# Distributed Blinding for ElGamal Re-encryption*

Lidong Zhou†, Michael A. Marsh‡,
Fred B. Schneider§, and Anna Redz¶

January 2, 2004

**Abstract**

A protocol is given that allows a set of $n$ servers to cooperate and produce an ElGamal ciphertext encrypted under one key from an ElGamal ciphertext encrypted under another, but without plaintext ever becoming available. The protocol is resilient to $\lfloor (n-1)/3 \rfloor$ of the servers being compromised and requires no assumptions about execution speeds or message delivery delays. Two new building blocks employed—a distributed blinding protocol and verifiable dual encryption proofs—could have uses beyond re-encryption protocols.

*Supported in part by AFOSR grant F49620-00-1-0198 and F49620-03-1-0156, Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement number F30602-99-1-0533, National Science Foundation Grant 9703470, and a grant from Intel Corporation. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.

†Microsoft Research Silicon Valley, 1065 La Avenida, Mountain View, CA 94043; email: `lidongz@microsoft.com`.

‡UMIACS, University of Maryland, College Park, Maryland 20742; email: `mmarsh@umiacs.umd.edu`.

§Department of Computer Science, Upson Hall, Cornell University, Ithaca, New York 14853; email: `fbs@cs.cornell.edu`.

¶Department of Numerical Analysis and Computer Science, Royal Institute of Technology, Sweden; email: `anna@nada.kth.se`.

# 1 Introduction

A *re-encryption protocol* (a form of proxy cryptography [3, 21]) produces a ciphertext encrypted under one key from a ciphertext encrypted under another but without plaintext becoming available during intermediate steps. This rules out first decrypting the message and then encrypting it using another key.

The re-encryption protocol in this paper is designed for use by *distributed services*, each comprising a set of servers that work together to implement some service semantics unless a significant fraction of the servers become compromised. Here, secret sharing [27, 2] is typically employed to split the service private key among the servers, and threshold cryptography [5, 13] is used for cryptographic operations involving that private key. Instances of this architecture are found in COCA [31], e-vault [19], ITTC [29], Omega [26], and SINTRA [6].

Re-encryption is particularly useful in connection with distributed services because, in such systems, no single server can be trusted to have unencrypted secrets. So, for example, a re-encryption protocol would be employed if a secret stored by a service $A$ is being moved to some other service $B$, where each service stores its secrets encrypted under a different service public key. Our interest in the problem arose in connection with building an infrastructure to support publish/subscribe communications. In this application, a number of administrative domains each operated its own distributed service with separate cryptographic keys, and the need to securely transfer items between domains led to design the re-encryption protocol of this paper.

Blinding [8] is the core for our re-encryption protocol. An ElGamal encrypted [16] secret at service $A$ is blinded by a random *blinding factor*, then decrypted using $A$'s private key, and finally both encrypted using $B$'s public key and un-blinded using the original random blinding factor. A new *distributed blinding protocol* allows distributed services to perform the blinding and un-blinding; this distributed blinding protocol employs a new cryptographic building block called *verifiable dual encryption* to create proofs that, without disclosing the plaintext, certify two ciphertexts created under *different* public keys are (with high probability) for the same plaintext. We believe that both the distributed blinding protocol and the verifiable dual encryption protocol have uses outside of re-encryption protocols.

Since assumptions invariably translate into vulnerabilities (and opportunities for attackers), we eschew assumptions about execution speed and message delivery delays. So we instead adopt the *asynchronous* model of computation, which has no assumptions about timings. But deterministic solutions to the consensus problem cannot exist in such settings [18], and that creates challenges for the protocol designer who nonetheless must implement any required server coordination. In the protocols contained herein, selection and agreement on a blinding factor is avoided by instead computing multiple equivalent candidates along with a unique label for each; the labels allow a server to choose one of the blinding factors and have any subsequent computations by its peers be consistent with this choice.

The rest of the paper is organized as follows. Section 2 describes the system model. In Section 3, ElGamal encryption is reviewed and re-encryption by blinding is explained. Our distributed blinding protocol is the subject of Section 4. Section 5 discusses alternative re-encryption schemes and other related work. Appendix A sketches proofs for our protocol.

# 2 System Model

Consider a distributed service $S$ comprising $n$ servers, where each server has a unique public/private key pair with the public key known to the other servers, service public key $K_S$ is widely known, and service private key $k_S$ is distributed among the servers according to an $(n, f)$ secret sharing scheme.[1] Consequently, servers can communicate with each other securely and the service can, using threshold cryptography, perform decryption and generate digital signatures provided at least $f + 1$ servers cooperate.

We also make the following assumptions about the environment:

**Compromised Servers:** Servers are either *correct* or *compromised*. A compromised server might stop, deviate arbitrarily from its specified protocols (i.e., Byzantine failure), and/or disclose information stored locally. At most $f$ of the $n$ servers are compromised, where $3f + 1 = n$ holds.[2]

**Asynchronous System Model:** There is no bound on message delivery delay or server execution speed.

So an adversary can control the behavior of and obtain all information available to as many as $\lfloor (n-1)/3 \rfloor$ of the servers. Also, an adversary could conduct denial-of-service attacks that delay messages or slow down servers by arbitrary finite amounts. As customary, the capability of the adversary is limited to that of a probabilistic polynomial-time Turing machine.

## 3 ElGamal Re-encryption Using Blinding

ElGamal public key encryption is based on large prime numbers $p$ and $q$ such that $p = 2q + 1$. Let $\mathcal{G}_p$ be a cyclic subgroup (of order $q$) of $\mathbb{Z}_p^* = \{i \mid 1 \le i \le p-1\}$, and let $g$ be some generator of $\mathcal{G}_p$.

Any $k \in \mathbb{Z}_q^*$ can be an ElGamal private key, and then $K = (p, q, g, y)$ with $y = g^k \mod p$ is the corresponding public key. To simplify notation, modular calculations will henceforth be left implicit. Thus, "mod $p$" is omitted when computing exponentiations and discrete logarithms, and "mod $q$" is omitted when performing computation on exponents.

An ElGamal ciphertext $E(m)$ for plaintext $m \in \mathcal{G}_p$ is a pair $(g^r, my^r)$ with $r$ uniformly and randomly chosen from $\mathbb{Z}_q^*$. Ciphertext $E(m) = (a, b)$ is decrypted by computing $b/a^k$, since (for some $r$)

$$b/a^k \;=\; my^r/(g^r)^k \;=\; m(g^k)^r/(g^r)^k \;=\; m.$$

Where needed, we write $E(m, r)$ to indicate the value of $r$ used in computing $E(m)$ and we write $\mathcal{E}(m)$ to denote the set $\{E(m, r) \mid r \in \mathbb{Z}_q^*\}$ of all possible ciphertexts for $m$.

