Traceable Secret Sharing and Applications

Vipul Goyal[1,2], Yifan Song[1], and Akshayaram Srinivasan[3]

[1] Carnegie Mellon University, Pittsburgh, USA
goyal@cs.cmu.edu, yifans2@andrew.cmu.edu
[2] NTT Research, Sunnyvale, USA
[3] Tata Institute of Fundamental Research, Mumbai, India
akshayaram.srinivasan@tifr.res.in

Abstract. Consider a scenario where Alice stores some secret data s on n servers using a t-out-of-n secret sharing scheme. Trudy (the collector) is interested in the secret data of Alice and is willing to pay for it. Trudy publishes an advertisement on the internet which describes an elaborate cryptographic scheme to collect the shares from the n servers. Each server who decides to submit its share is paid a hefty monetary reward and is guaranteed "immunity" from being caught or prosecuted in a court for violating its service agreement with Alice. Bob is one of the servers and sees this advertisement. On examining the collection scheme closely, Bob concludes that there is no way for Alice to prove anything in a court that he submitted his share. Indeed, if Bob is rational, he might use the cryptographic scheme in the advertisement and submit his share since there are no penalties and no fear of being caught and prosecuted. Can we design a secret sharing scheme which Alice can use to avoid such a scenario?

We introduce a new primitive called as *Traceable Secret Sharing* to tackle this problem. In particular, a traceable secret sharing scheme guarantees that a cheating server always runs the risk of getting traced and prosecuted by providing a valid evidence (which can be examined in a court of law) implicating its dishonest behavior. We explore various definitional aspects and show how they are highly non-trivial to construct (even ignoring efficiency aspects). We then give an efficient construction of traceable secret sharing assuming the existence of a secure two-party computation protocol. We also show an application of this primitive in constructing traceable protocols for multi-server delegation of computation.

1 Introduction

Secret sharing [Sha79, Bla79] allows a client to store a secret on n servers such that an authorized subset of the servers can recover the secret, while any unauthorized set learns no information about the secret. Now, consider a scenario where the client Alice stores her secret s (some proprietary dataset) across n different servers (or cloud providers) using secret sharing to enhance privacy. Alice divides her secret into n shares using (say) a t-out-of-n secret sharing scheme and stores one share on each server. Let us call these shares $\mathsf{share}_1, \ldots, \mathsf{share}_n$.

Trudy (the collector) is highly interested in learning Alice's secret and is willing to pay for it. Therefore, Trudy publishes an advertisement on the internet. The advertisement has an elaborate cryptographic scheme to collect shares from the servers. Each server who decides to submit its share is paid \$100. The collection scheme guarantees "cryptographic immunity" from being caught or prosecuted in a court (e.g., for violating its service agreement with Alice). The elaborate collection scheme has the following components: (1) a description of functions $f_1, \ldots, f_n$ (called as the collector's functions), and (2) description of a pirate reconstruction box $\mathsf{Rec}^\star$. The i-th server P_i is supposed to submit $f_i(\mathsf{share}_i)$ to the collector (in exchange for \$100).[4] If enough such $f_i(\mathsf{share}_i)$ are collected, the reconstruction box $\mathsf{Rec}^\star$ would output the secret s (or some information about s). The functions $\{f_i\}_{i\in[n]}$ and pirate reconstruction box $\mathsf{Rec}^\star$ are constructed very carefully to guarantee that even if Alice gets her hands on them (and even on $f_i(\mathsf{share}_i)$ for all i), it would not be possible for Alice to prove anything in a court and seek damages from any of the servers.

Bob is one of the servers and sees this advertisement. Competition from bigger cloud providers is tough, and, at this point, \$100 could really help Bob keep the service afloat and pay the staff salaries. Bob is worried however that if he gives out $f_i(\mathsf{share}_i)$ and somehow it reaches Alice, she will be able to trace him and sue him in a court for damages. This would surely mean bankruptcy given Bob's service agreement with Alice. However, upon examining the collection scheme and the reconstruction box closely, Bob concludes that there is no way for Alice to prove anything in a court even if he submitted $f_i(\mathsf{share}_i)$ (and it falls into Alice's hands). After all, Alice could have computed $f_i(\mathsf{share}_i)$ even on her own.

What if share_i was generated using a secure 2-party computation between Alice and Bob s.t. Alice doesn't know share_i? share_i could potentially even have identifying information about Bob. However we note that the function f_i may have been cleverly designed to remove this identifying information and only leave the "essence" of the share intact. In general, the function f_i might even encrypt share_i with a public key (s.t. only the reconstruction box has the corresponding secret key). The reconstruction box code may even be "obfuscated" in some way. Indeed if Bob is rational, he might submit $f_i(\mathsf{share}_i)$ to the collector to get \$100 since there are no penalties and no fear of being caught and prosecuted. After all, if he was the only one submitting the share, the collector anyway can get no information about Alice's secret. On the other hand, if a large number of servers are participating in the collection, Bob's does not want to be the one missing out on \$100.

The main goal of our paper is to try to design a secret sharing scheme in which the servers are held accountable for cheating. In particular, any server which cheats should run the risk of giving out a "proof of cheating" to the outside world. Given any collection scheme consisting of $f_1, \ldots, f_n$, the reconstruction box $\mathsf{Rec}^\star$, and the collected shares $\{f_i(\mathsf{share}_i)\}_{i\in M}$ where M is the set of malicious servers,

[4] To ensure that the server cannot claim a false reward by submitting f_i evaluated on some dummy value, the collector can presumably check the correctness of all the submitted values by, e.g., checking that they lie on a single polynomial.

Alice should be able to prove in front of a Judge that, for some i, P_i leaked its share. In other words, there does not exist a collection scheme which guarantees immunity to the cheating servers. We call such a secret sharing scheme a *traceable secret sharing*. The notion of traceable secret sharing seems to be relevant in natural scenarios such as secure multi-party computation in the client server model [IK00], and, in threshold cryptosystems [DF90, Fra90, DDFY94].

1.1 Our Results

We initiate the study of traceable secret sharing (TSS) and explore various definitional aspects. TSS schemes turn out to be highly non-trivial to construct even ignoring efficiency aspects. We first start with the high-level description of this primitive.

Definition. In a traditional threshold secret sharing scheme, there is a sharing phase where the dealer generates a set of n shares of his secret and distributes it to the servers. The reconstruction algorithm allows any set of t servers to come together to get back the secret. In a traceable secret sharing scheme, there are two additional algorithms, namely, Trace and Judge. At a high-level, the Trace algorithm uses the set of n collector functions $f_1, \ldots, f_n$, the collected shares $f_i(\mathsf{share}_i)$ (for all i), the pirate reconstruction box and the view of the dealer during the sharing phase. It outputs the identity of a traitor along with an evidence that this is indeed a traitor. This evidence is later analyzed by the Judge algorithm which pronounces whether the server is guilty or not. We assume that the honest servers never submit their shares and the malicious servers submit $f_i(\mathsf{share}_i)$. A way to model this (which we follow in this work) is to consider the collector's function corresponding to an honest server to be a constant function.

In addition to correctness and statistical privacy properties of a threshold secret sharing, we require a traceable secret sharing scheme to satisfy two additional properties. The first property is *traceability* which roughly states that if the pirate reconstruction box is able to distinguish between the shares of two different secrets with non-negligible advantage (where the probability is over the random coins of the sharing phase, random coins of the collectors functions and the internal coins of the reconstruction box), then the Trace algorithm, with non-negligible probability, outputs the identity of a traitor along with a valid evidence that is accepted by the Judge algorithm. The second property, called as *non-imputability*, protects an honest server against a cheating dealer. Roughly, this property requires that a cheating dealer, even if it colludes with every other party, cannot produce a valid evidence that implicates an honest server.

On the Model. We now make a couple of comments on the model.

- We require the Trace algorithm to take the description of the collector functions, the reconstruction box Rec^* as well as $\{f_i(\mathsf{share}_i)\}_{i \in M}$ submitted by the malicious servers as input. These components might be available to Alice if Trudy was Alice's agent, or if Trudy later sells them anonymously to Alice, or if Trudy gets caught by the law enforcement authorities and these

are submitted as evidence in the court of law. We note that if, for instance, $\{f_i(\mathsf{share}_i)\}_{i\in M}$ is not available to the trace algorithm, then there is no hope of identifying a traitor. Indeed, the reconstruction box does not have any secrets, and it is useless unless it is run on $\{f_i(\mathsf{share}_i)\}_{i\in[M]}$. This is, in fact, a key difference between traitor tracing (where the trace algorithm only requires access to the decryption box) and our notion of traceable secret sharing. We elaborate more on the differences between these two notions in Section 2.

- In this work, we consider a model where the collector specifies a set of functions $(f_1, \ldots, f_n)$ and asks the servers to submit $f_i(\mathsf{share}_i)$. However, it is possible to consider more general cases where the collector may ask the servers to run a distributed protocol and get the output of the protocol. Specifically, the collector and the servers might run a general MPC protocol that computes the reconstruction function and gives the output to the collector. We leave the study of such stronger models for future work. We note that in general, any tracing system (including broadcast encryption with traitor tracing) has its limitations and serves more as a deterrence rather than providing "foolproof security". In broadcast encryption with traitor tracing, the traitor might decrypt the broadcast and stream on an anonymous channel and then there is no hope of tracing the traitor. In spite of these limitations, traitor tracing has been widely deployed in practice (see, Fiat and Naor's ACM Paris Kanellakis Theory and Practice Award citation [ACM17]) and we take the first direction towards defining and constructing a similar primitive for the case of secret sharing.

Construction. In this work, we provide an efficient construction of traceable secret sharing scheme under standard cryptographic assumptions. Specifically, we show the following theorem:

Informal Theorem 1 *Assuming the existence of a secure two-party computation protocol, there exists an explicit construction of t-out-of-n threshold traceable secret sharing scheme for $t \geq 4$ in the PKI model.*[5] *In particular, for secrets of length λ,*

- *The construction satisfies statistical privacy.*
- *If there exists a set of n collector functions and a pirate reconstruction box that can distinguish between shares of two different secrets with advantage at least ϵ, then there exists a tracing algorithm that makes $\mathrm{poly}(\lambda, 1/\epsilon)$ oracle calls to the pirate reconstruction box and outputs the identity of a traitor along with a valid evidence with probability $\Omega(\frac{n\epsilon/(n-t+1)}{1+(n-1)\epsilon/(n-t+1)})$.*
- *With all but negligible probability, a (polynomially bounded) cheating dealer cannot provide a valid evidence against an honest party even if it colludes with every other party.*

Extensions. We also consider a couple of extensions to our setting of traceable secret sharing. The first extension is the collusion-resistant setting. Here, we

[5] See Remark 2 on why PKI is necessary for traceable secret sharing.

consider a scenario where a group of upto $t-1$ servers could come together and pool in their shares, apply a collector's function on their pooled shares and then submit the output. (Note that if we allow more than t servers to come together, then the servers could just reconstruct the secret without any collection, and TSS becomes meaningless.) We show that a simple modification to the construction from the above theorem actually satisfies this stronger definition. The second extension is that the tracing algorithm is now required to output the identities of *multiple* traitors along with a valid evidence implicating each of them. We note that in this case, it not possible to output the identities of more than t traitors as the reconstruction box can simply ignore the collected shares from some of the parties if more than t parties submit their shares. We are able to design a tracing algorithm that outputs the identities of at least $t-1$ traitors (which is nearly optimal) along with a valid evidence against each one of them.

Going Beyond Storage: Delegating Computation. We show an application of our traceable secret sharing in constructing offline-online multi-server delegation of computation on private data. In this setting, there is a single client who wants to delegate an expensive computation on a private input to a set of n servers. We are specifically interested in constructing offline-online secure computation protocols for this task. In the offline phase, the client learns the circuit that it wants to evaluate and engages in a protocol with the n servers. In the online phase, the client learns its private input and runs the online phase of the protocol. At the end of the online phase, the client can reconstruct the output of the computation. We require the online computation cost of the client to only grow with the input and output length of the computation and is otherwise independent of the size of the circuit.

