

On Quorum Controlled Asymmetric Proxy Re-encryption

Markus Jakobsson

Information Sciences Research Center, Bell Laboratories
Murray Hill, NJ 07974
`markusj@research.bell-labs.com`

Abstract. We present a scheme for quorum controlled asymmetric proxy re-encryption, with uses ranging from efficient key distribution for pay-tv to email applications. We prove that the scheme, which is based on El-Gamal encryption, leaks no information as long as there is no dishonest quorum of proxy servers. Of potential independent interest is a method providing publicly verifiable *translation certificates*, proving that the input and output encryptions correspond to the same plaintext message, without leaking any information about the plaintext to either the verifier or a subset of the servers of the prover. The size of the certificate is small, and independent of the number of prover servers.

Keywords: asymmetric proxy re-encryption, translation certificate, El Gamal encryption, quorum control, robustness, privacy.

1 Introduction

With an increasing importance of encryption methods for privacy and protection of business secrets, and with an increasing need for a flexible infrastructure, we foresee the need for many new secure and flexible primitives extending the basic communication and encryption capabilities available today. One such primitive is *proxy re-encryption*, which was recently introduced by Blaze, Bleumer and Strauss [2]. Proxy re-encryption is a translation between ciphertexts from one encryption key to another encryption key. It can be used to forward encrypted messages without having to expose the cleartexts to the participants involved, a primitive with many potential commercial uses.

In *symmetric* proxy encryption, which was exhibited by Blaze, Bleumer and Strauss, the proxy (who is the entity performing the translation, and who is typically not a distributed entity) needs to know a function of the secret keys associated with both the incoming and outgoing transcripts. The proxy needs to be trusted not to collude with one of the participants holding these secret keys, or the other secret key can be derived. (This characterization has the same meaning as that of [2], in which no trust has to be placed in the proxy, but the two other participants need to trust each other.)

On the other hand, in *asymmetric* proxy re-encryption, it is not necessary for the proxy to any function of the secret key corresponding to the produced ciphertext, which is advantageous given that we often want to forward encrypted

messages to parties who do not trust us with their secret key. In a quorum control setting, the proxy only needs to know the secret key corresponding to the incoming ciphertext. It is natural that the party whom the incoming ciphertext is for needs to trust the proxy to some extent, since the proxy controls how incoming messages are re-encrypted and forwarded. However, if the proxy is quorum controlled and the re-encryption is robust, then this party only needs to trust that there is no dishonest quorum of proxy servers.

In this paper, we demonstrate how to implement asymmetric proxy re-encryption, which was posed as an open problem in [2]. For security, the transformation is performed under quorum control: This guarantees that if there is no dishonest quorum, then the plaintext message whose encryption is being transformed is not revealed to the proxy servers. Our solution is efficient; allows tight control over actions (by the use of quorum cryptography); does not require any pre-computation phase to set up shared keys; and has a trust model appropriate for a variety of settings. We believe that such a mechanism may be very useful in many applications:

- It allows the proxy to transform encrypted messages to encryptions with a variety of different recipient public keys, to allow for categorization of the encryptions. The categorization may be performed either as a function of the transcripts and their origins, randomly (e.g., assigning an examiner to an electronically submitted patent), or as a function of time, and may be used to sort the encrypted messages according to priority or security clearance. A practical and concrete example is that you want somebody to be able to read your email while you are on vacation, but you do not want to force the senders of the messages to have to know that the messages are being re-routed. In a situation like this, a symmetric model is inappropriate. Also, some form of control is desirable, guaranteeing that the messages are being handled according to the instructions.
- It allows more efficient communication to a large number of recipients that are physically clustered around the proxy; the sender would only need to send *one* encryption, along with an authenticated list of recipients. This may prove very useful for pay-tv, general multi-cast, and related applications.
- Last but not least, we believe that asymmetric proxy encryption may become a useful building block in the design of secure and efficient multi-party protocols.

A partial result of potential independent interest is a non-interactive proof that the correct translation between encryptions was performed, i.e., that the incoming and outgoing encryptions indeed encrypt the same message. The transcript, which we call a *translation certificate* is publicly verifiable, is compact (using standard¹ security parameters, it is a mere 396 bytes long, independently of the number of provers), and does not leak any information about the plaintext to verifiers or to a subset of the provers.

