Verifiable Distributed Oblivious Transfer

Sheng Zhong Yang Richard Yang
Department of Computer Science, Yale University
{sheng.zhong, yang.r.yang}@yale.edu

Abstract

Distributed oblivious transfer (OT) [20] is an extension of traditional OT in which there is a group of *semi-honest* servers between the sender and the receiver. In this paper, we further extend the notion of distributed OT to consider *verifiable distributed OT*, in which the servers are potentially *malicious*. We design a protocol that protects the privacy of both the sender and the receiver under malicious attacks. Our protocol uses a novel technique to verify the consistency of encrypted secret shares and thus verify the correctness of the transfer. Our protocol also identifies cheating servers without compromising the privacy of either the sender or the receiver, using consistency verification through re-randomization.

Keywords: Oblivious Transfer, Verifiable Secret Sharing, Consistency Verification, Re-randomization

1 Introduction

Distributed OT and Verifiable Distributed OT Oblivious Transfer is a fundamental cryptographic primitive introduced by Rabin [24]. In [20], Naor and Pinkas introduced distributed OT, in which there is a group of servers between the sender and the receiver. In distributed OT, the servers are assumed to be *semi-honest* (honest-but-curious); that is, they follow the protocol when communicating with other participants but attempt to compromise the privacy of the sender or the receiver, possibly through collusion. The privacy of the sender and the receiver is protected under the assumption that the number of colluding servers is below a threshold. A distributed OT protocol has two stages: an initialization stage and a transfer stage. In the initialization stage, the sender distributes data derived from her[1] two private items among the servers, in such a secure way that no single server can learn any information about either private item. In the transfer stage, the receiver interacts with each of the servers in a quorum (*i.e.*, a threshold number of servers). Combining the data he gathers from all the servers in the quorum, the receiver can reconstruct one and only one of the sender's two private items.

In this paper, we extend the study of distributed OT to consider *malicious* servers, *i.e.*, servers that can actively deviate from a protocol and carry out malicious attacks. We require that the privacy of the sender and the receiver still be protected even under malicious attacks. In particular, we consider malicious attacks that intend to mislead the receiver to reconstruct his chosen item incorrectly. We require that the receiver be able to detect such malicious

[1] Hereafter, we use "she" to refer to the sender and "he" to the receiver.

attacks and accuse the cheating servers. In addition, to protect an honest server, we require that the receiver never be able to succeed in falsely accusing a server. To emphasize that the correctness of a transfer is verifiable by the receiver and that the correctness of a receiver's accusation is verifiable by the public, we call our new model *verifiable distributed OT*.

Why is the Problem Non-trivial? The problem of verifiable distributed OT may look trivial at a first glance, and one might suggest that the problem could be easily solved by a secret-sharing scheme with oblivious transfer of each share. However, there are two main technical challenges. First, the receiver must be able to verify the correctness of *both* items; otherwise, a malicious server could violate the receiver's privacy by tampering with its share of one item and observing whether or not this attack is detected by the receiver. However, at the same time, in order to protect the sender's privacy, the receiver should be able to reconstruct only one of the two items. In summary, our first challenge is to allow the receiver to reconstruct only one item but verify the correctness of both items. Second, when the receiver detects that some shares have been tampered with, the receiver must be able to identify the cheating servers and accuse them with evidence verifiable by the public. In addition, during the identification procedure, the privacy of both the sender and the receiver needs to be protected. In summary, our second challenge is to allow the receiver to identify and publicly accuse the cheating servers without compromising the privacy of either the sender or the receiver.

Protocol Overview Our protocol uses novel techniques to address the above two challenges. An overview of our protocol is as follows. During initialization, a Feldman Verifiable Secret Sharing (VSS) of keys is set up among the servers. An advantage of this setup is that the consistency of secret shares encrypted using ElGamal can be verified. Before each transfer, the sender distributes the shares of her both items (using a variant of the Shamir scheme) among the servers. During the transfer procedure, the receiver invokes the one-round OT protocol by Bellare and Micali [1, 5], with each server in a quorum (called *main servers*) in order to get the share of the item he chooses. Although the receiver can reconstruct only one item, he can verify the consistency of both items through the help of the remaining servers (called *verification servers*), because the encrypted shares of both items are transfered to the receiver during the OT. If cheating is detected, the receiver can perform the cheater-identification procedure, which uses the novel technique *cheater identification through re-randomization* and allows the receiver to identify the cheating servers but still protects the privacy of the sender and the receiver.

Related Work Oblivious transfer has been an active research area for many years, *e.g.*, [24, 9, 4, 18, 19]. As we discussed before, our verifiable distributed OT extends the notion of distributed OT [20]. However, distributed OT in [20] has an additional restriction that the receiver can only contact a quorum of servers in the transfer stage. Furthermore, the distributed OT protocol in [20] achieves more efficiency than traditional OT protocols. Therefore, the application of verifiable distributed OT will be different from that of distributed OT.

Private information retrieval (PIR) [6, 15], in particular, symmetric PIR (SPIR) [13], is similar to OT. One difference between SPIR and OT is that the former further requires small-communication overhead. Interestingly, Gertner *et al.* [12] introduced auxiliary servers to PIR, just as Naor and Pinkas introduced a group of servers to OT. However, in their model, the database itself is still involved in the protocol after the initialization stage, and the auxiliary servers may contain no information about the data at all. Therefore, it is significantly dif-

ferent from distributed OT and verifiable distributed OT. The relationship between OT and PIR/SPIR is further studied in [8].

Another related topic is Verifiable Secret Sharing (VSS), which can be used as a tool to detect cheating. Although various VSS schemes have been proposed, *e.g.*, in [22] and [25], our protocol employs the Feldman VSS [10]. Besides VSS, coding-theory-based techniques [17, 2] and cheating-immune secret sharing [23, 27] can also be used to detect cheating; however, the contexts of these schemes are different from this paper.

The problem of cheater identification looks like the Byzantine generals problem [16]. However, the context of our protocol is different. In particular, we require that the privacy of the sender and the receiver be protected in our procedure. Consequently, our solution is different from the solutions to the Byzantine generals problem.