Now, consider a scenario as before where there is a collector who is interested in learning the secret data of the client and publishes an advertisement describing a set of collector functions $f_1, \ldots, f_n$ and a reconstruction box Rec^*. The servers can submit the output of the collector functions applied on their *entire view* (as opposed to just the shares) during the protocol execution and the reconstruction box outputs some information about the client's input. We would like to design a protocol such that any server who submits this information always runs a risk of getting traced and prosecuted. This means that there are two additional algorithms (Trace, Judge) (that have the same semantics as in the traceable secret sharing scheme) that are respectively able to trace and verify the identities of the cheating servers. Specifically, given a set of n collector functions $f_1, \ldots, f_n$, the collected views of the servers $f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n)$ and a pirate reconstruction box Rec^* that is able to distinguish two different client inputs x_0, x_1 (that may not even lead to the same output), we require the Trace algorithm to output a valid evidence (that is accepted by the Judge algorithm) against a cheating server. We show the following theorem.

Informal Theorem 2 *Assuming the existence of a secure two-party computation protocol, there exists an explicit construction of n servers offline-online,*

delegation of computation protocol tolerating t passive server corruptions in the PKI model. In particular,

- *For any two client inputs x_0, x_1, the views of any set of $t-1$ servers when the client's input is x_0 is statistically close to their views when the client's input is x_1.*
- *For any two client inputs x_0, x_1, if there exists a set of n collector functions and a pirate reconstruction box that can distinguish the views where the client's inputs are x_0 and x_1 with advantage at least ϵ, then there exists a tracing algorithm that makes $\mathrm{poly}(|C|, \lambda, 1/\epsilon)$ (where C is the circuit to be evaluated) oracle calls to the pirate reconstruction box and outputs the identity of a traitor along with a valid evidence with probability $\Omega(\frac{n\epsilon/(n-t+1)}{|C|+(n-1)\epsilon/(n-t+1)})$.*
- *With all but negligible probability, a (polynomially bounded) cheating client cannot provide a valid evidence against an honest server even if it colludes with every other server.*

We note that this theorem statement *does not* follow as a direct consequence of traceable secret sharing (more on this in the next section) and in fact, the main challenge is to ensure that the shares of the intermediate wire values are also traceable. Indeed, if the starting shares of the inputs are traceable while the shares that the servers receive of the intermediate wire values are non-traceable, the servers can safely submit these intermediate shares to a collector (which still leaks non-trivial information).

1.2 Related Work

To the best of our knowledge, the notion of traceable secret sharing has never been studied directly. We discuss a few related notions that have appeared before in the literature. In the next subsection, we argue why techniques developed in the context of these problems fail in the TSS setting.

Traitor Tracing in Broadcast Encryption. A closely related notion to traceable secret sharing is that of traitor tracing [CFN94]. In the setting of traitor tracing [CFN94], there is a central party (also called as the broadcaster) who samples a set of public parameters along with n secret keys and distributes the secret keys to a set of parties (also called as subscribers). The broadcaster can use the public parameters to encrypt some message to a set of authorized parties and the authorized parties can use their secret key to decrypt this ciphertext. Now, when a group of subscribers come together to create a pirate decryption box that allows even an unauthorized party to decrypt the broadcast, a tracing algorithm can trace a party which was involved in creating this decryption box. There has been a long line of work focusing on obtaining efficient constructions of traitor tracing [BS95, KD98, NP98, BF99, FT99, NP01, SW00, KY01, NNL01, KY02, DF03, CPP05, BSW06, BW06, BN08, BZ14, NWZ16, GKW18] and several works which considered the setting where the broadcaster could be malicious [Pfi96, PS96, PW97]. Broadcast encryption with traitor tracing has been widely used in practice to protect digital content.

Fingerprinting Codes. Fingerprinting codes, introduced by Boneh and Shaw [BS95] are information theoretic objects used in the construction of traitor tracing schemes. It consists of a code generator that outputs a set of codewords along with a tracing key. We assign each codeword in the set to a different party. If a group of parties collude and create a new word (using some restricted operations) then the trace algorithm takes the tracing key and this new word and outputs a subset of the parties that were used in constructing this word. Subsequent to their introduction, more efficient constructions of fingerprinting codes have been proposed in [KD98, SSW01, Tar03]. The main difference between this notion and that of traceable secret sharing is that it doesn't allow to share a secret and additionally, the operations that are allowed to create a new word are somewhat restricted.

Accountable Authority IBE. An Accountable-Authority Identity based Encryption [Goy07] was introduced by Goyal to reduce the trust on the private key generator (PKG) in a IBE scheme. Specifically, if the PKG was behaving dishonestly and was leaking information of individual party's secret key, then there is an algorithm that can produce a proof that is accepted by a judge implicating the dishonest behavior of the PKG. There have been some extensions to this notion like Black-Box Accountable Authority IBE [GLSW08].

2 Technical Overview

In this section, we will give a high-level overview of our construction of traceable secret sharing and also give details of the proof. We will also give an overview of our traceable delegation protocol. Before describing our construction of traceable secret sharing, we will first explain why existing secret sharing schemes are not traceable.

Limitations of Existing Secret Sharing Schemes. Existing secret sharing schemes (such as Shamir secret sharing) do not satisfy non-imputability property. In these constructions, the dealer knows the entire share that is given to a party and hence, a malicious dealer will be able to easily implicate an honest party by coming up with his own collector functions, collected shares and a reconstruction box which serve as valid evidence against this party. To prevent this attack, we may try to run a secure multiparty computation protocol between the dealer and the parties where the dealer provides his secret and the parties receive the shares at the end. This prevents the dealer from learning the shares that the parties receive. It turns out if the underlying secret sharing scheme has some additional properties such as each share having sufficient min-entropy even conditioned on the other shares[6], then this modification can be proved to satisfy non-imputability. However, in this case it is not clear if the traceability holds.

[6] The constructions of leakage-resilient secret sharing schemes given in [SV19, ADN+19] satisfies this property.

Comparison with Related Notions. A major difference between related notions such as traitor tracing and a traceable secret sharing is in the restrictions placed on the tracing algorithm. In a traceable secret sharing, we are not trying to extract some secret from the pirate box but rather, we are trying to extract some information from (possibly obfuscated/encrypted) input given to the pirate box. This means we are only given a single sample and we must work with this sample. Indeed, we can produce fresh samples on our own and try to run the reconstruction box on these samples but in this case, the secret we are trying to extract is lost. Hence, its not even clear apriori how invoking the pirate box multiple times can help. One way to get around this issue would be to use the given input sample to produce multiple (correlated) samples s.t. the target secret is somehow present in all of them. However, this makes the construction and the analysis more subtle. We also note that a simple construction for broadcast encryption with traitor tracing exists based on any public key encryption. The problem becomes interesting only while considering the efficiency aspects. On the other hand, for traceable secret sharing, even getting a feasibility result is an interesting problem because of the above mentioned reason.

In the next subsection, we give details of our construction of traceable secret sharing scheme.

2.1 Our First Construction

The main idea behind our first construction is to partition the share of each party into two parts. The first part is a secret that is known only to this party and is unknown to the dealer and the second part is a share of a secret such that the secret is known only to the dealer (unknown to any individual party). Intuitively, the first part which is unknown to the dealer prevents a cheating dealer from implicating an honest party and the secret in the second part enables a dealer to trace a traitor. With this insight, let us now give details of our construction.

- To share a secret s, the dealer uses Shamir sharing to split s into n shares, namely, $\mathsf{ssh}_1, \ldots, \mathsf{ssh}_n \in \{0,1\}^\lambda$. The threshold t used here is the same as the required threshold for TSS.
- For every $j \in [\lambda]$, the dealer chooses a random mask R_j uniformly from $\{0,1\}^\lambda$ and splits R_j into n Shamir shares $R_{1,j}, \ldots, R_{n,j}$ (again using threshold t).
- Now, the party P_i and the dealer engage in a secure two-party computation protocol that computes the following function. The function takes the i-th Shamir share ssh_i, the shares $\{R_{i,j}\}_{j\in[\lambda]}$, and all masks $\{R_j\}_{j\in[\lambda]}$ from the dealer. It then samples $L_{i,j}$ for each $j \in [\lambda]$ randomly such that $\langle L_{i,j}, R_j\rangle = \mathsf{ssh}_{i,j}$ where $\mathsf{ssh}_{i,j}$ refers to the j-th bit of ssh_i and $\langle \cdot, \cdot \rangle$ denotes the inner product. It finally provides $\mathsf{owf}(L_{i,j})$ as output to the dealer and $\{L_{i,j}, R_{i,j}\}_{j\in[\lambda]}$ to P_i. Here, owf is an one-way function.

– The share of P_i (denoted by share_i) consists of $\{L_{i,j}, R_{i,j}\}_{j\in[\lambda]}$. The view of the dealer at the end of the sharing phase includes the Shamir shares $\mathsf{ssh}_1, \ldots, \mathsf{ssh}_n$, the shares $\{R_{i,j}\}_{i\in[n],j\in[\lambda]}$ and $\{\mathsf{owf}(L_{i,j})\}_{i\in[n],j\in[\lambda]}$.[7]
– In order to implicate the party P_i, the tracing algorithm is required to output any $L_{i,j}$ that is a valid pre-image.

Notice that the dealer's secrets $\{R_j\}_{j\in[\lambda]}$ are in fact secret shared among the parties. This means that even if you fix share_i for a party P_i, the value of R_j can still be freely decided by sampling $\{\mathsf{share}_k\}_{k\neq i}$ appropriately. This observation would be very useful when we design the tracing algorithm.

Non-Imputability. It can be easily shown that the above construction protects an honest party from a cheating dealer. In particular, it follows from the security of two-party computation that the dealer learns no information about a party's $L_{i,j}$ except learning that the inner-product of $L_{i,j}$ and R_j is $\mathsf{ssh}_{i,j}$. Thus, one can argue from the one-wayness property of owf (which hold even if there is a single bit of leakage) that the probability that a malicious dealer provides a valid pre-image is negligible and hence the probability that an honest party is implicated by a malicious dealer is negligible.

Tracing Algorithm Overview. Recall that the tracing algorithm receives the collector's functions $f_1, \ldots, f_n$, the collected shares $f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)$, the view of the dealer, and a pirate reconstruction box that is guaranteed to distinguish between the secret shares of s_0 and s_1 with noticeable advantage. The goal of the tracing algorithm is to extract one of $L_{i,j}$ that serves as a valid evidence against party P_i. However, to extract this evidence, the tracing algorithm must overcome the following challenges.

Challenge-1: Extraction from Single-Bit of Information. The first challenge is that the reconstruction box only gives a single bit of information about the evidence against P_i. However, recall that a valid evidence against P_i is one of $\{L_{i,j}\}_{j\in[\lambda]}$ where each $L_{i,j}$ is λ bits long. Furthermore, the reconstruction box is guaranteed to distinguish between the shares of s_0 and s_1 only with noticeable advantage and this means that the answer that the reconstruction box gives could sometimes be erroneous. So, the tracing algorithm must somehow use this single bit of information (which could further be erroneous) to extract a λ-bit long string.