¹ We use $|p| = 1024$, $|q| = 160$.

Our techniques draw on ideas used in the work on proactive security (e.g., [7, 9, 10]), on methods for undeniable signatures (e.g., [3, 4]), Schnorr signatures [14], and methods for information-theoretical secret sharing [12].

Outline: We start in section 2 by reviewing related work. We then discuss the requirements on our scheme in section 3. In section 4, we then present our basic scheme for proxy re-encryption, followed in section 5 by a protocol for generating translation certificates. We end by stating and proving claims in section 7.

2 Review

Public and Secret Information: Let p, q be primes such that $p = 2q + 1$, and g be a generator of G_p . The proxy servers share a secret key x_1 using a (k, n) threshold scheme (see [15, 11]); their corresponding public key is $y_1 = g^{x_1} \bmod p$. (Onwards, we assume all arithmetic to be modulo p where applicable, unless otherwise stated.) Likewise, the recipient has a secret key x_2 with a corresponding public key $y_2 = g^{x_2}$.

ElGamal: Our protocol uses ElGamal encryption [6]: To encrypt a value² m using the public key y , a value $\gamma \in_u Z_q$ is picked uniformly at random, and the pair $(a, b) = (my^\gamma, g^\gamma)$ calculated. Thus, (a, b) is the encryption of m . In order to decrypt this and obtain m , $m = a/b^x$ is calculated.

The Decision Diffie-Hellman Assumption: Let $p = 2q + 1$, for primes p and q , and let m, g be generators of a subgroup of order q . Then, the pairs (m, m^x, g, g^x) and (m, m^r, g, g^x) are indistinguishable, for random and unknown values $r, x \in Z_q$, $m, g \in G_p$.

3 Preliminaries

An entity with public key y_1 assigns a proxy, agrees with the proxy on rules for re-encryption, and distributes shares of his secret key x_1 to the servers of the proxy. Later, the proxy receives a transcript E_1 , which is an ElGamal encryption of a message m using public key y_1 . The proxy produces and outputs a transcript E_2 , which is an ElGamal encryption of the same message m , but using a given public key y_2 , which is chosen according to the rules set by the entity associated with y_1 . We note the re-encryption method can be extended to long messages by replacing m by a symmetric key used for encryption of a long message m_{long} whose ciphertext is passed along unaltered to the entity associated with y_2 .

The transformation is controlled by the use of quorum actions. Informally, the requirements on our scheme are:

1. *Correctness:* Any quorum Q of proxy servers, sharing a secret key x_1 , will be able to perform the above re-encryption.

² Here, $m = (\frac{M}{p})M$ for an original message $M \in [1 \dots \frac{p-1}{2}]$, where $(\frac{M}{p})$ is the Jacobi symbol of M .

2. *Robustness*: If any participant in the transformation protocol would output incorrect transcripts, then this will be detected by all honest participants. The protocol will allow the honest participants to determine what participants cheated, and to substitute these.
3. *Public Verifiability*: Anybody must be able to verify that the correct transformation was performed, without having or receiving knowledge of any secret information. The corresponding proof, the *translation certificate*, must be compact and be verifiable without interaction.
4. *Asymmetry*: The proxy servers must need no information about the secret key x_2 corresponding to the receiver's public key y_2 in order to perform the computation, and the receiver will need no information about x_1 or y_1 in order to decrypt E_2 .
5. *Privacy*: The proxy re-encryption (including the generation of the translation certificate, and other robustness mechanisms) does not leak any information about m to any set of proxy servers smaller than a quorum.

In section 7, we formalize these requirements and prove that our proposed scheme satisfies the same.

4 Gradual and Simultaneous Proxy Re-Encryption

The concept of our solution is to use *gradual* and *simultaneous* translation of transcripts. The translation is called *gradual*, since it is performed by quorum action, and each server's contribution to the computation is only a partial translation. We call it *simultaneous* since each server performs one partial decryption *and* one partial encryption, outputting such gradual re-encryptions without the cleartext ever being exposed. This approach makes all the partial translations simultaneous in the sense that no result is obtained until all the portions are accounted for.

We first consider a non-robust version of the proxy re-encryption, and then add on a proof to guarantee robustness.

