Threshold Garbled Circuits and Ad Hoc Secure Computation

Michele Ciampi[1], Vipul Goyal[2], and Rafail Ostrovsky[3]

[1] The University of Edinburgh, Edinburgh (UK) michele.ciampi@ed.ac.uk
[2] NTT Research and CMU, Pittsburgh, PA (USA) goyal@cs.cmu.edu
[3] UCLA Department of Computer Science and Department of Mathematics, Los Angeles, CA (USA)
rafail@cs.ucla.edu

Abstract. Garbled Circuits (GCs) represent fundamental and powerful tools in cryptography, and many variants of GCs have been considered since their introduction. An important property of the garbled circuits is that they can be evaluated securely if and only if exactly 1 key for each input wire is obtained: no less and no more. In this work we study the case when: 1) some of the wire-keys are missing, but we are still interested in computing the output of the garbled circuit and 2) the evaluator of the GC might have both keys for a constant number of wires. We start to study this question in terms of non-interactive multi-party computation (NIMPC) which is strongly connected with GCs. In this notion there is a fixed number of parties (n) that can get correlated information from a trusted setup. Then these parties can send an encoding of their input to an evaluator, which can compute the output of the function. Similarly to the notion of *ad hoc secure computation* proposed by Beimel et al. [ITCS 2016], we consider the case when less than n parties participate in the online phase, and in addition we let these parties colluding with the evaluator. We refer to this notion as *Threshold NIMPC*. In addition, we show that when the number of parties participating in the online phase is a fixed threshold $l \leq n$ then it is possible to securely evaluate any l-input function. We build our result on top of a new secret-sharing scheme (which can be of independent interest) and on the results proposed by Benhamouda, Krawczyk and Rabin [Crypto 2017]. Our protocol can be used to compute any function in NC^1 in the information-theoretic setting and any function in P assuming one-way functions.
As a second (and main) contribution, we consider a slightly different notion of security in which the number of parties that can participate in the online phase is not specified, and can be any number c above the threshold l (in this case the evaluator cannot collude with the other parties). We solve an open question left open by Beimel, Ishai and Kushilevitz [Eurocrypt 2017] showing how to build a secure protocol for the case when c is constant, under the Learning with Errors assumption.

1 Introduction

Garbled Circuits (GCs) have played a central role in cryptography. The basic version of GCs has been shown to be useful for secure computation as well as various other areas in cryptography because of its non-interactive nature [Yao86,BMR90,FKN94,NPS99,IK00,LP09]. Various GC variants with additional properties have also played an important role: e.g. GC with free-XOR [KS08], adaptive GC [HJO$^+$16,JSW17,JW16], information-theoretic GCs [Kol05], covert-garbled circuit [CGOS07], and arithmetic GC [AIK11]. Moreover, in general, a garbled circuit can be viewed as a randomized encoding which in turn has played an important role even beyond cryptography in complexity theory [App17]. A key property of a garbled circuit is its "decomposability", i.e., different input wire keys can be computed independently based on the value on that wire (also referred to as decomposable randomized encodings). This for example allows to use a separate 1-out-of-2 Oblivious Transfer (OT) for each input wire. In various applications, this property has played an important role, like in building functional encryption from attribute based encryption [GKP$^+$13], and in building Non-Interactive Multi-Party Computation (NIMPC) [BGI$^+$14] where different parties hold input values corresponding to different input wires. An important property of the garbled circuits is that they can be evaluated securely if and only if exactly 1 key for each input wire is obtained: no less and no more. Moreover, if the evaluator of the garbled circuit has more than one keys (even for a single wire) the security of the garbled circuit is (in general) compromised.

In this work, we ask the following natural question: *what if 1) the keys corresponding to some of the input wires are missing and 2) more than one key for a subset of wires is leaked to the adversary?*

In particular, suppose that a function is well defined even if only a subset of the inputs are present (e.g., the function simply computes the majority, some aggregate statistics like the median or the sorting on the inputs). Furthermore, suppose we only have the wire keys exactly for say l wires (less than the total number of wires n) and that more than one key for a constant number of wires can be leaked to the adversary. *Can we obtain a garbled circuit construction that still allows one to securely compute the function output in this case?*

Here l can be seen as a parameter for the GC construction. This notion, besides being intriguing and interesting in its own right, can also be seen as having natural applications to NIMPC. In NIMPC we can distinguish three main phases: *setup*, *online* and *evaluation*. In this, various parties with inputs and auxiliary information obtained during the setup phase, can encode their inputs and send this encoding to an *evaluator* during an online phase. The evaluator can then compute the output of the function without further interaction with the other parties. Basic constructions of NIMPC readily follows from GC. That is, the setup generates a garbled circuit with n input wires for the function that needs to be computed. Each party p_i receives two wire keys (one for the input 0 and one for the input 1) for the i-th wire. During the online phase each party sends the wire key which corresponds to its input to the evaluator. The evaluator, which now has n wire keys, can evaluate the garbled circuit and obtain the output. Frequently cited example applications of NIMPC are voting and auctions [BKR17,BGI$^+$14]. However, in the case of voting, it is conceivable that several voters might never show up. Can we obtain a system where if a threshold number of voter votes, the result can be obtained? One could also even consider "attribute-based voting" where your attributes determine whether or not you are eligible to vote. For example, in deciding a tenure case, only voters having the attributes of "full professor" and "computer science department" might be eligible. The number and identity of such voters may not necessarily be known at the time of the NIMPC setup (and only an upper-bound on the number of voters is known). Let n be total number of parties, the question we study in this paper is the following:

"Is it possible to obtain a construction of garbled circuits for a function having n input wires s.t. if the wire keys corresponding of $l \leq n$ wires are available, then the output can be securely computed even if both the keys for a constant number of wires are leaked to the adversary?"

A partial answer to the above question has been given in [BIK17], where the authors show how to obtain such a NIMPC protocol under the assumption that the evaluator does not collude with any of the other parties. Another partial answer has been given in [BKR17], where the authors show how to obtain a NIMPC protocol that tolerates a constant number of corruption only for the case where $l = n$, where n is the total number of parties involved in the protocol. However, to the best of our knowledge, we are the first to study the combination of the two problems. In [BIK17] the authors consider another interesting notion called (l, k)*-secure ad hoc private simultaneous messages (PSM)*. This notion is similar to the notion of NIMPC, with the difference that 1) the parties cannot collude with the evaluator and 2) any number k of parties might participate in the online phase of the protocol, with $k \geq l$. Beimel et al. [BIK17] proved that such a notion (for generic values of l and k) would imply obfuscation[4], and left open the following question:

"Is it possible to obtain $(l, l+c)$-secure ad hoc PSM protocol for a constant c?".

1.1 Our Contributions

Our contribution lies in studying the above questions, providing a formal definition, and obtaining various constructions. Our most basic result is the following:

Theorem 1 (informal). *If there exists an l-party NIMPC protocol for the l-input function f which tolerates up to t corruptions, then there exists an n-party Threshold NIMPC protocol that tolerates up to t corruptions that can securely evaluate f when only l of the n parties participate in the online phase.*

This can also naturally be seen a threshold garbled circuit where the message received by the evaluator during the setup phase corresponds to the garbled circuit, whereas the two messages corresponding to two different possibilities of the input (i.e., either 0 or 1) for party p_i can be seen as the two possible wire-keys for the i-th input wire. Our construction also relies on a conceptual tool which we call *positional secret sharing (PoSS)*, which we instantiate information theoretically. Please see the technical overview for more details. We note that our construction, additionally, has the feature that it can handle up to a constant number of corruptions (assuming the input of each player is a single bit). We build upon the construction of Benhamouda et al. [BKR17] with tolerates up to a constant number of corruptions. Informally, this means that the evaluator may be able to compute multiple outputs of the function by flipping the input of the corrupted parties (since the corrupted parties can generate an encoding of both the inputs 0 and 1). However, the evaluator learns no more than having access to an ideal functionality which allows for computing such multiple outputs. As noted in [BKR17], a construction tolerating an arbitrary number of corruptions in this setting implies indistinguishability obfuscation (iO) [BGI$^+$01]. Our second (and main) technical construction is a protocol that retains its security even if more than l input wire keys are given to an evaluator. Going back to the example of voting, while one may have an estimate on how the voter turnout will be (e.g., based on historical data), it might be hard to know the exact number of voters in advance. If the actual number of voters turns out to be even $l+1$ (as opposed to l), all security guarantees cease to exist and our previous construction may become entirely insecure. Towards that end, we ask the following question:

"Is it possible to design construction of garbled circuits where if anywhere between l and $l+c$ inputs wire keys are obtained, the function output can be securely computed?

In other words: can we have an $(l, l+c)$-secure ad hoc PSM protocol? Note that in this setting, the evaluator can compute multiple outputs by selecting any l-sized subset of the received inputs. While ideally, we would like to have $l+c = n$ (for a generic c), such a construction necessarily implies iO and indeed, using iO, a construction where $l+c = n$ can be readily obtained (we recall that n is the total number of parties). However, since our focus is on using standard falsifiable assumptions, we restrict our attention to the case where c is a constant. In addition, our construction allows the input of each party to be a string of arbitrary size. Our main theorem is the following:

[4] The authors of [BIK17] propose inefficient constructions for general functions.

Theorem 2 (informal). *If the LWEs assumption holds, then there exists an n-party $(l, l+c)$-secure ad hoc PSM protocol that can securely evaluate an l-input function f when N parties participate in the online phase with $N \leq l+c \leq n$ for a constant c.*

We stress that N does not need to be known in the setup phase. The last notion that we consider in this paper is *adaptive-ad-hoc PSM*. This notion, in addition to the notion of ad hoc PSM, gives to the evaluator the possibility to evaluate an N-input function f_N, where N is the number of parties that participate in the online phase, with $N \leq l+c \leq n$. This notion gives the same security guarantees as to the notion of $(l, l+c)$-secure ad hoc PSM, but it allows an honest evaluator to evaluate a function even if more than l parties participate in the online phase. It should be easy to see that such a notion can be easily realized using multiple instantiations of an ad hoc PSM scheme. Even in this case, the input of each party can be a string of arbitrary (bounded) length.