For $E(m_1) = (a_1, b_1)$, $E(m_2) = (a_2, b_2)$, and $E(m) = (a, b)$, define the following operations:

$$\begin{aligned} E(m)^{-1} &\equiv (a^{-1}, b^{-1}) \\ m' \cdot E(m) &\equiv (a, m'b) \\ E(m_1) \times E(m_2) &\equiv (a_1a_2, b_1b_2) \end{aligned}$$

The following properties then hold:

**ElGamal Inverse:** $E(m)^{-1} \in \mathcal{E}(m^{-1})$.

[1] By limiting the visibility of server public keys to only servers comprising this service, clients and other services are shielded from changes to these keys (including proactive refresh of private key shares) and shielded from changes to the composition of the service itself.

[2] The protocols are easily extended to cases where $3f + 1 < n$ holds.

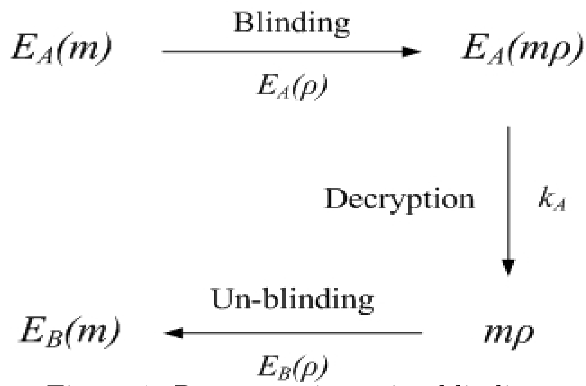


Figure 1: Re-encryption using blinding.

**ElGamal Juxtaposition:** $m' \cdot E(m, r) = E(m'm, r)$.

**ElGamal Multiplication:** $E(m_1, r_1) \times E(m_2, r_2) \in \mathcal{E}(m_1 m_2)$ if $r_1 + r_2 \in \mathbb{Z}_q^*$.

Note that side condition $r_1 + r_2 \in \mathbb{Z}_q^*$ in ElGamal Multiplication is easily checked without knowledge of $r_1$ or $r_2$. This is because

$$(a, b) = E(m_1, r_1) \times E(m_2, r_2) = (g^{r_1+r_2}, m_1 m_2 y^{r_1+r_2}),$$

so by checking that $a \neq 1$ holds, we conclude $r_1 + r_2 \neq 0$ which, by closure of group $\mathbb{Z}_q^*$, implies that $r_1 + r_2 \in \mathbb{Z}_q^*$ holds as well.

In those rare instances where $r_1 + r_2 = 0$ holds, plaintext $m_1 m_2$ is disclosed. This is not a concern for our protocols, because ElGamal Multiplication is used only in connection with random factors that are being multiplied to obtain a (random) encrypted blinding factor; new values can thus be requested whenever $r_1 + r_2 = 0$ is found to hold.[3] Our protocols omit such details, leaving implicit the checking of this side condition and any additional communications required to fetch suitable ElGamal encrypted values.

## Blinding and Un-blinding with ElGamal

Let $E_S(m)$ denote plaintext $m$ encrypted according to the public key $K_S$ of a service $S$ and let $D_S(c)$ denote ciphertext $c$ decrypted with the corresponding private key. Figure 1 summarizes how a service $A$ performs re-encryption using blinding and un-blinding. Each arrow is labeled by an operation (above) and its parameters (below). So we see that $E_A(m)$ is first blinded using $E_A(\rho)$, where $\rho$ is a random blinding factor; that result is decrypted to obtain $m\rho$; and finally $m\rho$ is unblinded using $E_B(\rho)$.

Figure 2 gives the actual protocol for re-encryption using blinding. Step 4 works because, letting $E_B(\rho)$ be $E_B(\rho, r)$ we have:

$$\begin{aligned}
& (m\rho) \cdot (E_B(\rho, r))^{-1} \\
= \;& \text{(ElGamal Inverse)} \\
& (m\rho) \cdot E_B(\rho^{-1}, -r) \\
= \;& \text{(ElGamal Juxtaposition)} \\
& E_B(m\rho\rho^{-1}, -r)
\end{aligned}$$

[3]The obvious denial of service attack of repeatedly requesting new values is prevented by accompanying such a request with evidence $E(m_1, r_1)$ and $E(m_2, r_2)$.

1. Pick a random blinding factor $\rho \in \mathcal{G}_p$; compute $E_A(\rho)$ and $E_B(\rho)$.
2. Compute blinded ciphertext $E_A(m\rho) := E_A(m) \times E_A(\rho)$.
3. Employ threshold decryption to obtain blinded plaintext $m\rho$ from blinded ciphertext $E_A(m\rho)$ computed in step 2.
4. Compute $E_B(m) := (m\rho) \cdot E_B(\rho)^{-1}$.

Figure 2: Re-encryption protocol executed by service $A$.

$$
\begin{aligned}
&= \text{(Cancellation)}\\
&\quad E_B(m, -r)\\
&\in \text{(definition of } \mathcal{E}_B(m))\\
&\quad \mathcal{E}_B(m)
\end{aligned}
$$

The possibility of compromised servers makes choosing $\rho$ and computing $E_A(\rho)$ and $E_B(\rho)$ in step 1 tricky to implement. A *distributed blinding protocol* to accomplish this task is the subject of the next section.

# 4 Distributed Blinding Protocol

We start by giving a protocol for a relatively benign environment; modifications for tolerating malicious attacks are then incorporated. This form of exposition, though perhaps a bit longer, elucidates the role played by each element of the protocol.

Given two related ElGamal public keys $K_A = (p, q, g, y_A)$ and $K_B = (p, q, g, y_B)$ with the same parameters $p$ and $g$, the distributed blinding protocol must satisfy the following correctness requirements.

**Randomness-Confidentiality:** Blinding factor $\rho \in \mathcal{G}_p$ is chosen randomly and kept confidential from the adversary.

**Consistency:** The protocol outputs a pair of ciphertexts $E_A(\rho)$ and $E_B(\rho)$ for blinding factor $\rho$.

## 4.1 Defending Against Failstop Adversaries

Replace Compromised Servers assumption by:

**Failstop Adversaries**: Compromised servers are limited to disclosing locally stored information or halting prematurely.[4] Assume at most $f$ out of $n$ servers are compromised, where $3f+1 = n$ holds.

Now to compute a confidential blinding factor $\rho$, it suffices to calculate $\prod_{i \in I} \rho_i$, where $I$ is a set of at least $f + 1$ servers and each server $i \in I$ generates a random *contribution* $\rho_i$. Confidentiality of $\rho$ follows because, with at most $f$ compromised servers, one server in $I$ is not compromised. This correct server picks a contribution that is random and unknown to the adversary; and the Failstop Adversaries assumption means all compromised servers necessarily select contributions that are independent of choices made by the correct servers.

[4]Thus, a failstop adversary is equivalent to an honest but curious server that can halt.