Let (a_1, b_1) be an ElGamal encryption of a message m w.r.t. a public key y_1 , and let x_1 be the corresponding secret key, which is shared by the proxy servers using a threshold scheme. The proxy servers wish to compute the ElGamal encryption (a_2, b_2) of m w.r.t. the public key y_2 . They wish not to expose m to any set of dishonest proxy servers (or any other set of servers); according to our assumptions, they do not know the secret key x_2 of y_2 .

For simplicity of denotation, we assume that x_{1j} is the Lagrange-weighted secret key (using the methods in [11]) of proxy server j w.r.t. a given active quorum Q ; $y_{1j} = g^{x_{1j}}$ is the corresponding public key share. The servers in the quorum perform the following computation:

1. Server j selects a random value δ_j uniformly at random from Z_q , and computes $(c_j, d_j) = (b_1^{-x_{1j}} y_2^{\delta_j}, g^{\delta_j})$. This pair is sent to the other proxy servers.
2. The servers (or alternatively, a non-trusted gateway) compute the pair $(a_2, b_2) = (a_1 \prod_{j \in Q} c_j, \prod_{j \in Q} d_j)$. The pair (a_2, b_2) is output.

The above protocol for proxy re-encryption is made robust by use of translation certificates.

5 Generating the Translation Certificate

We want to produce a translation certificate, i.e., a non-interactive proof that $a_1/b_1^{x_1} = a_2/b_2^{x_2}$, or in other words, a proof that (a_1, b_1) and (a_2, b_2) are encryptions of the same message, for secret decryption keys x_1 resp. x_2 of the two encryptions. The certificate must not leak any information to the verifier or to any non-quorum of prover servers. Also, it must not require knowledge of the second secret key, x_2 , since this is not assumed to be known by the prover. Finally, it must be publicly verifiable. Our solution will produce such certificates that are short, and whose length does not depend on the number of provers.

More specifically, we need to prove that $(a_2, b_2) = (a_1 b_1^{-x_1} y_2^\delta, g^\delta)$, for $y_1 = g^{x_1}$. In the proof, we will use a new generator, h , whose discrete log w.r.t. g is not known to any set of parties. We will also use a hash function $hash$, which is assumed to be collision free, and whose output is in Z_q . The proof has two components: One proving knowledge of the secret keys corresponding to two “public keys”, the other proving that the output has the claimed relation to these two public keys. The version we show first is, for clarity, the single-prover version. We then explain how this is extended to a distributed prover, and how cheating provers are detected and traced.

In order to increase the readability of the protocol, we will rename certain variables to obtain a more uniform naming. To this extent, we will use the variable names $(z_1, z_2, w_1, w_2, \sigma, \mu_1, \mu_2)$ to mean $(y_1, b_2^{-1}, x_1, -\delta, a_2/a_1, b_1, y_2)$. Thus, wanting to prove that $a_2 = a_1 b_1^{-x_1} y_2^\delta$, for $(y_1, b_2) = (g^{x_1}, g^\delta)$ is the same as wanting to prove that $\sigma = \mu_1^{w_1} \mu_2^{w_2}$ for $(z_1, z_2) = (g^{w_1}, g^{w_2})$.

Initialization:

P computes and outputs $(\bar{z}_1, \bar{z}_2) = (h^{w_1}, h^{w_2})$.

Part I:

1. P selects $\alpha \in_u Z_q$, and computes $(\mathcal{G}, \mathcal{H}) = (hash([g^\alpha]_p), hash([h^\alpha]_p))$.
2. P computes a pair of challenges $(\epsilon_1, \epsilon_2) = (hash(\mathcal{G}, \mathcal{H}, 1), hash(\mathcal{G}, \mathcal{H}, 2))$.
3. P computes the response $\delta = [\alpha - \epsilon_1 w_1 - \epsilon_2 w_2]_q$. He outputs $(\mathcal{G}, \mathcal{H}, \delta)$.

Part II:

1. P selects $\beta_1, \beta_2 \in_u Z_q$, and computes $(\mathcal{F}, \mathcal{M}) = (hash([g^{\beta_1} h^{\beta_2}]_p), hash([\mu_1^{\beta_1} \mu_2^{\beta_2}]_p))$.
2. P computes a challenge $e = hash(\mathcal{F}, \mathcal{M})$.
3. P computes the response $(d_1, d_2) = ([\beta_1 - e w_1]_q, [\beta_2 - e w_2]_q)$. He outputs $(\mathcal{F}, \mathcal{M}, d_1, d_2)$.