2 Technical overview

We start illustrating a new secret sharing scheme which is instrumental for our constructions. Then we show how to use such a secret sharing scheme to construct a threshold NIMPC and an (l, k)-Ad Hoc PSM protocol.

2.1 Positional Secret Sharing (PoSS)

We consider the setting where there is a dealer, n non-colluding parties $\{p_1, \ldots .p_n\}$ and an evaluator. A PoSS scheme allows a dealer to compute a secret sharing of l secrets $x_1, \ldots, x_l$ with respect to a party index j and distribute these shares among the n parties. Let $S = (s_1, \ldots, s_n)$ be the output shares computed by the dealer. Any subset of parties of size l can send their shares to an evaluator, and if the j-th party has the α-th greatest index among these l parties, then the evaluator can reconstruct the α-th secret. If the party p_j does not send its share then none of the secrets can be reconstructed (the j-th share goes always to the party p_j). To construct such a scheme we use a standard t-out-of-m secret sharing scheme. In more detail, the dealer computes 3-out-of-3 secret sharing of x_i obtaining x_i^0, $\tilde{x}_i$ and x_i^1. Then computes 1) an $(i-1)$-out-of-$(j-1)$ secret sharing of x_i^1 thus obtaining the shares $s_{i,1}, \ldots, s_{i,j-1}$, 2) an $(l-i)$-out-of-$(n-j)$ secret sharing of x_i^0 obtaining $s_{i,j+1}, \ldots, s_{i,n}$ and 3) defines $s_{i,i} := \tilde{x}_i$. The output of the sharing algorithm corresponds to $(s_1, \ldots, s_n)$ with $s_i := (s_{1,i}, \ldots, s_{l,i})$ for each $i \in [n]$. Intuitively, if the evaluator receives the shares $S' = (s_{i_1}, \ldots, s_{i_l})$ with $0 \leq i_1 < \cdots < i_l \leq n$ where $j = i_\alpha$ for some α, then she can reconstruct x_α^0 using the shares $s_{i_1}, \ldots, s_{i_{\alpha-1}}$, x_α^1 using the shares $s_{i_{\alpha+1}}, \ldots, s_{i_l}$ and $\tilde{x}_\alpha$, which corresponds to the share s_{i_α}. Note that all the other secrets x_j are protected since there are not enough shares to either reconstruct x_k^0 or x_k^1 for each $k \in [l] - \{\alpha\}$. In the case where there is no i_α with $\alpha = j$, then none of the secrets can be reconstructed since one share of the 3-out-of-3 secret sharing will be missing for each of the secrets.

2.2 Threshold NIMPC

Let f be an l-input function. To obtain a Threshold NIMPC for f that tolerates t corruptions we use a PoSS scheme in combination with a *standard* NIMPC protocol that supports t corruptions and that can be used to evaluate l-input functions. Let $p_1, \ldots, p_n$ be the parties that could participate an execution of the protocol (we recall that a threshold NIMPC is parametrized by l, which represents the maximum number of parties that can participate in the online phase). The idea is to precompute an encoding of the input 0 (that we denote with m_j^0) and of the input 1 (that we denote with m_j^1) for each input slot $j \in [l]$ of the NIMPC scheme. Then we run two instantiations of a PoSS for each party p_i. The first instantiation of the PoSS scheme is run on input the secrets $m_1^0, \ldots, m_l^0$ (and the index i of the party) whereas the second is run using the secrets $m_1^1, \ldots, m_l^1$ (and the index i of the party). Let $(s_{i,1}^0, \ldots, s_{i,n}^0)$ be the output shares of the first instantiation of the PoSS scheme, and $(s_{i,1}^1, \ldots, s_{i,n}^1)$ be the output of the second instantiation for the party p_i. All these shares are then distributed among the n parties. During the online phase each party p_i acts as follows. If the input of p_i is $b_i = 0$ then p_i sends all the shares but the one related to the second instantiation of the PoSS scheme for the index i (i.e., p_i does not send $s_{i,i}^1$), if $b_i = 1$ then p_i sends all the shares but the one related to the first instantiation of the PoSS scheme for the index i (i.e., p_i does not send $s_{i,i}^0$). The security of the PoSS scheme guarantees that if a party p_i does not send the share for one instantiation of PoSS that is run with respect to i, then nothing can be learned about the secrets encoded in that instantiation. In addition, for the case when p_{i_α} sends the share s_{i_α,i_α}^b (with $b \in \{0,1\}$), the PoSS security guarantees that only the secret in position i_α can be learned. Hence, the evaluator can compute $m_1^{b_{i_1}}, \ldots, m_l^{b_{i_l}}$ by running the reconstruction algorithms for the l instantiations of the PoSS scheme for which at least l shares have been provided.[5] These messages then can be used to run the evaluation algorithm of NIMPC protocol to obtain the output of f. In addition, if the NIMPC protocol used in the above construction supports up to t-corruption, so does our scheme. We allow only the corruption of the parties that are participating in the protocol. That is, if l parties provide an input then the corrupted parties belong to this set of parties. We give no security guarantees in any other case (which would give to the colluding evaluator an additional share for the PoSS scheme reaching the total of $l+1$ shares, compromising the security of the PoSS scheme, and in turn, the security of the underling NIMPC protocol). Given the implication of NIMPC with iO, for our construction we consider only the case when the input of each party is a bit, exactly as in [BKR17] (our other constructions do not have this limitation).

2.3 (l,k)-Secure Ad Hoc PSM

The notion of (l,k)-secure ad hoc PSM is similar to the notion of threshold NIMPC with the following two differences: 1) provides the best possible security guarantees in the case when N parties participate in the online phase for an unknown N with $l \leq N \leq k$ and 2) the security holds only if the evaluator does not collude with the other parties. In this work we want to construct a $(l, l+c)$-secure ad hoc PSM for a constant c. Moreover, we want to construct a scheme that allows the input of each party being a bit-string (instead of one bit like in the previous construction). One might think that a threshold NIMPC protocol already satisfies this security notion. We start by describing what are the problems in trying to prove that our threshold NIMPC is an ad hoc PSM, even considering the case when the input of each party is a bit, and then show how our construction works in an incremental fashion. In the threshold NIMPC showed above, if more than l parties are participating to the online phase then more than one secret from each instantiation of the PoSS scheme would be leaked (by the definition of PoSS). Hence, it might be possible for a corrupt evaluator to learn an encoding of different messages for the same input-slots of the NIMPC protocol. Note that this problem could be mitigated if the underlying NIMPC protocol was secure

[5] The shares of the PoSS scheme need to be opportunely permuted to not give a trivial advantage to the adversary. We refer the reader to the technical part of the paper for more detail.

against an arbitrary number of corruptions, but any such a scheme would imply iO. Luckily, we do not really need a NIMPC protocol that supports an arbitrary number of corruptions, but we need a protocol that remains secure in the case when an evaluator, given a set of input $X := (x_{i_1}, \dots, x_{i_{l+c}})$, could run the NIMPC protocol on any subset of size l of X. This property is clearly not enjoyed by a NIMPC protocol that supports a constant number of corruptions. Moreover, even if the problem of corruption and the problem that we are describing here seem related, it looks like a completely different technique is required. To see the problem from a different perspective, the issue of obtaining a secure NIMPC protocol in the case of corruption is related to the fact that an adversary could evaluate the function on strings that have hamming distance at most t from each other. That is, an adversary can flip up to t-bits, obtaining up to 2^t different inputs. In our case, even for $c = 1$, an adversary obtains inputs that have hamming distance l (where l is a polynomial). This is because the adversary, for example, could remove one input in the first position and add a new input in the last position thus causing the shift of the inputs that have not been replaced. Therefore, if the strings are close in terms of editing distance, they could have more than l hamming distance. For this reason, it is not clear how the techniques used to achieve security against corrupted parties (for example those used in [BKR17]) would be helpful in our case.

Quasi-secure ad hoc PSM. We now describe how, at a very high level, our protocol works. We provide an incremental description, starting from a protocol that is not secure, and gradually modifying it until we reach our final result. Let us consider the simplified scenario where we have only four parties p_1, p_2, p_3 and p_4 and we want to construct a $(3, 4)$-Ad Hoc PSM protocol for the 3-input function f. As a main tool, we consider two simple two-party NIMPC protocols (that tolerate no corruption): Π_1 that realizes the function g, Π_2 that realizes the function g_{OUT}. The function g, on input two values (z_1, z_2) concatenates them and creates an encoding of $z_1||z_2$ for the first input slot of Π_2. The function g_{OUT} takes the two inputs $(z_1||z_2, z_3)$ and outputs $f(z_1, z_2, z_3)$.

Given Π_1 and Π_2, each party p_i now prepares an encoding of its input x_i for the first and the second input slot of Π_1 (let us call these encodings Msg_i^0 and Msg_i^1). In addition, each party p_i computes an encoding of x_i for the second input slot of Π_2 (let us call this Msg_i^2). For each party p_i then we run an instantiations of a PoSS scheme with input $(\mathsf{Msg}_i^1, \mathsf{Msg}_i^2, \mathsf{Msg}_i^3, i)$. The security of the PoSS schemes guarantees that if the parties that are participating in the online phase are, for example, p_1 p_2 and p_4, then the evaluator will be able to get $(\mathsf{Msg}_1^1, \mathsf{Msg}_2^2, \mathsf{Msg}_4^3)$ only. The evaluator, at this point can evaluate the function g with the inputs of p_1 and p_2 by running the evaluation algorithm for Π_1 on input Msg_1^1 and Msg_2^2. The output of Π_1 can then be used in combination with Msg_4^3 to run the evaluation algorithm of Π_2 to compute the final output. It should be easy to see that this scheme is a threshold-NIMPC protocol that tolerates no corruption. But we are now interested in the security of the protocol in the case when four parties participate in the online phase. In this case, the PoSS scheme allows the evaluator to get, for example, $(\mathsf{Msg}_1^1, \mathsf{Msg}_2^2, \mathsf{Msg}_4^3)$ and $(\mathsf{Msg}_2^1, \mathsf{Msg}_3^2, \mathsf{Msg}_4^3)$ at the same time. This means that the evaluator can run the evaluation algorithm of Π_1 using $(\mathsf{Msg}_1^1, \mathsf{Msg}_2^2)$ and $(\mathsf{Msg}_2^1, \mathsf{Msg}_3^2)$ thus obtaining two different encodings for different values for the first input slot of Π_2 (assuming that the $x_1||x_2 \neq x_2||x_3$). This corresponds to the case in which the evaluator can collude with a party to generate encodings of multiple inputs for the first input slot of Π_2. Since we do not want to assume that Π_2 is resilient against such an attack[6], we modify the protocol as follows:

[6] We recall that we do not know any NIMPC protocol that is secure in this setting when the inputs of Π_2 are bit strings unless from assuming iO.