1. Coordinator $C_j$ initiates the protocol by sending to every server in $A$ an init message.

$$C_j \longrightarrow A : id, \mathsf{init}$$

2. Upon receipt of an init message from $C_j$, a server $i$:
   (a) Generates an independent random number $\rho_i$.
   (b) Computes encrypted contribution $(E_A(\rho_i), E_B(\rho_i))$.
   (c) $i \longrightarrow C_j : id, \mathsf{contribute}, i, E_A(\rho_i), E_B(\rho_i)$

3. Upon receipt of contribute messages from a set $I$ comprising $f+1$ servers in $A$:
   (a) $C_j$ computes: $E_A(\rho) = \times_{i \in I} E_A(\rho_i)$ and $E_B(\rho) = \times_{i \in I} E_B(\rho_i)$.
   (b) $C_j \longrightarrow A : id, \mathsf{finished}, E_A(\rho), E_B(\rho)$

Figure 3: Failstop Adversary Distributed Blinding Protocol

Ciphertext $E_A(\rho)$ can thus be obtained by calculating $\times_{i \in I} E_A(\rho_i)$, due to ElGamal Multiplication.[5] Similarly, ciphertext $E_B(\rho)$ can be obtained by calculating $\times_{i \in I} E_B(\rho_i)$. So a service $A$ can satisfy the confidentiality requirement for blinding factor $\rho$ if each server $i$ outputs as its *encrypted contribution* the ciphertext pair $(E_A(\rho_i), E_B(\rho_i))$.

To solicit encrypted contributions and then combine them into $E_A(\rho)$ and $E_B(\rho)$, we postulate a coordinator $C_j$ and (unrealistically) assume the server $j$ executing $C_j$ is never compromised:[6]

**Correct Coordinator**: Coordinator $C_j$ is correct.

We then have the distributed blinding protocol in Figure 3. There, we write

| | |
|---|---|
| $i \longrightarrow j : m$ | to specify that a message $m$ is sent by $i$ to $j$ |
| $i \longrightarrow A : m$ | to specify that a message $m$ is sent by $i$ to every server comprising service $A$ |

and $id$ identifies the *instance* of the protocol execution; $id$ contains, among other things, the identifier for the coordinator.

**Coping with Faulty Coordinators.** To eliminate the Correct Coordinator assumption, the protocol must tolerate coordinator disclosure of locally stored information or premature halting. Disclosure causes no harm, because the only locally stored information is the encrypted contributions from servers; to compute the blinding factor from these encrypted contributions, the adversary would have to know the private key of service $A$ or service $B$. A coordinator halting would prevent protocol termination, but this is easily tolerated by using $f+1$ different coordinators instead of just one. With $f+1$ coordinators, at least one will be correct and will complete the protocol. And if more than one coordinator is correct, then multiple blinding factors will be produced, which causes no difficulty.

[5] Use of ElGamal Multiplication to conclude $E_A(\rho) = E_A(\rho_1, r_1) \times E_A(\rho_2, r_2) \times \cdots \times E_A(\rho_{f+1}, r_{f+1})$ requires that $r_1 + r_2 + \cdots + r_{f+1} \in \mathbb{Z}_q^*$ hold. As before, this can be checked by seeing whether the first component of $E_A(\rho)$ equals 1 and soliciting new contributions if it does.

[6] This assumption is relaxed later in this section.

Employing multiple coordinators does imply a performance penalty. In the worst case, run-time costs are inflated by a factor of $f$, since as many as $f$ of the coordinators are superfluous. This cost, however, can be reduced by delaying when $f$ of the coordinators commence their execution. Since our protocol is designed for an asynchronous system, execution of coordinators can be delayed without adversely affecting correctness. So, one server acts as the *designated* coordinator and the others become coordinators only if the designated coordinator fails to complete execution within a specified period of time.

### 4.2 Defending Against Malicious Attacks

Relax the Failstop Adversaries assumption, returning to the original Compromised Servers assumption, and three noteworthy forms of misbehavior become possible.

- Servers choosing contributions that are not independent.
- The encrypted contribution from each server $i$ not being of the form $(E_A(\rho_i), E_B(\rho'_i))$ where $\rho_i = \rho'_i$.
- Servers and coordinators not following the protocol in other ways.

This section describes corresponding defenses.

**Randomness-Confidentiality.** Randomness-Confidentiality for the protocol of Figure 3 hinges on the contribution from at least one server being confidential and independent from contributions of all the others. It suffices to focus on a single run if, when engaging with different coordinators, a correct server selects random contributions that are independent. Unfortunately, even here a single compromised server can falsify the premise that its contribution is independent from the contributions of all other servers. That compromised server simply selects its contribution after seeing encrypted contributions from all other servers, exploiting the malleability of ElGamal encryption and choosing a contribution that cancels out the encrypted contributions from the other servers.

Specifically, a compromised server proceeds as follows to ensure that $\hat{\rho}$ becomes the blinding factor generated by the protocol. Suppose

$$\{(E_A(\rho_i), E_B(\rho_i)) \mid 1 \leq i \leq f\}$$

is the set of encrypted contributions received from the $f$ other servers at the start of step 3 in Figure 3. After receiving these, the compromised server generates two ciphertexts $E_A(\hat{\rho})$ and $E_B(\hat{\rho})$ and constructs as its encrypted contribution:

$$\left(E_A(\hat{\rho}) \times (\times_{i=1}^{f} E_A(\rho_i))^{-1},\ E_B(\hat{\rho}) \times (\times_{i=1}^{f} E_B(\rho_i))^{-1}\right) \tag{1}$$

Due to ElGamal Multiplication and ElGamal Inverse, the second factor in each element of this encrypted contribution will cancel the encrypted contributions from the other servers, so the resulting blinding factor is $\hat{\rho}$.

An obvious defense is to prevent servers that have not published an encrypted contribution from learning the encrypted contributions of others. So we modify the protocol of Figure 3 accordingly. Instead of sending an encrypted contribution to the coordinator, each server sends a *commitment*, which is a cryptographic hash (e.g., SHA1) of that encrypted contribution. And only after the coordinator has received $2f+1$ commitments does it solicit encrypted contributions from the servers.[7] Waiting for $2f+1$ commitments is necessary to ensure the coordinator will ultimately receive $f+1$ encrypted contributions, since as many as $f$ of the servers sending the $2f+1$ commitments could be compromised.

**Encrypted Contribution Consistency.** A compromised server might create an encrypted contribution that is not of the form $(E_A(\rho_i), E_B(\rho'_i))$ where $\rho_i = \rho'_i$ holds. Such *inconsistent* encrypted contributions cause the Consistency requirement for our distributed blinding protocol to be violated. Decrypting $E_A(\rho_i)$ and $E_B(\rho'_i)$ would be one way to check for inconsistent encrypted contributions, but having that plaintext would also undermine maintaining the confidentiality of $\rho$. So our protocol instead employs a new cryptographic building block called *verifiable dual encryption* that checks whether $\rho_i = \rho'_i$ holds given two ElGamal ciphertexts $E_A(\rho_i)$ and $E_B(\rho'_i)$.

Verifiable dual encryption is based on the non-interactive zero-knowledge proof, which we refer to as `DLOG`, for the equality of two discrete logarithms, as first proposed by Chaum and Pederson [9]. Given $a$, $g$, $X = g^a$, $Y$, and $Z = Y^a$, $\texttt{DLOG}(a, g, X, Y, Z)$ shows that[8] $a = \log_g X = \log_Y Z$ without disclosing $a$. (The protocols for `DLOG` are given in Appendix A.2.)