The translation certificate is the transcript $(\bar{z}_1, \bar{z}_2, \mathcal{G}, \mathcal{H}, \delta, \mathcal{F}, \mathcal{M}, d_1, d_2)$. The proof can be distributively generated, with tracing of dishonest provers, as shown in the Appendix. It is verified by the verifier V as follows:

Verification:

1. V computes $(\epsilon_1, \epsilon_2) = (\text{hash}(\mathcal{G}, \mathcal{H}, 1), \text{hash}(\mathcal{G}, \mathcal{H}, 2))$, and accepts part I iff $\mathcal{G} = \text{hash}([g^\delta z_1^{\epsilon_1} z_2^{\epsilon_2}]_p)$ and $\mathcal{H} = \text{hash}([h^\delta \bar{z}_1^{\epsilon_1} \bar{z}_2^{\epsilon_2}]_p)$.
2. V computes $e = \text{hash}(\mathcal{F}, \mathcal{M})$, and accepts part II iff $\mathcal{F} = \text{hash}([g^{d_1} h^{d_2} (z_1 \bar{z}_2)^e]_p)$ and $\mathcal{M} = \text{hash}([\mu_1^{d_1} \mu_2^{d_2} \sigma^e]_p)$.
3. If V accepted both part I and part II, then he outputs accept, otherwise he rejects.

6 Claims

The protocol for generation of translation certificates is *correct* (lemma 1,) *sound* (lemma 2,) and *zero-knowledge in the random oracle model* (lemma 3.)

The protocol for robust proxy re-encryption satisfies the previously stated requirements: it satisfies *correctness* (theorem 1,) *robustness* (theorem 2,) *asymmetry* (theorem 3,) and *privacy* (theorem 4.)

These lemmae and theorems are proven in section 7.

7 Proofs of Claims

Lemma 1: The protocol generating translation certificates is *correct*, i.e., if the prover is honest, then the verifier will accept with an overwhelming probability.

Proof of Lemma 1:

We assume that the prover is honest. Four equations have to be satisfied in order for the verifier to accept.

- 1: $g^\delta z_1^{\epsilon_1} z_2^{\epsilon_2} = g^{\alpha - \epsilon_1 w_1 - \epsilon_2 w_2} g^{w_1 \epsilon_1} g^{w_2 \epsilon_2} = g^\alpha = \text{hash}^{-1}(\mathcal{G})$.
- 2: $h^\delta \bar{z}_1^{\epsilon_1} \bar{z}_2^{\epsilon_2} = h^{\alpha - \epsilon_1 w_1 - \epsilon_2 w_2} h^{w_1 \epsilon_1} h^{w_2 \epsilon_2} = h^\alpha = \text{hash}^{-1}(\mathcal{H})$.
- 3: $g^{d_1} h^{d_2} (z_1 \bar{z}_2)^e = g^{\beta_1 - e w_1} h^{\beta_2 - e w_2} (g^{w_1} h^{w_2})^e = g^{\beta_1} h^{\beta_2} = \text{hash}^{-1}(\mathcal{F})$.
- 4: $\mu_1^{d_1} \mu_2^{d_2} \sigma^e = \mu_1^{\beta_1 - e w_1} \mu_2^{\beta_2 - e w_2} (\mu_1^{w_1} \mu_2^{w_2})^e = \mu_1^{\beta_1} \mu_2^{\beta_2} = \text{hash}^{-1}(\mathcal{M})$.

By the definition of $\mathcal{G}, \mathcal{H}, \mathcal{F}, \mathcal{M}$, these relations hold. \square

It can easily be seen that the robust and distributed version of the protocol for generating translation certificates is correct if the single-server version is.

Lemma 2: The protocol for generating translation certificates is *sound*: If a participant has a non-negligible probability of answering three³ or more challenges

³ Normally, soundness is defined as the claimed relationship must hold if the prover can answer *two* queries with a non-negligible probability. However, since this is only to bound the probability of a verifier accepting an incorrect proof, any polynomial number works.

correctly, then this participant can be used as a black-box to extract the secret key for decryption.

