Consistent Adaptive Two-Party Computations

Sven Laur[1] and Helger Lipmaa[2]

[1] Helsinki University of Technology, Finland
[2] University College London, UK

Abstract. Secure protocols for complicated tasks are usually constructed in two phases. Initially, one designs a protocol that is secure in a semihonest model. Second, zero-knowledge correctness proofs are added to assure correctness and privacy against malicious adversaries. Often the corresponding communication and computational overhead makes this approach intractable in practice. Therefore, we define an intermediate notion of security—*consistency*. In a consistent protocol, participants always learn if their opponent cheats but cannot complain without violating their own privacy. Thus, a potential victim has to choose between the utility of the correct output and a potential privacy breach. In many contexts, where the long-term reputation of a service provider is more important than the privacy of an individual query, such security notion provides adequate protection with a minimal overhead. Private inference control and adaptive oblivious transfer are the most straightforward examples of such protocols.
Keywords. Commitment scheme, hash tree, oblivious transfer, private inference control, two-party computation.

1 Introduction

The security of a two-party protocol is defined by comparing real protocol runs with an ideal world implementation. In the ideal world, the parties send their private inputs to a trusted third party (TTP), and then the TTP answers by forwarding them their private outputs. If the protocol is adaptive, then the ideal world implementation consists of several adaptive rounds. To gain indistinguishability between the real and ideal world, one must guarantee consistency between the outputs. Usually, this is done by proving in zero-knowledge that for *any* input the output would be correct. The latter provides *security* but is prohibitively resource demanding if the input size is big [9, p. 599].

For many functionalities, the only known method of reducing the communication of secure protocols is the method outlined in [9], where the parties use a communication-efficient commitment scheme to commit their inputs. After that, they use communication-efficient zero-knowledge proofs that are based on the PCP theorem: the statement that a party has behaved correctly belongs to NP, and thus has a PCP witness. However, while the resulting protocols have sublinear communication, they are still quite resource demanding: the participant needs to compute a PCP witness, and in the complexity-theoretic model the protocol has at least 2 messages in the initialization phase, and 4 additional messages per query. Moreover, for a concrete protocol instance the communication can be intractable in practice, especially, when the inputs are megabytes long. Also, the round complexity is 4 messages per query instead of the

optimal 2 messages per query. To avoid this complexity, many papers propose protocols that achieve security only in the semihonest model where correctness proofs are unnecessary.

A common alternative security definition for asymmetric two-party protocols is *relaxed security*, see, e.g., [1]. In asymmetric protocols, only one party (*the client*) obtains the output while the other party (*the server*) learns nothing. A protocol is relaxed-secure if it is secure against malicious clients and semihonest servers. Relaxed security is a standard security notion for say oblivious transfer protocols. Many other interesting functionalities can be relaxed-securely implemented with two messages [7]. In particular, for many tasks like oblivious transfer, there exist low-degree polylogarithmic-communication 2-message protocols.

However, relaxed security does not provide the client a correctness guarantee. E.g., consider private inference control (PIC, [12]) in the next setting. The server holds a database of private keys that are used to encrypt various content, e.g., radio or television channels. The clients have acquired different credentials and server's task is to release correct keys. Obviously, the client must distinguish between the denial of service attacks, where the server acts maliciously, and legitimate denials, where the client has no right to obtain a corresponding key. On the other hand, the server should not get to know which (legal) key the client obtained. Client's complaint must be verifiable without interacting with the server, as she might not want to disclose her full input in court.

Our contribution. To accommodate such a design goal, we introduce a new security notion. *Consistent protocols* guarantee consistency of client's outputs and allow to detect and prove server's malicious behavior based only on the protocol transcript. See Sect. 3 for formal definitions and Sect. 4 and 5 for concrete implementations.

For relaxed-secure protocols, such as adaptive oblivious transfer, it is impossible to guarantee that the server does not change her input during the protocol. Also, for an honest server it is much harder or even impossible to defend against false accusations. For many relaxed-secure protocols, the server must publicly reconstruct all the computations and even that might not be sufficient to invalidate a false accusation.

Differently from secure protocols, consistent protocols do not guarantee that the outputs are correct for any possible input, i.e., server's malicious behavior may cause a protocol failure only for a limited set of client's inputs. Thus, issuing a valid complaint reveals a single bit of information about client's inputs. To quantify such an information leakage, we define special halting predicates π_j. Predicates $\pi_1(q_1)$, $\pi_2(q_1, q_2)$,..., $\pi_m(q_1, q_2, \ldots, q_m)$ are *enforceable* if a malicious server can force the honest client with inputs $q_1, \ldots, q_m$ to halt whenever $\pi_j(q_1, \ldots, q_j)$ holds for some $j \in [m]$. Different consistent protocols have different sets of plausible halting predicate classes $\mathcal{P}_1, \ldots, \mathcal{P}_m$. Protocols with more restrictive predicate classes are more "secure", see Sect. 6, but on the other hand, lead to a smaller number of PIC-like applications.

Our central construction is a transformation from a relaxed-secure 1-out-of-n oblivious transfer protocol to an m-out-of-n consistent adaptive oblivious transfer protocol AOT. In the transformed protocol, the server first publishes a short list commitment to the whole database. List commitment schemes, see Sect. 2, allow to use short certificates (partial openings) for opening committed database elements, the best-known (although not hiding) list commitment scheme is the hash tree. During execution of every 1-out-of-n oblivious transfer protocol, the client obtains the corresponding private output with

a certificate that is sufficient to verify that the retrieved output is consistent with the list commitment. If the list commitment is statistically hiding and oblivious-transfer protocols are statistically server-private, the resulting AOT protocol is statistically server private, otherwise the security guarantees are only computational. (See Sect. 4.)

Consistent PIC protocols (see Sec 5) can be build on top of conditional disclosure of secrets (CDS) that is strengthened similarly to oblivious transfer. Recall that conditional disclosure of secrets [1, 7] allows to reveal database elements only if a certain public predicate (access restriction) holds. Clearly, if we can build a consistency check into the CDS protocol, then an honest client can always prove a denial of service attack without the help from the server. Sect. 5 presents a consistent CDS protocol. We also construct a general consistent protocol for a relatively large class of other functionalities. We conclude with a number of open questions.

Notation. For $t \in \mathbb{Z}^+$, let $[t] := \{1, \ldots, t\}$. Let k be the security parameter. Throughout this paper, we denote the number of composed protocols by m.