- Instead of considering one protocol Π_2 that realizes the function g_{OUT}, we consider λ protocols[7]: $\Pi_2^1, \dots, \Pi_2^\lambda$.
- Each input of g now comes with two random values v_1 and v_2 that each party samples. Hence, the inputs of g now can be seen as $(z_1||v_1, z_2||v_2)$.
- The function g, on input $z_1||v_1$ and $z_2||v_2$ computes $y = z_1||z_2$ and the hash $\mathsf{H}(v_1 \oplus v_2)$ thus obtaining $\mathsf{sel} \in [\lambda]$. Then g encodes y accordingly to the protocol Π_2^{sel}.
- The party p_3 and p_4 now compute an encoding of their input for the second input slot for all the protocols $\Pi_2^1, \dots, \Pi_2^\lambda$.

This mechanism now partially solves the problem of the previous protocol. This is because a different combination of inputs for Π_1 yields to an encoding for a different protocol Π_2^{sel}, with $\mathsf{sel} \in [\lambda]$. Indeed, if the Π_1 is run using the input contributed by p_1 and p_2 then the output of Π_1 corresponds to an encoding of the concatenation of $x_1||x_2$ for the protocol Π_2^{sel} with $\mathsf{sel} = \mathsf{H}(v_1 \oplus v_2)$. If instead Π_1 is run using the input contributed by p_1 and p_3, then we have that $\mathsf{H}(v_1 \oplus v_2) \neq \mathsf{H}(v_1 \oplus v_3) = \mathsf{sel}'$ with some probability $1/p$ (that depends on the choice of λ and on the random coins of the parties). Hence, the output of Π_1 corresponds to an encoding for the protocol $\Pi_2^{\mathsf{sel}'}$. Clearly, λ needs to be polynomially related to the security parameter. This means that the probability of founding a collision for H is non-negligible (and if there is a collision then the security of this protocol collapses back to the security of the previous protocol). Later in this section we show how to solve this problem using the LWE assumption. Before discussing that, we note that this protocol has yet another issue. As we said, the evaluator can get the values $(\mathsf{Msg}_1^1, \mathsf{Msg}_2^2, \mathsf{Msg}_4^3)$ and $(\mathsf{Msg}_2^1, \mathsf{Msg}_3^2, \mathsf{Msg}_4^3)$ when all the parties participate in the online phase. Given that Msg_1^1 and Msg_2^1 represent the encoding of different values for the first input slot of Π_1, then we have an issue similar to the one that we have just discussed. This time, we can solve this problem easily. We simply consider an instantiation of a NIMPC protocol that realizes the function g which we denote with $\Pi_1^{i,j}$, which can be used only by the party i, j, with $i \in \{1, 2\}$ and $j \in \{2, 3, 4\}$. Then, for example, the party p_1 will compute an encoding for the first input slot of $\Pi_1^{1,2}$, $\Pi_1^{1,3}$ and $\Pi_1^{1,4}$, and use all of them as the input of the first instantiation of the PoSS scheme. For the protocol that we have just described, we can prove that for a suitable choice of λ (given that c is a constant value) the probability that there are no collisions in H is $1/p$ where p is a polynomial. Hence, we can prove that the execution of our protocol is secure with probability $1/p$. We note that in this discussion we have assumed that the security of the PoSS scheme is not compromised even when more than l parties provide their shares. In the technical part of the paper we show that our construction of PoSS enjoys a stronger notion, that is indeed sufficient to construct the protocol that we have just described. To extend the above construction to the case when the number of party is more than 4, and the threshold l is an arbitrary value, we just need to consider a longer *chain* of 2-party NIMPC protocols. However, this generalization has to be done carefully to avoid an exponential blowup in the size of the messages. For more details on that, we refer the reader to Sec. 5.

Fully Secure ad hoc PSM. We denote the protocol that we have just described with Π^{PSM} and show how to use it to obtain an ad hoc PSM that is $(l, l+c)$-secure. To amplify the security of Π^{PSM} we make use of a homomorphic secret sharing (HSS) scheme for the function f (we recall that f is the l-input function that we want to evaluate). At a high level, a HSS allows each party i to compute m shares of its input x_i and distribute them among m servers using the algorithm $\mathsf{Share}^{\mathsf{HSS}}$ so that x_i is hidden from any $\mathsf{m}-1$ colluding servers. Each server j can apply a local evaluation algorithm

[7] We discuss the size of λ later in the paper.

$\mathsf{Eval}^{\mathsf{HSS}}$ to its share of the l inputs, and obtain an output share y_j. By combining all the output shares it is possible to obtain the output of the function, that is $y_1 \oplus \cdots \oplus y_{\mathsf{m}} = f(x_1, \ldots, x_l)$.[8] At a very high level, our protocol consists of m instantiations of Π^{PSM} where the e-th instantiation evaluates the function G_e with $e \in [\mathsf{m}]$. The Function G_e takes as input l shares of the HSS scheme, and uses them as input of $\mathsf{Eval}^{\mathsf{HSS}}$ together with the *server index* e (see the bottom of Fig. 8 for a formal specification of G_e). Each party p_i that wants to participate in the protocol computes a secret sharing of its input thus obtaining m shares $(s_1, \ldots, s_{\mathsf{m}})$. Then p_i uses the e-th share as input of the e-th instantiation of Π^{PSM}. The evaluator runs the evaluation algorithm of the e-th instantiation of Π^{PSM} thus obtaining y_e (which corresponds to the output of $\mathsf{Eval}^{\mathsf{HSS}}$ on input the e-th shares of all the parties) for each $e \in [\mathsf{m}]$. The output of the evaluation phase then corresponds to $y_1 \oplus \cdots \oplus y_{\mathsf{m}}$. We show that this protocol is secure as long as there is at least one execution of Π^{PSM} that is secure (i.e., simulatable). Moreover, by choosing m opportunely we can prove that at least one execution of Π^{PSM} is secure with overwhelming probability. Hence, at least one share of each of the inputs of the honest parties will be protected. Therefore, because of the security offered by the HSS, also the input of the parties will be protected.

Adaptive-ad-hoc PSM. It is straightforward to construct an adaptive-ad-hoc PSM having a $(l, l+c)$ ad hoc PSM Π^{APSM}. Indeed, we just need to run c instantiation of Π^{APSM}, where each instantiation computes a function f_α with arity α for each $\alpha \in \{l, \ldots, l+c\}$.

2.4 Related work

The study of MPC protocols with restricted interaction was initiated by Halevi, Lindell, and Pinkas [HLP11,HIJ+16]. We have mentioned the work of Benhamouda et al. [BKR17] which provides the first NIMPC protocol that tolerates up to a constant number of corruptions for all functions in P under OWFs. In addition, the authors show how to obtain a more efficient NIMPC protocol for symmetric functions. The work [BGIK16] introduces the notion of ad hoc PSM and in [BIK17] the authors propose many instantiations of such a primitive in the information-theoretic and computational setting. A result of [BIK17] that is very related to our first contribution, is the construction of an ad hoc PSM protocol for a k-argument function $f : X^k \to Y$ from a NIMPC protocol for a *related* n-argument function $g : (X \cup \{\perp\})^n \to Y$. More precisely, the function g outputs $\perp$ if there are more than $n-k$ inputs that are $\perp$, it outputs the output of f if there are exactly $n-k$ inputs that are $\perp$, in any other cases the output of g is undefined. The compiler that we propose is more generic and it preserves its security against colluding parties (if any). Always in [BIK17] the authors propose an $(l, l+c)$-secure ad hoc PSM protocol for symmetric functions whose complexity is exponential in l, and prove that an (l, k)-ad hoc PSM protocols for simple functions with generic (l, k) already implies obfuscation for interesting functions. In [BKN18] the authors improve the efficiency of the protocols proposed in [BIK17]. The work [HIJ+16] try to make reusable the setup assuming more interactions between the parties, or assuming specific graphs of interaction patterns. In [HIJ+17] the authors successfully remove the need of the parties to obtain correlated randomness from the setup phase via a PKI supplemented with a common random string under the iO assumption. In addition, the construction proposed in [HIJ+17] tolerates arbitrary many corruptions.

[8] In our work we assume that the HSS is additive.

3 Background

Preliminaries. We denote the security parameter by λ and use "$||$" as concatenation operator (i.e., if a and b are two strings then by $a||b$ we denote the concatenation of a and b). For a finite set Q, $x \overset{\$}{\leftarrow} Q$ denotes a sampling of x from Q with uniform distribution. We use "$=$" to check equality of two different elements (i.e. $a = b$ then...), "$\leftarrow$" as the assigning operator (e.g. to assign to a the value of b we write $a \leftarrow$ b). and $:=$ to define two elements as equal. We use the abbreviation PPT that stands for probabilistic polynomial time. We use $\mathsf{poly}(\cdot)$ to indicate a generic polynomial function. We say a function ν is *negligible* if for every positive integer c there is an integer N_c such that for all $x > N_c$, $|\nu(x)| < 1/x^c$. We denote with $[n]$ the set $\{1, \ldots, n\}$, $\mathbb{N}_0$ the set of non-negative integers and with $\mathbb{N}$ the set of positive integer.

