Distributed Blinding for Distributed ElGamal Re-encryption

Lidong Zhou
Microsoft Research Silicon Valley
Mountain View, CA
lidongz@microsoft.com

Michael A. Marsh
Institute for Advanced Computer Studies
University of Maryland
mmarsh@umiacs.umd.edu

Fred B. Schneider
Department of Computer Science
Cornell University
fbs@cs.cornell.edu

Anna Redz
Department of Numerical Analysis
and Computer Science
Royal Institute of Technology, Sweden
anna@nada.kth.se

Abstract

A protocol is given to take an ElGamal ciphertext encrypted under the key of one distributed service and produce the corresponding ciphertext encrypted under the key of another distributed service, but without the plaintext ever becoming available. Each distributed service comprises a set of servers and employs threshold cryptography to maintain its service private key. Unlike prior work, the protocol requires no assumptions about execution speeds or message delivery delays. The protocol also imposes fewer constraints on where and when various steps are performed, which can bring improvements in end-to-end performance for some applications (e.g., a trusted publish/subscribe infrastructure.) Two new building blocks employed—a distributed blinding protocol and verifiable dual encryption proofs—could have uses beyond re-encryption protocols.

1 Introduction

Cryptographic protocols intended for distributed systems are usually evaluated in terms of quantitative measures, like number of messages exchanged or total computing time. Virtually no attention has been paid to supporting flexibility in when and where the protocol steps are executed. Yet there are applications where such *step flexibility* is useful, as we recently discovered in designing a trusted publish/subscribe application.

This application required an infrastructure for transferring secrets from publishers to subscribers through a collection of interacting *distributed services*. A distributed service comprises a set of servers that together implement some desired semantics provided not too many of the servers are compromised. Secret sharing [34, 2] is typically employed to split the service private key among the set of servers, and threshold cryptography [4, 13] is used for cryptographic operations involving that private key. Instances of this architecture are found in COCA [37], e-vault [21], ITTC [35], Omega [32], SINTRA [7], and CODEX [27].

We decided to employ a *re-encryption protocol* [23] so that one distributed service could propagate a secret (encrypted under its service public key) to another distributed service; re-encryption—a form of proxy cryptography [3]—produces a ciphertext encrypted under one key from a ciphertext encrypted under another but without plaintext becoming available during intermediate steps.[1] And a re-encryption protocol that admits step flexibility allows certain optimizations:

- Computation that does not depend on the secret being transferred can be performed beforehand and, therefore, moved out of the critical path so that it does not contribute to end-to-end latency.

- For a secret being sent from a single service to multiple recipients, computation that does not rely on the sender's private key can be relocated from the sender to the receivers, thereby alleviating a potential bottleneck at the sender.

[1] The requirement that the plaintext not be disclosed during re-encryption is crucial for distributed services, because individual servers storing the plaintext might become compromised. Notice that decrypting the ciphertext with the first private key and then encrypting it using the second public key is now precluded, though.

Extant re-encryption protocols (e.g., [23]) did not admit step flexibility, so we developed a new one that does; it is the subject of this paper.

Blinding [9] is the core for our new 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. Use of the distributed blinding protocol supports flexible allocation of computation, because the distributed blinding protocol requires no knowledge of the original ciphertext or of A's private key. Consequently, the distributed blinding protocol can be executed before the original ciphertext is generated (thereby enabling pre-computation) and on servers other than A (thereby enabling offloading).

Our 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 conjecture that both the distributed blinding protocol and the verifiable dual encryption protocol have uses outside of re-encryption protocols.

Finally, since assumptions invariably translate into vulnerabilities (and opportunities for attackers), we eschewed assumptions about execution speed and message delivery delays in designing our protocols for the publish/subscribe application. So, unlike prior work in re-encryption, we adopted 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, followed by concluding remarks in Section 6.