To overcome this challenge, we rely on Goldreich-Levin decoding [GL89]. Indeed, our construction is designed to be able to use Goldreich-Levin decoding from the start. Before we go into the details of our solution, we first recall the setting of Goldreich-Levin decoding. Suppose there exists an oracle Ora that has a secret input $x \in \{0,1\}^\lambda$ hard-wired in its description. The oracle accepts

[7] We note that our construction satisfies statistical privacy even though we rely on secure two party computation protocol. This is because the dealer's inputs to any set of $t-1$ of these secure two-party computation corresponds to $t-1$ Shamir shares and it follows from the perfect privacy of Shamir secret sharing that these shares do not reveal anything about the secret that was shared.

queries $y \in \{0,1\}^\lambda$ and produces an output $z \in \{0,1\}$. If for a uniformly chosen query y, the probability that the oracle's output z is equal to $\langle x, y\rangle$ is noticeably more than $1/2$, Goldreich-Levin decoding algorithm gives a way of obtaining x hardwired in the oracle's descirption with overwhelming probability. Coming back to our setting, we will treat $L_{i,j}$ as the secret input x and use the pirate reconstruction box to simulate the working of the oracle Ora. The trace algorithm will then run the Goldreich-Levin decoder to extract out the secret $L_{i,j}$. However, for this task to be possible, we need the ability to set the query y to be equal to R_j so that we can use the reconstruction box to predict $\langle L_{i,j}, y\rangle = \langle L_{i,j}, R_j\rangle$. But the tracing algorithm only gets $f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)$ which could contain "encrypted" versions of $L_{i,j}$ and the shares of R_j and it is not clear upfront on how to set the R_j to be equal to the query y. This is where we use an earlier observation about our construction where we showed that is possible to fix share_i (that contains $L_{i,j}$) and resample the other shares in such a way that R_j is fixed to the oracle query y. We will then run the pirate reconstruction box on the fixed $f_i(\mathsf{share}_i)$ along with outputs of the other collector functions applied on the freshly sampled shares and use the output of the reconstruction box to predict $\langle L_{i,j}, y\rangle$.

A subtle but an important point that was ignored in the above paragraph is how does the tracing algorithm determine which $L_{i,j}$ to extract. The above description assumed that the tracing algorithm already knows which party is the traitor and then tries to extract the $L_{i,j}$ from this party. This brings us to the second challenge.

Challenge-2: A Careful Hybrid Argument. To determine the identity of a cheating party, the tracing algorithm will define a sequence of distributions or hybrids starting from the distribution where the shares correspond to the secret s_0 and ending with a distribution where the shares correspond to the secret s_1. Specifically, for every $i \in [n]$ and $j \in [\lambda + 1]$, the tracing algorithm defines $\mathsf{Hyb}_{i,j}$ as the distribution where $\{\mathsf{ssh}_{i'}\}_{i'<i}$ are valid Shamir shares of s_1 and $\{\mathsf{ssh}_{i'}\}_{i'>i}$ are valid Shamir shares of s_0. Further, the first $j-1$ bits of the i-th share are changed from a share of s_0 to a share of s_1. Now, via a standard averaging argument, it follows that if the pirate reconstruction box can distinguish between shares of s_0 and s_1 with advantage ϵ, then there exists an $i \in [n], j \in [\lambda+1]$ such that the reconstruction box can distinguish between $\mathsf{Hyb}_{i,j}$ and $\mathsf{Hyb}_{i,j+1}$ with advantage $\epsilon/O(n\lambda)$. This means that party P_i is a traitor (as otherwise, $\mathsf{Hyb}_{i,j} \equiv \mathsf{Hyb}_{i,j+1}$) and the tracing algorithm tries to extract an incriminating evidence against P_i. However, in order to determine if the reconstruction box can distinguish between $\mathsf{Hyb}_{i,j}$ and $\mathsf{Hyb}_{i,j+1}$ with noticeable advantage, we need the tracing algorithm to generate samples from both these distributions. To generate a sample from $\mathsf{Hyb}_{i,j}$ or $\mathsf{Hyb}_{i,j+1}$, we need to change the inner product of $L_{i,j}$ with R_j. However, we do not know $\{L_{i,j}\}_{j\in[\lambda]}$ that is available in share_i (recall that it only gets $f_i(\mathsf{share}_i)$) and hence, there does not seem to be a way for it to sample R_j such that the inner product of $L_{i,j}$ with R_j is a particular value.

To solve this issue, we slightly change the sequence of hybrids by introducing a "fine-grained structure". Specifically, instead of defining λ hybrids for changing

the i-th Shamir share, we define $2\lambda+1$ *small* hybrids $\mathsf{Hyb}_{i,j}$ indexed by $j \in \{0,\ldots,2\lambda\}$. These hybrids first change ssh_i from a valid Shamir sharing of s_0 (associated with $\mathsf{ssh}_{i+1},\ldots,\mathsf{ssh}_n$) to a random string one bit at a time, then change the random string to a valid Shamir sharing of s_1 (associated with ssh_1, $\ldots,\mathsf{ssh}_{i-1}$) again one bit at a time. Now, via a similar averaging argument we can show that there exists a $i \in [n], j \in [0, 2\lambda-1]$ such that the pirate reconstruction box can distinguish between $\mathsf{Hyb}_{i,j}$ and $\mathsf{Hyb}_{i,j+1}$ with advantage $\epsilon/O(n\lambda)$. For simplicity, assume that such a $j \in [0, \lambda-1]$. The key advantage of this fine-grained hybrid structure is that it additionally allows the tracing algorithm to sample from $\mathsf{Hyb}_{i,j}$ or $\mathsf{Hyb}_{i,j+1}$. We now give the details below.

In a thought experiment, the tracing algorithm first fixes share_i. This means that all $\{L_{i,j}\}_{j\in[\lambda]}$ are fixed but these values are unknown to the tracing algorithm. For every $k > j$, it fixes R_k as in the sharing phase. This means that the inner product of $L_{i,k}$ with R_k remains the same as the k-th bit of the i-th share of s_0. For every $k < j$, we sample an independent R'_k and this is possible due to an earlier observation that conditioned on fixing any share, the dealer's secrets are uniformly distributed. This means that for every $k < j$, the inner product of $L_{i,k}$ with the new R'_k is an uniformly chosen random bit. Now, if we fix R_j as in the sharing phase, we get an sample from $\mathsf{Hyb}_{i,j}$; else, if we sample R'_j uniformly at random, we get a sample from $\mathsf{Hyb}_{i,j+1}$.

Completing the tracing. The tracing algorithm will go over every i, j and determine if the pirate reconstruction box can distinguish between $\mathsf{Hyb}_{i,j}$ and $\mathsf{Hyb}_{i,j+1}$ with noticeable advantage. Eventually, it will reach $\mathsf{Hyb}_{i,j}$ and $\mathsf{Hyb}_{i,j+1}$ such that the pirate reconstruction box can distinguish between these two hybrids with probability at least $\epsilon/O(n\lambda)$. It will now use Goldreich-Levin decoder to extract $L_{i,j}$. For completeness, we provide the details below.

- The tracing algorithm starts running the Goldreich-Levin decoder and simulates the access to the oracle Ora.
- When the decoder queries the oracle on a uniform y, the tracing algorithm does the following:
  - It fixes the collected share of party P_i, i.e., $f_i(\mathsf{share}_i)$. In addition to this, it also fixes the random masks $\{R_k\}_{k>j}$ which are available from the view of the dealer. By fixing these random masks, the tracing algorithm has fixed the inner product of $L_{i,k}$ and R_k for $k > j$ to be the same as the bits of the initial Shamir share ssh_i that was used in the sharing phase.
  - It then randomly samples $\mathsf{ssh}'_{i+1},\ldots,\mathsf{ssh}'_n$ such that these correspond to the last $n-i$ shares of a Shamir sharing of the secret s_0. It also samples $\mathsf{ssh}'_1,\ldots,\mathsf{ssh}'_{i-1}$ such that $(\mathsf{ssh}'_1,\ldots,\mathsf{ssh}'_{i-1})$ correspond to the first $(i-1)$ shares of a Shamir sharing of s_1.
  - It sets $R'_j = y$ (where y is the query) and samples $R'_{1,j},\ldots,R'_{i-1,j},R'_{i+1,j},$ $\ldots,R'_{n,j}$ such that these values together with $R_{i,j}$ corresponds to a valid Shamir sharing of R'_j.
  - For $k < j$, it samples R_k uniformly from $\{0,1\}^\lambda$. Then the shares of $\{R_k\}_{k\neq j}$ are randomly sampled such that they are consistent with the

fixed share_i and it samples $\mathsf{share}'_1, \ldots, \mathsf{share}'_{i-1}, \mathsf{share}'_{i+1}, \ldots, \mathsf{share}'_n$ that are consistent with the above sampled values.

- The tracing algorithm then runs the pirate reconstruction box on $f_1(\mathsf{share}'_1)$, $\ldots, f_{i-1}(\mathsf{share}'_{i-1}), f_i(\mathsf{share}_i), f_{i-1}(\mathsf{share}'_{i-1}), \ldots, f_n(\mathsf{share}'_n)$). We show that using the output of the reconstruction box, one can predict the value of the inner-product between $L_{i,j}$ and y with probability noticeably better than half.

A minor subtlety that arises because of fixing $(\mathsf{share}_i, \{R_k\}_{k>j})$ is that for the Goldreich-Levin decoding to work, we require that conditioned on fixing these values, the reconstruction box still distinguishes between $\mathsf{Hyb}_{i,j}$ and $\mathsf{Hyb}_{i,j+1}$ with non-negligible advantage. We note that we can rely on Markov's inequality to show that for $\frac{\epsilon}{O(n\lambda)}$ fraction of values of $(\mathsf{share}_i, \{R_k\}_{k>j})$, the reconstruction box still distinguishes between $\mathsf{Hyb}_{i,j+1}$ and $\mathsf{Hyb}_{i,j}$ with probability at least $\frac{\epsilon}{O(n\lambda)}$ conditioned on fixing these values. This allows the tracing algorithm to use the pirate reconstruction box to simulate the oracle and thus, enabling the Goldreich-Levin decoder to extract $L_{i,j}$.

2.2 Boosting Tracing Probability

The analysis explained before shows us how to trace a traitor with overwhelming probability conditioned on $\mathsf{share}_i, \{R_k\}_{k>j}$ belonging to a "good" set. Also, we argued via Markov's inequality that the probability that this value is "good" is at least $\frac{\epsilon}{O(\lambda n)}$. Thus, the probability of tracing a traitor is roughly, $\frac{\epsilon}{O(\lambda n)}$. We now show how to increase the success probability of the tracing algorithm in a sequence of steps. The first step will show how to increase it to $O(\epsilon/n)$, the second step will increase the tracing probability to $O(\epsilon)$ and the final step will show how to increase it to $O(\frac{n\epsilon/(n-t+1)}{1+(n-1)\epsilon/(n-t+1)})$. In this informal overview, we will focus only on the first two steps and leave the third step to the main body.

First Step: $O(\epsilon/n)$. We note that to implicate P_i, it is sufficient to extract one of $\{L_{i,j}\}_{j\in[\lambda]}$ as the evidence. The above analysis tried to extract one specific L_{ij} and hence, suffered from a bad success probability. In the first boosting step, we analyze the success probability of the tracing algorithm in extracting *any* one of the $\{L_{ij}\}_{j\in[\lambda]}$. Since the tracing algorithm has more "slots" to extract a valid evidence, this increases the success probability by a proportional factor.

Towards this goal, we define $(\mathsf{share}_i, \{R_k\}_{k\in[\lambda]})$ output in the initial sharing phase to be *traceable* if there exists $j \in [\lambda]$ (or $j \in \{\lambda+1\ldots, 2\lambda\}$) such that $(\mathsf{share}_i, \{R_k\}_{k>j})$ (or $(\mathsf{share}_i, \{R_k\}_{k>2\lambda-j+1})$) is "good". In this case, we note that we can use the strategy mentioned above to extract $L_{i,j}$ (or $L_{i,2\lambda-j+1}$).

The main technical lemma that we show in this step is the following. Let us consider two large hybrids Hyb_i and Hyb_{i+1}, and if ϵ_i is the advantage of the pirate reconstruction box in distinguishing between Hyb_i and Hyb_{i+1}, then with probability $O(\epsilon_i - \epsilon/(Cn))$, $(\mathsf{share}_i, \{R_k\}_{k\in[\lambda]})$ output in the initial sharing phase is traceable (where C is a some large enough constant). By observing that there exists an $i \in [n]$ such that, $\epsilon_i \geq \epsilon/n$ (via an averaging argument), we show

that probability of tracing is $O(\epsilon/n)$. We now give an overview of this lemma by assuming without loss of generality that that the distinguishing advantage between $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,\lambda}$ is at least $\epsilon_i/2$.

