma 5. *A LEA learns no non-trivial information on the private key of the user during decryption.*

Proof. The signature scheme in [10] by Shoup is proven secure in the random oracle model. This means that since each exponentiation step in the escrow protocol are exactly the same as a Shoup threshold signature computation they're by themselves secure.

The different results after each exponentiation offer no information either since such tuples can be generated by the adversary himself. This can be done by picking r and then compute $(g^r, h^r, (h^r)^n, \cdots, (h^r)^{n^s})$.

Lemma 6. *Users, LEAs and t or less KEA servers cooperating are unable to calculate d.*

Proof. Firstly t or less KEA servers has no information on d, since it is secret shared with a threshold of t.

Users can only get correctly constructed ciphertexts decrypted. This means that the values the KEA servers are raising to the secret exponent d_j is on the form:

$$(h^r)^{\beta n^s} \bmod n$$

which is in Q_n. But this is the exact same type of values that are exponentiated in [10]. If an adversary exists against this step then an adversary exist against Shoup's scheme, which was proven secure in the random oracle model.

The rest follows from the proof of lemma 5, namely that the rest of the values can be simulated by the LEAs themselves, without help from the KEA servers.

References

1. A. Shamir: *How to Share a Secret*, Communications of the ACM, vol. 22, no. 11, pp. 612-613, 1979.

2. T. ElGamal: *A public-key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory. IT-31(4), pp 469-472, July 1985.
3. R. Cramer, I. Damgård and B. Schoenmakers: *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, Proceedings of Crypto '94, Springer-Verlag LNCS 839, pp. 174-187.
4. A. Shamir: *Partial Key Escrow: A New Approach to Software Key Escrow*, NIST FIPS Key Escrow Workshop, National Institute of Standards and Technology, Gaithersburg, Md., September 15, 1995.
5. S. Micali: *Guaranteed Partial Key Escrow*, MIT laboratory of computer science, Technical Memo 537, September 1995.
6. S. Micali and A. Shamir: *Partial Key Escrow*, Manuscript, February 1996.
7. M. Bellare and S. Goldwasser: *Verifiable Partial Key Escrow*, Proceedings of ACM Conference on Computer and Communications Security, 1997, pp 78-91.
8. W. Mao: *Publicly Verifiable Partial Key Escrow*, Proceedings of ICICS'97, Springer Verlag LNCS series 1334, pp. 409-413.
9. P. Paillier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt '99, Springer Verlag LNCS series 1592, pp. 223-238.
10. V. Shoup: *Practical Threshold Signatures*, Proceedings of EuroCrypt '00, Springer Verlag LNCS 1807, pp. 207-220.
11. I. Damgård and M. Jurik: *A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System*, Proceedings of Public Key Cryptography 2001, Springer Verlag LNCS series 1992, pp. 119-136.
12. J. Algesheimer, J. Camenisch and V. Shoup: *Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products*, Cryptology ePrint Archive, Report 2002/029, `http://eprint.iacr.org/`, March 2002.
13. J. Shaoquan and Z. Yufeng: *Partial Key Escrow Monitoring Scheme*, Cryptology ePrint Archive, Record 2002/039, `http://eprint.iacr.org/`, March 2002.
14. I. Damgård and M. Jurik: *A Length-Flexible Threshold Cryptosystem with Applications*, BRICS Report Series, RS-03-16, `http://www.brics.dk/Publications/`, March 2003.

Recent BRICS Report Series Publications

RS-03-22 **Ivan B. Damgård and Mads J. Jurik.** ***Scalable Key-Escrow.*** **May 2003. 15 pp.**

RS-03-21 **Ulrich Kohlenbach.** ***Some Logical Metatheorems with Applications in Functional Analysis.*** **May 2003. 55 pp.**

RS-03-20 **Mads Sig Ager, Olivier Danvy, and Henning Korsholm Rohde.** ***Fast Partial Evaluation of Pattern Matching in Strings.*** **May 2003. 16 pp. Final version to appear in Leuschel, editor,** ***ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*****, PEPM '03 Proceedings, 2003. This report supersedes the earlier BRICS report RS-03-11.**

RS-03-19 **Christian Kirkegaard, Anders Møller, and Michael I. Schwartzbach.** ***Static Analysis of XML Transformations in Java.*** **May 2003. 29 pp.**

RS-03-18 **Bartek Klin and Paweł Sobociński.** ***Syntactic Formats for Free: An Abstract Approach to Process Equivalence.*** **April 2003. 41 pp.**

RS-03-17 **Luca Aceto, Jens Alsted Hansen, Anna Ingólfsdóttir, Jacob Johnsen, and John Knudsen.** ***The Complexity of Checking Consistency of Pedigree Information and Related Problems.*** **March 2003. 31 pp. This paper supersedes BRICS Report RS-02-42.**

RS-03-16 **Ivan B. Damgård and Mads J. Jurik.** ***A Length-Flexible Threshold Cryptosystem with Applications.*** **March 2003. 19 pp.**

RS-03-15 **Anna Ingólfsdóttir.** ***A Semantic Theory for Value–Passing Processes Based on the Late Approach.*** **March 2003. 48 pp.**

RS-03-14 **Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard.** ***From Interpreter to Compiler and Virtual Machine: A Functional Derivation.*** **March 2003. 36 pp.**

RS-03-13 **Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard.** ***A Functional Correspondence between Evaluators and Abstract Machines.*** **March 2003. 28 pp.**

RS-03-12 **Mircea-Dan Hernest and Ulrich Kohlenbach.** ***A Complexity Analysis of Functional Interpretations.*** **February 2003. 70 pp.**