3.1 Secret Sharing

A secret sharing scheme allows a dealer to share a secret m among n parties $\mathcal{P} = \{p_1, \ldots, p_m\}$ such that any authorized subset (if any) of $\mathcal{P}$ can reconstruct the secret m, while the other parties learn nothing about m.

We now give the definition of l-out-of-n secret sharing.

Definition 1 (l-out-of-n secret sharing). *A l-out-of-n secret sharing scheme over a message space $\mathcal{M}$ is a pair of* PPT *algorithms (*Share, Reconstruct*) where:*

- Share *on input $x \in \mathcal{M}$ outputs n shares $(s_1, \ldots, s_n)$;*
- Reconstruct *on input l values (shares) outputs a message in $\mathcal{M}$;*

satisfying the following requirements.

- Correctness. $\forall x \in \mathcal{M}$, $\forall S = \{i_1, \ldots, i_l\} \subseteq \{1, \ldots, n\}$ *of size* l,
  Prob $[\, x \leftarrow \mathsf{Reconstruct}(s_{i_1}, \ldots, s_{i_l}) : (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(x)\,] = 1$.
- Security. $\forall x, x' \in \mathcal{M}$, $\forall S \subseteq \{1, \ldots, n\}$ *s.t.* $|S| < l$, *the following distributions are identical:*
  $\{(s_i)_{i \in S} : (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(x)\}$
  $\{(s'_i)_{i \in S} : (s'_1, \ldots, s'_n) \leftarrow \mathsf{Share}(x')\}$.

3.2 Homomorphic Secret Sharing (HSS)

We consider HSS scheme that supports the evaluation of a function f on shares of inputs $x_1, \ldots x_n$ that are originated from different clients. In this notion each client i can compute m shares of its input x_i and distribute them between m servers using the algorithm $\mathsf{Share}^{\mathsf{HSS}}$ so that x_i is hidden from any $m - 1$ colluding servers. Each server j can apply a local evaluation algorithm $\mathsf{Eval}^{\mathsf{HSS}}$ to its share of the n inputs, and obtains an output share y_j.

The output $f(x_1, \ldots, x_n)$ is reconstructed by applying a decoding algorithm $\mathsf{Dec}^{\mathsf{HSS}}$ to the output shares $y_1, \ldots, y_m$.

Definition 2 (HSS [BGI+18]). *An n-client, m-server, t-secure homomorphic secret sharing scheme for a function $f : (\{0,1\}^\star)^{n+1} \to \{0,1\}^\star$, or (n, m, t)-HHS for short, is a triple of* PPT *algorithms* $(\mathsf{Share}^{\mathsf{HSS}}, \mathsf{Eval}^{\mathsf{HSS}}, \mathsf{Dec}^{\mathsf{HSS}})$ *where:*

- $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, i, x)$: *On input 1^λ (security parameter), $i \in [n]$ (client index) and $x \in \{0,1\}^\star$ (client input), the sharing algorithm* $\mathsf{Share}^{\mathsf{HSS}}$ *outputs m input shares $(x^1, \ldots, x^m)$.*

- $\mathsf{Eval}^{\mathsf{HSS}}(j, x_0, (x_1^j, \dots, x_n^j))$: *On input $j \in [m]$ (server index), $x_0 \in \{0,1\}^\star$ (common server input), and $x_1^j, \dots, x_n^j$ (j-th share of each client input), the evaluation algorithm $\mathsf{Eval}^{\mathsf{HSS}}$ outputs $y^j \in \{0,1\}^\star$, corresponding to the server j's share of $f(x_0; x_1, \dots, x_n)$.*
- $\mathsf{Dec}^{\mathsf{HSS}}(y^1, \dots, y^m)$: *On input $(y^1, \dots, y^m)$ (list of output shares), the decoding algorithm $\mathsf{Dec}^{\mathsf{HSS}}$ computes a final output $y \in \{0,1\}^\star$.*

The algorithm $(\mathsf{Share}^{\mathsf{HSS}}, \mathsf{Eval}^{\mathsf{HSS}}, \mathsf{Dec}^{\mathsf{HSS}})$ should satisfy the following correctness and security requirements:

- **Correctness:** *For any $n+1$ inputs $x_0, \dots, x_n \in \{0,1\}^\star$,*
  $\mathrm{Prob}[\forall i \in [n] (x_i^1, \dots x_i^m) \xleftarrow{\$} \mathsf{Share}^{\mathsf{HSS}}(1^\lambda, i, x_i),\ \forall j \in [m]\ y^j \xleftarrow{\$} \mathsf{Eval}^{\mathsf{HSS}}(j, x_0, (x_1^j, \dots, x_n^j)) : \mathsf{Dec}^{\mathsf{HSS}}(y^1, \dots, y^m) = f(x_0; x_1, \dots, x_n)] = 1 - \nu(\lambda)$.
- **Security:** *: Consider the following semantic security challenge experiment for corrupted set of server $T \subset [m]$:*
  1. *The stateful adversary gives challenge index and inputs $(i, x_0, x_1) \leftarrow \mathcal{A}(1^\lambda)$, with $i \in [n]$ and $|x_0| = |x_1|$.*
  2. *The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(x^1, \dots, x^m) \xleftarrow{\$} \mathsf{Share}^{\mathsf{HSS}}(1^\lambda, i, x_b)$.*
  3. *The adversary outputs $b' \leftarrow \mathcal{A}((x^j)_{j \in T})$ given the shares for corrupted T.*

  Denote by $a := \mathrm{Prob}[\, b = b'\,] - 1/2$ the advantage of $\mathcal{A}$ in guessing b in the above experiment, where probability is taken over the randomness of the challenger and of $\mathcal{A}$. For circuit size bound $S = S(\lambda)$ and advantage bound $\alpha = \alpha(\lambda)$, we say that an (n, m, t)-HSS scheme Π is (S, α)-secure if for all $T \subset [m]$ of size $|T| \leq t$, and all non-uniform adversaries $\mathcal{A}$ of size $S(\lambda)$, we have $a \leq \alpha(\lambda)$. We say that Π is computationally secure if it is $(S, 1/S)$-secure for all polynomials S.

In this work we consider only *additive* HSS schemes. An HHS scheme is additive if $\mathsf{Dec}^{\mathsf{HSS}}$ outputs the exclusive or of the m output shares. For our construction we make use of an additive $(n, m, m-1)$-HSS scheme. Such a scheme can be constructed from the LWEs assumption [BGI+18,DHRW16].

4 Our Model

In this section we propose the formal definition of NIMPC. We give a more general definition that captures the case when up to t parties can collude with the evaluator, and following [HLP11,HIJ+16,BKR17], we refer to this notion as t-robust NIMPC. Then we give our new definition of threshold NIMPC which can be seen as a combination of the notion of NIMPC with the notion of ad hoc PSM proposed in [BGI+14]. Let $\mathcal{X}$ be non-empty sets and let $\mathcal{X}$ denote the Cartesian product $\mathcal{X}^n := \mathcal{X} \times \cdots \times \mathcal{X}$.

Definition 3 (NIMPC Protocol. [BKR17]). *Let $\mathcal{F} = (\mathcal{F}_n)_{n \in \mathbb{N}}$ be an ensemble of sets $\mathcal{F}_n$ of functions $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is a finite set. A non-interactive secure multiparty computation (NIMPC) protocol for $\mathcal{F}$ is a tuple of three algorithms $\Pi := (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Eval})$, where:*

- Setup *takes as input unary representations of n and of the security parameter λ, and a representation of function $f \in \mathcal{F}_n$ and outputs a tuple $(\rho_0, \rho_1, \dots, \rho_n)$;*
- Msg *takes as input a value ρ_i, and an input $x_i \in \mathcal{X}$, and deterministically outputs a message m_i;*
- Eval *takes as input a value ρ_0 and a tuple of n messages $(m_1, \dots, m_n)$ and outputs an element in $\mathcal{Y}$ satisfying the following property:*

C*orrectness. For any* $n \in \mathbb{N}$*, security parameter* $\lambda \in \mathbb{N}_0$*,* $f \in \mathcal{F}_n$*,* $x := (x_1, \ldots, x_n) \in \mathcal{X}$*, and* $(\rho_0, \ldots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(1^n, 1^\lambda, f)$*,*
$\mathsf{Eval}(\rho_0, \mathsf{Msg}(\rho_1, x_1), \ldots, \mathsf{Msg}(\rho_n, x_n)) = f(x)$*.*

While the previous definition is abstract, in the sequel, we will often see NIMPC protocols as protocols with n parties $p_1, \ldots, p_n$ with respective inputs $x_1, \ldots, x_n$ and an evaluator p_0. A polynomial-time NIMPC protocol for $\mathcal{F}$ is an NIMPC protocol $(\mathsf{Setup}, \mathsf{Msg}, \mathsf{Eval})$ where Setup, Msg, and Eval run in polynomial time in n and λ. In particular, functions $f \in \mathcal{F}$ should be representable by polynomial-size bit strings.

Robustness. For a subset $T = \{i_1, \ldots, i_t\} \subseteq [n]$ and $x = (x_1, \ldots, x_n)$, we denote by $\overline{x}_T$ the t-coordinate projection vector $(x_{i_1}, \ldots, x_{i_t})$. For a function $f : \mathcal{X}^n \to \mathcal{Y}$, we denote by $f|_{\overline{T}, x_{\overline{T}}}$ the function f with the inputs corresponding to positions $\overline{T}$ fixed to the entries of the vector x.