The main idea in the proof of the lemma is the following (informal) duality condition. We show that for every $j \in [\lambda]$, we can either use $(\mathsf{share}_i, \{R_k\}_{k>j})$ to extract $L_{i,j}$ or the distinguishing advantage between $\mathsf{Hyb}_{i,j-1}$ and $\mathsf{Hyb}_{i,j}$ is "small". Since we know that that the distinguishing advantage between $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,\lambda}$ is at least $\epsilon_i/2$, we get a lower bound on the probability that there exists a $j \in [\lambda]$, such that $(\mathsf{share}_i, \{R_k\}_{k>j})$ can be used to extract $L_{i,j}$. The actual proof is involved and uses a delicate partitioning argument. We refer the reader to the main body for the full details.

Second Step: $O(\epsilon)$: We note that the previous analysis showed that the probability that $(\mathsf{share}_i, \{R_k\}_{k\in[\lambda]})$ is traceable is at least $O(\epsilon_i - \epsilon/(Cn))$. This in particular means that P_i can be traced with probability at least $O(\epsilon_i - \epsilon/(Cn))$. The key trick in this step is that if any two parties can be traced independently, then we may take advantage of the pairwise independence and boost the success probability. However, to trace a party, we need $(\mathsf{share}_i, \{R_k\}_{k\in[\lambda]})$ to be traceable, which means the event that one party can be traced is correlated with the event that another party can be traced.

To break the above mentioned correlation, we modify our construction as follows. In the sharing phase, instead of sampling R_j and using Shamir secret sharing to split it, the dealer samples a polynomial $p_j(\cdot)$ of degree at most $t-1$ and sets $R_{i,j}$ to be $p_j(\alpha_i)$ (for some fixed element α_i). Furthermore, instead of sampling $L_{i,j}$ such that $\mathsf{ssh}_{i,j} = \langle L_{i,j}, R_j \rangle$, the sharing protocol samples $L_{i,j}$ such that $\mathsf{ssh}_{i,j} = \langle L_{i,j}, p_j(\beta_i) \rangle$ (for some fixed element β_i). In this new construction, to trace a party P_i, we need $(\mathsf{share}_i, \{p_k(\beta_i)\}_{k\in[\lambda]})$ to be traceable. We observe that if $t \geq 4$, the random variables $(\mathsf{share}_i, \{p_k(\beta_i)\}_{k\in[\lambda]})$ and $(\mathsf{share}_{i'}, \{p_k(\beta_{i'})\}_{k\in[\lambda]})$ for any $i \neq i'$ are pairwise independent. We rely on this observation and make use of standard inequalities like Cauchy-Schwartz to get a lower bound on the probability that at least for one $i \in \{1, \ldots, n\}$, $(\mathsf{share}_i, \{p_k(\beta_i)\}_{k\in[\lambda]})$ is traceable. This allows us to get an improved analysis and thus improving the success probability to $O(\epsilon)$.

2.3 Traceable Delegation

In this subsection, we show an application of traceable secret sharing to constructing traceable multi-server delegation of computation in the offline-online setting.

The Setting. In our model, there is a single client and n servers. The client wants to delegate the computation of a circuit C on some private input x to the n servers. We consider the offline-online setting where the client gets the circuit to be computed in the offline phase but learns the private input in the online phase. The offline computational cost of the client can grow with the size of the circuit C but we require the online computation of the client to be

extremely fast. In particular, it should only grow proportional to the input length x and the output length of C and is otherwise, independent of the size of C. We require the standard correctness and the privacy properties from the protocol, meaning that the client always reconstructs the correct output and the views of t servers provide no information about the client's private input x. Additionally, we require the protocol to be traceable, meaning that given any set of collector functions $f_1, \ldots, f_n$ and a pirate reconstruction box that can distinguish between the cases where the client's input was x_0 and x_1 with noticeable advantage, then we require a tracing algorithm to output a valid evidence (accepted by a judge) against one of the cheating servers.

Why natural approaches fail? A natural approach to construct such a traceable MPC protocol is for the client to use our traceable secret sharing scheme to secret share its private input x among the n servers. Then, the servers can run standard MPC protocols like BGW [BOGW88] or GMW [GMW87] to compute a secret share of the output which can finally be reconstructed by the client. However, this approach fails in our setting because these protocols crucially rely on the secret sharing scheme to be linear whereas our traceable secret sharing scheme is non-linear. To get around this problem of non-linearity, one might think that for every gate, we might run a mini MPC protocol that takes the traceable shares of the inputs, reconstructs the input values, computes the output of the gate and then reshares it using a traceable secret sharing scheme. However, this requires the mini MPC protocol itself to be traceable and we are back to square one. In conclusion, the main difficulty we face is in making the shares of the intermediate wire values to be traceable.

Our protocol. The main idea behind our protocol is to "secret share" the circuit rather than secret sharing the input. Towards building the main intuition behind the protocol, let us start with a trivial case where the circuit is just a single gate g that takes in two input values and has a single output value. In the offline phase of our computation, the client "garbles the truth table" of this gate. Specifically, for every input wire and the output wire, the client chooses a random masking bit. Let us call these masking bits to be r_1, r_2 corresponding to the input wires and r_3 corresponding to the output wire. Further, the client generates a table with 4 entries where the (a, b)-th entry of the table for $a \in \{0, 1\}$ and $b \in \{0, 1\}$ is given by $g(a \oplus r_1, b \oplus r_2) \oplus r_3$. After generating all the entries of the "garbled table", the client uses our traceable secret sharing to secret share each entry of the garbled truth table to the n servers. This completes the offline phase of the protocol and at the end of the offline phase, each of the servers hold a secret share for every entry of the garbled truth table. In the online phase, the client learns its input $(x_1, x_2) \in \{0, 1\} \times \{0, 1\}$ and it sends to each of the servers $(x_1 \oplus r_1, x_2 \oplus r_2)$. Now, each of the servers hold the masked values of the input wires, and they just choose the share corresponding to the entry given by $(x_1 \oplus r_1, x_2 \oplus r_2)$ in the truth table, and reconstruct this particular value by broadcasting the chosen shares. It is easy to see that the reconstructed value will be the actual output

of the gate masked with r_3. Now, this value will be sent back to the client who can unmask this value and learn the output of the computation.

To give the main idea behind tracing, notice that in the online evaluation phase executed by the servers, there are three secret shares that are left untouched. Further, the entry of the gate table that is reconstructed does not give any information about the client's input due to the one-time pad security. This means that if we change any one of the untouched shares to a secret sharing of the revealed value and if the reconstruction box is able to detect this change, then we are back to the standard setting of traceable secret sharing. With this intuition in mind, let us now give the details about tracing. Towards this, let us first assume that we have a set of n collector functions $f_1, \ldots, f_n$ and a pirate reconstruction box that can distinguish between the cases where the input of the client was (x_1, x_2) from the case the input was (x'_1, x'_2) with noticeable advantage. The tracing algorithm defines a sequence of 6 hybrids starting from the case where the input was (x_1, x_2) and ending with the case where the input was (x'_1, x'_2). The first three hybrids change each entry of the garbled truth table to be $g(x_1 \oplus r_1, x_2 \oplus r_2) \oplus r_3$. That is, at the end of these changes, all the 4 secrets that were shared during the offline phase are equal to $g(x_1 \oplus r_1, x_2 \oplus r_2) \oplus r_3$. Notice that once we have done this change, we can rely on the one-time pad security to make the views of all the servers to be independent of the input. In particular, we can change the masked inputs which were sent during the online phase to be $(x'_1 \oplus r_1, x'_2 \oplus r_2)$ and the entries of the garbled table to be $g(x'_1 \oplus r_1, x'_2 \oplus r_2) \oplus r_3$. The next sequence of 3 hybrids will just reverse these changes ending with the actual view of the servers when the client's input was (x'_1, x'_2). If the reconstruction box distinguishes between the cases where the client's inputs were (x_1, x_2) from (x'_1, x'_2) with advantage ϵ, then via a standard averaging argument, it follows that there exists two intermediate hybrids Hyb and Hyb' in this sequence such that the reconstruction box is able to distinguish between these two hybrids with advantage $\epsilon/6$. Notice that the only difference between any two subsequent hybrids is the secret that was shared in a particular gate entry. Thus, fixing all other gate entries and their corresponding shares, we can now directly rely on our tracing algorithm to catch a specific traitor.

An astute reader might have noticed the similarities between our approach and the point-and-permute trick in garbled circuits [BMR90]. Indeed, we can extend the toy example in a straightforward way to computing an arbitrary circuit C composed of many gates via the point-and-permute trick. Specifically, we ask the client to choose an independent random masking bit for each wire of the circuit (including the output wires) and generate the garbled truth table for each gate as explained above. In the offline phase, the client secret shares each entry of each garbled table using our traceable secret sharing scheme and sends it over to the servers. In the online phase, the client sends the masked values of its input. Then, the servers compute the masked output of each gate in the topological order, starting from the input gates and ending in the output gates exactly as explained above. Once the servers have the masked value of the output,

then can simply send this to the client who unmasks this and reconstructs the actual output.

A subtlety. A minor subtlety that arises with the above approach is in proving the non-imputability property. Let us once again consider the toy example above where there is a single gate. In the online phase, when the servers broadcast the shares corresponding to the $(x_1 \oplus r_1, x_2 \oplus r_2)$-th entry of the garbled truth table, they also need to broadcast the $\{L_{i,j}\}_{j\in[\lambda]}$ corresponding to these shares. However, broadcasting these values allow a cheating client that colluded with one other server to easily implicate an honest server. To prevent this attack, we make use of the specific structure of our shares. Recall that the share corresponding to the i-th server comprises of $\{L_{i,j}, R_{i,j}\}_{j\in[\lambda]}$. Instead of asking the servers to naively broadcast this share in its entirety, we first ask the servers to broadcast $\{R_{i,j}\}_{j\in[\lambda]}$. This allows the servers to first reconstruct $\{R_j\}_{j\in[\lambda]}$. Once this is done, the servers can take the inner produce of each $L_{i,j}$ with R_j to reconstruct the i-th Shamir share ssh_i. The servers then broadcast this value and this allows them to reconstruct the actual secret without revealing $\{L_{i,j}\}_{j\in[\lambda]}$ to any party.

2.4 Extensions

Trace t-1 Parties. In this extension, we are interested in tracing many traitors. By using the construction in the previous step, we note that $(\mathsf{share}_1, \{p_{1,k}(0)\}_{k\in[\lambda]})$, $\ldots, (\mathsf{share}_n, \{p_{n,k}(0)\}_{k\in[\lambda]})$ are $(t-1)$-wise independent. We use the trick of explained before to identify $t-1$ "special" parties such that each of them can be traced with probability $O(\epsilon/(n-t+1))$. Therefore, we can trace $t-1$ parties with probability $O((\epsilon/(n-t+1))^{t-1})$.

Disjoint Collusion Setting. We also consider the setting where up to $t-1$ parties can collude. We focus on the disjoint collusion setting where each party can be in at most one collusion. We model the collusion by allowing the collector to specify functions $f_{i_1,\ldots,i_k}$ for collusion of $k \leq t-1$ parties, where $f_{i_1,\ldots,i_k}$ takes $\mathsf{share}_{i_1}, \ldots, \mathsf{share}_{i_k}$ as input.

The main idea of the tracing algorithm would be the same as before. However, to generate a valid random sample which is either in $\mathsf{Hyb}_{i,j}$ or $\mathsf{Hyb}_{i,j-1}$, in addition to fixing $(\mathsf{share}_i, \{p_k(\beta_i)\}_{k>j})$, we also need to fix $\{\mathsf{share}_{i'}, \{p_k(\beta_{i'})\}_{k\in[\lambda]}\}_{i'\in\mathcal{C}_i}$, where $\mathcal{C}_i$ denotes the set of parties which collude with P_i. Because we need to use the collected share sent by P_i, which requires that the shares of P_i and all parties who collude with P_i should be the same as that generated in the initial phase. Furthermore, since the tracing algorithm does not know $\{L_{i',k}\}_{i'\in\mathcal{C}_i,k\in[\lambda]}$, we need to also reuse $\{p_k(\beta_{i'})\}_{i'\in\mathcal{C}_i,k\in[\lambda]}$ so that the inner product between $L_{i',j}$ and $p_k(\beta_{i'})$ is known to the tracing algorithm. However, for $p_j(\cdot)$, we need to fix $2t-3$ values, which has already determined $p_j(\cdot)$. It disables us to change the value of $p_j(\beta_i)$.