2 List Commitments

In this section we define list commitment schemes as commitment schemes where one can commit to a list and then later open separately list elements. List commitment schemes are natural generalizations of hash trees, but because nobody has before defined them formally, we next provide formal definitions and a simple construction.

Recall that a commitment scheme $\mathcal{C}om$, specified by a triple of algorithms $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$, is assumed to be hiding and binding. The initialization algorithm Gen generates public parameters pk. The commitment algorithm $\mathsf{Com}_{\mathsf{pk}} : \mathcal{M} \times \mathcal{R} \to \mathcal{C} \times \mathcal{D}$ is used to get a randomized pair $(c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}(m; r)$, where the commitment c is sent to another party and the state d is kept for later use. We omit r and use $\mathsf{Com}_{\mathsf{pk}}(m)$ instead if r is chosen uniformly at random from $\mathcal{R}$. The opening algorithm $\mathsf{Open}_{\mathsf{pk}} : \mathcal{C} \times \mathcal{D} \to \mathcal{M} \cup \{\perp\}$ opens $c \in \mathcal{C}$, given access to the state d, so that for all $x \in \mathcal{M}$, $\mathsf{Open}_{\mathsf{pk}}(\mathsf{Com}_{\mathsf{pk}}(x)) = x$.

A *list commitment scheme* is a commitment scheme for lists $\boldsymbol{x} = (x_1, \ldots, x_n)$, with $n \leq \mathrm{poly}(k)$. A list commitment scheme $\mathcal{LC}$ is specified by quadruple $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Cert}, \mathsf{Open})$, where Gen and Com are defined as in the case of usual commitment schemes. In addition, partial decommitment values (*certificates*) can be obtained from the state d via $\mathsf{Cert}_{\mathsf{pk}} : \mathcal{D} \times \mathbb{N} \to \mathcal{D}$. I.e, for every $\boldsymbol{x} \in \mathcal{M}^n$ and every $i \in [n]$, if $(c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}(\boldsymbol{x})$ then $\mathsf{Cert}_{\mathsf{pk}}(d, i)$ returns a value s_i such that $\mathsf{Open}_{\mathsf{pk}}(c, s_i) = (i, x_i)$. For incorrect certificates, Open returns $\perp$. Sometimes, Gen is an interactive protocol between the client and the server. We emphasize the interactive nature by explicit notation $\mathsf{Gen}\langle R, S\rangle(1^k)$, where S is the party who is going to commit and R is the client of the commitment.

The simplest list commitment consists of the list of ordinary commitment values $(c_1, \ldots, c_n)$ to $(x_1, \ldots x_n)$, with linear-in-n commitment length $|\mathsf{Com}_{\mathsf{pk}}(\boldsymbol{x})|$. Another famous example of a binding (though not hiding, see below for formal definitions) list commitment is hash tree, where Gen generates a random h from a collision-resistant hash function family, $\mathsf{Com}_h(\boldsymbol{x})$ returns the hash tree root on $\boldsymbol{x}$ and a description of the shape of the hash tree as a commitment value c, and sets $d \leftarrow \boldsymbol{x}$. A partial opening $\mathsf{Cert}_h(\boldsymbol{x}, i)$ is the hash certificate of the ith node, i.e., the minimum amount of information required

to recompute the root value from x_i located at the ith position in the corresponding tree shape. As we can always fix a canonical shape for each number of leafs, we assume that the commitment value c consists of the root hash and of the leaf count. We assume that the certificate returned by $\mathsf{Cert}_h(\boldsymbol{x}, i)$ contains the value x_i.

Like usual commitments schemes, a list commitment scheme must be binding and hiding. More precisely, for a non-uniform stateful adversary $A = \{A_k\}$ and a honest client R and honest server S, define $\mathsf{Adv}^{\mathsf{bind}}_{\mathcal{LC}}(A, k)$ to be the probability that A_k can generate a c and two certificates to the same index i, so that the certificates allow to open c to different values of x_i, i.e.,

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathcal{LC}}(A, k) := \Pr\left[\begin{array}{l} \mathsf{pk} \leftarrow \mathsf{Gen}\langle A_k, R\rangle(1^k), (c, s_0, s_1) \leftarrow A_k(\mathsf{pk}), \\ (i_b, x_b) \leftarrow \mathsf{Open}_{\mathsf{pk}}(c, s_b), \text{ for } b \in \{0, 1\} \\ i_0 = i_1 \wedge \bot \neq x_0 \neq x_1 \neq \bot \end{array}\right] .$$

Also, define $\mathsf{Adv}^{\mathsf{hide}}_{\mathcal{LC}}(A, k)$ to be the probability that A_k can distinguish commitments to two databases, given c and oracle access to certificates, i.e.,

$$\mathsf{Adv}^{\mathsf{hide}}_{\mathcal{LC}}(A, k) := 2 \cdot \left| \Pr\left[\begin{array}{l} \mathsf{pk} \leftarrow \mathsf{Gen}\langle S, A_k\rangle(1^k), \left(\boldsymbol{x}^0, \boldsymbol{x}^1\right) \leftarrow A_k(\mathsf{pk}), \\ b \leftarrow \{0, 1\}, (c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}\left(\boldsymbol{x}^b\right) : \\ \boldsymbol{x}^0, \boldsymbol{x}^1 \in \mathcal{M}^n \wedge A_k^{\mathsf{Cert}_{\mathsf{pk}}(d,\cdot)}(\mathsf{pk}, c) = b \end{array}\right] - \frac{1}{2} \right| .$$

Here, the probability is taken over coin tosses of all relevant algorithms, A is allowed to query $\mathsf{Cert}(d, i)$ only on indices i with $\boldsymbol{x}^0_i = \boldsymbol{x}^1_i$, and the size of committed lists must be polynomial. A list commitment scheme is (1) *computationally hiding* if for any non-uniform polynomial time adversary $A = \{A_k\}$, $\mathsf{Adv}^{\mathsf{hide}}_{\mathcal{LC}}(A, k) \leq k^{-\omega(1)}$, and (2) *computationally binding* if for any non-uniform polynomial time adversary $A = \{A_k\}$, $\mathsf{Adv}^{\mathsf{bind}}_{\mathcal{LC}}(A, k) \leq k^{-\omega(1)}$. In the case of unbounded adversaries, we speak about *statistical hiding* and *statistical binding*.