We now recall the notions of robustness for NIMPC protocols. Informally, T-robustness $T \subseteq \{1, \ldots, n\}$ for a set T of colluding parties means that if $x_{\overline{T}}$ represents the inputs of the honest parties, then an evaluator colluding with the parties in set T can compute the residual function $f|_{\overline{T}, x_{\overline{T}}}$ on any input $x_{\overline{T}}$ but cannot learn anything else about the input of the honest parties. This describes the best privacy guarantee attainable in this adversarial setting. The formal definition is stated in terms of a simulator that can generate the view of the adversary (evaluator plus the colluding parties in set T) with sole oracle access to the residual function $f|_{\overline{T}, x_{\overline{T}}}$.

Definition 4 (NIMPC Robustness [BKR17]). *Let* $n \in \mathbb{N}$ *and* $T \subseteq \{1, \ldots, n\}$*. A NIMPC protocol* Π *is perfectly (resp., statistically, computationally)* T*-robust if there exists a* PPT *algorithm* Sim *(called simulator) such that for any* $f \in \mathcal{F}_n$ *and* $x_{\overline{T}} \in \mathcal{X}_{\overline{T}}$*, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:* $\{\mathsf{Sim}^{f|_{\overline{T}, x_{\overline{T}}}}(1^n, 1^\lambda, T)\}, \{\mathsf{View}(1^n, 1^\lambda, f, T, x_{\overline{T}})\}$*, where* $\{\mathsf{View}(1^n, 1^\lambda, f, T, x_{\overline{T}})\}$ *is the view of the evaluator* p_0 *and of the colluding parties* p_i *(for* $i \in T$*) from running* $\Pi := (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Eval})$ *on input* $x_{\overline{T}}$ *for the honest parties: that is,* $((m_i)_{i \in \overline{T}}, \rho_0, (\rho_i)_{i \in T})$ *where* $(\rho_0, \ldots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(1^n, 1^\lambda, f)$ *and* $m_i \leftarrow \mathsf{Msg}(\rho_i, x_i)$ *for all* $i \in \overline{T}$ *where* $x_{\overline{T}} := (x_i)_{i \in \overline{T}}$*.*

Let $t \in \mathbb{N}_0$ *be a function of* n*, then a NIMPC protocol* Π *is perfectly (resp., statistically, computationally)* t*-robust if for any* $n \in \mathbb{N}$ *and any* $T \subseteq \{1, \ldots, n\}$ *of size at most* $t = t(n)$*,* Π *is perfectly (resp., statistically, computationally)* T*-robust.*

Robustness does not necessarily imply that the simulator Sim is the same for any n and T. In this and in the following notions we consider only PPT simulators since in this paper we focus only on efficiently simulatable protocols.

4.1 Threshold NIMPC

We introduce the new notion of *Threshold NIMPC*. A Threshold NIMPC is parametrized by n and l with $0 \leq l \leq n$, where n denotes the number of parties and l represents a threshold. Given a set of n parties $\mathcal{P}$, any subset of $\mathcal{P}' \subseteq \mathcal{P}$ of size l can evaluate the function $f : \mathcal{X}^l \to \mathcal{Y}$, where $\mathcal{Y}$ is a finite set and $\mathcal{X} = \{\{0,1\}^\lambda, \{1, \ldots n\}\}$. In more details, we assume that any party in $\mathcal{P}$ is univocally identified by an index $i \in [n]$. The setup algorithm and the algorithm used by the parties to generate an encoding of their inputs have the same interface as the algorithms of a NIMPC protocol. The difference is in the evaluation algorithm. In this notion we do not require all the n parties to participate in the protocol in order to evaluate a function. That is, any subsets of $\mathcal{P}$ of size l would allow the evaluator to compute the function f. Without loss of generality, we consider only functionalities whose output depends on the inputs of the parties, and on the indexes of the parties that contributed with these inputs. Formally, the class of function supported by our protocol is described in Fig. 1 (where g can be any function).

Input: $\big((x_{i_1}, i_1), \ldots (x_{i_l}, i_l)\big)$ where $\{i_1, \ldots, i_l\} \subseteq [n]$, $x_{i_1}, \ldots, x_{i_l} \in \mathcal{X}$, $l \leq n$ and $n \in \mathbb{N}$.
Output: Let $(j_1, \ldots, j_l)$ be a permutation of the values $(i_1, \ldots, i_l)$ such that $1 \leq j_1 < j_2 < \cdots < j_{l-1} < j_l \leq n$ and output $\perp$ if such a permutation does not exist, else, output $g\big(x_{j_1}, \ldots, x_{j_l}\big)$

Fig. 1: Class of functionalities supported by our threshold NIMPC protocol.

Definition 5 (Threshold NIMPC Protocol). *Let* $\mathcal{F} = (\mathcal{F}_l)_{l \in \mathbb{N}}$ *be an ensemble of sets* $\mathcal{F}_l$ *of functions* $f : \mathcal{X} \to \mathcal{Y}$*, a Threshold NIMPC protocol for* $\mathcal{F}$ *is a tuple of three algorithms* $(\mathsf{Setup}^{\mathsf{th}}, \mathsf{Msg}^{\mathsf{th}}, \mathsf{Eval}^{\mathsf{th}})$*, where:*

- $\mathsf{Setup}^{\mathsf{th}}$ *takes as input unary representations of* n*,* l *and of the security parameter* λ *with* $1 \leq l \leq n$*, and a representation of function* $f \in \mathcal{F}_l$ *and outputs a tuple* $(\rho_0, \rho_1, \ldots, \rho_n)$*;*
- $\mathsf{Msg}^{\mathsf{th}}$ *takes as input a value* ρ_i*, and an input* $x_i \in \mathcal{X}$*, and deterministically outputs a message* m_i*;*
- $\mathsf{Eval}^{\mathsf{th}}$ *takes as input a value* ρ_0 *and a tuple of* n *messages* $(m_{j_1}, \ldots, m_{j_l})$ *with* $1 \leq j_1 < \cdots < j_l \leq n$ *and outputs an element in* $\mathcal{Y}$*;*

satisfying the following property:

C*orrectness. For any* $n \in \mathbb{N}$*, security parameter* $\lambda \in \mathbb{N}_0$*,* $f \in \mathcal{F}_l$*,* $x := \big((x_{j_1}, j_1), \ldots, (x_{j_l}, j_l)\big) \in \mathcal{X}$*, with* $1 \leq j_1 < \cdots < j_l \leq n$ *and* $(\rho_0, \ldots, \rho_n) \overset{\$}{\leftarrow} \mathsf{Setup}^{\mathsf{th}}(1^n, 1^l, 1^\lambda, f)$*,*

$$\mathsf{Eval}^{\mathsf{th}}(\rho_0, \mathsf{Msg}^{\mathsf{th}}(\rho_{j_1}, x_{j_1}), \ldots, \mathsf{Msg}^{\mathsf{th}}(\rho_{j_l}, x_{j_l})) = f\big((x_{j_1}, j_1), \ldots, (x_{j_l}, j_l)\big).$$

Definition 6 (Threshold NIMPC Security). *Let* $n \in \mathbb{N}$*,* $K := \{j_1, \ldots, j_l\}$ *with* $1 \leq j_1 < \cdots < j_l \leq n$*,* $T \subseteq K$ *and* $\overline{T} := K - T$*. A Threshold NIMPC protocol* Π *is perfectly (resp., statistically, computationally)* T*-secure if there exists a* PPT *algorithm* Sim *(called simulator) such that for any* $f \in \mathcal{F}_l$ *and* $x_{\overline{T}} \in \mathcal{X}_{\overline{T}}$*, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:*

$$\{\mathsf{Sim}^{f|_{\overline{T}, x_{\overline{T}}}}(1^n, 1^l, 1^\lambda, T, K)\}, \{\mathsf{View}(1^n, 1^l, 1^\lambda, f, T, K, x_{\overline{T}})\}$$

where $\{\mathsf{View}(1^n, 1^l, 1^\lambda, f, T, K, x_{\overline{T}})\}$ *is the view of the evaluator* p_0 *and of the colluding parties* p_i *(for* $i \in T$*) from running* Π *on input* $x_{\overline{T}}$ *for the honest parties: that is,* $((m_i)_{i \in \overline{T}}, \rho_0, (\rho_i)_{i \in T})$ *where* $(\rho_0, \ldots, \rho_n) \overset{\$}{\leftarrow} \mathsf{Setup}(1^n, 1^l, 1^\lambda, f)$ *and* $m_i \leftarrow \mathsf{Msg}(\rho_i, x_i)$ *for all* $i \in \overline{T}$*.*[9] *Let* $t, l, n \in \mathbb{N}_0$ *be such that* $0 \leq t \leq l \leq n$*, a Threshold NIMPC protocol* Π *is perfectly (resp., statistically, computationally)* t*-secure if for any* $K \subseteq [n]$ *with* $|K| \leq l$*, and any* $T \subseteq K$ *such that* $K = T \cup \overline{T}$ *with* $|T| \leq t$*,* Π *is perfectly (resp., statistically, computationally)* T*-secure.*

[9] $f|_{\overline{T}, x_{\overline{T}}}$ works as before, with the difference that it outputs $\perp$ in the case where less than $|K| < l$.

4.2 Ad Hoc PSM

An (l,t)-secure ad hoc PSM protocol $\mathit{\Pi}$ is a 0-secure threshold NIMPC that remains secure even if more than l (and less than t) parties participate in the online phase. In other words, the evaluator cannot collude with any of the other parties, but the protocol remains secure for any number N of parties participating in the protocol with $N \leq t$. Moreover, the evaluator can compute the output if $N \geq l$. By secure here we mean that the adversary can evaluate the function f on any combination of size l of the inputs provided by the honest parties and learns nothing more than that. More formally, if $\overline{x} := ((x_{i_1}, i_1), \ldots, (x_{i_{\text{fl}}}, i_N))$ represents the inputs of the N parties participating in the online phase, then a malicious party can compute f on any input $\overline{x}_K$ where $K := \{j_1, \ldots, j_l\}$ with $1 \leq j_1 < \cdots < j_l \leq n$, $K \subseteq \{i_1, \ldots, i_N\}$ but cannot learn anything else. This describes the best privacy guarantee attainable in this setting. The formal definition is stated in terms of a simulator that can generate the view of the adversary with sole oracle access to $\mathcal{O}_f$, where $\mathcal{O}_f$ takes as input a set $K := \{j_1, \ldots, j_l\}$ with $1 \leq j_1 < \cdots < j_l \leq n$, $K \subseteq \{i_1, \ldots, i_N\}$ and returns $f\big((x_{j_1}, j_1), \ldots, (x_{j_l}, j_l)\big)$.