To solve this issue, we modify the construction as follows. In the sharing phase, for $j \in [\lambda]$ instead of only using one polynomial $p_j(\cdot)$, the dealer samples n polynomials $p_{1,j}(\cdot), \ldots, p_{n,j}(\cdot)$ of degree at most $t-1$. Every party will receive

$R^1_{i,j} = p_{1,j}(\alpha_i), \ldots, R^n_{i,j} = p_{n,j}(\alpha_i)$. Instead of sampling $L_{i,j}$ such that $\mathsf{ssh}_{i,j} = \langle L_{i,j}, p_j(\beta_i)\rangle$, the sharing protocol samples $L_{i,j}$ such that $\mathsf{ssh}_{i,j} = \langle L_{i,j}, p_{i,j}(0)\rangle$. In this way, we only fix $t-1$ values of $p_{i,j}(\cdot)$ and therefore can still change the value of $p_{i,j}(0)$. The first step of boosting success probability still works in the new construction. Therefore, we can trace a party with probability $O(\epsilon/(n-t+1))$ in the collusion setting.

3 Preliminaries

Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathrm{poly}(\cdot)$ there exists λ_0 such that for all $\lambda > \lambda_0$ we have $\mu(\lambda) < \frac{1}{\mathrm{poly}(\lambda)}$. We will use $\mathrm{negl}(\cdot)$ to denote an unspecified negligible function and $\mathrm{poly}(\cdot)$ to denote an unspecified polynomial function.

We assume reader's familiarity with the digital signature schemes.

3.1 Goldreich-Levin Lemma

Lemma 1. *Suppose* owf *is a one-way function. If there is an oracle* $\mathsf{Ora}(X, \star)$ *with* X *hard-coded where* $X \in \{0,1\}^\lambda$ *such that*

$$\Pr_{Y \sim \{0,1\}^\lambda}[\mathsf{Ora}(X, Y) = \langle X, Y\rangle] \geq 1/2 + \eta(\lambda),$$

then there exists a probabilistic algorithm Inv*, which takes* owf *and* $\mathsf{owf}(X)$ *as input, has the access to* $\mathsf{Ora}(X, \star)$*, runs in* $\mathrm{poly}(1/\eta(\lambda), \lambda)$ *and makes* $\mathrm{poly}(1/\eta(\lambda), \lambda)$ *oracle queries, such that*

$$\Pr[X' \leftarrow \mathsf{Inv}^{\mathsf{Ora}(X,\star)}(\mathsf{owf}(\cdot), \mathsf{owf}(X)) : \mathsf{owf}(X') = \mathsf{owf}(X)] \geq 1 - \mathrm{negl}(\lambda).$$

We use $\mathsf{Inv}^{\mathsf{Ora}(X,\star)}(\mathsf{owf}(X))$ for simplicity and ignore the input of the description of owf when it is evident from the context.

4 Traceable Secret Sharing

In Section 4.1, we give the definition of a traceable secret sharing. In Section 4.2, we give our construction. We refer the readers to the full version of this paper for the proof of security.

4.1 Definition

A traceable secret sharing scheme consists of four algorithms ($\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Judge}$). The ($\mathsf{Share}, \mathsf{Rec}$) have the same syntax as that of a normal secret sharing scheme. The algorithm Trace takes in a set of n collector functions $f_1, \ldots, f_n$, the set of collected shares, a pirate reconstruction box, the view of the dealer during the sharing phase and outputs the identity of a traitor party $i^\star$ who has submitted its share to the collector along with a proof $\pi_{i^\star}$. The Judge algorithm takes in this proof and pronounces whether $i^\star$ is guilty or not. We give the formal definition below.

Definition 1. *A Traceable Secret Sharing (TSS) is a tuple of four algorithms* $(\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Judge})$ *with the following syntax:*

- $\mathsf{Share}(1^\lambda, s, t, n)$: *On input the security parameter* 1^λ*, a secret* s*, the threshold* t *and the number of players* n*, the dealer* D *runs the* Share *protocol with* n *players* $P_1, \ldots, P_n$*. At the end of the protocol, the player* P_i *outputs its share* share_i *and the dealer outputs its view* view_D*. We will ignore the security parameter when it is evident from the context.*
- $\mathsf{Rec}(\mathsf{share}_{i_1}, \ldots, \mathsf{share}_{i_t})$: *This is a deterministic algorithm such that given any set of* t *shares, denoted by* $\mathsf{share}_{i_1}, \ldots, \mathsf{share}_{i_t}$*, outputs a secret* s*.*
- $\mathsf{Trace}^{\mathsf{Rec}^\star}(f_1, \ldots, f_n, f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n), \mathsf{view}_D, s_0, s_1)$: *The collector publishes the description of the functions* $f_1, \ldots, f_n$ *along with a pirate reconstruction box* $\mathsf{Rec}^\star$*. The collector receives shares from a set of parties after applying the collector functions. If a party* P_i *is honest and has not submitted its share, we will replace* f_i *with a constant function. Formally, if* H *is the set of honest parties, then* f_i *is a constant function for* $i \in H$*. The* Trace *algorithm takes the* n *collector functions* $f_1, \ldots, f_n$*, the collected shares* $f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)$*, the view of* D*, two secrets* s_0, s_1*, and with oracle access to a pirate reconstruction box* $\mathsf{Rec}^\star$ *outputs an index* $i^\star \in [n]$ *and a proof* $\pi_{i^\star}$*.*
- $\mathsf{Judge}(i^\star, \pi_{i^\star}, \mathsf{view}_D)$: *This is a deterministic algorithm that takes the alleged traitor identity* $i^\star \in [n]$*, the proof* $\pi_{i^\star}$ *and the view* view_D *of the dealer and outputs* guilty *or* $\mathsf{not-guilty}$*.*

We say a scheme is a t*-out-of-*n δ*-traceable secret sharing if it satisfies the following properties.*

- ***Correctness.*** *For any secret* s *and any* $T = \{i_1, \ldots, i_t\}$ *where each* $i_j \in [n]$*, we require that*

$$\Pr_{\mathsf{Share}(s,t,n)}[\mathsf{Rec}(\mathsf{share}_{i_1}, \ldots, \mathsf{share}_{i_t}) = s] = 1$$

- ***Statistical Privacy.*** *For any two secrets* s_0, s_1 *and any* $T \subseteq [n]$ *with* $|T| \leq t-1$*, we require that*

$$\{(\mathsf{share}_1, \ldots, \mathsf{share}_n) \leftarrow \mathsf{Share}(s_0, t, n) : \mathsf{share}_T\} \approx_s$$
$$\{(\mathsf{share}_1, \ldots, \mathsf{share}_n) \leftarrow \mathsf{Share}(s_1, t, n) : \mathsf{share}_T\}$$

- ***Traceability.*** *If there exists a set of* n *collector functions* $f_1, \ldots, f_n$ *(where* f_i *is a constant function if* P_i *is honest) and a pirate reconstruction box* $\mathsf{Rec}^\star$ *such that for two secrets* s_0, s_1*,*

$$\Big|\Pr_{\mathsf{Share}(s_0,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] -$$
$$\Pr_{\mathsf{Share}(s_1,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0]\Big| \geq \epsilon$$

then,

$$\Pr[(\mathsf{share}_1, \ldots, \mathsf{share}_n, \mathsf{view}_D) \leftarrow \mathsf{Share}(s_0, t, n);$$
$$(i^\star, \pi_{i^\star}) \leftarrow \mathsf{Trace}^{\mathsf{Rec}^\star}(f_1, \ldots, f_n, f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n), \mathsf{view}_D, s_0, s_1) :$$
$$\mathsf{Judge}(i^\star, \pi_{i^\star}, \mathsf{view}_D) = \mathsf{guilty}] \geq \delta(\epsilon)$$

Furthermore, the number of queries that Trace *makes to the pirate reconstruction box* $\mathsf{Rec}^\star$ *is* $\mathrm{poly}(\lambda, 1/\epsilon)$.

– ***Non-imputability.*** *For any secret* s*, honest player* $P_{i^\star}$ *and any computationally bounded algorithm* $\widetilde{D}$*,*

$$\Pr_{\mathsf{share}(1^\lambda,s,t,n)}[(\mathsf{view}'_D, i^\star, \pi_{i^\star}) \leftarrow \widetilde{D}(\mathsf{view}_D, \mathsf{share}_{[n]\setminus\{i^\star\}}) : \mathsf{Judge}(i^\star, \pi_{i^\star}, \mathsf{view}'_D) = \mathsf{guilty}] \leq \mathrm{negl}(\lambda)$$

Remark 1. We can consider a stronger definition wherein the parties apply the collector's functions on not only the shares received but also on its entire view during the execution of the sharing protocol. In fact, our construction satisfies this stronger definition.

Tracing More Traitors. In the previous definition, it was sufficient for the Trace algorithm to output the identity of one of the traitors $i^\star$ along with a proof $\pi_{i^\star}$. It is natural to consider a stronger formulation where Trace is required to output the identities of all the traitors along with a valid proofs against each one of them. We note that it is generally impossible to output the identities of more than t traitors as the reconstruction box could simply ignore some of the collected shares. So, the best we could hope for from a tracing algorithm is to output the identities along with valid evidence of at most t traitors.

Collusion-Resistant Setting. In the previous formulation, we considered the setting where the individual parties submit their shares without colluding. Here, we consider a stronger formulation where the collector publishes the description of the functions which can take a set of shares as input. To be more precise, we consider the disjoint collusion setting (though stronger formulations are indeed possible) where each party can appear in at most one collusion. We model this collusion by allowing the collector to specify functions $f_{\{i_1,\ldots,i_k\}}$ for collusion of $k \leq t-1$ players, where $f_{\{i_1,\ldots,i_k\}}$ takes $\mathsf{share}_{i_1}, \ldots, \mathsf{share}_{i_k}$ as input. The trace algorithm takes in the description of these collector's functions, collected shares and the view of the dealer and outputs the identity of a traitor along with a proof by making oracle access to the reconstruction box. We note that if t or more parties collude together they can then recover the secret and submit some information about the secret to the collector. Thus, we restrict the size of the collusions to be at most $t-1$.

4.2 Construction

Setting. Let n denote the number of players and λ denote the security parameter. We further set the length of the secret to be λ. In the full version of this paper, we will show that our construction is traceable under parallel composition so that larger length secrets can be chopped into blocks of length λ bits each. We use P_i to represent the i-th player. Let $\mathbb{F} = \mathrm{GF}(2^\lambda)$. Let owf be an one-way function. Let $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \in \mathbb{F} \setminus \{0\}$ be $2n$ distinct fixed elements. The pair of elements (α_i, β_i) is assigned to P_i. Each P_i also has a pair of keys $(\mathsf{sk}_i, \mathsf{vk}_i)$ generated by Gen of a digital signature scheme and we assume that vk_i is public

and is known to every other party including the judge algorithm (similar to the PKI infrastructure). Alternatively, we may assume that at the end of the sharing protocol, the dealer and the server come together and sign on the transcript of the sharing protocol. In this way, the transcript available with the dealer's view can be verified by the judge.

Remark 2. We note that the PKI assumption seems a necessary condition for a traceable secret sharing scheme. Intuitively, if without the PKI assumption, a corrupted server can simply deny the messages and the corresponding signatures sent to the client when this sever is caught by the tracing algorithm. Essentially, there would be no way for the judge to check whether the messages are sent by the server or not.