Going back to the example of hash trees, the certificate s_i is valid when it is consistent with the root hash c, otherwise $\mathsf{Open}_h(d, s_i) = \bot$. Clearly, the computational binding property follows from the collision-resistance of the function family $\mathcal{H}$ against non-uniform adversaries: if an adversary can find a double opening $(i, x_i) \neq (i, \widehat{x}_i)$ then there must be a collision in the root path, since the shape of the tree is fixed. Thus, collision resistance of $\mathcal{H}$ implies that polynomial-size hash trees are computationally binding. On the other hand, a pure hash tree is not hiding since the hash certificate $\mathsf{Cert}_h(\boldsymbol{x}, i)$ reveals not only x_i but also $x_{i\oplus 1}$, the sibling of x_i in the tree. However, one can make hash trees hiding by using hiding commitments in the leafs.

Theorem 1. *Assume that $\mathcal{C}om$ is a computationally (resp., statistically) hiding commitment scheme and that $\mathcal{H}$ is a collision-resistant hash function family. Then there exists a computationally (resp., statistically) hiding list commitment scheme $\mathcal{LC}$ with efficiency comparable to the hash tree.*

Proof. Let $\mathcal{C}om = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$. Construct the list commitment scheme $\mathcal{LC}$ as follows. Given $\boldsymbol{x} \in \mathcal{M}^*$, compute a commitment and decommitment vector $(\boldsymbol{c}, \boldsymbol{d})$, where $(c_i, d_i) \leftarrow \mathsf{Com}_{\mathsf{pk}}(x_i)$. Then use the non-hiding hash-tree list commitment scheme on

$\boldsymbol{c}$ and output the corresponding root value and the shape of the hash tree as the commitment value. The corresponding global decommitment value is the vector $(\boldsymbol{c}, \boldsymbol{d})$. The certificate s_i for x_i consists of d_i and of the ith hash tree certificate. In order to verify a certificate s_i, one has to verify that the corresponding hash tree certificate is correct and that $\mathsf{Open}_{\mathsf{pk}}(c_i, d_i) \neq \perp$. Clearly, this construction is (resp., statistically) hiding if $\mathcal{C}om$ is (resp., statistically) hiding. Similarly, $\mathcal{LC}$ is computationally binding if $\mathcal{H}$ is a collision-resistant hash function family and $\mathcal{C}om$ is computationally binding. □

Note that if the commitment scheme is statistically hiding for each pk then the public parameters $h \leftarrow \mathcal{H}$ and $\mathsf{pk} \leftarrow \mathsf{Gen}$ can be generated the client. Since the commitment scheme is statistically hiding for every pk, then the client can only harm herself by choosing pk differently form the output distribution of Gen.

3 Secure Adaptive Asymmetric Two-party Computation

The standard security definition of a two-party protocol Π^f for functionality f is given via comparison with its ideal model implementation and real protocol runs. In the ideal world, participants forward their inputs to a trusted third party (TTP) that computes the desired functionality f and sends back the corresponding outputs one by one. Since both parties can halt in the real world, the TTP is allowed to halt if one of the participants sends a halting instruction in the ideal world.

We focus on adaptive asymmetric computations between the client and the server, where the server holds an input $\boldsymbol{x}$ and the client makes m adaptive queries $q_1, \ldots, q_m$ to retrieve the outputs of some functions $f_1, \ldots, f_m$. In the ideal world, the TTP receives $\boldsymbol{x}$ from the server and for every client's input q_j sends $\perp$ to the client if the server issues an abort command and $f_j(q_j, \boldsymbol{x})$ otherwise. Such formalism captures all client-server protocols where server's input is non-adaptive. $\Pi^{\boldsymbol{f}}$ is *correct*, if the client obtains the desired outputs $f_1(q_1, \boldsymbol{x}), \ldots, f_m(q_m, \boldsymbol{x})$ provided that both parties act honestly. In particular, protocol's inputs must be from a valid range, and it may give arbitrary outputs for incorrect inputs. The following definitions of privacy, relaxed-security and security are standard, and given here (without all the details) for the sake of completeness.

Privacy. $\Pi^{\boldsymbol{f}}$ is computationally *client-private*, if for any non-uniform polynomial-time malicious server, there exists a non-uniform polynomial-time adversary acting as the server (*simulator*) in the ideal world such that the output distributions are computationally indistinguishable if the client is honest. A protocol is computationally *server-private*, if for every non-uniform polynomial time malicious client, there exists a non-uniform polynomial-time adversary acting as the client (*simulator*) in the ideal world such that the output distributions are computationally indistinguishable when the server is honest. Protocol is statically private if malicious participants and the corresponding simulators are unbounded and the output distributions are statistically indistinguishable.

Relaxed-security. $\Pi^{\boldsymbol{f}}$ is computationally (resp., statistically) *relaxed-secure* if it is computationally client-private and computationally (resp., statistically) server-private in the malicious model. The dual case when the server is unbounded is known to require linear communication in the case of the computationally-private information retrieval and is therefore not very interesting from practical viewpoint.

Security. Π^f is secure if it is correct even if the server is malicious. More precisely, consider an arbitrary non-uniform polynomial-time malicious server and an honest client in the real world. A protocol is *secure* if there exists a non-uniform polynomial time adversary acting as the server in the ideal world such that the joint output distribution of the server and the client is computationally indistinguishable from the joint output in the real world. In the ideal world the honest client just submits his inputs and outputs the values received from TTP. In particular, security means that the client can detect server's malicious behavior that might change his output value for some valid input. Thus, the client can freely complain, as the server herself knows that she acts maliciously.

One can artificially make some relaxed-secure protocols secure. Consider for example 1-out-of-2 oblivious transfer where the client wants to learn one of the database elements (x_1, x_2). Assume that the client interprets a protocol failure as output 0. Consider a simulator that acts as an honest client, queries both (x_1, x_2), submits the outputs to TTP, and outputs server's last output message. Then the output distributions of the real and ideal world are indistinguishable. The latter is an artifact of missing consistency requirements—one can always reconstruct a valid database from any protocol outputs. Thus, one often constructs verifiable protocols where the parties first commit their inputs and then verify the consistency between the inputs and the outputs. A secure adaptive protocol needs explicit consistency, as one input value must correspond to all outputs.

Consistency. As already mentioned in the introduction, a secure protocol seems to need 4 messages per query in the complexity-theoretic model. Moreover, due to the use of the PCP theorem, existing secure sublinear-communication protocols are not easy to implement. On the other hand, there exist several relaxed-secure oblivious transfer protocols with low-degree polylogarithmic communication [8, 5] and 2 messages per query. Such protocols exist also for many other practical functionalities [7]. Our goal is to strengthen security requirements so that simple and communication efficient two round protocols still exist. In particular, client should be able to detect and prove inconsistency between outputs; this is extremely important as the server can change her input during a straightforward sequential composition of relaxed-secure protocols.