Our Contributions Our main contributions can be summarized as follows. First, we extend the notion of distributed OT to consider malicious servers. Second, we develop a novel technique to achieve consistency verification of encrypted secret shares. Third, we identify malicious servers without compromising the privacy of either the sender or the receiver, using consistency verification through re-randomization. Our protocol is based on these novel techniques.

Organization of the Paper The rest of this paper is organized as follows. In Section 2, we formulate the problem of verifiable distributed OT. In Section 3, we address the first main challenge and present consistency verification of encrypted secret shares. A summary of our protocol (excluding the cheater-identification procedure) can also be found in this section. In Section 4, we address the second challenge and present the cheater-identification procedure. In Section 5, we give the security properties of our protocol. Finally, we conclude in Section 6.

2 Definitions

We formulate the problem of verifiable distributed OT by expanding the formulation in [20].

Let k be a security parameter. A verifiable distributed OT protocol involves a sender, a receiver, and a group of servers, $T_1, \ldots, T_t$, where $t = O(\log(k))$. Each of these parties is a probabilistic Turing machine. We assume that the computational power of each party is polynomially bounded in k. In the sequel, unless specified otherwise, when we say polynomial complexity, we mean polynomial in k. We assume an authenticated, untappable channel between the sender (resp., receiver) and each server. Let $s_0, s_1 \in Z_Q$ be the two items held privately by the sender, where Q is a prime of length k. Let $\sigma \in \{0, 1\}$ be a private input of the receiver.

A verifiable distributed OT protocol consists of an initialization stage and a transfer stage. In the initialization stage, the sender sends function $F_j : \{0,1\}^* \rightarrow \{0,1\}^*$ to each server T_j, where F_j depends on (s_0, s_1) and the sender's coin tosses. In the transfer stage, in order to learn s_σ, the receiver first carries out the share-transfer procedure by sending query q_j to server T_j and receiving reply $r_j = F_j(q_j)$ from T_j. Since the receiver may not send his queries all at once, q_j may depend on the replies to previous queries. After receiving replies from the servers, the receiver decides either to accept the replies (and reconstruct s_σ) or to reject the replies. In the latter case, he may further carry out the cheater-identification procedure and

interact with the servers in order to identify the cheater(s). We use μ to denote the receiver's view in the further interaction, if there is any. The receiver finally accuses cheating servers with evidence e.

We summarize the security requirements of a verifiable distributed OT protocol as follows. Below, when we say *high probability*, we mean probability p such that for any polynomial f, there exists k_0 such that, for any $k > k_0$, $p > 1 - \frac{1}{f(k)}$. When we say that two views are *computationally indistinguishable*, we mean that, if these two views are used as inputs to any *probabilistic polynomial time (PPT)* distinguisher, then the two outputs would have equal distributions with high probability.

Definition 1 *(*correctness*) A verifiable distributed OT protocol is* correct*, if there exists a PPT algorithm $\mathcal{R}$ for the receiver that computes s_σ from $(F_1(q_1), \ldots, F_t(q_t))$ when all parties follow the protocol.*

Definition 2 *(*receiver's privacy*) In a verifiable distributed OT protocol, σ is private against a coalition of the sender and τ_1 servers if, for any colluding group of a dishonest sender and τ_1 dishonest servers, there exists a PPT simulator S that can replace the receiver in the following sense: The view of the above colluding group interacting with S is computationally indistinguishable from the view of this group interacting with the receiver.*

Definition 3 *(*sender's privacy*) One of the two items, s_0 and s_1, is private against a coalition of the receiver and τ_2 servers if, for any colluding group of a dishonest receiver and τ_2 dishonest servers, there exists a PPT simulator S' that takes only one of the two items as input, and can replace the sender in the following sense: The view of the above colluding group interacting with S' and honest servers is computationally indistinguishable from the view of this group interacting with the sender and the honest servers.*

We consider two types of verifiability — *verifiability of reconstruction* and *verifiability of accusation*. For verifiability of reconstruction, we require that cheating be detected if it may lead the receiver to compute a false s_σ. On the other hand, if the cheating behavior of some servers does not affect correct reconstruction of s_σ, it will be unnecessary to detect it.

Definition 4 *(*verifiability of reconstruction*) A verifiable distributed OT protocol is reconstruction-verifiable if there exists a PPT algorithm for the receiver that accepts the replies $(r_1, \ldots, r_t)$ when $(r_1, \ldots, r_t) = (F_1(q_1), \ldots, F_t(q_t))$ and rejects when*

$$\mathcal{R}(r_1, \ldots, r_t) \neq \mathcal{R}(F_1(q_1), \ldots, F_t(q_t)).$$

Intuitively, when one or more servers try to cheat, an *accusation-verifiable* protocol allows the receiver to identify at least one of the cheating servers and show convincing evidence to the public. At the same time, an accusation-verifiable protocol does not allow the receiver to succeed in falsely accusing any server.

Definition 5 *(*verifiability of accusation*) A verifiable distributed OT protocol is accusation-verifiable if there exists a PPT cheater-identification algorithm for the receiver and a PPT*

verification algorithm for the public. If cheating is detected, the cheater-identification algorithm outputs a non-empty set of T_j such that $r_j \neq F_j(q_j)$ and computes e, evidence of cheating, from $(r_1, \ldots, r_t)$ and μ. The public's verification algorithm accepts e if and only if all servers accused by the receiver really cheated in the accused way.[2]

3 Consistency Verification of Encrypted Shares and the Share-Transfer Procedure

In this section, we address the first challenge and present the share-transfer procedure of our protocol. This section is organized as follows. We first review an adapted version of the Bellare-Micali OT protocol, which can be understood as transferring both items encrypted using ElGamal. Then we review a variant of the Shamir scheme and the Feldman VSS. Next we show how they can be adapted to verify the consistency of secret shares encrypted using ElGamal. Finally we present the share-transfer procedure of our protocol, which is based on the Bellare-Micali OT and the Feldman VSS.