Proof of Lemma 2: (*Sketch*)

In this proof, we assume that it is not feasible to find hash collisions. Then, the only time the verifier will accept is if $g^\alpha = g^\delta z_1^{\epsilon_1} z_2^{\epsilon_2}$ and $h^\alpha = h^\delta \bar{z}_1^{\epsilon_1} \bar{z}_2^{\epsilon_2}$. We now further assume that the challenges are randomly generated. We consider the two parts of the proof independently:

Part I: Let (ϵ_1, ϵ_2) , $(\epsilon'_1, \epsilon'_2)$ and $(\epsilon''_1, \epsilon''_2)$ be three different pairs of challenges, and let δ , δ' , and δ'' be the corresponding correct responses. Then, given that we have three equations for these, with common choices of α , and we only have two unknowns (w_1 and w_2), we can solve the equations for these. Therefore, if the prover can answer three or more challenges with a non-negligible probability, he must know w_1 and w_2 .

Part II: Using a similar argument to that above, we see that if the prover can answer two different challenges e and e' , then he can solve the response equations for w_1 and w_2 .

For these two cases, therefore, being able to answer three or more possible challenges out of all possible challenges can be used to compute the secret key w_1 , which corresponds to the secret key for decryption. \square

Lemma 3: The interactive scheme for proving valid exponentiation is *zero-knowledge in the random oracle model*.

This can be seen using a standard argument, by turning the protocol into an interactive protocol, where the challenges are randomly chosen instead of chosen as functions of previously seen transcripts. If the challenges are committed to at the beginning of the protocol, then a rewinding technique will allow a simulator to produce the expected outputs after having seen the challenges.

Theorem 1: The transformation scheme satisfies *correctness*, i.e., if E_1 is an encryption of m w.r.t. y_1 , then the output of the scheme will be E_2 , an encryption of m w.r.t. y_2 , for a value y_2 chosen by the proxy.

Proof of Theorem 1: (*Sketch*)

Assume that $(a_1, b_1) = (my_1^\gamma, g^\gamma)$, i.e., (a_1, b_1) is a valid ElGamal encryption of a message m w.r.t. the proxy's public key y_1 . We have that $(c_j, d_j) = (b_1^{-x_{1j}} y_2^{\delta_j}, g^{\delta_j})$, for an already Lagrange-weighted (w.r.t the quorum Q) secret key share x_{1j} of proxy server j , and a random number δ_j . Then, we have that $(a_2, b_2) = (a_1 \prod_{j \in Q} c_j, \prod_{j \in Q} d_j)$. We therefore have that $a_2 = a_1 b_1^{-x_1} y_2^\delta$, for $\delta = \sum_{j \in Q} \delta_j \text{ mod } q$, and $x_1 = \sum_{j \in Q} x_{1j} \text{ mod } q$. Recall that $y_1 = g^{x_1}$ and that $a_1/b_1^{x_1}$ is the plaintext m corresponding to the ciphertext (a_1, b_1) w.r.t. the public key y_1 . Thus, $a_2 = m y_2^\delta$, according to the decryption algorithm for ElGamal encryption. Since $b_2 = \prod_{j \in Q} d_j = g^\delta$, we have that (a_2, b_2) is a valid ElGamal encryption of the message m w.r.t. the public key y_2 , and thus, the transformation protocol is correct. \square

It follows automatically that the protocol that is made robust by the added use of translation certificates must be correct if the non-robust version is correct.

Theorem 2: The scheme satisfies *robustness*, i.e., if any participating proxy server outputs a transcript that would result in an incorrect end result, then the honest participants will detect this, and will be able to determine the cheating server's identity.

This follows from the soundness of the translation certificates, which was shown in Lemma 2: Only correct outputs (corresponding to valid re-encryptions) will have correct translation certificate. If an invalid translation certificate is found, the individual portions of this certificate can be verified for validity. This can be done without interaction. An invalid portion (w.r.t. the public key of the participant generating it) corresponds to a cheater.

Theorem 3: The scheme satisfies *asymmetry*.

This is obvious given the specification of the protocol; the proxy servers never need any secret information corresponding to the public key y_2 of the intended recipient, nor does the recipient need any secret information apart from this secret key in order to decrypt the received transcript.