Intuitively, a protocol is consistent when an honest client who does not detect the fraud in the real world obtains the same result as in the ideal world. However, the exact formalization is not so straightforward, as a malicious server can use different inputs. Instead we consider a ideal world, where the server first sends her input $\boldsymbol{x}$ to TTP, and then additionally specifies randomized halting predicates $\pi_1(q_1), \ldots, \pi_m(q_m)$. In a concrete protocol, the set $\mathcal{P}_j$ of enforceable predicates can be restricted, see Sect. 6.

More formally, the server sends a description of Halt-Machine that is a stateful randomized polynomial-time algorithm[1] with its running time polynomial in the security parameter k. Given the jth query q_j, TTP feeds q_j into Halt-Machine. The TTP sends to the client $f_j(q_j, \boldsymbol{x})$, if Halt-Machine outputs accepting state, and $\perp$, otherwise.[2] A correct and relaxed-secure protocol $\Pi^{\boldsymbol{f}}$ is *consistent* if for any non-uniform polynomial-time server there exists a non-uniform polynomial time adversary acting as the server

[1] Note that the server may send different descriptions of Halt-Machine and the polynomial time-bound means that there exists a constant c such that the working time of all Halt-Machine generated by the server S_k are bounded by k^c.

[2] Alternatively, the server could send the truth table directly to TTP but such approach leads to inefficient simulators as the size of the truth table can be exponential.

in the ideal world such that the restricted joint output distribution of the real world and ideal world are computationally indistinguishable. Additionally, after the jth input, Halt-Machine is restricted to compute only enforceable predicates $\pi_j \in \mathcal{P}_j$ on $(q_1, \ldots, q_j)$.

By the construction of ideal world, a malicious server might be able to cause protocol failure selectively for client's some inputs and thus potentially learn a single bit—the value of the halting predicate π_j on $(q_1, \ldots, q_j)$—of information about client's query. Clearly, If the client reaches the accepting state then he is guaranteed that his output is the same as it would be in the ideal world. Thus, a non-accepting client can prove to the court that server's behavior was indeed malicious, without anymore interacting with the server. The client must show the protocol transcript and prove to the court that his actions were correct. On the other hand, an honest server can now easily disprove false accusations, since server's honest behavior leads always to acceptance.

When a protocol is only relaxed-secure then an honest server must repeat her computations and prove (in zero-knowledge) that her actions were correct to falsify the accusation and even that might be insufficient, since sometime it is possible to change protocol inputs to hide malicious behavior. Moreover, to prove such claims, the server has to store all computations which is clearly impractical if she serves many clients.

The main drawback of consistent protocols is the potential 1-bit information leakage. One natural goal is to restrict the set of potential halting predicates. In Sec 6, we consider different subclasses of consistent protocols that have different sets of plausible halting predicates. On the other hand, consistent-but-not-secure protocols have their own applications, see Sect. 5.

4 Consistent Adaptive Oblivious Transfer

Consider a simple application, where the server sells some digital goods, such as access rights (encryption keys) to TV programs, or sensitive end-results of statistical queries. The client wants hide the information about purchased goods but on the same time he wants to get some guarantees that the obtained result was not tampered. The latter is especially important when the client has no prior knowledge about the product before obtaining it. Such problem can be formalized as adaptive oblivious transfer (AOT). In an *adaptive m-out-of-n oblivious transfer protocol,* the server has a static database $\boldsymbol{x} = (x_1, \ldots, x_n)$ and the client wants to adaptively fetch database elements $x_{q_1}, \ldots, x_{q_m}$. More formally, in each phase the client must learn $f(q, \boldsymbol{x}) = x_q$ when $q \in [n]$, and $\perp$ otherwise. The value of q_j may depend on the values of $x_{q_1}, \ldots, x_{q_{j-1}}$. In the case of a 1-out-of-n oblivious transfer protocol, only a single query can be submitted.

Assume that we are given a list commitment scheme $\mathcal{LC}$ for tuples consisting of ℓ-bit elements, where the length of certificates $s_i \leftarrow \mathsf{Cert}_{\mathsf{pk}}(d, i)$ is bounded by $\ell_{\mathrm{s}}(n)$. Let OT be a 1-out-of-n oblivious transfer protocol for $\ell_{\mathrm{ot}} = \ell + \ell_{\mathrm{s}}(n)$ bit strings. Then under suitable assumptions given below, Prot. 1 is a consistent m-out-of-n oblivious transfer protocol. Since it uses OT in a black-box way, it can be seen as a transformation from relaxed-secure oblivious transfer protocols to consistent oblivious transfer protocols. It is easy to see that Prot. 1 is correct.

Efficiency. Prot. 1 has total communication $|\mathsf{Gen}| + |\mathsf{Com}_{\mathsf{pk}}(\boldsymbol{s})| + m \cdot C(n, \ell_{\mathrm{ot}})$, where $|\mathsf{Gen}|$ is the communication of the key generation protocol and $C(m, \ell_{\mathrm{ot}})$ is the com-

COMMON INPUT: Both parties hold k, n, ℓ, m and $\ell_{\mathrm{ot}} := \ell + \ell_{\mathrm{s}}(n)$.
CLIENT'S INPUT: possibly adaptively chosen $(q_1, \ldots, q_m) \in [n]^m$
SERVER'S INPUT: $\boldsymbol{x} = (x_1, \ldots, x_n)$, $x_i \in \{0,1\}^{\ell}$.
UNDERLYING PROTOCOLS:
An oblivious transfer protocol OT for ℓ_{ot}-bit strings.
A list commitment scheme $\mathcal{LC} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Cert})$.

Initialization phase:

1. The client and the server generate jointly a $\mathsf{pk} \leftarrow \mathsf{Gen}(1^k)$.
2. Server sets $\mathsf{counter} \leftarrow 1$ and computes $(c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}(\boldsymbol{x})$, stores a database $\boldsymbol{s} = (s_1, \ldots, s_n)$, where $s_i \leftarrow \mathsf{Cert}_{\mathsf{pk}}(d, i)$ for $i \in [n]$ and send c to the client.
3. The client stores c.

Protocol execution on a single query q:

1. The server aborts if $\mathsf{counter} > m$. Otherwise, the server increments $\mathsf{counter}$.
2. The client and the server execute a single run of OT on the database $\boldsymbol{s}$.
3. Let $\hat{s}_q$ be client's private output in OT.
4. The client outputs $\hat{x}_q$ if $\mathsf{Open}_{\mathsf{pk}}(c, \hat{s}_q) = (q, \hat{x}_q) \neq \bot$, and $\bot$, otherwise.