For $k \in [n]$ and $k \geq t$, we say a vector (or a set) of pairs of values $((\alpha_{i_1}, v_{i_1}), \ldots, (\alpha_{i_k}, v_{i_k}))$ are valid t-Shamir shares of secret s, if there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$, such that $f(\alpha_{i_j}) = v_{i_j}$ for all $j \in [k]$ and $f(0) = s$.

Theorem 3. *Assume the existence of one-way functions, the PKI infrastructure and secure two-party computation protocols. For $t \geq 4, n \geq t$ and any $C = \mathrm{poly}(\lambda)$, there exists an explicit t-out-of-n δ-traceable secret sharing scheme with the size of each share $O(\lambda^2)$ where $\delta(\epsilon) = p(\epsilon)/(\frac{n-1}{n}p(\epsilon)+1) - \mathrm{negl}(\lambda)$, and*

$$p(\epsilon) = \frac{n\epsilon}{2(n-t+1)} - (\frac{t-1}{2} + n\lambda)\frac{\epsilon}{Cn\lambda}.$$

Without loss of generality, for $t \geq 4, n \geq t$, a set of n collector functions $f_1, \ldots, f_n$ and a pirate reconstruction box $\mathsf{Rec}^\star$ such that for two secrets s_0, s_1,

$$\begin{aligned}|&\Pr_{\mathsf{Share}(s_0,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] - \\ &\Pr_{\mathsf{Share}(s_1,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0]| \geq \epsilon,\end{aligned}$$

we assume

$$\begin{aligned}&\Pr_{\mathsf{Share}(s_0,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] - \\ &\Pr_{\mathsf{Share}(s_1,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] \geq \epsilon.\end{aligned}$$

To handle the case

$$\begin{aligned}&\Pr_{\mathsf{Share}(s_0,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] - \\ &\Pr_{\mathsf{Share}(s_1,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] \leq -\epsilon,\end{aligned}$$

one can first design a new $\widetilde{\mathsf{Rec}}^\star$ which always outputs the opposite bit of $\mathsf{Rec}^\star$ and then run Trace with access to $\widetilde{\mathsf{Rec}}^\star$.

Our construction works as below.

– $\mathsf{Share}(1^\lambda, s, t, n)$: The dealer D first randomly generates $((\alpha_1, \mathsf{ssh}_1), \ldots, (\alpha_n, \mathsf{ssh}_n))$ which are valid t-Shamir shares of secret s. For each $j \in [\lambda]$, D repeatedly samples a random polynomial $p_j(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ until $p_j(\cdot)$ satisfies that $p_j(\beta_i) \neq 0$ for all $i \in [n]$.[8] Here, $p_j(\beta_i)$ is used as "R_j" for P_i. See more discussion in the second step of Section 2.2. We require $p_j(\beta_i) \neq 0$ to ensure that the inner-product $\langle L_{i,j}, p_j(\beta_i)\rangle$ in Figure 1 is not a constant 0.
For every player P_i, let $\mathsf{ssh}_i = (\mathsf{ssh}_{i,1}, \ldots, \mathsf{ssh}_{i,\lambda})$ where $\mathsf{ssh}_{i,j} \in \{0,1\}$. The dealer D and P_i query $\mathcal{F}_{\mathsf{share}}$ which is described in Figure 1.
Let $\mathsf{view}_D = (\{\mathsf{vk}_i\}_{i\in[n]}, \{(\mathsf{owf}(L_{i,j}), \mathsf{Sign}(\mathsf{owf}(L_{i,j}), \mathsf{sk}_i))\}_{i\in[n],j\in[\lambda]}, \{(\alpha_i, \mathsf{ssh}_i)\}_{i\in[n]}, \{p_j(\cdot)\}_{j\in[\lambda]}, \{\beta_i\}_{i\in[n]})$.
– $\mathsf{Rec}(\mathsf{share}_{i_1}, \ldots, \mathsf{share}_{i_t})$: For $k \in [t]$, parse share_{i_k} as $(\alpha_{i_k}, \beta_{i_k}, (L_{i_k,1}, R_{i_k,1}), \ldots, (L_{i_k,\lambda}, R_{i_k,\lambda}))$. For $j \in [\lambda]$, compute the polynomial $p_j(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p_j(\alpha_{i_k}) = R_{i_k,j}$ for all $k \in [t]$. For $k \in [t]$ and $j \in [\lambda]$, let $\mathsf{ssh}_{i_k,j} = \langle L_{i_k,j}, p_j(\beta_{i_k})\rangle$ and $\mathsf{ssh}_{i_k} = (\mathsf{ssh}_{i_k,1}, \ldots, \mathsf{ssh}_{i_k,\lambda})$. Then reconstruct the secret s by using the reconstruction of the Shamir secret sharing scheme on $(\alpha_{i_1}, \mathsf{ssh}_{i_1}), \ldots, (\alpha_{i_t}, \mathsf{ssh}_{i_t})$.

1. $\mathcal{F}_{\mathsf{share}}$ receives ssh_i, $\{p_j(\cdot)\}_{j\in[\lambda]}$ from D and $(\mathsf{sk}_i, \mathsf{vk}_i)$ from P_i.
2. For every $j \in [\lambda]$, $\mathcal{F}_{\mathsf{share}}$ samples a random $L_{i,j}$ such that $\mathsf{ssh}_{i,j} = \langle L_{i,j}, p_j(\beta_i)\rangle$.
   - Let $R_{i,j} = p_j(\alpha_i)$. $\mathcal{F}_{\mathsf{share}}$ sets $\mathsf{share}_i = (\alpha_i, \beta_i, (L_{i,1}, R_{i,1}), \ldots, (L_{i,\lambda}, R_{i,\lambda}))$ and sends share_i to P_i.
   - For every $j \in [\lambda]$, $\mathcal{F}_{\mathsf{share}}$ sends $(\mathsf{owf}(L_{i,j}), \mathsf{Sign}(\mathsf{owf}(L_{i,j}), \mathsf{sk}_i))$ to D.

Fig. 1. Description of $\mathcal{F}_{\mathsf{share}}$

– $\mathsf{Trace}^{\mathsf{Rec}^\star}(f_1, \ldots, f_n, f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n), \mathsf{view}_D, s_0, s_1)$: Recall that:

$$\Pr_{\mathsf{Share}(s_0,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] -$$
$$\Pr_{\mathsf{Share}(s_1,t,n)}[\mathsf{Rec}^\star(f_1(\mathsf{share}_1), \ldots, f_n(\mathsf{share}_n)) = 0] \geq \epsilon.$$

For $i \in \{t, \ldots, n\}$ and $j \in \{0, \ldots, 2\lambda\}$, we define the distribution $\mathsf{Hyb}_{i,j}$ as follows:[9]

- If $j \leq \lambda$, $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{i-1}, \mathsf{ssh}'_{i-1}), (\alpha_i, \mathsf{ssh}''_i), (\alpha_{i+1}, \mathsf{ssh}'_{i+1}), \ldots, (\alpha_n, \mathsf{ssh}'_n))$ are sampled randomly such that $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{i-1}, \mathsf{ssh}'_{i-1}))$ are valid t-Shamir shares of s_1 and $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{t-1}, \mathsf{ssh}'_{t-1}), (\alpha_i, \mathsf{ssh}''_i), (\alpha_{i+1}, \mathsf{ssh}'_{i+1}), \ldots, (\alpha_n, \mathsf{ssh}'_n))$ are valid t-Shamir shares of s_0. Then the

[8] We note that $\mathsf{Share}(1^\lambda, s, t, n)$ is an expected probabilistic polynomial time algorithm. However, it can be made strict polynomial time with negligible error probability.

[9] We intentionally choose the index i starting from t since the first $t-1$ shares in the Shamir sharing of s_0 and s_1 are identical and uniformly distributed.

first j bits of ssh''_i are replaced by random bits. Let ssh'_i be ssh''_i after replacement. $p'_1, \ldots, p'_\lambda$ are then sampled in the same way as that in $\mathsf{Share}(1^\lambda, s, t, n)$. $(\mathsf{share}'_1, \ldots, \mathsf{share}'_n)$ are generated in the same way as that in $\mathcal{F}_{\mathsf{share}}$.

- If $j > \lambda$, $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{i-1}, \mathsf{ssh}'_{i-1}), (\alpha_i, \mathsf{ssh}''_i), (\alpha_{i+1}, \mathsf{ssh}'_{i+1}), \ldots, (\alpha_n, \mathsf{ssh}'_n))$ are sampled randomly such that $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{i-1}, \mathsf{ssh}'_{i-1}), (\alpha_i, \mathsf{ssh}''_i))$ are valid t-Shamir shares of s_1 and $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{t-1}, \mathsf{ssh}'_{t-1}), (\alpha_{i+1}, \mathsf{ssh}'_{i+1}), \ldots, (\alpha_n, \mathsf{ssh}'_n))$ are valid t-Shamir shares of s_0. Then the first $2\lambda - j$ bits of ssh''_i are replaced by random bits. Let ssh'_i be ssh''_i after replacement. $p'_1, \ldots, p'_\lambda$ are then sampled in the same way as that in $\mathsf{Share}(1^\lambda, s, t, n)$. $(\mathsf{share}'_1, \ldots, \mathsf{share}'_n)$ are generated in the same way as that in $\mathcal{F}_{\mathsf{share}}$.

Let $\eta(\epsilon) = \frac{\epsilon}{Cn\lambda}$ where $C = \mathrm{poly}(\lambda)$. Let $\mathsf{Inv}^{\mathsf{Ora}(X,\star)}$ be the algorithm in the Goldreich-Levin Lemma, where $\mathsf{Ora}(X, \star)$ is an oracle with X hard-coded and X is an element in $\mathbb{F}$, such that $\Pr[Y \sim \mathbb{F} : \mathsf{Ora}(X, Y) = \langle X, Y \rangle] \geq 1/2 + \eta(\epsilon)/2$.
For every $i \in \{t, \ldots, n\}$ and $j \in \{1, \ldots, \lambda\}$, Trace starts running $\mathsf{Inv}^{\mathsf{Ora}(L_{i,j},\star)}(\mathsf{owf}(L_{i,j}))$ by simulating the access to $\mathsf{Ora}(L_{i,j}, \star)$ as below:

- On receiving a query Y, if $Y = 0$, Trace outputs 0. Otherwise, Trace randomly generates $(\mathsf{share}'_1, \ldots, \mathsf{share}'_{i-1}, \mathsf{share}'_{i+1}, \ldots, \mathsf{share}'_n)$ such that, after combining with share_i (which is unknown to Trace), it is a sample in $\mathsf{Hyb}_{i,j}$ and $p'_j(\beta_i) = Y$, $p'_k(\beta_i) = p_k(\beta_i)$ for $k > j$.
  To this end, Trace randomly samples ssh''_i such that $\mathsf{ssh}''_{i,k} = \mathsf{ssh}_{i,k}$ for $k > j$. Then randomly sample $(\mathsf{ssh}'_1, \ldots, \mathsf{ssh}'_{t-1}, \mathsf{ssh}'_{i+1}, \ldots, \mathsf{ssh}'_n)$ such that $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{t-1}, \mathsf{ssh}'_{t-1}), (\alpha_i, \mathsf{ssh}''_i), (\alpha_{i+1}, \mathsf{ssh}'_{i+1}), \ldots, (\alpha_n, \mathsf{ssh}'_n))$ are valid t-Shamir shares of s_0, and after that, generate $(\mathsf{ssh}'_t, \ldots, \mathsf{ssh}'_{i-1})$ such that $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{i-1}, \mathsf{ssh}'_{i-1}))$ are valid t-Shamir shares of s_1.
  For $k < j$, it repeatedly samples a random polynomial $p'_k(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p'_k(\alpha_i) = p_k(\alpha_i)$ (recall that $R_{i,k} = p_k(\alpha_i)$ is a component in share_i) until $p'_k(\cdot)$ satisfies that $p'_k(\beta_1), \ldots p'_k(\beta_n)$ are non-zero.
  For $k = j$, it repeatedly samples a random polynomial $p'_k(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p'_k(\alpha_i) = p_k(\alpha_i)$ and $p'_k(\beta_i) = Y \neq 0$ until $p'_k(\cdot)$ satisfies that $p'_k(\beta_1), \ldots p'_k(\beta_n)$ are non-zero.
  For $k > j$, it repeatedly samples a random polynomial $p'_k(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p'_k(\alpha_i) = p_k(\alpha_i)$ and $p'_k(\beta_i) = p_k(\beta_i) \neq 0$ until $p'_k(\cdot)$ satisfies that $p'_k(\beta_1), \ldots p'_k(\beta_n)$ are non-zero.
  Then, $\mathsf{share}'_1, \ldots, \mathsf{share}'_{i-1}, \mathsf{share}'_{i+1}, \ldots, \mathsf{share}'_n$ are generated in the same way as that in $\mathcal{F}_{\mathsf{share}}$.
- Let $\mathsf{share}'_i = \mathsf{share}_i$. Note that $f_i(\mathsf{share}'_i) = f_i(\mathsf{share}_i)$ is known to Trace. Let $b = \mathsf{Rec}^\star(f_1(\mathsf{share}'_1), \ldots, f_n(\mathsf{share}'_n))$. Intuitively, b indicates whether the sharing is in $\mathsf{Hyb}_{i,j-1}$ or $\mathsf{Hyb}_{i,j}$. See the formal analysis in the full version of this paper. Output $b \oplus \mathsf{ssh}''_{i,j}$, where $\mathsf{ssh}''_{i,j}$ is the j-th bit of ssh''_i which was generated in the last step.