Theorem 4: The scheme satisfies *privacy*: Let A be a set of proxy servers not containing a quorum. A can simulate transcripts such that these cannot be distinguished by A from transcripts of the transformation protocol, other than with a negligible probability.

Proof of Theorem 4: (*Sketch*)

We consider the interactive version of the translation certificate herein, to make the argument simple. Let E_2 be a value that cannot be distinguished by A from a valid re-encryption (according to the given public keys) of the input E_1 . (For ElGamal encryption, it is commonly believed that any pair of randomly chosen elements from G_p may be chosen as such a value E_2 , given no partial knowledge of the corresponding decryption key x_2 .) Let us assume that the secret key x_2 needed to decrypt the transformed encryption is not known by any proxy servers. Focusing on the non-robust transformation protocol only, one can then show that the view of a set of proxy servers not containing a quorum can be simulated, following the (somewhat space-consuming) method used in [10] for proving the simulability of two related protocols, namely those for proactive key update and for distributed signature generation. The same result will be obtained when such a protocol is interleaved (a constant and low number of times) with a protocol that is zero-knowledge. Therefore, the robust transformation protocol has the property that a partial view (corresponding to the views of a set of proxy servers not containing a quorum) is simulable in p-time, and the simulated transcripts cannot be distinguished (by the same set of proxy servers) from real transcripts. This argument holds for a serial concatenation of protocol executions (following the proof method in [10],) and so, is valid also when cheating servers corrupt the protocol and force a restart of the same.

In more detail, the simulator will compute transcripts according to the inputs given by A , and select transcripts for the appropriate distributions from the proxy servers not controlled by A . This is done so that the resulting output is E_2 .

The simulator then simulates the zero-knowledge proofs for the honest servers (i.e., those not controlled by A), giving transcripts showing that these transcripts are valid and correspond to the previously set outputs of these servers. We note that it will not be possible for A to distinguish transcripts in a simulation where a *false* statement is “proven” from transcripts from a simulation of a *true statement* (and therefore also not from real transcripts.) If this were not the case, then it would not be hard to decide whether a given input is valid or not, without the interaction of the prover, which in turn would violate our computational assumption. \square

Acknowledgements

Many thanks to Matt Blaze and Daniel Bleichenbacher for helpful comments.

References

1. M. Bellare, P. Rogaway, "Random Oracles are Practical: a paradigm for designing efficient protocols," 1st ACM Conference on Computer and Communications Security, pp. 62-73, 1993.
2. M. Blaze, G. Bleumer, M. Strauss, "Divertible Protocols and Atomic Proxy Cryptography," Eurocrypt '98, pp. 127-144
3. D. Chaum, H. Van Antwerpen, "Undeniable Signatures," Crypto '89, pp. 212-216
4. D. Chaum, "Zero-Knowledge Undeniable Signatures," Eurocrypt '90, pp. 458-464
5. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to Share a Function Securely," STOC '94, pp. 522-533
6. T. ElGamal "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," Crypto '84, pp. 10-18
7. Y. Frankel, P. Gemmell, P. MacKenzie, M. Yung, "Proactive RSA," Proc. of CRYPTO '97, pp. 440-454
8. S. Goldwasser and S. Micali, "Probabilistic Encryption," J. Comp. Sys. Sci. 28, pp 270-299, 1984.
9. A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, "Proactive Secret Sharing, or How to Cope with Perpetual Leakage," Crypto '95, pp. 339-352
10. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, "Proactive Public Key and Signature Systems," Proceedings of the 4th ACM Conference on Computer and Communications Security, 1997, pp. 100-110
11. T. P. Pedersen. A threshold cryptosystem without a trusted party. In D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pp. 522 – 526. Springer-Verlag, 1991.
12. T. P. Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing," Crypto '91, pp. 129-140
13. D. Pointcheval, J. Stern, "Security Proofs for Signature Schemes," Eurocrypt '96, pp. 387-398
14. C. P. Schnorr, "Efficient Signature Generation for Smart Cards," Advances in Cryptology - Proceedings of Crypto '89, pp. 239-252
15. A. Shamir, "How to Share a Secret," Communications of the ACM, Vol. 22, 1979, pp. 612-613
16. D. Tygar, B. Yee, "Strongbox: A System for Self Securing Programs," CMU Computer Science: 25th Anniversary Commemorative, Addison-Wesley, 1991
17. B. Yee, D. Tygar, "Secure Coprocessors in Electronic Commerce Applications," Proceedings of the First USENIX Workshop on Electronic Commerce, New York, New York, July, 1995
18. B. Yee, "Using Secure Coprocessors," Ph.D. Thesis, Carnegie Mellon University, CMU-CS-94-149, 1994