Protocol 1: A new consistent m-out-of-n oblivious transfer protocol

munication of OT for ℓ_{ot}-bit strings. Since C is at most linear in ℓ_{ot} and $\ell_{\mathrm{s}}(n)$ is logarithmic in n for good list commitment schemes, the communication overhead is minimal. For hash-tree based list commitments, $\ell_{\mathrm{s}}(n) = (\log_2 n - 1)\lambda + \Theta(\ell)$, where λ is the output length of the underlying hash function and $\Theta(\ell)$ terms counts the size of leaf commitment and decommitment values. As one can use oblivious transfer protocols with low-degree polylogarithmic-communication [8, 5] in conjunction with communication-efficient list commitment schemes, the total communication of Prot. 1 is low-degree polylogarithmic $\Theta(m \cdot \mathrm{poly}(\log n))$. As there are commitment schemes where the client can choose the public parameters pk, Prot 1 can be implemented with $2m + 2$ rounds.

Theorem 2 (Relaxed-security). *If $\mathcal{LC}$ is statistically hiding for every $\mathsf{pk} \leftarrow \mathsf{Gen}\langle R, S\rangle$ and computationally binding and OT is statistically relaxed-secure, then Prot. 1 is statistically relaxed-secure.*

Proof. *Computational client-privacy* follows straightforwardly, as OT is client-private and $\mathsf{Gen}\langle R, S\rangle(1^k)$ is independent from client's inputs.

Statistical server-privacy follows from the next standard hybrid argument. Consider three worlds: World_0 corresponds to the ideal world, in World_1 only the instances of OT are replaced with ideal implementations of 1-out-of-n oblivious transfer and World_2 is the real world. Clearly, any malicious client in World_2 can be converted to a malicious client in World_1 such that their output distributions are statistically close, as OT is statistically server-private. As the commitment scheme is statistically hiding for every pk, then there exist unbounded algorithm that can open $\mathsf{Com}_{\mathsf{pk}}(\mathbf{0})$, where $\mathbf{0} = (0, \ldots, 0)$, to any other vector $\boldsymbol{x} \in \{0,1\}^{n\ell}$ with negligible failure. Moreover, such algorithm must be able to change unopened elements on the fly or otherwise the commitment is not statis-

tically hiding. Hence, we can convert a malicious client $\hat{R}_1$ from World_1 to a malicious client $\hat{R}_0$ for World_0 as follows:

1. Run protocol $\mathsf{Gen}\langle \hat{R}_1, S\rangle$ to get pk. Compute $(c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}(\mathbf{0})$ and sent c to $\hat{R}_1$.
2. Given query q_j obtain x_{q_j}, compute a $\hat{s}_j$ such that $\mathsf{Open}_{\mathsf{pk}}(c, \hat{s}_j) = (q_j, x_{q_j})$.
3. Output the output of $\hat{R}_1$.

As commitment is statistically hiding for every pk the output distributions for $\hat{R}_0$ and $\hat{R}_1$ are statistically indistinguishable. The claim follows. □

The transformation from World_1 to World_0 fails if we cannot generate faked certificates on the fly. Therefore, the proof fails for computationally hiding commitments unless list commitment does not have a necessary trapdoor, i.e., is equivocable, see App. A for the corresponding definitions and the proof of Cor. 1.

Corollary 1. *If $\mathcal{LC}$ is computationally hiding, equivocable and binding and* OT *is relaxed-secure, then Prot. 1 is computationally relaxed-secure.*

The proof of consistency consists of two main steps. First, we must extract suitable input from the malicious server and second, we must show that if an honest client reaches the accepting state then his output has same distribution as in the ideal model. The extraction part is tricky, since a malicious server might act honestly only under specific conditions. Thus, we have to probe the server with all possible queries.

Theorem 3 (Consistency). *Prot. 1 is computationally consistent, if it is relaxed-secure, $\mathcal{LC}$ is computationally binding, and n^m is polynomial in the security parameter.*

Proof. Consider an input extractor that first chooses randomness for the honest client and the malicious server $\hat{S}$ and then sends all possible n^m queries to $\hat{S}$, reconstructs from the replies the corresponding database $\boldsymbol{x}$ and an n^m-element truth table with $\mathsf{H}[q_1, \ldots, q_m] = \mathsf{true}$ only if the client does not halt. Second, it submits $\boldsymbol{x}$ and $\mathsf{Halt\text{-}Machine}$ that uses H to answer queries to TTP and outputs the last output of $\hat{S}$.

By the construction, the output distributions of the real and ideal world w.r.t. the same random inputs coincide unless the probing reveals double openings (i, x_i) and $(i, \hat{x}_i)$ that do not allow to reconstruct $\boldsymbol{x}$. If some x_i is not revealed then it is also never queried, and we can set $x_i \leftarrow 0$. The probability that some queries reveal a double opening is negligible because otherwise the input extractor together with a malicious server breaks binding property of $\mathcal{LC}$. Since the protocol is client-private by assumption the joint distributions are computationally indistinguishable. As simulator's working time is dominated by the execution of n^m oblivious transfer protocols, the simulator runs in polynomial time. □

In particular, the simulator runs in polynomial time if m is an arbitrary constant. Alternatively, we can make a stronger assumption that $\mathcal{LC}$ is binding against non-uniform adversaries of subexponential size; such an assumption is quite common. Then, Prot. 1 is computationally consistent for $m = o(n/\log n)$.

Comparison with previous work. Currently several 2-round client-private 1-out-of-n private information retrieval protocols with low-degree polylogarithmic-communication are known [8, 5]. All of these protocols can be converted to relaxed-secure 1-out-of-n

oblivious transfer protocols using well-known communication efficient transformation techniques from [10, 1, 7]. Thus, it is known how to construct 2-round relaxed-secure oblivious transfer with low-degree polylogarithmic communication. On the other hand all practical protocols for adaptive oblivious transfer that achieve security are based on zero-knowledge proofs and have thus communication $\Omega(mn)$. The latter can be escaped with explicit use of PCP theorem [9] but these approaches aren't currently practical, as they are optimal only in asymptotic sense. Therefore, Prot. 1 can be viewed as the most communication efficient transformation from private information retrieval to consistent adaptive oblivious transfer.

5 Consistent Conditional Disclosure of Secrets

Consistent conditional disclosure of secrets (CDS) is the most natural way to achieve private inference control (PIC, [12]) where the server holds a database of private keys and has to distribute them according to some public rules, e.g., in micro-payment protocols. In such scenarios, consistency provides necessary protection for the server and the client, as the client can detect and prove unjustified denials of service and the honest server can protect herself against false accusations. The concept of homomorphic CDS was first introduced in [1] and latter extended in [7].