The definition that we provide is essentially the same as the one provided in [BIK17], we just use a different terminology to be consistent with our other definitions.

Definition 7 (Ad Hoc PSM). *Let $n, l, t, \lambda \in \mathbb{N}_0$ and $K := \{j_1, \ldots, j_N\}$ with $0 \leq j_1 < \cdots < j_N \leq n$ such that $0 \leq N \leq t$. An ad hoc PSM protocol is perfectly (resp., statistically, computationally) K-secure if there exists a* PPT *algorithm* Sim *(called simulator) such that for any $f \in \mathcal{F}_l$, $\overline{x} := \big((x_{j_1}, j_1), \ldots, (x_{j_N}, j_N)\big)$, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:*

$$\{\mathsf{Sim}^{\mathcal{O}_f}(1^n, 1^l, 1^\lambda, K)\}, \{\mathsf{View}(1^n, 1^l, 1^\lambda, f, K, \overline{x})\}$$

where $\{\mathsf{View}(1^n, 1^l, 1^\lambda, f, K, \overline{x})\}$ is the view of the evaluator p_0 from running $\mathit{\Pi}$ on input $\overline{x}$ for the honest parties: that is, $((m_i)_{i \in K}, \rho_0)$ where $m_i \leftarrow \mathsf{Msg}(\rho_i, x_i)$ for all $i \in K$ and $(\rho_0, \ldots, \rho_n) \overset{\$}{\leftarrow}$ $\mathsf{Setup}(1^n, 1^l, 1^\lambda, f)$. We say that an ad hoc PSM protocol $\mathit{\Pi}$ is perfectly (resp., statistically, computationally) (l,t)-secure if for any $N \leq t$, any $K := \{j_1, \ldots, j_N\}$, $\mathit{\Pi}$ is perfectly (resp., statistically, computationally) K-secure.

4.3 Adaptive-ad-hoc PSM

An adaptive-ad-hoc PSM protocol is parametrized by the number of parties n, the threshold l, an integer t with $0 \leq t \leq n$ and a set of functions $f_l, \ldots, f_\beta$, and allows an honest evaluator to obtain the evaluation of a function f_N if the number of parties that are participating in the protocol is $l \leq N \leq \beta$, for any $N \in \{l, \ldots, \beta\}$. Informally, an adaptive-ad-hoc PSM protocol can be seen as a protocol that allows evaluating a function that accepts a variable number of inputs. The formal definition is stated in terms of a simulator that can generate the view of the adversary with sole oracle access to $\mathcal{O}$, where $\mathcal{O}$ takes as input an index $N \in \{l, \ldots, \beta\}$ for the function f_N, a set $K := \{j_1, \ldots, j_N\}$ with $0 \leq j_1 < \cdots < j_N \leq n$, $K \subseteq \{i_1, \ldots, i_N\}$ and returns $f_N\big((x_{j_1}, j_1), \ldots, (x_{j_l}, j_N)\big)$.[10]

Definition 8 (Adaptive-ad-hoc PSM Protocol). *Let $\mathcal{F} = (\mathcal{F}_n)_{n \in \mathbb{N}}$ be an ensemble of sets $\mathcal{F}_n$ of functions $f : \mathcal{X} \to \mathcal{Y}$, an adaptive-ad-hoc PSM protocol for $\mathcal{F}$ is a tuple of three algorithms* $(\mathsf{Setup}^{\mathsf{AdT}}, \mathsf{Msg}^{\mathsf{AdT}}, \mathsf{Eval}^{\mathsf{AdT}})$*, where:*

[10] Also in this case, the oracle outpus $\perp$ in the case where $N < l$.

- $\mathsf{Setup}^{\mathsf{AdT}}$ *takes as input unary representations of* n, l, t *and of the security parameter* λ *with* $0 \leq t \leq n$, *and a representation of the functions* $\overline{f} := f_l, \dots f_\beta$ *with* $f_\alpha \in \mathcal{F}_\alpha$ *for each* $\alpha \in \{l, \dots, \beta\}$, *and outputs a tuple* $(\rho_0, \rho_1, \dots, \rho_n)$;
- $\mathsf{Msg}^{\mathsf{AdT}}$ *takes as input a value* ρ_i, *and an input* $x_i \in \mathcal{X}$, *and deterministically outputs a message* m_i;
- $\mathsf{Eval}^{\mathsf{AdT}}$ *takes as input a value* ρ_0 *and a tuple of* N *messages* $(m_{j_1}, \dots, m_{j_N})$ *with* $l \leq N \leq t$, $0 \leq j_1 < \cdots < j_N \leq n$, *and outputs an element in* $\mathcal{Y}$;

  satisfying the following property:

C*orrectness. For any* $n, t, l, N \in \mathbb{N}$, *security parameter* $\lambda \in \mathbb{N}_0$, $\overline{f}$, $x := \big((x_{j_1}, j_1), \dots, (x_{j_N}, j_N)\big) \in \mathcal{X}$, *with* $l \leq N \leq \beta \leq n$, $0 \leq j_1 < \cdots < j_N \leq n$ *and* $(\rho_0, \dots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(1^n, 1^l, 1^t, 1^\lambda, \overline{f})$, $\mathsf{Eval}(\rho_0, \mathsf{Msg}(\rho_{j_1}, x_{j_1}), \dots, \mathsf{Msg}(\rho_{j_N}, x_{j_N})) = f_N\big((x_{j_1}, i_1), \dots, (x_{j_N}, j_N)\big)$.

Definition 9 (Adaptive-ad-hoc PSM security). *Let* $n, l, t, \lambda \in \mathbb{N}$, $K := \{j_1, \dots, j_N\}$ *with* $0 \leq j_1 < \cdots < j_N \leq n$ *such that* $0 \leq N \leq t \leq n$. *An adaptive-ad-hoc PSM protocol is perfectly (resp., statistically, computationally)* K*-secure if there exists a* PPT *algorithm* Sim *(called simulator) such that for any* $\overline{f}$, $\overline{x} := \big(x_{j_1}, j_1), \dots, (x_{j_N}, j_N\big)$, *the following distributions are perfectly (resp., statistically, computationally) indistinguishable:*

$$\{\mathsf{Sim}^{\mathcal{O}}(1^n, 1^l, 1^c, 1^\lambda, K)\}, \{\mathsf{View}(1^n, 1^l, 1^c, 1^\lambda, \overline{f}, K, \overline{x})\}$$

where $\{\mathsf{View}(1^n, 1^l, 1^c, 1^\lambda, f, K, \overline{x})\}$ *is the view of the evaluator* p_0 *from running* Π *on input* $\overline{x}$ *for the honest parties: that is,* $((m_i)_{i \in K}, \rho_0)$ *where* $m_i \leftarrow \mathsf{Msg}(\rho_i, x_i)$ *for all* $i \in K$ *and* $(\rho_0, \dots, \rho_n) \xleftarrow{\$} \mathsf{Setup}(1^n, 1^l, 1^\lambda, f)$.

We say that an adaptive-ad-hoc PSM protocol Π *is perfectly (resp., statistically, computationally)* (l, t)*-secure if for any* $K := \{j_1, \dots, j_N\}$ Π *is perfectly (resp., statistically, computationally)* K*-secure with* $N \leq t$.

5 Positional Secret Sharing (PoSS)

In this section we propose new notions of secret sharing schemes, and provide an information theoretical instantiation of them. These new definitions represent one of the main building block of our NIMPC protocols. We now introduce the first notion that we call *Positional Secret Sharing (PoSS)*. Let $\mathcal{P} := \{p_1, \dots, p_n\}$ be a set of parties and $X := (x_1, \dots, x_l)$ be a sequence of secrets. A PoSS scheme is defined with respect to a party $p_j \in \mathcal{P}$. In a PoSS scheme a dealer can compute a secret sharing of X thus obtaining $s_1, \dots, s_n$ and distribute s_i to p_i for all $i \in \{1, \dots, n\}$. Let $\mathcal{P}' := \{p_{j_1}, \dots, p_{j_l}\}$ be an arbitrary chosen set of l parties with $0 \leq j_1 < j_2 < \cdots < j_{l-1} < j_l \leq n$. On input $(s_{j_1}, \dots, s_{j_l})$ with $j_\alpha = j$ for some $\alpha \in \{1, \dots, l\}$ an evaluator can compute x_α and nothing more. If there is no $j_\alpha = j$ or less than l shares are available then all the secrets remain protected. We now propose a formal definition of PoSS.

Definition 10 (Positional Secret Sharing). *A PoSS scheme over a message space* $\mathcal{M}$ *is a pair of* PPT *algorithms (*$\mathsf{Share}^{\mathsf{PoSS}}$, $\mathsf{Reconstruct}^{\mathsf{PoSS}}$*) where:*

- $\mathsf{Share}^{\mathsf{PoSS}}$ *takes as input* $X := (x_1, \dots, x_l)$, *the number of parties* n *and an index* $j \in [n]$, *and outputs* n *shares* $(s_1, \dots, s_n)$;
- $\mathsf{Reconstruct}^{\mathsf{PoSS}}$ *takes as input* l *values (shares), the index* j *and outputs a message in* $\mathcal{M}$ *(where* $\mathcal{M}$ *denotes the message space);*

satisfying the following requirements.