Then Trace receives the output of $L'_{i,j} = \mathsf{Inv}^{\mathsf{Ora}(L_{i,j},\star)}(\mathsf{owf}(L_{i,j}))$ and checks that whether $\mathsf{owf}(L_{i,j}) = \mathsf{owf}(L'_{i,j})$. If they are the same, Trace adds $(i, (j, L'_{i,j}))$ into the output list.

For every $i \in \{t, \ldots, n\}$ and $j \in \{\lambda, \ldots, 2\lambda - 1\}$, Trace starts running $\mathsf{Inv}^{\mathsf{Ora}(L_{i,2\lambda-j},\star)}(\mathsf{owf}(L_{i,2\lambda-j}))$ by simulating the access to $\mathsf{Ora}(L_{i,2\lambda-j}, \star)$ as below:

- On receiving a query Y, if $Y = 0$, Trace outputs 0. Otherwise, Trace randomly generates $(\mathsf{share}'_1, \ldots, \mathsf{share}'_{i-1}, \mathsf{share}'_{i+1}, \ldots, \mathsf{share}'_n)$ such that, after combining with share_i (which is unknown to Trace), it is a sample in $\mathsf{Hyb}_{i,j}$ and $p'_{2\lambda-j}(\beta_i) = Y$, $p'_k(\beta_i) = p_k(\beta_i)$ for $k > 2\lambda - j$.
  To this end, Trace randomly samples ssh''_i such that $\mathsf{ssh}''_{i,k} = \mathsf{ssh}_{i,k}$ for $k > 2\lambda - j$. Then randomly sample $(\mathsf{ssh}'_1, \ldots, \mathsf{ssh}'_{i-1})$ such that $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{i-1}, \mathsf{ssh}'_{i-1}), (\alpha_i, \mathsf{ssh}''_i))$ are valid t-Shamir shares of s_1, and after that, generate $(\mathsf{ssh}'_{i+1}, \ldots, \mathsf{ssh}'_n)$ such that $((\alpha_1, \mathsf{ssh}'_1), \ldots, (\alpha_{t-1}, \mathsf{ssh}'_{t-1}), (\alpha_{i+1}, \mathsf{ssh}'_{i+1}), \ldots, (\alpha_n, \mathsf{ssh}'_n))$ are valid t-Shamir shares of s_0.
  For $k < 2\lambda - j$, repeated sample a random polynomial $p'_k(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p'_k(\alpha_i) = p_k(\alpha_i)$ (recall that $R_{i,k} = p_k(\alpha_i)$ is a component in share_i) until $p'_k(\cdot)$ satisfies that $p'_k(\beta_1), \ldots p'_k(\beta_n)$ are non-zero.
  For $k = 2\lambda - j$, repeated sample a random polynomial $p'_k(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p'_k(\alpha_i) = p_k(\alpha_i)$ and $p'_k(\beta_i) = Y \neq 0$ until $p'_k(\cdot)$ satisfies that $p'_k(\beta_1), \ldots p'_k(\beta_n)$ are non-zero.
  For $k > 2\lambda - j$, repeated sample a random polynomial $p'_k(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p'_k(\alpha_i) = p_k(\alpha_i)$ and $p'_k(\beta_i) = p_k(\beta_i) \neq 0$ until $p'_k(\cdot)$ satisfies that $p'_k(\beta_1), \ldots p'_k(\beta_n)$ are non-zero.
  Then, $\mathsf{share}'_1, \ldots, \mathsf{share}'_{i-1}, \mathsf{share}'_{i+1}, \ldots, \mathsf{share}'_n$ are generated in the same way as that in $\mathcal{F}_{\mathsf{share}}$.
- Let $\mathsf{share}'_i = \mathsf{share}_i$. Note that $f_i(\mathsf{share}'_i) = f_i(\mathsf{share}_i)$ is known to Trace. Let $b = \mathsf{Rec}^\star(f_1(\mathsf{share}'_1), \ldots, f_n(\mathsf{share}'_n))$. Intuitively, b indicates whether the sharing is in $\mathsf{Hyb}_{i,j}$ or $\mathsf{Hyb}_{i,j+1}$. See the formal analysis in the full version of this paper. Output $\bar{b} \oplus \mathsf{ssh}''_{i,2\lambda-j}$, where $\mathsf{ssh}''_{i,2\lambda-j}$ is the $(2\lambda-j)$-th bit of ssh''_i which was generated in the last step.

Then Trace receives the output of $L'_{i,2\lambda-j} = \mathsf{Inv}^{\mathsf{Ora}(L_{i,2\lambda-j},\star)}(\mathsf{owf}(L_{i,2\lambda-j}))$ and checks that whether $\mathsf{owf}(L_{i,2\lambda-j}) = \mathsf{owf}(L'_{i,2\lambda-j})$. If they are the same, Trace adds $(i, (2\lambda - j, L'_{i,2\lambda-j}))$ into the output list.

In the end, if the output list is empty, Trace outputs $\perp$. Otherwise, Trace outputs the first pair $(i, (j, L'_{i,j}))$ in the output list.

– $\mathsf{Judge}(i^\star, \pi_{i^\star}, \mathsf{view}_D)$: Judge first parses $\pi_{i^\star}$ as $(j, L'_{i^\star,j})$. Then output $\mathsf{Verify}(\mathsf{owf}(L'_{i^\star,j}), \sigma_{i^\star,j}, \mathsf{vk}_{i^\star})$ where $\sigma_{i^\star,j}$ is the signature available from view_D.

Proof and Extensions. In the full version of this paper, (1) we give the formal proof of our construction, (2) we show how to improve the tracing probability of our construction, (3) we extend our construction to the collision-resistant setting and tracing more than one servers, and (4) we show the parallel composition of

our construction. We refer the readers to the full version of this paper for more details.

5 Traceable Multi-server Delegation of Computation

In this section, we define and construct a traceable multi-server delegation of computation from our traceable secret sharing. A traceable multi-server delegation of computation is an offline-online protocol between a client and n servers denoted by $P_1, \ldots, P_n$. In the offline phase, the client's input is a circuit C and it engages in a protocol with the severs. In the online phase, the client learns the input x and sends a single message to each of the servers. The servers engage in a protocol and at the end of the protocol, each server sends a single message back to the client. The client reconstructs $C(x)$ from these messages. We require the online computational cost of the client to only grow with the input and output length and is otherwise, independent of the size of the circuit. Let us denote the view of the i-th server with $\mathsf{view}_i(C, x)$ and the view of the client as $\mathsf{view}_D(C, x)$. When it is clear from the context, we use view_i to denote $\mathsf{view}_i(C, x)$. We say $(\Pi, \mathsf{Trace}, \mathsf{Judge})$ (where Trace and Judge have the same semantics of the secret sharing scheme) to be a traceable delegation of computation if it satisfies the following properties.

Definition 2. *An offline-online multi-server delegation of computation protocol* $(\Pi, \mathsf{Trace}, \mathsf{Judge})$ *with threshold* t *is said to be* δ*-traceable if it satisfies the following properties.*

- ***Correctness.*** *The correctness requirement states that for every circuit* C *and every input* x*, the client reconstructs* $C(x)$ *with probability 1.*
- ***Security.*** *For every circuit and any two inputs* x_0, x_1 *and for any subset* T *of the servers of size at most* $t-1$*, we require that*

$$\mathsf{view}_T(C, x_0) \approx_s \mathsf{view}_T(C, x_1)$$

- ***Traceability.*** *If there exists a set of* n *collector functions* $f_1, \ldots, f_n$ *(where* f_i *is a constant function if* P_i *is honest) and a pirate reconstruction box* $\mathsf{Rec}^\star$ *such that for two inputs* x_0, x_1*,*

$$|\Pr_{\Pi(C,x_0)}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n)) = 0] - \Pr_{\Pi(C,x_1)}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n)) = 0]| \geq \epsilon$$

then,

$$\Pr[(\mathsf{view}_1, \ldots, \mathsf{view}_n, \mathsf{view}_D) \leftarrow \Pi(C, x_0);$$
$$(i^\star, \pi_{i^\star}) \leftarrow \mathsf{Trace}^{\mathsf{Rec}^\star}(f_1, \ldots, f_n, f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n), \mathsf{view}_D, x_0, x_1) :$$
$$\mathsf{Judge}(i^\star, \pi_{i^\star}, \mathsf{view}_D) = \mathsf{guilty}] \geq \delta(\epsilon)$$

Furthermore, the number of queries that Trace *makes to the pirate reconstruction box* $\mathsf{Rec}^\star$ *is* $\mathrm{poly}(|C|, \lambda, 1/\epsilon)$*.*

- ***Non-imputability.*** *For any circuit* C *and input* x*, an honest server* $P_{i^\star}$ *and any computationally bounded client* $\widetilde{D}$*,*

$$\Pr_{\Pi(C,x)}[(\mathsf{view}'_D, i^\star, \pi_{i^\star}) \leftarrow \widetilde{D}(\mathsf{view}_D, \mathsf{view}_{[n]\setminus\{i^\star\}}) : \mathsf{Judge}(i^\star, \pi_{i^\star}, \mathsf{view}'_D) = \mathsf{guilty}] \leq \mathrm{negl}(\lambda)$$

5.1 The Protocol

In this subsection, we give the details of our traceable delegation of computation.