Consider an encrypted contribution $(E_A(\rho_i), E_B(\rho'_i))$ where

$$\begin{array}{rcccl} E_A(\rho_i) & = & (\delta_1, \gamma_1) & = & (g^{r_1}, \rho_i y_A^{r_1}) \\ E_B(\rho'_i) & = & (\delta_2, \gamma_2) & = & (g^{r_2}, \rho'_i y_B^{r_2}). \end{array}$$

corresponding to encryption using ElGamal public keys $K_A = (p, q, g, y_A)$ and $K_B = (p, q, g, y_B)$. We can show $\rho_i = \rho'_i$ holds by proving

$$\gamma_1/\gamma_2 = g^{k_A r_1 - k_B r_2} \tag{2}$$

because if $\rho_i = \rho'_i$ holds then

$$\gamma_1/\gamma_2 = (\rho_i y_A^{r_1})/(\rho'_i y_B^{r_2}) = (\rho_i/\rho'_i)(g^{k_A r_1}/g^{k_B r_2}) = g^{k_A r_1 - k_B r_2}.$$

Since $g^{k_A r_1 - k_B r_2} = g^{(k_A+k_B)(r_1-r_2)} g^{k_A r_2}/g^{k_B r_1}$ holds, equation (2) is satisfied if the following three conditions hold:

$$G_{12} = g^{k_A r_2} \tag{3}$$

$$G_{21} = g^{k_B r_1} \tag{4}$$

$$\gamma_1/\gamma_2 = g^{(k_A+k_B)(r_1-r_2)} G_{12}/G_{21} \tag{5}$$

Recall, a server that generates ciphertexts $E_A(\rho_i)$ and $E_B(\rho'_i)$ knows both $r_1$ and $r_2$, and thus is able to generate a *verifiable dual encryption proof*, denoted $\texttt{VDE}(E_A(\rho_i), E_B(\rho'_i))$, by constructing `DLOG` proofs for the conditions defined by equations (3) though (5).

$\texttt{VDE}(E_A(m), E_B(m))$ is obtained by showing:

[7] Here, we use the random oracle model [1], which has limitations [7]. A non-malleable [15] commit protocol (e.g., [12]) might be the basis for a scheme that ensures (informally speaking) server contributions are un-related with respect to any polynomial time relation. However, a non-malleable commit protocol would not by itself suffice, because this ensures the encrypted contributions are unrelated but not that the contributions themselves are unrelated. A non-malleable proof of plaintext knowledge [22] might be needed.

[8] Note, all operations are in domain $\mathbb{Z}_p$.

Pr1: $\texttt{DLOG}(r_2, g, g^{r_2}, y_A, G_{12})$ proves that $G_{12} = y_A^{r_2} = (g^{k_A})^{r_2}$ holds. Therefore, condition (3) is satisfied.

Pr2: $\texttt{DLOG}(r_1, g, g^{r1}, y_B, G_{21})$ proves that $G_{21} = y_B^{r_1} = (g^{k_B})^{r_1}$ holds. Therefore, condition (4) is satisfied.

Pr3: $\texttt{DLOG}(r_1 - r_2, g, g^{r_1 - r_2}, y_A y_B, (\gamma_1/\gamma_2)(G_{21}/G_{12}))$ proves that

$$(\gamma_1/\gamma_2)(G_{21}/G_{12}) = (y_A y_B)^{r_1 - r_2} = (g^{k_A + k_B})^{r_1 - r_2} = g^{(k_A + k_B)(r_1 - r_2)}$$

holds and therefore condition (5) is satisfied.

Thus, it suffices that every server $i$ attach $\texttt{VDE}(E_A(\rho_i), E_B(\rho_i))$ when sending encrypted contribution $(E_A(\rho_i), E_B(\rho_i))$ to the coordinator. The coordinator, in turn, only uses encrypted contributions that are accompanied by valid proofs—at least $f + 1$ will be, because at least $f + 1$ servers are correct out of the $2f + 1$ from which the coordinator received commitments.

**Constraining Malicious Coordinators.** It only remains to deal with compromised servers and coordinators that cause disruption by taking overt action. In a distributed system, such action is limited to sending messages.

We dealt above with two attacks that servers might launch through interaction with coordinators: (i) revealing encrypted contributions prematurely and (ii) sending inconsistent encrypted contributions. Compromised coordinators have corresponding attacks, and a compromised coordinator might:

- cause some servers to reveal encrypted contributions before other (presumably compromised) servers have selected theirs,
- fabricate an encrypted value for the blinding factor rather than computing that value from $f + 1$ encrypted server contributions.

For these and all attacks that involve sending bogus messages, we employ a single, general defense: each message sent is made *self-verifying* as in COCA [31], so that a receiver of the message can check whether the message is *valid*, based solely on message contents. A valid message is, by definition, one that is consistent with the sender following the protocol. Thus, if messages that are not valid are ignored then attacks involving bogus messages become indistinguishable from lost messages.

A message is made self-verifying by attaching *evidence* that establishes its plausibility. In general, it suffices that any message produced by a protocol step be signed by the sender and include as evidence all messages that served as the inputs to that protocol step, where these included messages are themselves self-verifying. For example, returning to the attacks mentioned above for compromised coordinators, messages might be made self-verifying as follows.

- The message requesting servers to reveal their encrypted contributions would be signed by the coordinator and include signed messages from $2f + 1$ servers containing the commitment for that server's encrypted contribution.
- The message conveying $(E_A(\rho), E_B(\rho))$ would be signed by the coordinator and also contain
  - signed messages from $2f + 1$ servers containing the hash of that server's encrypted contribution,
  - signed messages from $f + 1$ servers containing their encrypted contributions and corresponding valid verifiable dual encryption proofs.

1. Coordinator $C_j$ initiates protocol instance $id$ with an $\mathsf{init}$ message:

$$C_j \longrightarrow A : \langle id, \mathsf{init} \rangle_{C_j}$$

2. Upon receipt of a valid $\mathsf{init}$ message, a server $i$:

   (a) Generates an independent random value $\rho_i$.

   (b) Computes encrypted contribution $(E_A(\rho_i), E_B(\rho_i))$ and corresponding commitment $\kappa(E_A(\rho_i), E_B(\rho_i))$.