In the sequel, let P, Q be two large primes such that $P = 2Q + 1$. Let G_Q be the quadratic residue subgroup of Z_P^*. Let g be a generator of G_Q. The standard Decisional Diffie-Hellman (DDH) assumption states that, for x, y, z picked uniformly at random from Z_Q, (g^x, g^y, g^z) is computationally indistinguishable from (g^x, g^y, g^{xy}) [3]. This implies that no PPT algorithm can compute the discrete logarithm of a random element in G_Q.

Bellare-Micali OT Assume that there exists a public random source. In this adapted version of Bellare-Micali OT, the receiver first picks $\delta \in G_Q$ using the public random source. Since the receiver has no control over the public random source, he does not know $\log_g \delta$, the discrete logarithm of δ. The receiver then picks $\beta \in Z_Q$ and sets $G_\sigma = g^\beta$, $G_{1-\sigma} = \delta / g^\beta$. Note that the receiver knows $\log_g G_\sigma$ but not $\log_g G_{1-\sigma}$. The receiver sends G_0, G_1, δ to the sender, along with a proof that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$, using a result by Cramer *et al.* [7]. The sender first verifies that δ has been chosen properly according to the public random source, and $\delta = G_0 G_1$. Then the sender computes, for $b = 0, 1$, $\hat{s}_b = s_b G_b^\kappa$, where $\kappa \in Z_Q$ is the sender's private key and g^κ her public key.

This OT protocol can be understood as transferring both shares in ElGamal ciphertext. Recall that in the ElGamal encryption scheme, which is semantically secure under the DDH assumption, when plaintext $s \in G_Q$ is encrypted with private key κ using random string $r \in Z_Q$, the ciphertext will be $(sg^{\kappa r}, g^r)$. In the Bellare-Micali OT above, $\hat{s}_b$ can be understood as the first element of the ElGamal ciphertext of s_b, encrypted using random string $\log_g G_b$. For convenience, hereafter we often refer to the first element of an ElGamal ciphertext as *the ciphertext*. In order to decrypt $\hat{s}_b$, a party not knowing κ (*e.g.*, the receiver) must know $\log_g G_b$, the random string used for encryption.

The sender gives both $\hat{s}_0$ and $\hat{s}_1$ to the receiver. Since $G_\sigma^\kappa = (g^\beta)^\kappa = (g^\kappa)^\beta$, the receiver can

[2] If for any accused T_j, $r_j = F_j(q_j)$, then obviously the verification algorithm should reject. But even if for each accused T_j, $r_j \neq F_j(q_j)$, the verification algorithm may or may not accept — it accepts only when e is valid evidence of the accused type of cheating. In other words, if a server cheats in one way and the cheater-identification algorithm falsely accuses the server of another type of cheating, the verification algorithm should still reject. The distinct types of cheating are described in Sections 3 and 4.

reconstruct s_σ by computing $s_\sigma = \hat{s}_\sigma/(g^\kappa)^\beta$, where g^κ is public and β is known to the receiver. However, since the receiver does not know $\log_g G_{1-\sigma}$, he cannot compute $s_{1-\sigma}$.

Since a server sends its two shares to the receiver as ElGamal ciphertexts, our first technical challenge can be addressed if we can verify the consistency of secret shares encrypted using ElGamal. Before presenting our scheme, we review a variant of the Shamir scheme and the Feldman VSS.

A Variant of the Shamir Scheme The variant of the Shamir scheme we use is defined on G_Q. It is constructed by applying the homomorphic mapping $\alpha \to g^\alpha$ to the original Shamir scheme on Z_Q. Therefore, with a (t, τ)-secret sharing of $s \in G_Q$, the jth share of the secret is $s_{(j)} = s \prod_{\theta=1}^{\tau-1} g^{a_\theta j^\theta}$ (for $j = 1, \ldots, t$),[3] where a_θ's are random coefficients. To reconstruct the secret, assuming that the known shares are $\{s_{(j)} | j \in J\}$ such that $|J| = \tau$, we can reconstruct s by the following formula (derived from the Lagrange interpolation):

$$s = \prod_{j \in J} s_{(j)}^{\prod_{\theta \in J}^{\theta \neq j} \frac{-\theta}{j-\theta}}.$$

Correctness of the above reconstruction assumes that the shares used have not been tampered with. We say a share is a *true share* if it has not been tampered with by its holder; otherwise, we say the share is a *false share*. If $\{s_{(j)} | j \in J\}$ are all true shares, and $s_{(\pi)}$ is also a true share, where $\pi \in \{1, 2, \ldots, t\} - J$, we have

$$\prod_{j \in J} s_{(j)}^{\prod_{\theta \in J}^{\theta \neq j} \frac{\pi-\theta}{j-\theta}} = s_{(\pi)}. \quad (1)$$

Definition 6 *In the above variant of the Shamir scheme, let $s_{(1)}, \ldots, s_{(t)}$ be the shares submitted by the servers,* which may or may not be true. *If (1) is satisfied, then we say that $s_{(\pi)}$ is consistent with $\{s_{(j)} | j \in J\}$; otherwise, we say that $s_{(\pi)}$ is inconsistent with $\{s_{(j)} | j \in J\}$.*

Lemma 7 *If $s_{(\pi)}$ is consistent with $\{s_{(j)} | j \in J\}$, then for any $j_0 \in J$, $s_{(j_0)}$ is consistent with $\{s_{(j)} | j \in J - \{j_0\} \cup \{\pi\}\}$, and the two sets of shares, $\{s_{(j)} | j \in J\}$ and $\{s_{(j)} | j \in J - \{j_0\} \cup \{\pi\}\}$, return the same s if used for reconstructing s.*

Remark Intuitively, this lemma indicates that exchanging $s_{(\pi)}$ with $s_{(j_0)}$ ($j_0 \in J$) does not break the consistency or change the reconstruction, provided that $s_{(\pi)}$ is consistent with $\{s_{(j)} | j \in J\}$. We omit the proof, which is simple elementary algebra.