2 System Model and Problem Definition

Each distributed service S comprises n servers along with a widely known service public key K_S and a service private key k_S that is distributed among the servers according to an (n, f) threshold cryptography scheme. Furthermore, each server is assumed to have a unique public/private key pair, with the public key known to the other servers.[2] Servers thus 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 assume:

> **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.[3]
>
> **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 $\lfloor (n-1)/3 \rfloor$ servers. Also, an adversary could conduct denial-of-service attacks that delay messages or slow down servers by arbitrary finite amounts. As is customary, the capability of the adversary is limited to that of a probabilistic polynomial-time Turing machine.

A re-encryption protocol for conveying a secret m from one distributed service A to another distributed service B must ensure that neither confidentiality nor integrity of m is compromised. In particular, given a ciphertext for m encrypted under K_A, a re-encryption protocol must produce an output subject to the following criteria (which are formalized in [36]):

> **Progress:** The protocol must terminate with correct servers in B having received the output.
>
> **Integrity:** The output produced by the re-encryption protocol is a ciphertext of m encrypted under K_B.
>
> **Confidentiality:** The protocol discloses no information about the plaintext m.

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 $\mathbb{Z}_p^* = \{i \mid 1 \leq i \leq p - 1\}$, and let

[2] By limiting the visibility of server public keys to only servers comprising the 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.

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

g be some generator of $\mathcal{G}_p$. We assume that all ElGamal keys share the same parameters p, q, g, and $\mathcal{G}_p$.

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:

$$E(m)^{-1} \equiv (a^{-1}, b^{-1})$$

$$m' \cdot E(m) \equiv (a, m'b)$$

$$E(m_1) \times E(m_2) \equiv (a_1 a_2, b_1 b_2)$$

The following properties then hold:

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

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

ElGamal Multiplication:[4] If $r_1 + r_2 \in \mathbb{Z}_q^*$ then $E(m_1, r_1) \times E(m_2, r_2) \in \mathcal{E}(m_1 m_2)$.

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

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

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.[5] Our protocols omit such details, leaving implicit the checking of this side condition and any additional communications required to fetch suitable ElGamal encrypted values.

[4]This property is often referred to as the *homomorphic* property of a public key cryptosystem.

[5]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 $E(m_2, r_2)$.

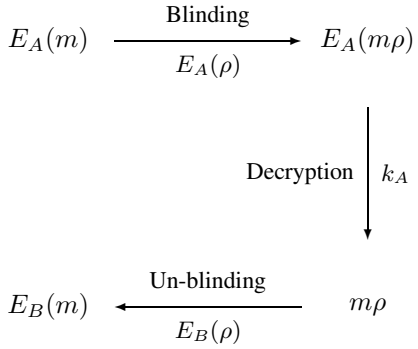

Figure 1: Re-encryption using blinding.

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 to perform re-encryption using blinding and un-blinding. Each arrow is labeled by an operation and its parameters. So we see that $E_A(m)$ is first blinded using $E_A(\rho)$, where ρ 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:

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

Note that step 1 can be performed by service B instead of service A and can be done before $E_A(m)$ is known. All other steps must be carried out after $E_A(m)$ is available. Step 3 must be executed on A because it requires knowledge of A's private key for threshold decryption. In the rest of the paper, we assume that step 1 is done by service B, although it could be easily done by A.

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

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.

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 $K_B = (p, q, g, y_B)$, 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 ρ.

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.[6] Assume at most f out of n servers are compromised, where $3f + 1 = n$ holds.

Now to compute a confidential blinding factor ρ, 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* ρ_i. Confidentiality of ρ 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.

[6]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 B an init message.

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

2. Upon receipt of an init message from C_j, a server i:
   (a) Generates an independent random number ρ_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 B:
   (a) C_j computes: $E_A(\rho) = \bigtimes_{i \in I} E_A(\rho_i)$ and $E_B(\rho) = \bigtimes_{i \in I} E_B(\rho_i)$.
   (b) $C_j \longrightarrow A : id, E_A(\rho), E_B(\rho)$.

Figure 3: Failstop adversary distributed blinding protocol.