   (c) Replies to $C_j$:

$$i \longrightarrow C_j : \langle id, \mathsf{commit}, i, \kappa(E_A(\rho_i), E_B(\rho_i)) \rangle_i$$

3. Upon receipt of a set $M$ of valid $\mathsf{commit}$ messages from a set $I$ comprising $2f+1$ servers, $C_j$ requests the corresponding encrypted contributions.

$$C_j \longrightarrow A : \langle id, \mathsf{reveal}, M \rangle_{C_j}$$

4. Upon receipt from $C_j$ of a valid $\mathsf{reveal}$ message $R$ containing server $i$'s commitment, server $i$ responds:

$$i \longrightarrow C_j : \langle id, \mathsf{contribute}, i, R, (E_A(\rho_i), E_B(\rho_i)), \mathtt{VDE}(E_A(\rho_i), E_B(\rho_i)) \rangle_i$$

5. Upon receipt of a set $M'$ of valid $\mathsf{contribute}$ messages from a set $I' \subset I$ of $f+1$ servers, $C_j$:

   (a) Computes $E_A(\rho) := \bigtimes_{i \in I'} E_A(\rho_i)$

   (b) Computes $E_A(m\rho) := E_A(m) \times E_A(\rho)$

   (c) Invokes at service $A$ threshold decryption for $E_A(m\rho)$ with $M'$ included as evidence to make the decryption request self-verifying; obtains $m\rho$ and evidence $V^{id}_{m\rho}$ that the decryption result is correct.

   (d) Computes $E_B(\rho) := \bigtimes_{i \in I'} E_B(\rho_i)$

   (e) Computes $E_B(m) := (m\rho) \cdot (E_B(\rho))^{-1}$

   (f) Invokes at service $A$ threshold signature protocol on $(A, E_A(m), B, E_B(m))$, with $(m\rho, V^{id}_{m\rho})$ included as evidence to make the request self-verifying; obtains $\langle (A, E_A(m), B, E_B(m)) \rangle_A$.

   (g) $C_j \longrightarrow A : \langle id, \mathsf{done}, \langle (A, E_A(m), B, E_B(m)) \rangle_A \rangle_{C_j}$

Figure 4: Complete Re-encryption Protocol.

**Putting it Together.** Applying these defenses, we obtain the re-encryption protocol of Figure 4, where $\langle m \rangle_i$ denotes a message $m$ that is signed by $i$, and $\kappa$ is a cryptographic hash function. Criteria for validity of self-verifying messages used in the protocol are given in Figure 5. See Appendix A for the proof that this protocol works correctly in environments satisfying the Compromised Servers and Asynchronous System Model assumptions of §2.

# 5 Related Work

**Ciphertext Transformation.** Re-encryption protocols transform one ciphertext to another without ever revealing the plaintext. We are not the first to study the problem.

Mambo and Okamoto [23] introduced the notion of *proxy cryptosystems* to support delegation of decryption. In their scheme, $A$ can endow $B$ with the power to decrypt messages that have been

| type | check |
|---|---|
| init | The message is correctly signed. |
| commit | The message is correctly signed. |
| reveal | The message is (i) correctly signed and (ii) the messages it contains in $M$ are valid. |
| contribute | The message is (i) correctly signed, (ii) includes a valid verifiable dual encryption proof, and (iii) the encrypted contribution corresponds to the commitment in the included reveal message. |

Figure 5: Validity of Self-Verifying Messages

encrypted using public key $K_A$ but without disclosing to $B$ corresponding private key $k_A$. Delegation is accomplished by $A$ transforming a ciphertext encrypted under $K_A$ into another ciphertext that $B$ can decrypt; the transformed ciphertext is decrypted by using a *proxy key* that $B$ receives from $A$ when the proxy is initially set up. This is in contrast to our scheme, where re-encryption produces ciphertext under $B$'s public key.

Blaze, Bleumer, and Strauss [3] coined the term *atomic proxy cryptography*, which applies not only to encryption but also to other cryptographic operations (such as identification and signature). An atomic proxy encryption scheme involves an *atomic proxy function*, which converts ciphertexts for decryption by a first key into ciphertexts for a second key. The atomic proxy function is public, so any entity (even an untrusted one) can perform the transformation, making an encrypted message available to holders of the second key. With our re-encryption protocol, a distributed service $A$, which knows the first key (private key $k_A$), converts the ciphertext to the second key. And because $A$ is a distributed service, the individual servers of $A$ are not themselves trusted. Thus, a crucial difference between atomic proxy encryption and our re-encryption protocol concerns where trust is being placed.

**Jakobsson's Re-Encryption Scheme.** Jakobsson's quorum-controlled proxy re-encryption scheme [21], like ours, gives a way for a distributed service $A$ to transform $E_A(m)$ to $E_B(m)$ without disclosing $m$ to individual servers in $A$. The scheme leverages the observation that a ciphertext encrypted using $A$'s public key can first be encrypted using $B$'s public key, after which decryption using $A$'s private key yields a ciphertext under $B$'s public key.[9] Because Jakobsson's scheme also assumes a distributed service, the encryption and decryption operations are performed jointly by servers, with servers carrying out a partial encryption and a partial decryption (in parallel).

To ensure robustness of the scheme, a *translation certificate* is generated for $(E_A(m), E_B(m))$. This certificate is a non-interactive proof showing that $E_A(m)$ and $E_B(m)$ are encryptions of the same plaintext under public keys $K_A$ and $K_B$ respectively. Translation certificates are thus similar in function to our verifiable dual encryption proofs. The mechanisms differ, however, in what private information is known to a prover: For a translation certificate, the prover knows $A$'s private key and the random number used in the encryption to generate $E_B(m)$; for verifiable dual encryption, the prover does not know $A$'s private key but does knows both random numbers used in the encryption to generate $E_A(m)$ and $E_B(m)$.

[9] More precisely, given $A$'s public key $(p, q, g, y_A)$ and $B$'s public key $(p, q, g, y_B)$, consider a ciphertext $E_A(m, r) = (g^r, my_A^r)$. Encrypting $my_A^r$ using $B$'s public key produces $(g^{r'}, my_A^r y_B^{r'})$, and subsequent decryption using $A$'s private key yields $my_B^{r'}$. Note that $(g^{r'}, my_B^{r'}) = E_B(m, r')$ is a ciphertext of $m$ under $B$'s public key.

In Jakobsson's scheme, the translation certificate is not created locally by a single server—as our verifiable dual encryption proofs are—but instead constructed using a distributed protocol, because $E_B(m)$ is generated through a distributed threshold decryption scheme. This increases the complexity of the protocol and requires a synchronous model of computation. Whether the distributed protocol could be adapted to work in the less restrictive Asynchronous System Model of this paper is an open question. Furthermore, whereas the distributed protocol to generate Jakobsson's translation certificate and the threshold decryption protocol must be executed on service $A$, our scheme also supports shifting execution of the distributed blinding protocol to service $B$.[10] This shift is appealing for load balancing and for reducing the likelihood of denial-of-service attacks against $A$.

**Proactive Secret-Sharing.** A premise of our work is that encryption is being used to store secret information securely. An alternative is to use secret sharing [2, 27]. Rather than storing $E_A(m)$ on servers comprising $A$, now shares of $m$ are distributed among those servers.

- To retrieve secret information stored in this manner, a client establishes secure links to the servers and retrieves enough shares to reconstruct the secret. Verifiable secret sharing [10, 17, 25] allows correctness of the shares to be checked.

- To transmit the secret information from a service $A$ to a service $B$, a new, independent sharing of the secret information is constructed and distributed among the servers comprising $B$. Proactive secret sharing (PSS) protocols [20] are easily adapted to solve this problem, as shown in [14, 30].