Lemma 8 *One share is true if and only if it is consistent with any set of τ true shares.*

Remark This is also a simple fact that will be used to prove Lemma 9.

Lemma 9 *Assume that the number of false shares is less than $\frac{t-\tau}{2}$. For any $J \subseteq \{1, \ldots, t\}$, $|J| = \tau$, $\{s_{(j)} | j \in J\}$ is a set of true shares if and only if more than $\frac{t-\tau}{2}$ shares with indices outside J are consistent with $\{s_{(j)} | j \in J\}$.*

[3] We add a pair of parenthesis to the subscript of the jth share, $s_{(j)}$, in order to distinguish it from the two private items, s_0 and s_1.

Proof: If $\{s_{(j)}|j \in J\}$ is a set of true shares, then the left hand side of (1) computes the untampered value of $s_{(\pi)}$. Therefore, each true share $s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}$. Because there are fewer than $\frac{t-\tau}{2}$ false shares, there must be more than $\frac{t-\tau}{2}$ true shares with indices outside J. These true shares are all consistent with $\{s_{(j)}|j \in J\}$.

If more than $\frac{t-\tau}{2}$ shares with indices outside J are consistent with $\{s_{(j)}|j \in J\}$, let $J_0 = J \cup \{\pi|s_{(\pi)}$ is consistent with $\{s_{(j)}|j \in J\}\}$. Because $|J_0| > \tau + \frac{t-\tau}{2}$, and because there are fewer than $\frac{t-\tau}{2}$ false shares, $\{s_{(j)}|j \in J_0\}$ must include more than τ true shares. Take a set of τ true shares, and we know that all other shares with indices in J_0 are consistent with this set of τ true shares, by Lemma 7. Therefore, all shares with indices in J_0 are true, by Lemma 8. Specifically, $\{s_{(j)}|j \in J\}$ is a set of true shares. □

Remark Lemma 9 will be crucial to the proof of Lemma 10. It will also be useful in the proofs of our verifiability results, Claim 14 and Claim 15.

Feldman VSS With both Shamir's secret sharing and the above variant, we can view the Feldman VSS as a combination of the two schemes. Assume $s \in Z_Q$ is a secret shared in the Feldman VSS, and $s_{(j)}$ the jth share of s in the Feldman VSS, held by server T_j. Then $s_{(j)}$ is exactly the jth share of s in the Shamir scheme, while $g^{s_{(j)}}$, T_j's commitment to $s_{(j)}$, can be understood as the jth share of g^s in the variant of the Shamir scheme using the same random coefficients. As we will see shortly, this setup is important to the verification of the consistency of encrypted shares.

One way to set up the Feldman VSS among the t servers is to use a single trusted party. Note that if we use a trusted party to set up the Feldman VSS for a protocol, the protocol can be compromised if the trusted party is corrupted. Another possible solution without using a trusted party is to use a general-purpose protocol for secure multi-party computation, which is less efficient. However, we need to set up VSS only once, and then we can reuse the set-up for multiple transfers. Therefore, both of the solutions above will be acceptable. (Although Pedersen's protocol [21] was proposed to efficiently set up VSS without a trusted party, it has been pointed out to be flawed [11].)

Consistency Verification on Encrypted Shares With a setup of the Feldman VSS, we can check the consistency of secret shares encrypted using ElGamal. The following lemma articulates this fact.

Lemma 10 *Consider an ElGamal cryptosystem using generator g. Suppose that t servers use the Feldman VSS with threshold τ to share an implicit global private key κ. Each server T_j regards its share κ_j as its own private key, and uses the commitment to its share, g^{κ_j}, as its public key. Suppose that each server T_j encrypts plaintext $s_{(j)}$ using ElGamal, where $s_{(j)}$ is the (possibly-tampered) jth share of a secret s in the variant of the Shamir scheme using threshold τ. If all servers use the same random string $r \in Z_Q$ for encryption, then there exists a PPT algorithm that takes the ciphertexts of any $\tau + 1$ shares of s as input and outputs whether or not these shares are consistent.*

Proof: The shares are consistent if and only if (1) is satisfied. On the other hand, because κ_j is the untampered jth share of κ in the Feldman VSS, g^{κ_j} is the untampered jth share of g^{κ}

in the variant of the Shamir scheme. Therefore, g^{κ_j}'s are consistent, *i.e.*, they satisfy

$$\prod_{j\in J}(g^{\kappa_j})^{\prod_{\theta\in J}^{\theta\neq j}\frac{\pi-\theta}{j-\theta}} = g^{\kappa_\pi}. \tag{2}$$

Assume that the common random string used for encryption is r. Then (2) implies that (1) $\Leftrightarrow$

$$\prod_{j\in J}(s_{(j)}g^{\kappa_j r})^{\prod_{\theta\in J}^{\theta\neq j}\frac{\pi-\theta}{j-\theta}} = s_{(\pi)}g^{\kappa_\pi r}. \tag{3}$$

Because T_j's ciphertext is $(s_{(j)}g^{\kappa_j r}, g^r)$, (3) can be checked with only ciphertexts. □

Remark This is a central result that makes our verifiable distributed OT protocol possible.

In order to simplify formulae, we define

$$CShare(\{c_j|j\in J\},\pi) = \prod_{j\in J} c_j^{\prod_{\theta\in J}^{\theta\neq j}\frac{\pi-\theta}{j-\theta}}. \tag{4}$$

Therefore, for consistency verification on ciphertexts $\{c_j|j\in J\}$ and c_π, it is sufficient to compare $CShare(\{c_j|j\in J\},\pi)$ with c_π.

Building the Share-Transfer Procedure of our Protocol Given the above OT and secret sharing schemes, we address the first challenge as follows. The Feldman VSS of keys is established among the servers; the sender shares s_0^2, s_1^2 in the variant of the Shamir scheme[4] and distributes the shares among the servers; an instance of Bellare-Micali OT of shares is invoked between each server and the receiver; the receiver can only reconstruct one private item, but he can verify the consistency of the shares of either item, using the ciphertexts transfered.