Ciphertext $E_A(\rho)$ can thus be obtained by calculating $\bigtimes_{i \in I} E_A(\rho_i)$, due to ElGamal Multiplication.[7] Similarly, ciphertext $E_B(\rho)$ can be obtained by calculating $\bigtimes_{i \in I} E_B(\rho_i)$. So a service A can satisfy the confidentiality requirement for blinding factor ρ 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:[8]

> **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, $i \longrightarrow B : m$ to specify that a message m is sent by i to every server comprising service B, 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

[7]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.

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

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. The same techniques of using multiple coordinators were used in [37] and [6].

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 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 $\rho_i = \rho_i'$, and
- servers and coordinators not following the protocol in other ways.

This section describes corresponding defenses.

4.2.1 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 could proceed 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:

$$(E_A(\hat{\rho}) \times (\bigtimes_{i=1}^{f} E_A(\rho_i))^{-1}, \quad E_B(\hat{\rho}) \times (\bigtimes_{i=1}^{f} E_B(\rho_i))^{-1}) \quad (1)$$

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.[9] 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.

4.2.2 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 ρ. 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

[9] Here, we use the random oracle model [1], which has limitations [8]. A non-malleable [15] commit protocol (e.g., [12]) might be the basis for a scheme that ensures (informally speaking) server contributions are unrelated 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 [24] might be needed.

equality of two discrete logarithms, as first proposed by Chaum and Pedersen [10]. Given a, g, $X = g^a$, Y, and $Z = Y^a$, $\texttt{DLOG}(a, g, X, Y, Z)$ shows that[10] $a = \log_g X = \log_Y Z$ without disclosing a. (Protocols for DLOG are given in [36].)

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 verifying

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

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

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

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:

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

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

holds and therefore condition (5) is satisfied.

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

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.

4.2.3 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 or
- 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* [29, 25] as in COCA [37], 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*. 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.

4.2.4 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 κ is a cryptographic hash function. Criteria for validity of self-verifying messages used in the protocol are given in Figure 5. See [36] for the proof that this protocol works correctly in environments satisfying the Compromised Servers and Asynchronous System Model assumptions of Section 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 [26] 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 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 [23], like ours, gives a way for a distributed service A to transform

1. Coordinator C_j initiates protocol instance id with an init message:
$$C_j \longrightarrow B : \langle id, \mathsf{init} \rangle_{C_j}$$
2. Upon receipt of a valid init message, a server i:
   (a) Generates an independent random value ρ_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 commit messages from a set I comprising $2f + 1$ servers, C_j requests the corresponding encrypted contributions.
$$C_j \longrightarrow B : \langle id, \mathsf{reveal}, M \rangle_{C_j}$$
4. Upon receipt from C_j of a valid 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 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 $E_B(\rho) := \bigtimes_{i \in I'} E_B(\rho_i)$
   (c) Invokes at service B threshold signature protocol on $(id, \mathsf{blind}, A, E_A(\rho), B, E_B(\rho))$, with M' included as evidence to make the request self-verifying; obtains $\langle id, \mathsf{blind}, A, E_A(\rho), B, E_B(\rho) \rangle_B$.
   (d) $C_j \longrightarrow A : \langle id, \mathsf{blind}, A, E_A(\rho), B, E_B(\rho) \rangle_B$
6. Upon receipt of a valid $M'' = \langle id, \mathsf{blind}, A, E_A(\rho), B, E_B(\rho) \rangle_B$ from C_j, server l in service A:
   (a) Computes $E_A(m\rho) := E_A(m) \times E_A(\rho)$
   (b) 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.
   (c) Computes $E_B(m) := (m\rho) \cdot (E_B(\rho))^{-1}$
   (d) Invokes at service A threshold signature protocol on $(\mathsf{done}, A, E_A(m), B, E_B(m))$, with $(m\rho, V^{id}_{m\rho})$ included as evidence to make the request self-verifying; obtains $\langle \mathsf{done}, A, E_A(m), B, E_B(m) \rangle_A$.
   (e) $l \longrightarrow B : \langle \mathsf{done}, A, E_A(m), B, E_B(m) \rangle_A$

Figure 4: Complete Re-encryption Protocol.

type	check
init	The message is correctly signed.
commit	The message is correctly signed.
reveal	The message (i) is correctly signed and (ii) contains a set M of $2f+1$ different valid commit messages with a matching id.
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.
blind	The message is correctly signed.