The PSS-based solution does have advantages. Our re-encryption protocol is restricted to a particular public key cryptosystem (ElGamal) whereas the PSS-based solution imposes no such restrictions. Also, the PSS-based solution does not involve threshold cryptographic operations, thereby avoiding a complicated and expensive computation that is required with our re-encryption protocol.

The PSS-based solution, however, requires secure communication links between each server in $A$ and every server in $B$, so individual server public keys must be known outside of each service. Periodic refresh of server keys now becomes problematic. Our re-encryption protocol requires only that service public keys be known and, therefore, refresh is transparent outside the service. (Refreshing the service's private key shares does not change the service public key.)

Furthermore, in the presence of a mobile adversary [24], the PSS-based solution would require use of proactive secret sharing, periodically refreshing shares of all secret information the service stores. A service that stores a lot then incurs a significant recurring overhead. Our re-encryption protocol only involves one set of secret shares—the service private key—and thus the overhead of defending against mobile adversaries is considerably lower. In fact, it was this cost, in connection with the design of a publish/subscribe service, that prompted us to design a re-encryption protocol.


## Acknowledgments

We are grateful to Mark Lindermann for suggesting the problem and for discussions as this work progressed. We also benefited from helpful feedback from Dan Simon, Ilya Mironov, E. Gun Sirer, and Cynthia Dwork.


[10] If service $B$ is to execute distributed blinding protocol and send the results to $A$, then, at the end of the distributed blinding protocol, $B$ must initiate a threshold signature protocol to sign output $(E_A(\rho), E_B(\rho))$; a correct server in $B$ will participate in signing only if the output is verified against a set of at least $f+1$ valid contribute messages. Servers in $A$, which knows $B$'s service public key, can then verify the signature and be assured that $(E_A(\rho), E_B(\rho))$ is generated correctly by the distributed blinding protocol.

# A Correctness Proofs

## A.1 Properties of ElGamal Operations

**ElGamal Multiplication:** $E(m_1, r_1) \times E(m_2, r_2) \in \mathcal{E}(m_1 m_2)$ if $r_1 + r_2 \in \mathbb{Z}_q^*$.

*Proof:* If $r_1 + r_2 \in \mathbb{Z}_q^*$ then, by definition, $E(m_1 m_2, r_1 + r_2) \in \mathcal{E}(m_1 m_2)$ holds, so it suffices to prove $E(m_1, r_1) \times E(m_2, r_2) = E(m_1 m_2, r_1 + r_2)$:

$$\begin{aligned} E(m_1, r_1) \times E(m_2, r_2) &= (g^{r_1} g^{r_2}, m_1 y^{r_1} m_2 y^{r_2}) \\ &= (g^{r_1 + r_2}, m_1 m_2 y^{r_1 + r_2}) \\ &= E(m_1 m_2, r_1 + r_2). \end{aligned}$$

■

**ElGamal Inverse:** $E(m)^{-1} \in \mathcal{E}(m^{-1})$.

*Proof:* If $r \in \mathbb{Z}_q^*$ holds then so does $-r \in \mathbb{Z}_q^*$ and, by definition, $E(m^{-1}, -r) \in \mathcal{E}(m^{-1})$ holds. So it suffices to prove that $E(m, r)^{-1} = E(m^{-1}, -r)$ holds for every $r \in \mathbb{Z}_q^*$:

$$E(m, r)^{-1} = (g^{-r}, m^{-1} y^{-r}) = E(m^{-1}, -r).$$

■

**ElGamal Juxtaposition:** $m' \cdot E(m, r) = E(m'm, r)$.

*Proof:* $m' \cdot E(m, r) = (g^r, m'my^r) = E(m'm, r)$. ■

## A.2 Correctness of the `DLOG` protocol

We prove completeness, soundness, and zero-knowledge of the `DLOG` proof under the random oracle model [1]. `DLOG` is specified by:

**Public input:** $(p, q, g, X, Y, Z)$, a cryptographic hash function $H$
**Prover $P$'s private input:** $a = log_g(X)$

Prover $P$:

1. Select $s \in_R \mathbb{Z}_q$. Compute $U := g^s$ and $Q := Y^s$.
2. Compute $h := H(g, X, Y, Z, U, Q)$ using the hash function $H$.
3. Compute $w := s + ha$.
4. Publish $\texttt{DLOG}(g, X, Y, Z) = (U, Q, w)$ as the proof.

Verifier $V$:

1. Compute $h := H(g, X, Y, Z, U, Q)$
2. Verify the proof by checking the validity of the following equations:

$$g^w = UX^h \tag{6}$$

$$Y^w = QZ^h. \tag{7}$$

**Lemma A.2.1 (`DLOG` Completeness)** *Given $g$, $X = g^a$, $Y$, and $Z = Y^a$, a correct prover can always generate a* `DLOG` *proof that passes the verification (i.e., making Equations (6) and (7) hold).*

*Proof Sketch:* A correct prover always chooses $s$ and $w$, such that $w = s + ha$ holds. This ensures equations (6) and (7) hold because $UX^h = g^s(g^a)^h = g^{s+ha} = g^w$ and $QZ^h = Y^{s+ha} = Y^w$ hold. ■

**Lemma A.2.2 (`DLOG` Soundness)** *If a* `DLOG`$(a, g, X, Y, Z)$ *proof verifies then, with high probability,* $a = \log_g X = \log_Y Z$ *holds.*

*Proof Sketch:* Assume $X = g^a$, $Z = Y^{a'}$, $U = g^s$ and $Q = B^{s'}$ all hold, where $a \neq a'$ holds. For equations (6) and (7) to hold, the prover must ensure that $w = s+ha = s'+ha'$ holds. But $h$ is fixed once $a$, $a'$, $s$, and $s'$ are fixed. The probability of finding an $h$ such that both $h = H(g, X, Y, Z, U, Q)$ and $h = (s - s')/(a' - a)$ hold is negligible. ■

**Lemma A.2.3 (`DLOG` Zero-Knowledge)** `DLOG`$(a, g, X, Y, Z)$ *is a non-interactive zero-knowledge algorithm in the random oracle model (following the definitions in [1, 11]) under the decision Diffie-Hellman assumption [4].*

*Proof Sketch:* A simulator with access to a random oracle $\mathcal{O}$, but no access to $a$, can be constructed as follows.

1. Select $w, h \in_R \mathbb{Z}_q$.
2. Compute $U = g^w/A^h$ and $Q = B^w/C^h$.
3. Let $\mathcal{O}(g, X, Y, Z, U, Q) = h$ and output $(U, Q, w)$.