In order to protect the privacy of the servers, we need to reduce the number of shares the receiver can learn. For this end, although there is no difference in the servers with regard to the shares, we distinguish between two classes of servers: the first τ servers, $T_1,\ldots,T_\tau$, are called *main* servers, while the rest servers are called *verification* servers. Only the main server's shares will be transfered to the receiver using the Bellare-Micali OT. Then the receiver computes the ciphertexts of the verification servers' shares from the ciphertexts of the main servers' shares, using (4), and sends them to the verification servers for comparison. The results of these comparisons reflect whether or not the shares are consistent.

We summarize the initialization stage and the share-transfer procedure of our protocol in Figure 1.

4 Cheater Identification and Accusation

In the previous section, we have presented the share-transfer procedure of our protocol and shown how the receiver verifies the consistency of the encrypted shares. In this section, we present a procedure that can further identify and publicly accuse at least one of the cheating servers.

[4]We use the bijection $\alpha\rightarrow\alpha^2$ to map the secret from Z_Q to G_Q. Note that there is an efficient algorithm to compute the modular square root because P is a prime.

Overall System-Initialization A Feldman VSS of keys as described in Lemma 10 is set up among $T_1, \ldots, T_t$.

Sender-Initialization The sender distributes the shares of s_0^2 and $s_1^2 \in G_Q$ (in the variant of Shamir's scheme with threshold τ), respectively, among $T_1, \ldots, T_t$.

The Share-Transfer Procedure

Step 1: The receiver picks $\delta \in G_Q$ uniformly at random according to the public random source. He also picks $\beta \in Z_Q$ uniformly at random, and computes $G_\sigma = g^\beta$ and $G_{1-\sigma} = \delta / g^\beta$. He sends query (G_0, G_1, δ) to each main server, along with a proof that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$.

Step 2: Each main server T_j first verifies that 1) there is no previous query from the receiver in this session; 2) δ is chosen properly according to the public random source; and 3) $\delta = G_0 G_1$. If all the three conditions are satisfied, T_j computes, for $b = 0, 1$,

$$\hat{s}_{b,j} = s_{b,j} G_b^{\kappa_j}, \tag{5}$$

and sends $(\hat{s}_{0,j}, \hat{s}_{1,j})$ to the receiver, together with a receipt of (G_0, G_1).

Step 3: Using the public key of main server T_j, the receiver computes a share of s_σ^2 by $s_{\sigma,j} = \hat{s}_{\sigma,j} / (g^{\kappa_j})^\beta$. Then the receiver computes

$$s_\sigma^2 = \prod_{j \in J} s_{\sigma,j}^{\prod_{\theta \in J}^{\theta \neq j} \frac{-\theta}{j-\theta}}, \tag{6}$$

where $J = \{1, \ldots, \tau\}$. He further computes s_σ from s_σ^2.

Step 4: The receiver computes, for $b = 0, 1$ and $\pi = \tau + 1, \ldots, t$, $\hat{s'}_{b,\pi} = CShare(\{\hat{s}_{b,j} | j \in J\}, \pi)$. Then he sends $(G_0, G_1, \hat{s'}_{0,\pi}, \hat{s'}_{1,\pi})$ to each verification server T_π.

Step 5: Each verification server T_π tests, for $b = 0, 1$,

$$\hat{s'}_{b,\pi} \quad = \quad s_{b,\pi} G_b^{\kappa_\pi}, \tag{7}$$

and sends the results of comparisons back to the receiver, together with a receipt of $(G_0, G_1, \hat{s'}_{0,\pi}, \hat{s'}_{1,\pi})$.

Step 6: If for both $b = 0$ and $b = 1$, more than half of the verification servers reply with "yes" (*i.e.*, reply that (7) holds), the receiver outputs s_σ and accuses those verification servers who reply with "no" for either $b = 0$ or $b = 1$. This is called "type-1 accusation" and the evidence consists of the replies from all servers. The protocol finishes successfully. If for either $b = 0$ or $b = 1$, the number of verification servers replying with "yes" is less than half (we require $t \not\equiv \tau \pmod 2$ in order to avoid a tie), a cheater-identification procedure is invoked (see Figure. 2).

Figure 1: The Initialization and Share-Transfer Procedure

Before the cheater identification, the receiver has already collected τ encrypted shares, $\{\hat{s}_{b,j} | j \in J\}$, for $b = 0, 1$. In addition, the consistency/inconsistency of each $\hat{s}_{b,\pi} = s_{b,\pi} G_b^{\kappa_\pi}$ ($\pi \notin J$) with $\{\hat{s}_{b,j} | j \in J\}$ has also been revealed by T_π, maybe incorrectly. In order to identify the cheaters, the receiver needs to check the consistency/inconsistency of $\hat{s}_{b,\pi}$ ($\pi \notin J'$) with $\{\hat{s}_{b,j} | j \in J'\}$, where $J' \subseteq \{1, \ldots, t\}$, $|J'| = \tau$ and $J' \neq J$. On the other hand, in order to protect the privacy of the sender, the servers should not reveal more shares to the receiver. That is, the above checking of consistency should not reveal any $\hat{s}_{b,\pi}$ for $\pi \notin J$ to the receiver.

Our solution to this problem is called *consistency verification through re-randomization.*

Consistency Verification through Re-Randomization The main idea of consistency verification through re-randomization is that each server T_j $(j = 1, \ldots, t)$ encrypts $s_{b,j}$ using $r' \in Z_Q$, a common random string *that is unknown to all parties.* Specifically, each server T_j $(j = 1, \ldots, t)$ calculates a new ciphertext of its share, $(s_{b,j} g^{\kappa_j r'}, g^{r'})$, and sends the new ciphertext to the receiver. The receiver can then check for consistency among the new ciphertexts.