Background. The core of a CDS protocol is the *disclose-if-equal* (DIE) subprotocol where the client submits an encryption $\mathsf{Enc}_{\mathsf{pk}}(q)$, and learns a secret x if and only if q is equal to server's input y; the server learns $\mathsf{Enc}_{\mathsf{pk}}(q)$. In such a protocol, the client generates a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}$ of an additively homomorphic cryptosystem and sends pk to the server. When client sends $\mathsf{Enc}_{\mathsf{pk}}(q)$, the server computes reply as $u \leftarrow (\mathsf{Enc}_{\mathsf{pk}}(q)\mathsf{Enc}_{\mathsf{pk}}(-y))^s \mathsf{Enc}_{\mathsf{pk}}(x) = \mathsf{Enc}_{\mathsf{pk}}(s(q-y)+x)$, for $s \leftarrow \mathbb{Z}_q$. If the plaintext space has prime order q, like in the additively homomorphic lifted ElGamal, then the resulting protocol is relaxed secure and perfectly sender-private [1]. For cryptosystems with composite plaintext order, one needs slightly more elaborate protocols [7]. It is straightforward to extend basic DIE to and-DIE protocol, where the client learns a secret x only if his encrypted inputs $\mathsf{Enc}_{\mathsf{pk}}(q_1), \dots, \mathsf{Enc}_{\mathsf{pk}}(q_m)$ satisfy constraint $[q_1 = y_1 \wedge \cdots \wedge q_m = y_m]$. In the case of lifted ElGamal, sender's reply is $u \leftarrow (\mathsf{Enc}_{\mathsf{pk}}(q_1)\mathsf{Enc}_{\mathsf{pk}}(-y_1))^{s_1} \cdots (\mathsf{Enc}_{\mathsf{pk}}(q_m)\mathsf{Enc}_{\mathsf{pk}}(-y_m))^{s_m} \mathsf{Enc}_{\mathsf{pk}}(x)$. In an or-DIE protocol, defined similarly, the server must compute two replies, and the client can choose the correct one, for which $q_i = y_i$, and recover the secret. For more complex operations, the server must divide secrets using secret sharing and use simpler DIE subprotocols to transfer the sub-secrets, e.g. set $x_1 + x_2 = x$ and use sub-protocols to assure $[q_1 = y_1]$ and $[q_2 = y_2]$ [7]. As shown in [7], the server can efficiently release secret x if a predicate Ψ has a polynomial circuit size. Such protocol is called circuit-CDS [7].

Consistent circuit-CDS. As all access rules in PIC protocols are public then circuit-CDS protocol is exactly what is needed to implement PIC. More precisely, let Ψ be the circuit for the key release constraint, then the circuit-CDS protocol implements the desired functionality. However, the known CDS protocols are not consistent. It is straightforward to make DIE protocol consistent by letting the server first send a commitment to x, and then use the DIE protocol to transfer the decommitment string. Since the circuit-CDS protocol uses properly generated sub-secrets $x_1, \dots, x_n$ to transfer the main secret z,

COMMON INPUT: Both parties hold k, n, ℓ, m, $\ell_{\mathrm{ot}} := \ell + \ell_{\mathrm{s}}(n)$ and a formula Ψ.
CLIENT'S INPUT: is possibly adaptively chosen $(q_1, \ldots, q_m) \in [n]^m$ for Ψ-CDS.
SERVER'S INPUT: a master secret $\boldsymbol{z} \in \{0,1\}^{\ell}$.
UNDERLYING PROTOCOLS:
A relaxed-secure circuit-CDS protocol Ψ-CDS for ℓ_{ot} bit strings.
A list commitment scheme $\mathcal{LC} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Cert})$.

Initialization phase:

1. The client and the server generate jointly $\mathsf{pk} \leftarrow \mathsf{Gen}(1^k)$ for commitment
2. The client generates $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Gen}'(1^k)$ for CDS and sends pk' to the server.
3. The server computes all necessary sub-secrets $\boldsymbol{x} = (x_1, \ldots, x_n)$ needed for Ψ-CDS.
   She computes $(c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}(\boldsymbol{x})$ and $\boldsymbol{s} = (s_1, \ldots, s_n)$, where $s_i \leftarrow \mathsf{Cert}_{\mathsf{pk}}(d, i)$.
4. The server sends c to the client who stores c.

Modified Ψ-CDS protocol:

1. The server and client run Ψ-CDS protocol where sub-secrets are $\boldsymbol{s}$ instead of $\boldsymbol{x}$.
2. The client determines based on $\boldsymbol{q}$ and Ψ what sub-secrets $x_{r_1}, \ldots, x_{r_t}$ he should learn.
   In the modified protocol the client should learn $s_{r_1}, \ldots, s_{r_m}$.
3. The client computes $x_{r_i} = \mathsf{Open}_{\mathsf{pk}}(c, s_{r_i})$ for $i \in [t]$ and halts if any $x_{r_i} = \perp$.
   Finally, the client restores the master secret z from $x_{r_1}, \ldots, x_{r_t}$ by using the original Ψ-CDS.

Protocol 2: A new consistent conditional disclosure of secrets protocol

then sending commitments to all secrets x_i is a simple way to make the protocol consistent. A more communication-efficient way is just to send list commitment[3] $\mathsf{Com}_{\mathsf{pk}}(\boldsymbol{x})$, see Prot. 2.

Theorem 4. *Let Ψ-CDS be relaxed-secure. If $\mathcal{LC}$ is computationally hiding and equivocable and binding, then Prot. 2 is computationally relaxed-secure. If $\mathcal{LC}$ is also statistically hiding for every $\mathsf{pk} \leftarrow \mathsf{Gen}\langle R, S\rangle$ and Ψ-CDS is statistically server-private, then Prot. 2 is statistically server-private. If the set of satisfiable inputs for Ψ is polynomial in the security parameter then Prot. 2 is computationally consistent.*

Proof. The proof is analogous to the security analysis of Prot. 1 and thus omitted. □

Example applications. Since the server can restore encrypted inputs of clients consistent CDS protocols can be used in pay-per-view systems: one client's input corresponds to $\mathsf{Enc}_{\mathsf{pk}}(credit)$ stored an updated by the server. It is straightforward to test that $credit > 0$ by using consistent circuit-CDS. Another application is restricted access to private data, e.g., TV or military broadcasts with complex access policy, based on credentials that are represented as random keys.