Note that each possible `DLOG` proof can be generated by the simulator. This is done by setting $w$ and $h$ to be the corresponding values created in that `DLOG` proof in step 1. Therefore, an adversary cannot distinguish a real `DLOG`$(a, g, X, Y, Z)$ proof from one generated by the simulator. Because the simulator does not know $a$, the adversary learns nothing about $a$ from the `DLOG` proof. ■

## A.3 Correctness of the `VDE` protocol

The correctness of the `VDE` protocol builds on the results of §A.2. The proofs of this subsection assume, in addition, that the three `DLOG` proof instances use independent hash functions so they can be modeled by three independent oracles. This independence assumption can be discharged by adding a different (but fixed) prefix to the data before applying the hash function.

**Lemma A.3.1 (`VDE` Completeness)** *A server generating* $E_A(\rho_i)$ *and* $E_B(\rho_i)$ *can always construct* `VDE`$(E_A(\rho_i), E_B(\rho_i))$*.*

*Proof Sketch*: This follows from `DLOG` Completeness (Lemma A.2.1) for the three `DLOG` sub-proofs comprising `VDE`$(E_A(\rho_i), E_B(\rho_i))$. ■

**Lemma A.3.2 (`VDE` Soundness)** *If* `VDE`$(X, Y)$ *verifies then, with high probability,* $X = E_A(\rho_i)$ *and* $Y = E_B(\rho_i)$ *hold for some* $\rho_i$*.*

*Proof Sketch*: This follows from `DLOG` Soundness (Lemma A.2.2) for the three `DLOG` sub-proofs in `VDE`$(E_A(\rho_i), E_B(\rho_i))$. ■

**Lemma A.3.3 (`VDE` Zero-Knowledge)** `VDE` *is a non-interactive zero-knowledge algorithm in the random oracle model under the decisional Diffie-Hellman assumption.*

*Proof Sketch*: Construct a simulator, using the simulators for the three DLOG proofs constructed in DLOG Zero-Knowledge (Lemma A.2.3). This is possible because any DLOG-simulator is free to choose its own $w$ and $h$, compute $U$ and $Q$, and set its random oracle accordingly without affecting the other DLOG-simulators.

The resulting simulator is able to produce the same proof as in the real VDE proof without the knowledge of $\rho_i$, $r_1$, or $r_2$, where $\rho_i$ is the plaintext, and $r_1$ and $r_2$ are the random numbers used for ElGamal encryption. ■

## A.4 Correctness of the re-encryption protocol

Correctness of the re-encryption protocol in Figure 4 is established by proving separate theorems for Progress, Integrity, and Confidentiality. These proofs are based on the following assumptions.

- Hash functions used in different steps of the protocol or by different servers are different and uncorrelated.
- Threshold decryption and threshold signature protocols terminate with correct results if and only if correct evidence, as specified by the re-encryption protocol, is provided upon invocation.
- The adversary is unable to decrypt a ciphertext encrypted under $A$'s public key without invoking the threshold decryption protocol in step 5(c).
- The adversary is unable to sign a message using $A$'s private key without invoking the threshold signature protocol in step 5(f).

Although the protocol of Figure 4 was designed for networks with reliable links, it is easily augmented to operate with unreliable, but fair, links.[11] The modifications involve adding message re-transmission along the lines discussed in [31].

**Theorem A.4.1 (Progress)** *Assuming reliable links between correct servers, the re-encryption protocol is guaranteed to terminate in step 5(g) of Figure 4 by sending to all servers in $A$ a* done *message that is signed by some coordinator $C_j$ and that contains a tuple $(A, E_A(m), B, E)$ signed by $A$ for some $E$.*

*Proof Sketch*: There are at least $f + 1$ coordinators, so at least one is correct. Thus, it suffices to show that the protocol instance with a correct coordinator $C_j$ completes step 5(g) in Figure 4.

The only steps where $C_j$ could block are step 3 (waiting for $2f + 1$ commit messages), step 5 (waiting for $f+1$ contribute messages), step 5(c) (invocation of threshold decryption), and step 5(f) (invocation of threshold signature). We show that $C_j$ will not block at any of these steps.

- $C_j$ never blocks at step 3 because there are at least $2f+1$ correct servers in distributed service $A$. These correct servers will respond to the init message sent by $C_j$ in step 1, because that init message will be valid. Therefore, due to reliable links, $C_j$ will eventually receive at least $2f + 1$ messages from correct servers, and these messages are guaranteed to be valid.
- $C_j$ never blocks at step 5 because, for the $2f+1$ servers that $C_j$ receives the commit messages from in step 3, at least $(2f+1)-f = f+1$ of these servers are correct. Due to Lemma A.3.1, every of these servers is able to generate a VDE proof for its contribution and thus produce a valid contribute message. Therefore, due to reliable links, $C_j$ is guaranteed to receive valid contribute messages from at least $f + 1$ servers.

[11] With unreliable but fair links, messages sent infinitely often are assumed delivered infinitely often.

- The threshold decryption protocol in step 5(c) always returns because the coordinator always provides $M'$ as the evidence. Similarly, the threshold signature protocol in step 5(f) always returns because the coordinator always provides the correct evidence. ■

**Theorem A.4.2 (Integrity)** *If the re-encryption protocol of Figure 4 sends*

$$\langle id, \mathsf{done}, \langle (A, E_A(m), B, E) \rangle_A \rangle_{C_j}$$

*in step 5(g), then $E \in \mathcal{E}_B(m)$ holds with high probability.*

*Proof Sketch*: For a re-encryption protocol instance to reach step 5(g), some coordinator $C_j$ (compromised or correct) must have received a set $M'$ of valid contribute messages, where $|M'| = f + 1$ holds. Otherwise, $C_j$ would not be able to complete step 5(c) because the invocation of the threshold decryption requires such an $M'$ as input. Without completing step 5(c), $C_j$ would not be able to complete step 5(f) because invocation of the threshold signature protocol requires the output from step 5(c) as evidence. Without executing step 5(f), an adversary is unable to produce a valid signature using $A$'s private key.

If every encrypted contribution provided in $M'$ is consistent (i.e., two ciphertexts are for the same plaintext under $K_A$ and $K_B$), then, due to ElGamal Multiplication, the purported $E_A(\rho)$ computed in step 5(a) and the purported $E_B(\rho)$ computed in step 5(d) are consistent. Then, the purported $E_B(m)$ computed in step 5(e) is indeed in $\mathcal{E}_B(m)$:

$$\begin{aligned}
& (m\rho) \cdot (E_B(\rho, r))^{-1} \\
= & \text{ (ElGamal Inverse)} \\
& (m\rho) \cdot E_B(\rho^{-1}, -r) \\
= & \text{ (ElGamal Juxtaposition)} \\
& E_B(m\rho\rho^{-1}, -r) \\
= & \text{ (Cancellation)} \\
& E_B(m, -r) \\
\in & \text{ (definition of } \mathcal{E}_B(m)) \\
& \mathcal{E}_B(m)
\end{aligned}$$

Therefore, the only way to violate Integrity is to introduce into $M'$ encrypted contributions that are not consistent. However, every contribute message in $M'$ must be valid, which implies that the VDE proof attached verifies. Due to Lemma A.3.2, the probability of inconsistency for the encrypted contribution is thus negligible. ■

**Theorem A.4.3 (Confidentiality)** *The re-encryption protocol of Figure 4 discloses no information about the plaintext $m$ under the random oracle model and the decision Diffie-Hellman assumption.*

*Proof Sketch*: It suffices to prove that if we are given transcript $t$ for an execution of the re-encryption protocol for $m$, then we can construct for any given $m' \in \mathcal{G}_p$ a transcript $t'$ of the re-encryption protocol for $m'$, and $t'$ is indistinguishable from $t$.[12] We only provide the intuition behind the construction here, rather than giving the formal details.