In order to prevent servers from further tampering with their shares (or switching back from tampered shares to true shares), for $j \in J$, T_j should prove that its new ciphertext $\tilde{s}_{b,j}$, which is encrypted using random string r', is just a re-randomization of its previous ciphertext $\hat{s}_{b,j}$. For the same reason, for $\pi \notin J$, the receiver checks that the consistency/inconsistency regarding T_π's new ciphertext is the same as that regarding T_π's old ciphertext.

We summarize the cheater-identification procedure of our protocol in Figure 2.

5 Security Properties of the Protocol

Our verifiable distributed OT protocol has the following security properties:

Claim 11 *(correctness) This verifiable distributed OT protocol is correct.*

Proof: According to the definition of correctness, we need to show that there exists a PPT algorithm for the receiver to compute s_σ from the servers' responses when every party is honest. As described in the protocol, if all parties are honest, the response from each main server T_j is $(\hat{s}_{0,j}, \hat{s}_{1,j})$, which is computed from (5). By (5) we know

$$\begin{aligned} s_{\sigma,j} &= \hat{s}_{\sigma,j}/(G_\sigma)^{\kappa_j} \\ &= \hat{s}_{\sigma,j}/(g^\beta)^{\kappa_j} \\ &= \hat{s}_{\sigma,j}/(g^{\kappa_j})^\beta. \end{aligned}$$

Because g^{κ_j} is public, the receiver can easily compute $s_{\sigma,j}$ $(1 \le j \le \tau)$ from the (main) servers' responses. Because $s_{\sigma,1}, \ldots, s_{\sigma,\tau}$ are the shares of s_σ^2, the receiver can further reconstruct s_σ^2 from them, using formula (6). Since the modulus P is a prime number, the receiver can compute s_σ from s_σ^2 in polynomial time. □

Claim 12 *(receiver's privacy) σ is private against a coalition of the sender and all t servers.*

Proof: We constructs S as follows.

First, S picks $\delta, G_0 \in G_Q$, and sets $G_1 = \delta/G_0$. Then S sends (G_0, G_1, δ) to each main server, along with a proof that it knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$. Suppose that the response of the main server T_j to S is $(\hat{s}_{0,j}, \hat{s}_{1,j})$, where $(\hat{s}_{0,j}, \hat{s}_{1,j})$ may not have been computed properly because T_j may be dishonest. S computes $(\hat{s'}_{0,\pi}, \hat{s'}_{1,\pi})$, the query to each verification server T_π, by $\hat{s'}_{b,\pi} = CShare(\{\hat{s}_{b,j} | j \in J\}, \pi)$ and sends the query to the verification server.

Finally, if the majority of the verification servers reply with "yes" for both $b = 0$ and $b = 1$, S accuses the verification servers that reply with "no" for either $b = 0$ or $b = 1$. Otherwise, S invokes the cheater-identification procedure and behaves as if it were the receiver. Note that what S does in the cheater-identification procedure is based only on what the servers send to S, and thus is independent of whether S knows $\log_g G_0$ or $\log_g G_1$. □

The Cheater-Identification Procedure

Step 1: All the servers pick a common string, $G \in G_Q$, at random, according to the public random source. Each server T_j $(j = 1, \ldots, t)$ computes, for $b = 0, 1$,

$$\tilde{s}_{b,j} = s_{b,j} G^{\kappa_j},$$

and sends $\tilde{s}_{0,j}, \tilde{s}_{1,j}$ to the receiver. Each main server T_j proves to the receiver, in zero knowledge [26],

$$\log_{\frac{G}{G_b}} \frac{\tilde{s}_{b,j}}{\hat{s}_{b,j}} = \log_g g^{\kappa_j}. \tag{8}$$

Step 2: The receiver verifies all main servers' proofs. If a main server provides an invalid proof, the receiver aborts the protocol and accuses the main server. This is called "type-2 accusation" and the evidence is the invalid proof provided by the main server.

Step 3: The receiver verifies that, for $b = 0, 1$, and for each T_π $(\pi \notin J)$ that replied with "yes" for this b,

$$\tilde{s}_{b,\pi} = CShare(\{\tilde{s}_{b,j} | j \in J\}, \pi), \tag{9}$$

holds; he also verifies that, for $b = 0, 1$, and for each T_π $(\pi \notin J)$ that replied with "no" for this b, (9) does not hold. If the above is not true for any b and any T_π, the receiver aborts the protocol and accuses T_π. This is called "type-3 accusation" and the evidence for a type-3 accusation of T_π is all main servers' replies and proofs, and also T_π's reply together with $\tilde{s}_{b,1}, \ldots, \tilde{s}_{b,\tau}, \tilde{s}_{b,\pi}$.

Step 4: The receiver searches for a set $J' \subseteq \{1, \ldots, t\}$, $|J'| = \tau$ such that for $b = 0, 1$,

$$\tilde{s}_{b,\pi} = CShare(\{\tilde{s}_{b,j} | j \in J'\}, \pi) \tag{10}$$

holds for more than $\frac{t-\tau}{2}$ choices of $\pi \in \{1, \ldots, t\} - J'$.[a] The receiver accuses the T_π's for which Equation (10) does not hold for $b = 0$ or $b = 1$. This is called "type-4" accusation and the evidence consists of the replies from all servers, all main servers' proofs, $\tilde{s}_{b,1}, \ldots, \tilde{s}_{b,t}$, and set J'. After a type-4 accusation, the receiver uses

$$s_\sigma^2 = \prod_{j \in J'} s_{\sigma,j}^{\prod_{\theta \in J'}^{\theta \neq j} \frac{-\theta}{j-\theta}}$$

to compute s_σ^2, and then further computes s_σ. The protocol finishes successfully.

[a] The complexity of the search is exponential in t. However, since $t = O(\log(k))$, the complexity is still polynomial in k.

Figure 2: The Cheater-Identification Procedure

Claim 13 *(sender's privacy) Under the DDH assumption, one of the two items, s_0 and s_1, is private against a coalition of the receiver and any $\lfloor \frac{\tau-1}{2} \rfloor$ servers.*

Proof: We construct a simulator, S', for any given colluding group of a dishonest receiver and $\lfloor \frac{\tau-1}{2} \rfloor$ servers. Note that it is sufficient to consider a deterministic adversary that controls this group. (See page 22 of [14] for an argument of considering only deterministic adversaries.) Now we consider the possible strategies of this deterministic adversary.