- **Offline Phase.** In the offline phase, the client receives the circuit C and does the following.
  1. For every wire w of the circuit C, the client chooses a random mask $r_w \leftarrow \{0,1\}$. We assume the input wires are labeled from 1 to ℓ.
  2. For every gate g of the circuit with input wires i and j and the output wire k, the client generates a table with 4 entries where each entry is labeled with $(a,b) \in \{0,1\} \times \{0,1\}$. The (a,b)-th entry of the gate table is given by $g(a \oplus r_i, b \oplus r_j) \oplus r_k$.
  3. For every gate g and every entry of the gate table, the client and the servers run the sharing protocol of a t-out-of-n traceable secret sharing. Let $\mathsf{share}_i^{g,a,b}$ be the i-th share corresponding to the (a,b)-th entry of the gate g.
- **Online Phase.** In the online phase, the client receives its input x and sends $x \oplus r_{[\ell]}$ to each of the servers. The servers now starting running the online protocol. For every gate g (in the topological order),
  1. The servers hold $y_i \oplus r_j$ and $y_j \oplus r_j$ where y_i, y_j are the values carried by the i and j-th wires when the circuit C is evaluated on input x.
  2. Now, the i-th server parses $\mathsf{share}_i^{g,y_i \oplus r_i, y_j \oplus r_j}$ as $(\alpha_i, \beta_i, (L_{i,1}, R_{i,1}), \ldots, (L_{i,\lambda}, R_{i,\lambda}))$. The servers first exchange $R_{i,1}, \ldots, R_{i,\lambda}$ to each other. For $j \in [\lambda]$, the servers compute the polynomial $p_j(\cdot) \in \mathbb{F}[X]$ of degree at most $t-1$ such that $p_j(\alpha_i) = R_{i,j}$ for all $i \in [n]$. For every $j \in [\lambda]$, the i-th server computes $\mathsf{ssh}_{i,j} = \langle L_{i,j}, p_j(\beta_i) \rangle$ and $\mathsf{ssh}_i = (\mathsf{ssh}_{i,1}, \ldots, \mathsf{ssh}_{i,\lambda})$. The servers then broadcast the values of ssh_i and use the reconstruction of the Shamir secret sharing scheme to obtain $g(y_i, y_j) \oplus r_k = y_k \oplus r_k$.

  The servers finally send the masked values of the output to the client, who removes the output masks to learn $C(x)$.
- **Tracing algorithm.** Given $f_1, \ldots, f_n$, $f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n)$, the view of the client view_C, two inputs x_0, x_1 and oracle access to a reconstruction box Rec^*, the tracing algorithm does the following.
  1. It defines a sequence of hybrid distributions $\mathsf{Hyb}_{g,a,b}$ (starting from $\Pi(C, x_0)$) for every gate g and $(a,b) \in \{0,1\} \times \{0,1\}$ such that every $g' < g$ (with input wires i', j' and output wire k'), we change all the gate entries to $g(y_{i'} \oplus r_{i'}, y_{j'} \oplus r_{j'}) \oplus r_{k'}$. Further, for all entries that are less than (a,b) in the gate table of g (w.r.t. to some ordering), we change those entries to $g(y_i \oplus r_i, y_j \oplus r_j) \oplus r_k$. Notice that $\mathsf{Hyb}_{|C|,1,1}$ is independent of x_0 and hence, symmetrically, it defines $\mathsf{Hyb}'_{g,a,b}$ from $\Pi(C, x_1)$ where $\mathsf{Hyb}_{|C|,1,1} \equiv \mathsf{Hyb}'_{|C|,1,1}$.
  2. Notice that for any two intermediate hybrids in this sequence, the only difference is in the value that was secret shared in a particular gate entry. Thus, the tracing algorithm fixes the secret shares of all other gate entries and runs the corresponding tracing algorithm for the secret sharing scheme where the two secrets are the two different values in the subsequent hybrids corresponding to this gate table entry. It repeats this

process for every subsequent hybrid in the sequence. If in some iteration it succeeds in extracting a valid evidence from a party, it stops and outputs the evidence.

– **Judge algorithm.** The judge algorithm for the MPC runs the corresponding judge algorithm of the secret sharing scheme and outputs whatever it outputs.

Theorem 4. *If the protocol described above is instantiated with a δ-traceable secret sharing scheme, then it is an offline-online $\delta(\epsilon/8|C|)$-traceable n server delegation protocol with threshold t for a circuit C.*

Proof. The correctness of the protocol is easy to observe and we now show security, traceability and non-imputability.

Security. To show security, we need to show that for any two inputs x_0, x_1 and for any subset $T \subseteq [n]$ of size at most $t-1$, we have

$$\mathsf{view}_T(C, x_0) \approx_s \mathsf{view}_T(C, x_1).$$

We show security through a hybrid argument.

– Hyb_0 : This corresponds to $\mathsf{view}_T(C, x_0)$.
– Hyb_1 : In this hybrid, we generate the sharings of the gate entries differently. For every gate g with input wires i, j and output wire k, we generate the (a, b)-th entry for every $(a, b) \neq (y_i \oplus r_i, y_j \oplus r_j)$ as a secret sharing of 0. We output the view of the T servers. We note that $\mathsf{Hyb}_0 \approx_s \mathsf{Hyb}_1$ from the privacy of traceable secret sharing scheme.
– Hyb_2 : In this hybrid, for every wire i, we set $y_i \oplus r_i$ as an independently chosen random value. Hyb_2 is identically distributed to Hyb_1. Notice Hyb_2 is independent of the input x_0.

Via an identical argument, we can show that $\mathsf{view}_T(C, x_1)$ is computationally close to Hyb_2. This proves security.

Traceability. Let us fix the collector functions $f_1, \ldots, f_n$ and a pirate reconstruction box Rec^* such that for two inputs x_0, x_1,

$$\left| \Pr_{\Pi(C,x_0)}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n)) = 0] - \Pr_{\Pi(C,x_1)}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1), \ldots, f_n(\mathsf{view}_n)) = 0] \right| \geq \epsilon.$$

We now define a sequence of $4|C|$ hybrids starting from $\Pi(C, x_0)$. Specifically, for every gate g (with input wires i, j and output wire k) and $(a, b) \in \{0,1\} \times \{0,1\}$, we define $\mathsf{Hyb}_{g,a,b}$ where as a distribution where for every $g' < g$ (with input wires i', j' and output wire k'), we change all the gate entries to $g(y_{i'} \oplus r_{i'}, y_{j'} \oplus r_{j'}) \oplus r'_{k'}$. Further, for all entries that are less than (a, b) in the gate table of g (w.r.t. to some ordering), we change those entries to $g(y_i \oplus r_i, y_j \oplus r_j) \oplus r_k$. Note that once we make this change for every gate entry, the final hybrid is independent of x_0 and hence, we can reverse these hybrids one by one to get $\Pi(C, x_1)$. Without loss of generality, let us assume that

$$|\Pr_{\Pi(C,x_0)}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1),\ldots,f_n(\mathsf{view}_n))=0]-\Pr_{\mathsf{Hyb}_{|C|,1,1}}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1),\ldots,f_n(\mathsf{view}_n))=0]|\geq\epsilon/2$$

By an averaging argument, we infer that there exists two intermediate hybrids, Hyb and Hyb' in the sequence such that

$$|\Pr_{\mathsf{Hyb}}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1),\ldots,f_n(\mathsf{view}_n))=0]-\Pr_{\mathsf{Hyb}'}[\mathsf{Rec}^\star(f_1(\mathsf{view}_1),\ldots,f_n(\mathsf{view}_n))=0]|\geq\epsilon/(8|C|)$$

Notice that the only difference between Hyb and Hyb' is the value that was secret shared in a particular gate entry. Thus, it follows from the traceability of the underlying secret sharing scheme, that the MPC tracing algorithm outputs a valid evidence against a party with probability at least $\delta(\epsilon/(8|C|))$.

Non-imputability. Note that the offline view of the servers consists of the views of $4|C|$ different sharings of our traceable secret sharing scheme. Further, observe that the messages sent during step 2 of the online phase can be simulated using view_D. The non-imputability property follows directly from the underlying traceable secret sharing, as we can correctly guess the particular secret for which an adversarial dealer gives the correct evidence with $1/(4|C|\lambda)$ probability and hardcode the one-way function challenge in this position.

Acknowledgments.

V. Goyal, Y. Song—Supported in part by the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.
A. Srinivasan—Work partially done while at UC Berkeley and visiting CMU. Supported in part by AFOSR Award FA9550-19-1-0200, AFOSR YIP Award, NSF CNS Award 1936826, DARPA/ARL SAFEWARE Award W911NF15C0210, a Hellman Award and research grants by the Sloan Foundation, Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the authors and do not reflect the official policy or position of the funding agencies.

References

ACM17. ACM. ACM Paris Kanellakis Theory and Practice Award. Press Release, May 2017. https://awards.acm.org/binaries/content/assets/press-releases/2017/may/technical-awards-2016a.pdf.

ADN+19. Divesh Aggarwal, Ivan Damgard, Jesper Buus Nielsen, Maciej Obremski, Erick Purwanto, Joao Ribeiro, and Mark Simkin. Stronger leakage-resilient and non-malleable secret-sharing schemes for general access structures. CRYPTO, 2019. https://eprint.iacr.org/2018/1147.

BF99. Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, Heidelberg, August 1999.

Bla79. GR Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conf.*, volume 48, pages 313–317, 1979.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513. ACM Press, May 1990.

BN08. Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08: 15th Conference on Computer and Communications Security*, pages 501–510. ACM Press, October 2008.

BOGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, May 1988.

BS95. Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data (extended abstract). In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 452–465. Springer, Heidelberg, August 1995.

BSW06. Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 573–592. Springer, Heidelberg, May / June 2006.

BW06. Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 211–220. ACM Press, October / November 2006.

BZ14. Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499. Springer, Heidelberg, August 2014.

CFN94. Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, Heidelberg, August 1994.

CPP05. Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 542–558. Springer, Heidelberg, May 2005.

DDFY94. Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th Annual ACM Symposium on Theory of Computing*, pages 522–533. ACM Press, May 1994.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, Heidelberg, August 1990.

DF03. Yevgeniy Dodis and Nelly Fazio. Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 100–115. Springer, Heidelberg, January 2003.

Fra90. Yair Frankel. A practical protocol for large group oriented networks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 56–61. Springer, Heidelberg, April 1990.

FT99. Amos Fiat and Tamir Tassa. Dynamic traitor training. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 354–371. Springer, Heidelberg, August 1999.

GKW18. Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM Symposium on Theory of Computing*, pages 660–670. ACM Press, June 2018.

GL89. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32. ACM Press, May 1989.

GLSW08. Vipul Goyal, Steve Lu, Amit Sahai, and Brent Waters. Black-box accountable authority identity-based encryption. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08: 15th Conference on Computer and Communications Security*, pages 427–436. ACM Press, October 2008.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM Press, May 1987.

Goy07. Vipul Goyal. Reducing trust in the PKG in identity based cryptosystems. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 430–447. Springer, Heidelberg, August 2007.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304. IEEE Computer Society Press, November 2000.

KD98. Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 145–157. Springer, Heidelberg, May / June 1998.

KY01. Aggelos Kiayias and Moti Yung. Self protecting pirates and black-box traitor tracing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 63–79. Springer, Heidelberg, August 2001.

KY02. Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 450–465. Springer, Heidelberg, April / May 2002.

NNL01. Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *Advances in Cryptology*

– *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, Heidelberg, August 2001.

NP98. Moni Naor and Benny Pinkas. Threshold traitor tracing. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 502–517. Springer, Heidelberg, August 1998.

NP01. Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In Yair Frankel, editor, *FC 2000: 4th International Conference on Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Heidelberg, February 2001.

NWZ16. Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 388–419. Springer, Heidelberg, May 2016.

Pfi96. Birgit Pfitzmann. Trials of traced traitors. In *Information Hiding, First International Workshop, Cambridge, UK, May 30 - June 1, 1996, Proceedings*, pages 49–64, 1996.

PS96. Birgit Pfitzmann and Matthias Schunter. Asymmetric fingerprinting (extended abstract). In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 84–95. Springer, Heidelberg, May 1996.

PW97. Birgit Pfitzmann and Michael Waidner. Asymmetric fingerprinting for larger collusions. In *ACM CCS 97: 4th Conference on Computer and Communications Security*, pages 151–160. ACM Press, April 1997.

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

SSW01. Alice Silverberg, Jessica Staddon, and Judy L. Walker. Efficient traitor tracing algorithms using list decoding. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 175–192. Springer, Heidelberg, December 2001.

SV19. Akshayaram Srinivasan and Prashant Nalini Vasudevan. Leakage resilient secret sharing and applications. CRYPTO, 2019. `https://eprint.iacr.org/2018/1154`.

SW00. Reihaneh Safavi-Naini and Yejing Wang. Sequential traitor tracing. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 316–332. Springer, Heidelberg, August 2000.

Tar03. Gábor Tardos. Optimal probabilistic fingerprint codes. In *35th Annual ACM Symposium on Theory of Computing*, pages 116–125. ACM Press, June 2003.