Let $\rho$ be the blinding factor chosen in transcript $t$. Given $m$ and $m'$, there exists $\rho' \in \mathcal{G}_p$, such that $m\rho = m'\rho'$ holds. Without loss of generality, assume that $\{\rho_i \mid 1 \leq i \leq f+1\}$ are contributions

[12] Because the contributions picked by each correct server for different coordinators are independent, the construction, hence the proof, can be applied to every instance, one for each coordinator.

selected in $M'$ (step 5) of transcript $t$ and that $\rho_1$ is from a correct server $c$. Pick $\rho'_1$, such that $\rho'_1 \prod_{2\leq i\leq f+1} \rho_i = \rho'$.

So, we construct a new transcript $t'$ where all servers do the same as in $t$ except that server $c$ picks $\rho'_1$ instead of $\rho_1$. Server $c$ changes its commitment in step 2 and its contribute message in step 4 accordingly.

Under the random oracle model, the commitments for $(E_A(\rho_1), E_B(\rho_1))$ and $(E_A(\rho'_1), E_B(\rho'_1))$ are indistinguishable. Therefore, there exists a transcript $t'$ where the compromised servers whose contributions are selected in step 5 behave the same as with the original transcript $t$ in step 2.

We can ignore inputs from other compromised servers because these never influence results revealed in the later steps of the protocol. This is guaranteed, because any contribute message with no corresponding commit message in the reveal message in step 3 will not be included in $M'$.

At step 4, server $c$ reveals the new ciphertext $(E_A(\rho'_1), E_B(\rho'_1))$ and the corresponding VDE proof. Because of the semantic security of ElGamal under decision Diffie-Hellman assumption [28] and because of Lemma A.3.3, the new contribute message is indistinguishable from the original one in $t$. Therefore, there exists a transcript $t'$ (with the already constructed prefix up to step 4), where the coordinator (correct or compromised) constructs the same $M'$ in step 5.

Again, the new transcript $t'$ for step 5 is indistinguishable from step 5 in $t$ because of $m\rho = m'\rho'$ and of the semantic security of ElGamal under decision Diffie-Hellman assumption.

Therefore, we can construct a transcript $t'$ for any given $m'$, so that it is indistinguishable to the adversary from the given transcript for $m$. This proves that the re-encryption protocol reveals nothing about $m$ under the stated assumptions. ■

# References


[1] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *The First Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[2] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, 48, pages 313–317. American Federation of Information Processing Societies Proceedings, 1979.

[3] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In K. Nyberg, editor, *Advances in Cryptology – Eurocrypt'98 (Lecture Notes in Computer Science 1403)*, pages 127–144, Espoo, Finland, May 1998. Springer-Verlag.

[4] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423, pages 48–63. Springer-Verlag, 1998.

[5] C. Boyd. Digital multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Clarendon Press, 1989.

[6] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the Internet. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2002)*, pages 167–176. IEEE, June 2002.

[7] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th annual ACM symposium on Theory of Computing*, pages 209–218. ACM Press, 1998.

[8] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto'82*, pages 199–203, 1983.

[9] D. Chaum and T. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology — Crypto '92 (Lecture Notes in Computer Science 576)*, pages 89–105, Santa Barbara, CA USA, August 1992. Springer-Verlag.

[10] B. Chor, S. Goldwasser, S. Macali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneous broadcast. In *Proceedings of the 26th Symposium on Foundations of Computer Science*, pages 335–344, 1985.

[11] R. Cramer, I. Damgård, and B. Schoemakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *Advances in Cryptology — Crypto '94 (Lecture Notes in Computer Science 839)*, pages 174–187, Santa Barbara, CA USA, August 1994. Springer-Verlag.

[12] G. D. Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and non-interactive non-malleable commitment. In *Advances in Cryptology — Eurocrypt 2001 (Lecture Notes in Computer Science 2045*, pages 40–59. Springer-Verlag, May 2001.

[13] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology — Crypto '89 (Lecture Notes in Computer Science 435)*, pages 307–315, Santa Barbara, California, U.S.A., August 1990. Springer-Verlag.

[14] Y. Desmedt and S. Jajodia. Redistributing secret shares to new access structures and its applications. Technical Report ISSE_TR-97-01, George Mason University, July 1997.

[15] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[16] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[17] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, pages 427–437, 1987.

[18] M. J. Fischer, N. A. Lynch, and M. S. Peterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, April 1985.

[19] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin. Secure distributed storage and retrieval. In *Proc. 11th International Workshop on Distributed Algorithms (WDAG '97), LNCS (1320)*, pages 275–289, 1997.

[20] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In D. Coppersmith, editor, *Advances in Cryptology — Crypto '95 (Lecture Notes in Computer Science 963)*, pages 457–469, Santa Barbara, California, U.S.A., August 1995. Springer-Verlag.

[21] M. Jakobsson. On quorum controlled asymmetric proxy re-encryption. In H. Imai and Y. Zheng, editors, *Public Key Cryptography, Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography (PKC'99)*, volume 1560 of *Lecture Notes in Computer Science*, pages 112–121, Berlin, Germany, 1999. Springer-Verlag.

[22] J. Katz. Efficient and non-malleable proofs of plaintext knowledge and applications. Cryptology ePrint Archive (http://eprint.iacr.org/), 2002. 2002/027.

[23] M. Mambo and E. Olamoto. Proxy cryptosystem: Delegation of the power to decrypt ciphertexts. *IEICE TRANS. Fundamentals*, E80-A(1):54–63, January 1997.

[24] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[25] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology — Crypto'91 (Lecture Notes in Computer Science 576)*, pages 129–140, Santa Barbara, California, U.S.A., August 1992. Springer-Verlag.

[26] M. K. Reiter, M. K. Franklin, J. B. Lacy, and R. N. Wright. The Ω key management service. *Journal of Computer Security*, 4(4):267–297, 1996.

[27] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[28] Y. Tsiounis and M. Yung. On the security of El Gamal based encryption. In H. Imai and Y. Zheng, editors, *Public Key Cryptography, Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptography (PKC'98)*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134, Pacifico Yokohama, Japan, February 1998. Springer-Verlag.

[29] T. Wu, M. Malkin, and D. Boneh. Building intrusion tolerant applications. In *Proceedings of the 8th USENIX Security Symposium*, pages 79–91, 1999.

[30] L. Zhou, F. B. Schneider, and R. van Renesse. APSS:proactive secret sharing for asynchronous systems. TR 2002-1877, Computer Science Department, Cornell Univerisity, Ithaca, NY USA, October 2002.

[31] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.