To each honest main server T_j, the dishonest receiver will have to prove that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$. Therefore, there are only two possible

cases for a given strategy and a given honest main server T_j: (a) the receiver knows $\log_g G_0$ for the G_0 sent to T_j; (b) the receiver does not know $\log_g G_0$, and thus knows $\log_g G_1$ for the G_1 sent to T_j. For a given strategy, if the number of j's that falls into case (a) is greater than or equal to $\frac{\tau}{2}$, we define $\sigma' = 0$; otherwise, we define $\sigma' = 1$. Consequently, S', which takes $s_{\sigma'}$ as input, can be constructed as follows.

When the simulation begins, S' defines $s'_{\sigma'} = s_{\sigma'}$ and picks $s'_{1-\sigma'} \in Z_Q$ uniformly at random. S' computes the shares of ${s'_0}^2$ and ${s'_1}^2$, respectively, in the variant of the Shamir scheme, and distributes the shares among the servers, just as the sender distributes the shares of s_0^2 and s_1^2 in our protocol. Then the honest servers interact with the colluding group just as required in the protocol.

In order to see the computational indistinguishability of the views, we note the following fact. The shares of $s'_{1-\sigma'}$ can be divided into three classes: (1) at most $\frac{\tau}{2}$ shares of $(s'_{1-\sigma'})^2$ are multiplied by a value $G_{1-\sigma'}^{\kappa_j}$ with $\log_g G_{1-\sigma'}$ known to the adversary; (2) only $\lfloor \frac{\tau-1}{2} \rfloor$ shares of $(s'_{1-\sigma'})^2$ are held by the dishonest servers; (3) all other shares are either held by honest verification servers, or held by honest main servers and multiplied by a value $G_{1-\sigma'}^{\kappa_j} = (\frac{\delta}{G_{\sigma'}})^{\kappa_j}$ with $\log_g G_{\sigma'}$ known to the adversary. Because δ is picked according to the public random source, by the DDH assumption, the ciphertext of each share in the third class is indistinguishable from a uniformly random element. On the other hand, the total number of shares in the first two classes is less than τ, and a set of fewer than τ shares in the variant of Shamir scheme is indistinguishable from a set of uniformly random elements. □

Claim 14 *(verifiability of reconstruction)The protocol is reconstruction-verifiable if the number of dishonest servers is less than than* $\frac{t-\tau}{2}$.

Proof: See appendix. □

Claim 15 *(verifiability of accusation) The protocol is accusation-verifiable if the number of dishonest servers is less than* $\frac{t-\tau}{2}$.

Proof: See appendix. □

6 Conclusion

In this paper, we have considered the problem of verifiable distributed OT, in which the servers are potentially malicious. We design a protocol that allows the receiver to detect and expose the cheating behavior of malicious servers, but prevents him from falsely accusing any server. Our protocol also protects the receiver's privacy against all other parties, and protects the sender's privacy against a collusion of the receiver and up to $\lfloor \frac{t-\tau}{2} \rfloor$ servers.

Verifiable distributed OT can be further extended to consider a malicious sender, who may intentionally mislead the servers so that they can be accused of cheating. The protocol we have shown can be easily adapted to solve this problem, if revised slightly. We ignore this extension for simplicity.

The design of this protocol is aimed more at simplicity than at efficiency. We have ignored some minor optimizations that can slightly improve efficiency but may make the presentation less understandable.

References

[1] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557, 1990.

[2] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, 1968.

[3] Dan Boneh. The decision Diffie-Hellman problem. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63, 1998.

[4] Gilles Brassard, Claude Crepeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology - Proceedings of CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238, 1986.

[5] Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Muller. One-round secure computation and secure autonomous mobile agents. In *Automata, Languages and Programming, 27th International Colloquium,*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523, 2000.

[6] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–982, November 1998.

[7] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - Proceedings of CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[8] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138, 2000.

[9] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28:637–647, 1985.

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

[11] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, 1999.

[12] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *RANDOM'98*, volume 1518 of *Lecture Notes in Computer Science*, pages 200–217, 1998.

[13] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 151–160, 1998.

[14] Oded Goldreich. Secure multi-party computation. Working Draft Version 1.1, 1998.

[15] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[16] L. Lamport, R. Shostack, and M. Pease. The Byzantine generals problem. *ACM Transactions on Proggramming Languages and Systems*, 4(3):382–401, 1982.

[17] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[18] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 245–254, 1999.

[19] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *Advances in Cryptology - Proceedings of CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 573–590, 1999.

[20] Moni Naor and Benny Pinkas. Distributed oblivious transfer. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 205–219, 2000.

[21] T. Pedersen. A threshold cryptosystem without a trusted third party. In *Advances in Cryptology - Proceedings of EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, 1991.

[22] T. P. Pedersen. Non-interactive and information-theoretical secure verifiable secret sharing. In *Advances in Cryptology - Proceedings of CRYPTO 91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, 1992.

[23] Josef Pieprzyk and Xian-Mo Zhang. Cheating prevention in secret sharing over $GF(p^t)$. In *INDOCRYPT 2001*, volume 2247 of *Lecture Notes in Computer Science*, pages 79–90, 2001.

[24] M. Rabin. How to exchange secrets by oblivious transfer. *Tech. memo TR-81, Aiken Computation Laboratory, Havard U.*, 1981.

[25] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21th Annual ACM Symposium on the Theory of Computing*, pages 73–85, 1989.

[26] C. P. Schnorr. Efficient indentification and signatures for smart cards. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, 1990.

[27] Xian-Mo Zhang and Josef Pieprzyk. Cheating immune secret sharing. In *ICICS 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 144–149, 2001.