Figure 5: Validity of self-verifying messages.

$E_A(m)$ 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.[11] 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). This dictates that the re-encryption must be done entirely by service A.

In contrast, by employing the distributed blinding protocol, our scheme allows a flexible allocation of computation both over time and in location. Only step 6 in Figure 4 needs to be performed on service A after $E_A(m)$ is available—this essentially involves only one threshold decryption operation. (The threshold signature operation in step 6(d) simply makes the result verifiable by servers in B.) To achieve such flexibility, our scheme has to employ a new building block for robustness, namely, verifiable dual encryption, whereas Jakobsson's scheme employs *translation certificates*. A translation 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. The two building blocks differ in what private information is known to a prover and hence require entirely different constructs: 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 know both random numbers used in the encryption to generate $E_A(m)$ and $E_B(m)$.

[11] 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.

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, 34]. 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 [11, 17, 30] 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 [22] are easily adapted to solve this problem, as shown in [19, 14].

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 [28], 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.

6 Concluding Remarks

Distributed services and distributed trust [31, 20, 5, 33] constitute a general architecture for extending state-machine replication to obtain a system that is not only fault-tolerant but also resists attacks. With new architectures

come new needs. The protocols described in this paper—a re-encryption protocol, a distributed blinding protocol, and verifiable dual encryption—were developed to satisfy those needs. But beyond the protocol details, a contribution of this work is to signal the importance of two non-traditional requirements for cryptographic protocols:

- Cryptographic protocols should assume the asynchronous (instead of the synchronous) model of computation, since the result will then be an intrinsic defense against denial of service and other forms of timing attacks.
- Cryptographic protocols should admit what we have termed step flexibility, since this provides ways to reduce overall latency, which can be important.

So in that sense, our new protocols should be seen as but one piece of a far bigger picture.

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, Markus Jakobsson, Moti Yung, Cynthia Dwork, Yacov Yacobi, and the anonymous reviewers.

This work is 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, Department of Defense CIPIA Grant F49620-01-1-0312, 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.

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

[5] C. Cachin. Distributing trust on the Internet. In *Proceedings of International Conference on Dependable Systems and Networks (DSN-2001)*, pages 183–192, Göteborg, Sweden, June 30–July 4 2001. IEEE Computer Society Technical Committee on Fault-Tolerant Computing, IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance, IEEE Computer Society.

[6] C. Cachin. An asynchronous protocol for distributed computation of RSA inverses and its applications. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing (PODC 2003)*, pages 153–162, Boston, MA, USA, July 2003. ACM.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[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] Y. Frankel, P. Gemmel, P. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proceedings of the 38th Symposium on Foundations of Com-*

puter Science, pages 384–393, Miami Beach, FL USA, October 20–22 1997. IEEE.

[20] Y. Frankel and M. Yung. Cryptosystems robust against "dynamic faults" meet enterprise needs for organizational "change control". In M. K. Franklin, editor, *Proceedings of the Third International Conference on Financial Cryptography, FC'99 (Lecture Notes in Computer Science 1648)*, pages 241–252, Anguilla, British West Indies, Feb 1999. Springer-Verlag.

[21] 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.

[22] 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.

[23] 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.

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

[25] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

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

[27] M. A. Marsh and F. B. Schneider. CODEX: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing*, 1(1):34–47, January-March 2003.

[28] 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.

[29] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[30] 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.

[31] M. K. Reiter. Distributing trust with the Rampart toolkit. *Communications of the ACM*, 39(4):71–74, April 1996.

[32] 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.

[33] F. B. Schneider and L. Zhou. Distributed trust: Supporting fault-tolerance and attack-tolerance. Technical Report TR 2004-1924, Computer Science Department, Cornell University, Ithaca, NY USA, January 2004.

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

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

[36] L. Zhou, M. A. Marsh, F. B. Schneider, and A. Redz. Distributed blinding for ElGamal re-encryption. Technical Report TR 2004-1920, Cornell University, January 2004.

[37] 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.