Correctness. $\forall x_1, \ldots, x_l \in \mathcal{M}^l$, $\forall S = \{j_1, \ldots, j_l\} \subseteq \{1, \ldots, n\}$ *with* $j_1 < j_2 < \cdots < j_{l-1} < j_l$, *if there exists* $\alpha \in \{1, \ldots, l\}$ *such that* $j_\alpha = j$ *then*
$\text{Prob}\left[\, x_\alpha \leftarrow \mathsf{Reconstruct}^{\mathsf{PoSS}}(s_{j_1}, \ldots, s_{j_l}, j) : (s_1, \ldots, s_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x_1, \ldots, x_l), j) \,\right] = 1.$[11]

Standard security. $\forall (x_1, \ldots, x_l), (x'_1, \ldots, x'_l) \in \mathcal{M}^l$, $\forall S \subseteq \{1, \ldots, n\}$ *s.t.* $|S| < l$, *the following distributions are identical:*
$\{(s_i)_{i \in S} : (s_1, \ldots, s_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x_1, \ldots, x_l), j)\}$
$\{(s'_i)_{i \in S} : (s'_1, \ldots, s'_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x'_1, \ldots, x'_l), j)\}$

Positional security. $\forall (x_1, \ldots, x_l), (x'_1, \ldots, x'_l) \in \mathcal{M}^l$, $\forall S = \{j_1, \ldots, j_l\} \subseteq \{1, \ldots, n\}$ *with* $j_1 < j_2 < \cdots < j_{l-1} < j_l$:

1. *if there exists* $\alpha \in \{1, \ldots, l\}$ *such that* $j_\alpha = j$, *the following distributions are identical:*
   $\{(s_i)_{i \in S} : (s_1, \ldots, s_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x_1, \ldots, x_{\alpha-1}, x_\alpha, x_{\alpha+1} \ldots, x_l), j)\}$
   $\{(s'_i)_{i \in S} : (s'_1, \ldots, s'_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x'_1, \ldots, x'_{\alpha-1}, x_\alpha, x'_{\alpha+1}, \ldots, x'_l), j)\}$.
2. *if* $\nexists \alpha \in \{1, \ldots, l\}$ *such that* $j_\alpha = j$, *the following distributions are identical:* $\{(s_i)_{i \in S} : (s_1, \ldots, s_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x_1, \ldots, x_l), j)\}$
   $\{(s'_i)_{i \in S} : (s'_1, \ldots, s'_n) \overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}}((x'_1, \ldots, x'_l), j)\}$

5.1 PoSS: Our Construction

We denote our scheme with $(\mathsf{Share}^{\mathsf{PoSS}^\star}, \mathsf{Reconstruct}^{\mathsf{PoSS}^\star})$. $\mathsf{Share}^{\mathsf{PoSS}^\star}$ takes as input $X := (x_1, \ldots, x_l)$ and the index j and executes the following steps.

- Pick $\tilde{x}_1 \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and set $x_1^1 \leftarrow \tilde{x}_1 \oplus x_1$.
- Construct an $(l-1)$-out-of-$(n-j)$ secret sharing for x_1^1 thus obtaining $s_{1,j+1}, \ldots, s_{1,n}$.
- Define $s_{1,j} := \tilde{x}_1$ and $s_{1,1} := \perp, ..., s_{1,j-1} := \perp$.
- Pick $\tilde{x}_l \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and set $x_l^0 \leftarrow \tilde{x}_l \oplus x_l$.
- Construct an $(l-1)$-out-of-$(j-1)$ secret sharing for x_l^0 thus obtaining $s_{i,1}, \ldots, s_{i,j-1}$.
- Define $s_{l,j} := \tilde{x}_1$ and $s_{l,j+1} := \perp, ..., s_{l,n} := \perp$.
- For $i = 2, \ldots, l-1$
  1. Pick $x_i^0, x_i^1 \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and compute $\tilde{x}_i \leftarrow x_i^0 \oplus x_i^1 \oplus x_i$.
  2. Construct an $(i-1)$-out-of-$(j-1)$ secret sharing for x_i^0 thus obtaining $s_{i,1}, \ldots, s_{i,j-1}$.
  3. Construct a $(l-i)$-out-of-$(n-j)$ secret sharing for x_i^1 thus obtaining $s_{i,j+1}, \ldots, s_{i,n}$.
  4. Define $s_{i,j} := \tilde{x}_i$.
- For $i = 1, \ldots, n$ set $s_i = (s_{1,i}, \ldots, s_{l,i})$.
- Output $(s_1, \ldots, s_n)$.

The algorithm $\mathsf{Reconstruct}^{\mathsf{PoSS}^\star}$ takes as input $(s_{j_1}, \ldots, s_{j_l})$ and the index j, and executes the following steps.

1. If there does not exist α such that $j_\alpha = j$ then output $\perp$ else continue as follows.
2. For $i = 1, \ldots, l$ parse s_{j_i} as $(s_{1,j_i}, \ldots, s_{l,j_i})$.
3. If $\alpha = 1$ then use the shares $s_{1,j_2}, \ldots, s_{1,j_l}$ to reconstruct x_1^1 and output $x_1^1 \oplus s_{1,j_1}$.
4. If $\alpha = l$ then use the shares $s_{l,j_1}, \ldots, s_{l,j_{l-1}}$ to reconstruct x_1^0 and output $x_1^0 \oplus s_{1,j_l}$.
5. If $\alpha \in \{2, \ldots, l-1\}$ then use the shares $s_{\alpha,j_1}, \ldots, s_{\alpha,j_{\alpha-1}}$ to reconstruct x_α^0.

[11] We denote with $\mathcal{M}^l$ the l applications of the Cartesian product on $\mathcal{M}$.

6. Use the shares $s_{\alpha,j_{\alpha+1}},\dots,s_{\alpha,j_l}$ to reconstruct x_α^1.
7. Output $x_\alpha \leftarrow x_\alpha^0 \oplus x_\alpha^1 \oplus s_{\alpha,j_\alpha}$.

We note passing that a PoSS scheme could be constructed from monotone span programs [KW93]. However, for some of our applications we need a PoSS scheme that is also secure under a stronger notion (*enhanced PoSS*). For this reason we have provided one ad-hoc scheme that relies on standard k-out-of-m secret sharing and that can be proven secure under the notion of PoSS and its stronger variant.

Theorem 1. $(\mathsf{Share}^{\mathsf{PoSS}\star}, \mathsf{Reconstruct}^{\mathsf{PoSS}\star})$ *is a PoSS scheme.*

We refer the reader to App. A.1 for a formal proof of the theorem. We now present the notion of Enhanced Positional Secret Sharing (ePoSS). An ePoSS scheme is a PoSS scheme with an additional security property that guarantees the protection of some of the secret inputs even when an adversary obtains more than l shares. In more detail, the notion of PoSS guarantees that when l shares are available one of the l secret can be reconstructed, and nothing about the other $l-1$ secrets is leaked. The notion of ePoSS guarantees that even if an adversary has $l+c$ shares, then at least $l-c-1$ secrets remain protected. In the same spirit as in the definition of PoSS, the notion of ePoSS specifies also which secrets remain protected depending on the indexes of the dealer (the second input of the sharing algorithm). We show that the construction provided in the previous section already satisfies this additional security property. The formal definition follows.

Definition 11 (Enhanced Positional Secret Sharing). *An* Enhanced Positional Secret Sharing *scheme over a message space $\mathcal{M}$ is a PoSS scheme described by the* PPT *algorithms* $(\mathsf{Share}^{\mathsf{ePoSS}}, \mathsf{Reconstruct}^{\mathsf{ePoSS}})$ *which satisfies the following additional property.*
Enhanced positional security. $\forall (x_1,\dots,x_l),(x'_1,\dots,x'_l) \in \mathcal{M}^l$, $\forall S = \{j_1,\dots,j_{l+c}\} \subseteq \{1,\dots,n\}$ *with* $j_1 < j_2 < \dots < j_{l-1} < j_l < \dots < j_{l+c}$*:*

1. *If there exists* $\alpha \in \{1,\dots,l+c\}$ *such that* $j_\alpha = j$*, and* $c \le l$ *then*
   1.1. *If* $\alpha \le l$ *then the following distributions are identical (where* $\gamma = \min\{c, \alpha-1\}$*):*
   $$\{(s_i)_{i\in S} : (s_1,\dots,s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{ePoSS}}((x_1,\dots,x_{\alpha-\gamma-1},x_{\alpha-\gamma},\dots,x_{\alpha-1},x_\alpha,\dots,x_l),j)\}$$
   $$\{(s_i)_{i\in S} : (s_1,\dots,s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{ePoSS}}((x'_1,\dots,x'_{\alpha-\gamma-1},x_{\alpha-\gamma},\dots,x_\alpha,x'_{\alpha+1},\dots,x'_l),j)\}.$$
   1.2. *If* $\alpha > l$ *the following distributions are identical:*
   $$\{(s_i)_{i\in S} : (s_1,\dots,s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{ePoSS}}((x_1,\dots,x_{\alpha-c-1},x_{\alpha-c},\dots,x_{l-1},x_l),j)\}$$
   $$\{(s_i)_{i\in S} : (s_1,\dots,s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{ePoSS}}((x'_1,\dots,x'_{\alpha-c-1},x_{\alpha-c},\dots,x_{l-1},x_l),j)\}$$
2. *if* $\nexists \alpha \in \{1,\dots,l+c\}$ *such that* $j_\alpha = j$*, the following distributions are identical:*
   $$\{(s_i)_{i\in S} : (s_1,\dots,s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{ePoSS}}((x_1,\dots,x_l),j)\}$$
   $$\{(s'_i)_{i\in S} : (s'_1,\dots,s'_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{ePoSS}}((x'_1,\dots,x'_l),j)\}$$

It is easy to see that for $c = 0$ the properties of enhanced positional and positional security are equivalent and that for $c \ge l-1$ none of the secrets is protected. We refer to App. A.2 for the formal proof of the following theorem.