A Distributed Generation of Translation Certificates

Initialization:

$P_i, i \in Q$ has a (already Lagrange weighted w.r.t. Q) pair (w_{1i}, w_{2i}) . P_i computes and outputs $(\bar{z}_{1i}, \bar{z}_{2i}) = (h^{w_{1i}}, h^{w_{2i}})$.

Part I:

1. $P_i, i \in Q$ selects $\alpha_i \in_u Z_q$, computes and publishes $(G_i, H_i) = (g^{\alpha_i}, h^{\alpha_i})$. P_i computes $(G, H) = (\prod_{j \in Q} G_j, \prod_{j \in Q} H_j)$, and $(\mathcal{G}, \mathcal{H}) = (\text{hash}(G), \text{hash}(H))$.
2. $P_i, i \in Q$ computes a pair of challenges $(\epsilon_1, \epsilon_2) = (\text{hash}(\mathcal{G}, \mathcal{H}, 1), \text{hash}(\mathcal{G}, \mathcal{H}, 2))$.
3. $P_i, i \in Q$ computes and outputs $\delta_i = [\alpha_i - \epsilon_1 w_{1i} - \epsilon_2 w_{2i}]_q$. P_i computes $\delta = \sum_{j \in Q} \delta_j$. The triple $(\mathcal{G}, \mathcal{H}, \delta)$ is output.

Part II:

1. $P_i, i \in Q$ selects $\beta_{1i}, \beta_{2i} \in_u Z_q$, computes and outputs $(F_i, M_i) = (g^{\beta_{1i}} h^{\beta_{2i}}, \mu_1^{\beta_{1i}} \mu_2^{\beta_{2i}})$. P_i computes $(F, M) = (\prod_{j \in Q} F_j, \prod_{j \in Q} M_j)$, and $(\mathcal{F}, \mathcal{M}) = (\text{hash}(F), \text{hash}(M))$.
2. $P_i, i \in Q$ computes a challenge $e = \text{hash}(\mathcal{F}, \mathcal{M})$.
3. $P_i, i \in Q$ computes and outputs $(d_{1i}, d_{2i}) = ([\beta_{1i} - e w_{1i}]_q, [\beta_{2i} - e w_{2i}]_q)$. P_i computes $(d_1, d_2) = (\sum_{j \in Q} d_{1j}, \sum_{j \in Q} d_{2j})$. The quadruple $(\mathcal{F}, \mathcal{M}, d_1, d_2)$ is output.

Verification and Tracing (by provers):

1. P_i verifies that $\mathcal{G} = \text{hash}([g^\delta z_1^{\epsilon_1} z_2^{\epsilon_2}]_p)$, $\mathcal{H} = \text{hash}([h^\delta \bar{z}_1^{\epsilon_1} \bar{z}_2^{\epsilon_2}]_p)$, $\mathcal{F} = \text{hash}([g^{d_1} h^{d_2} (z_1 \bar{z}_2)^e]_p)$ and $\mathcal{M} = \text{hash}([\mu_1^{d_1} \mu_2^{d_2} \sigma^e]_p)$. If this holds, P_i accepts the transcript, otherwise he proceeds:
2. For all $j \in Q$, P_j is replaced if one of the following equations is not satisfied: $G_j = g^{\delta_j} z_{1j}^{\epsilon_1} z_{2j}^{\epsilon_2}$, $H_j = h^{\delta_j} \bar{z}_{1j}^{\epsilon_1} \bar{z}_{2j}^{\epsilon_2}$, $F_j = g^{d_{1j}} h^{d_{2j}} (z_{1j} \bar{z}_{2j})^e$, $M_j = \mu_1^{d_{1j}} \mu_2^{d_{2j}} \sigma^e$.

The generated transcripts are identical to those of the single-server case, and thus, the verification (by the verifier) is identical to what was previously presented.