6 Different Subclasses of Consistent Protocols

The main advantage of consistent computations is verifiability: an honest client can detect and prove that the server has managed to alter her input. On the other hand, a

[3] Public parameters of commitment scheme are different from public key of DIE protocol.

malicious server can cause selective protocol failures so that issuing a complaint leaks information about client's inputs. The definition of consistency limits the corresponding information gain to polynomially computable randomized predicates, however, we might want to quantify it in more fine-grained level.

Memoryless consistency. An adaptive protocol is *memoryless-consistent* if the halting predicates $\pi_1, \ldots, \pi_m$ are independent from previous queries, i.e., $\pi_i(q_1, \ldots, q_i) = \pi_i(q_i)$. As a result, the server cannot relate results of different queries.

Note that there is no black-box construction from an arbitrary stateful relaxed-secure 1-out-of-n oblivious transfer protocol OT to a memoryless-consistent m-out-of-n oblivious transfer. A formal separation is following. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a randomized encryption scheme. Now, if client augments all his queries with $\mathsf{Enc}(q_j)$ and accepts the OT reply only if the server adds an encryption of q_j to her reply, then the corresponding protocol is certainly relaxed-secure. However, if a malicious server replaces $\mathsf{Enc}(q_j)$ with the encryption $\mathsf{Enc}(q_{j-1})$ from the previous round then she can force an halting predicate $[q_{j-1} = q_j]$. More natural examples can be based on the relaxed-secure homomorphic oblivious transfer protocol from [1], where the client's message is just $\mathsf{Enc}(q_j)$. If we construct Prot. 1 so that all OT invocations share the same pk then the server can force many affine halting predicates. On the other hand, Prot. 1 is memoryless-consistent when the subsequent executions of oblivious transfer do not share random variables.

Theorem 5 (Memoryless consistency). *Relaxed-secure Prot. 1 is also computationally memoryless consistent, if $\mathcal{LC}$ is computationally binding and n^m is polynomial in the security parameter and oblivious transfer protocols do not share random variables.*

Proof. Assume that we have an adversary A that breaks the memoryless-consistent property of Prot. 1. That is, it can force the client to abort if and only if a predicate p_i holds on client's queries $(q_1, \ldots, q_i)$ thus far, where p_i is a non-trivial function of at least two different values q_a and q_b for $a < b \leq i$. Since the protocol is stateless then the adversary can play the role of the client in round $b > a$, to breach the privacy of the client in round a: given its knowledge of whether the client aborted in round b, it will have some advantage in guessing q_a, given the value $p_i(q_a, q_b)$. □

The same result holds also for consistent private inference control. If client generates each time a new encryption key then the corresponding protocols are memoryless-consistent. However, such approach might be impractical in some applications. Achieving memoryless-consistency by other means is an interesting open question.

Homomorphic consistency. Many efficient oblivious transfer protocols are based on homomorphic encryption [11, 1, 8, 7]. In such protocols, client's input is sent in encrypted form and thus it is possible to compute ciphertexts of affine combinations of inputs. More formally let $\mathcal{P}_{\text{aff}}(q)$ consist of all base predicates of type $[\alpha q = \beta]$ and their conjunctions; let $\mathcal{P}_{\text{b-aff}}(q)$ consist of all base predicates of type $\alpha_1 q[1] + \cdots \alpha_k q[k] = \beta$ where $q[k] \ldots q[1]$ is the bit-representation of q and their conjunctions. Then a protocol is *homomorphically consistent* if the halting predicates $\pi_1(q_1), \ldots, \pi_m(q_1, \ldots, q_m)$ belong either to the class of $\mathcal{P}_{\text{aff}}(q_1, \ldots, q_j)$ or to the class $\mathcal{P}_{\text{b-aff}}(q_1, \ldots, q_j)$. Note that such a set of predicates is rather restricted and minimal if we use oblivious transfer based on homomorphic encryption. One can straightforwardly prove that if we use ideal homomorphic encryption, that does not allow to compute any other operation on ciphertext, for the constructions [11, 1, 8, 7] then resulting protocols are homomorphically consistent.

Many researchers have worked on extensively on cryptographic protocols that use extensively the properties of existing homomorphic cryptosystems. There is no evidence that any non-affine predicate can be computed-on-ciphertexts ("cryptocomputed") in the case of any of such cryptosystems. Thus, it is reasonable to make the next seemingly novel gap assumption that only affine predicates can be computed-on-ciphertexts.

Assumption 1 (Gap homomorphic assumption for Π.) *Let Π be an additively homomorphic cryptosystem such as the Paillier. For any non-affine efficiently verifiable t-ary predicate π, $t \geq 1$, the probability that some polynomial-size probabilistic circuit A, given access to a randomly generated public key* pk *and to random encryptions of any t elements q_i, can output an encryption of $\pi(q_1, \ldots, q_t)$, is negligibly close to the probability that A can guess the value of $\pi(q_1, \ldots, q_t)$, given its knowledge of the distribution from which the plaintexts q_i are drawn.*

Alternatively, if Assumption 1 cannot be proven, one should still describe the class of computable predicates, as it provides a subset of predicates that can be efficiently cryptocomputed using additively homomorphic encryption. The set of cryptocomputable predicates that can be computed with constant communication is currently unknown and is one of the most interesting problems in the design of efficient cryptographic protocols.

7 Discussion and Open Problems

We showed how to build consistent m-out-of-n oblivious transfer given only client-private 1-out-of-n information retrieval protocols and binding list commitment scheme that is either statistically hiding or equivocable. One can verify that the existence of list commitment schemes is necessary. Using similar techniques to [6], one can build a list commitment scheme based on every (adaptive) sublinear consistent oblivious transfer protocol. Thus, in particular, a sublinear consistent oblivious transfer protocol exists if and only f there exists a communication efficient list commitment scheme. The requirement of equivocability is also rather standard though somewhat restricting. Therefore, one possible further research direction would be the elimination of equivocability using different construction or more powerful proof technique.

Minimizing assumptions. Prot. 1 uses list commitment schemes and oblivious transfer protocols. List commitment schemes can be built from hash trees—for which one needs collision-resistant hash functions—and from statistically-hiding commitment schemes. One can construct a collision-resistant hash family from a sublinear client-private oblivious transfer protocol [6], and a statistically hiding commitment scheme from an arbitrary collision-resistant hash family [3]. Therefore, one can also construct list commitment schemes from any sublinear client-private oblivious transfer protocol. As shown in [4], one can construct a relaxed-secure oblivious transfer protocol from any client-private oblivious transfer protocols. Thus Prot. 1 is an efficient "client-private oblivious transfer" to "consistent oblivious transfer" transformation without any need for extra assumptions like the existence of collision-resistant hash functions.