A Proofs of Verifiability

A.1 Proof of Claim 14

We construct an algorithm that accepts if no cheating is detected, and rejects otherwise. Obviously this algorithm accepts if all servers follow the protocol and reply to the receiver. Thus, it is sufficient to show that the algorithm rejects if the reconstruction of s_σ is incorrect. Equivalently, we show that the reconstruction is correct if no cheating is detected.

Let us extend the definition of $\hat{s}_{b,j}$ from the main servers to the verification servers. That is, we define, for $b = 0, 1$ and $\pi \notin J$, the untampered value of $\hat{s}_{b,\pi}$ as $s_{b,\pi} G_b^{\kappa_\pi}$. By Lemma 10, we know that, for $\pi \notin J$, the plaintext of $\hat{s}_{b,\pi}$ is consistent with the plaintexts of $\{\hat{s}_{b,j} | j \in J\}$ if and only if

$$CShare(\{\hat{s}_{b,j} | j \in J\}, \pi) = \hat{s}_{b,\pi}. \tag{11}$$

On the other hand, because the receiver is honest, $\hat{s'}_{b,\pi} = CShare(\{\hat{s}_{b,j} | j \in J\}, \pi)$. Therefore, the verification servers' testing of (7) is equivalent to the receiver's testing of (11) — if a verification server, T_π, claims that (7) holds (resp. does not hold), then we can imagine that, equivalently, T_π has provided the receiver with a value of $\hat{s}_{b,\pi}$ such that (11) holds (resp., does not hold); if T_π lies about the result of its comparison, we can equivalently imagine that T_π cheats in providing $\hat{s}_{b,\pi}$. Therefore, if no cheating is detected for $b = \sigma$, *i.e.*, if more than $\frac{t-\tau}{2}$ verification servers claim that (7) holds for $b = \sigma$, then (11) holds for more than $\frac{t-\tau}{2}$ choices of π when $b = \sigma$; that is, the plaintext of $\hat{s}_{\sigma,\pi}$ is consistent with the plaintexts of $\{\hat{s}_{\sigma,j} | j \in J\}$ for more than $\frac{t-\tau}{2}$ choices of π. By Lemma 9, this implies that the plaintexts of $\{\hat{s}_{\sigma,j} | j \in J\}$ are all true shares. In other words, these plaintexts are really $\{s_{\sigma,j} | j \in J\}$. This, in turn, ensures that the reconstruction is correct.

A.2 Proof of Claim 15

In the protocol description, we have shown the cheater-identification algorithm and also the evidence it will provide for each type of cheating. We hereby show a PPT verification algorithm (for the public) that decides to accept the evidence if the accused servers have cheated in the accused way, and to reject the evidence otherwise.

The algorithm works as follows. (For each type of accusation, we first describe what the algorithm does, and then briefly explain why the algorithm's decision is correct.)

- For a type-1 accusation, the algorithm first checks that every server has received the same G_0, G_1 and that the receiver's query to every verification server has been computed properly from the main servers' replies (recall that each server's reply contains a receipt of the query). Then, the algorithm accepts if and only if for $b = 0, 1$, more than $\frac{t-\tau}{2}$ verification servers' replies are "yes", and each accused verification server has replied with "no" for $b = 0$ or $b = 1$. (Recall that all communication is authenticated, *i.e.*, each reply has a signature.)
  - If more than $\frac{t-\tau}{2}$ verification servers have replied with "yes" for some b, the main servers' shares for this b are all true (see proof of Claim 14). Therefore, each verification server replying with "no" has cheated.

- For a type-2 accusation, the algorithm accepts if and only if each accused server has given an invalid zero-knowledge proof.

  – This is obvious.

- For a type-3 accusation, the algorithm first checks that every server has received the same G_0, G_1, that the receiver's query to each accused verification server has been computed properly from the main servers' replies, and that all proofs by the main servers are valid. Then the algorithm verifies that, for each accused verification server replying with "no" (resp., "yes"), (9) holds for that b (resp., does not hold for that b). The algorithm accepts if and only if all the above verify.

  – Suppose (9) holds. Then from (8), which has a zero-knowledge proof, we know (for $j \in J$)

  $$\tilde{s}_{b,j} = \hat{s}_{b,j}(\frac{G}{G_b})^{\kappa_j}. \tag{12}$$

  Plugging (12) into (9), we get

  $$s_{b,\pi} {G_b}^{\kappa_\pi} (\frac{G}{G_b})^{\kappa_\pi} = \prod_{j \in J} (\hat{s}_{b,j}(\frac{G}{G_b})^{\kappa_j})^{\prod_{\theta \in J}^{\theta \neq j} \frac{\pi - \theta}{j - \theta}}. \tag{13}$$

  Considering $\kappa_\pi = \sum_{j \in J} \kappa_j \prod_{\theta \in J}^{\theta \neq j} \frac{\pi - \theta}{\pi - j}$, we get

  $$s_{b,\pi} {G_b}^{\kappa_\pi} = \prod_{j \in J} \hat{s}_{b,j}^{\prod_{\theta \in J}^{\theta \neq j} \frac{\pi - \theta}{j - \theta}}, \tag{14}$$

  which means that T_π should have replied with "yes" instead of "no." Similarly, if (9) does not hold, T_π should have replied "no" instead of "yes."

- For a type-4 accusation, the algorithm first checks that every server has received the same G_0, G_1, that the receiver's query to each verification server has been computed properly from the main servers' replies, that all proofs by the main servers are valid, and that, for $b = 0, 1$ and for each verification server replying with "yes" (resp., "no") for b, (9) holds for that b (resp., does not hold for that b). Then the algorithm checks that, for $b = 0, 1$, more than $\frac{t-\tau}{2}$ choices of $\pi \in \{1, \ldots, t\} - J'$ satisfy (10), and for each accused server T_π, (10) is not satisfied for $b = 0$ or $b = 1$. The algorithm accepts if and only if all the above conditions are satisfied.

  – A type-4 accusation is similar to a type-1 accusation, except that here we have J' in the place of J, and $\tilde{s}_{b,j}$ in the place of $\hat{s}_{b,j}$ for $j = 1, \ldots, t$. Therefore the correctness of accusation is similar.