More efficient proof techniques. A shortcoming of the current consistency proof is its inherent inefficiency in the extraction phase, as simulator runs in time $\Theta(n^m)$ probing all possible inputs. The same problem appears also in the case of consistent private inference control protocols. As a result, simulator works in polynomial time if m is constant.

If $m = o(n/\ln n)$ then the oblivious transfer and the list commitment scheme must be secure against non-uniform adversaries that work in sub-exponential time. Such restriction seems rather artificial, especially, if we do not care about the revealed halting predicates. Therefore, we need more efficient extraction techniques to extend security proofs for many other interesting cases.

General construction for consistent computations. In the current article, we gave constructions for consistent oblivious transfer and conditional disclosure of secrets, however, we did not give a general construction. One can turn general relaxed-secure two-round circuit evaluation protocols for additively shared output into constant round consistent circuit evaluation protocols by running two copies of the protocol with changed identities and afterwards doing secure comparison. But such construction requires more than 4 rounds so the question whether round-efficient general consistent computation protocols exist remains open. Note that a slightly weaker version of consistency—where in the ideal model, the server sends the values of $f_j(q_j, \boldsymbol{x})$ for all possible values of q_j's and for a fixed value of $\boldsymbol{x}$ to the TTP—that is equivalent to the consistency in the case of oblivious transfer protocols, one can implement many functionalities by executing Prot. 1 with server's input $\boldsymbol{y}$, $y_i = (f_j(i, \boldsymbol{x}))$.

References

1. William Aiello, Yuval Ishai, and Omer Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer-Verlag.
2. Ivan Damgård and Jens Groth. Non-Interactive And Reusable Non-Malleable Commitment Schemes. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 426–437, San Diego, CA, USA, June 9–11 2003. ACM Press.
3. Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. Statistical secrecy and multibit commitments. *IEEE Transactions on Information Theory*, 44(3):1143–1151, 1998.
4. Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single Database Private Information Retrieval Implies Oblivious Transfer. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138, Bruges, Belgium, 14–18 May 2000. Springer-Verlag.
5. Craig Gentry and Zulfikar Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In Luis Caires, Guiseppe F. Italiano, Luis Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *The 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815, Lisboa, Portugal, 2005. Springer-Verlag.
6. Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient Conditions for Collision-Resistant Hashing. In Joe Kilian, editor, *The Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 445–456, Cambridge, MA, USA, February 10–12, 2005. Springer Verlag.
7. Sven Laur and Helger Lipmaa. Additive Conditional Disclosure of Secrets And Applications. Technical Report 2005/378, IACR, November 21 2005.
8. Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In Jianying Zhou and Javier Lopez, editors, *The 8th Information Security Conference (ISC'05)*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328, Singapore, September 20–23, 2005. Springer-Verlag.

9. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings of the Thirty-Third Annual ACM Symposium on the Theory of Computing*, pages 590–599, Heraklion, Crete, Greece, July 6–8 2001. ACM Press.
10. Moni Naor and Benny Pinkas. Oblivious Transfer and Polynomial Evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 245–254, Atlanta, Georgia, USA, May 1–4, 1999. ACM Press.
11. Julien P. Stern. A New and Efficient All or Nothing Disclosure of Secrets Protocol. In Kazuo Ohta and Dingyi Pei, editors, *Advances on Cryptology — ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 357–371, Beijing, China, October 18–22, 1998. Springer-Verlag.
12. David P. Woodruff and Jessica Staddon. Private Inference Control. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 188–197, Washington, DC, USA, October 25–29, 2004.

A Equivocable Commitments and Relaxed Security

In the first glance Prot. 1 seems relaxed-secure whenever the underlying list commitment $\mathcal{LC}$ is binding and hiding. However, there is a small subtlety: a malicious client can choose the queries based on $\mathsf{Com}_{\mathsf{pk}}(\boldsymbol{x})$ and released certificates s_i. The latter makes simulation hard, as the simulator must somehow iteratively put queried x_i under the commitment so that the client does not change his asked queries. For that we need a trapdoor so that we could successfully execute the simulation algorithm given in Thm 2. Such property is known as equivocability. An equivocable commitment scheme [2] has a special trapdoor functionality $(\mathsf{SimCom}, \mathsf{Equiv})$ that allows to forge decommitments. More specifically, a modified setup algorithm $\widehat{\mathsf{Gen}}$ can return extra information sk that allows to compute fake commitments $(c, \sigma) \leftarrow \mathsf{SimCom}_{\mathsf{sk}}$ such that c can be opened to any value using the function $\mathsf{Equiv}_{\mathsf{sk}}$ I.e. for all $(c, \sigma) \leftarrow \mathsf{SimCom}_{\mathsf{sk}}$, $x \in \mathcal{M}$ we have $\mathsf{Open}_{\mathsf{pk}}(c, \mathsf{Equiv}_{\mathsf{sk}}(x, c, \sigma)) = x$. Second, it should be infeasible to distinguish between true and faked commitments. More precisely, let D be a nonuniform polynomial time distinguisher that chooses x according to $\mathsf{pk} \leftarrow \mathsf{Gen}$ and $\widehat{x}$ according to $\widehat{\mathsf{pk}} \leftarrow \widehat{\mathsf{Gen}}$. Then triples (pk, c, d) and $(\widehat{\mathsf{pk}}, \widehat{c}, \widehat{d})$, where $(c, d) \leftarrow \mathsf{Com}_{\mathsf{pk}}(x)$ and $(\widehat{c}, \sigma) \leftarrow \mathsf{SimCom}_{\mathsf{sk}}$ and $\widehat{d} \leftarrow \mathsf{Equiv}_{\mathsf{sk}}(\widehat{x}, \widehat{c}, \sigma)$ must be computationally indistinguishable for D. The corresponding notion of equivocable list commitment schemes are defined analogously allowing D to ask also partial openings. Note that if we use equivocable commitments on the leafs of hash tree, we get equivocable list commitment.

Now, it is straightforward to augment the simulation algorithm in the proof of Thm. 2 by using $\widehat{\mathsf{Gen}}$, $\mathsf{SimCom}_{\mathsf{sk}}$ and $\mathsf{Equiv}_{\mathsf{sk}}$ to fake openings on the fly. Equivocability assures that the output of malicious client must be computationally indistinguishable from the output in World_1.