Theorem 2. $(\mathsf{Share}^{\mathsf{PoSS}\star}, \mathsf{Reconstruct}^{\mathsf{PoSS}\star})$ *is an Enhanced Positional Secret Sharing scheme*

6 Threshold NIMPC

In this section we show how to construct a t-secure NIMPC $\mathsf{NIMPC}^{\mathsf{th}} := (\mathsf{Setup}^{\mathsf{th}}, \mathsf{Msg}^{\mathsf{th}}, \mathsf{Eval}^{\mathsf{th}})$. That is, a threshold NIMPC protocol for n parties, with threshold l that supports up to t corruptions. For our construction we make use of the following tools.

- A t-robust NIMPC protocol $\mathsf{NIMPC} := (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Eval})$.
- A PoSS scheme $\mathsf{PSS} := (\mathsf{Share}^{\mathsf{PoSS}}, \mathsf{Reconstruct}^{\mathsf{PoSS}})$.

At a high level our protocol $\mathsf{NIMPC}^{\mathsf{th}}$ works as follows.

Setup: The algorithm $\mathsf{Setup}^{\mathsf{th}}$ runs the setup algorithm of the t-robust NIMPC protocol on input the unary representation of l (the number of parties that will participate in the computation) and the unary representation of the security parameter λ thus obtaining $\tilde{\rho}_0, \dots, \tilde{\rho}_l$. Then, for each $i \in \{1, \dots, l\}$, $\mathsf{Setup}^{\mathsf{th}}$ computes an encoding of the input 0 and of the input 1 using NIMPC: $\tilde{m}_i^0 \leftarrow \mathsf{Msg}(\tilde{\rho}_i, 0)$, $\tilde{m}_i^1 \leftarrow \mathsf{Msg}(\tilde{\rho}_i, 1)$. As a final step, for all $i \in \{1, \dots, l\}$, $\mathsf{Setup}^{\mathsf{th}}$ computes a positional secret sharing of the messages $(\tilde{m}_1^0, \dots, \tilde{m}_k^0)$ using index i thus obtaining $(s_{i,1}^0, \dots, s_{i,n}^0)$, and a positional secret sharing of the messages $(\tilde{m}_1^1, \dots, \tilde{m}_k^1)$, always for the index i, obtaining $(s_{i,1}^1, \dots, s_{i,n}^1)$. The output of $\mathsf{Setup}^{\mathsf{th}}$ corresponds to $(\tilde{\rho}_0, \rho_1, \dots, \rho_n)$ where $\rho_i := (s_{j,i}^0, s_{j,i}^1)_{j \in \{1,\dots,n\}}$ for all $i \in \{1, \dots, n\}$.

Online messages. The party p_i with input $\rho_i := (s_{j,i}^0, s_{j,i}^1)_{j \in \{1,\dots,n\}}$ and the input $x_i \in \{0,1\}$ sends $m_i := (s_{1,i}^0, s_{1,i}^1), \dots, s_{i,i}^{x_i}, \dots, (s_{n,i}^0, s_{n,i}^1)$

Evaluation. The evaluator p_0, on input $\tilde{\rho}_0, m_{j_1}, \dots, m_{j_l}$ with $0 \le j_1 < \dots < j_l \le n$, performs the following steps. For all $i \in \{1, \dots, l\}$, let $b_i \in \{0,1\}$ be such that $\tilde{m}_i \overset{\$}{\leftarrow} \mathsf{Reconstruct}^{\mathsf{PoSS}}(s_{j_i,j_1}^{b_i}, \dots, s_{j_i,j_i}^{b_i}, \dots, s_{j_i,j_l}^{b_i}, j_i)$ and $\tilde{m}_i \neq \perp$.[12] Then p_0 computes and outputs $\mathsf{Eval}(\tilde{\rho}_0, \tilde{m}_1, \dots, \tilde{m}_l)$.

It is easy to see that in the above construction a malicious evaluator can learn the input of the honest party p_i by only inspecting the bit b_i. To avoid this trivial attack we just need to permute the shares sent by the parties to the evaluator. We decided to not include this additional step into the informal description of the protocol to make it easier to read. We show how the complete scheme works in the formal description of the protocol proposed Fig. 2. Intuitively, the scheme is secure because of the following reasons:

1. The *standard security* property of the PoSS scheme exposes only one between $\mathsf{Msg}(\tilde{\rho}_j, 0)$ and $\mathsf{Msg}(\tilde{\rho}_j, 1)$ for all $j \in [l]$ when $i_j \in [n]$ is the index of an honest party p_{i_j}. Indeed, an honest party p_{i_j} will not send the share $s_{i_j,i_j}^{1-x_i}$ where x_{i_j} denotes the input bit of p_{i_j}. Hence, there would not be enough shares to reconstruct $\mathsf{Msg}(\tilde{\rho}_i, 1 - x_{i_j})$.
2. The *positional security* guarantees that the adversary, with respect to a corrupted party p_{i_k}, can obtain only the two messages $\mathsf{Msg}(\tilde{\rho}_k, 0)$ and $\mathsf{Msg}(\tilde{\rho}_k, 1)$ (where $i_k \in [n]$ and $k \in [l]$).
3. The security of the t-robust NIMPC guarantees that even if for the corrupted parties $p_{c_1}, \dots, p_{c_t}$ the adversary obtains $\mathsf{Msg}(\tilde{\rho}_i, 0)$ and $\mathsf{Msg}(\tilde{\rho}_i, 1)$ for each $i \in [t]$ this does not represent a problem.

Theorem 3. *If* NIMPC *is a t-robust NIMPC protocol, then* $\mathsf{NIMPC}^{\mathsf{th}}$ *is a t-secure Threshold NIMPC protocol.*

We refer to App. A.3 for the proof of the theorem.

[12] In this informal description of the protocol we assume that the algorithm $\mathsf{Reconstruct}^{\mathsf{PoSS}}$ outputs $\perp$ in the case that some of the input shares are ill formed (e.g., the input shares are the combination of different execution of the algorithm $\mathsf{Share}^{\mathsf{PoSS}}$).

```
Setup
  1. Run $\mathsf{Setup}(1^l, 1^\lambda, f)$ obtaining $\tilde{\rho}_0, \ldots, \tilde{\rho}_l$.
  2. For $i = 1, \ldots, l$ compute $\tilde{m}_i^0 \leftarrow \mathsf{Msg}(\tilde{\rho}_i, 0)$, $\tilde{m}_i^1 \leftarrow \mathsf{Msg}(\tilde{\rho}_i, 1)$
  3. For $i = 1, \ldots, n$ pick the permutation bit $b_i \xleftarrow{\$} \{0, 1\}$, run
     1. $\mathsf{PSS}(\tilde{m}_1^0, \ldots, \tilde{m}_l^0, i)$ thus obtaining $(s_{i,1}^{b_i}, \ldots, s_{i,n}^{b_i})$ and run
     2. $\mathsf{PSS}(\tilde{m}_1^1, \ldots, \tilde{m}_l^1, i)$ obtaining $(s_{i,1}^{1-b_i}, \ldots, s_{i,n}^{1-b_i})$.
  4. Output $(\rho_0, \rho_1, \ldots, \rho_n)$ where $\rho_0 := \tilde{\rho}_0$ and for $i = 1, \ldots, n$, $\rho_i := (b_i, (s_{j,i}^0, s_{j,i}^1)_{j \in \{1,\ldots,n\}})$.
Online messages
  1. On input $x_i \in \{0, 1\}$ and $\rho_i$ the party $p_i$ does the following.
     1.1 If $b_i = 0$ then set $s_{i,i} \leftarrow s_{i,i}^{x_i}$ and $d_i \leftarrow x_i$ else set $s_{i,i} \leftarrow s_{i,i}^{1-x_i}$ and $d_i \leftarrow 1 - x_i$.
     1.2 Sends $m_i := ((s_{1,i}^0, s_{1,i}^1), \ldots, s_{i,i}, \ldots, (s_{n,i}^0, s_{n,i}^1), d_i)$.
Evaluation
  1. On input $\rho_0, m_{j_1}, \ldots, m_{j_l}$ with $0 \leq j_1 < \cdots < j_l \leq n$, for $i = 1, \ldots, l$ compute $\tilde{m}_i \leftarrow$
     $\mathsf{Reconstruct}^{\mathsf{PoSS}}(s_{j_i,j_1}^{d_{j_i}}, \ldots, s_{j_i,j_i}, \ldots, s_{j_i,j_l}^{d_{j_i}}, j_i)$
  2. Compute and output $\mathsf{Eval}(\rho_0, \tilde{m}_1, \ldots, \tilde{m}_l)$.
```

Fig. 2: Our t-secure NIMPC

7 Ad Hoc PSM

We start by showing how to construct an $(l, l+c)$-secure ad hoc PSM protocol, for an arbitrary non-negative integer c, for a very simple functionality that we call *message selector* and denote with $f^{\mathsf{msg\_sel}}$. $f^{\mathsf{msg\_sel}}$ takes l inputs, and each input $i \in [l]$ consists of 1) a list of size l of λ-bit strings and 2) and integer i_o with $i_o \in [n]$ (this will represent the index of the party that is contributing to the input). The output of $f^{\mathsf{msg\_sel}}$ corresponds to the concatenation of l messages, where the message in position j corresponds to the $j-$th message in the input list of the party with the j-th greatest index that is participating in the online phase. We propose a formal description of the function in Fig. 3. We denote our protocol with $\Pi^{\mathsf{msg\_sel}} := (\mathsf{Setup}^{\mathsf{msg\_sel}}, \mathsf{Msg}^{\mathsf{msg\_sel}}, \mathsf{Eval}^{\mathsf{msg\_sel}})$ and provide an informal description of it for the simplified case in which the input of each party is a list of bits (instead of list of λ-bit strings). In the formal description we consider the generic case where the input of each party is a list of λ-bit strings. At a very high level, the protocol $\Pi^{\mathsf{msg\_sel}}$ works as follows.

Setup: For each party indexed by $i \in \{1, \ldots, n\}$, $\mathsf{Setup}^{\mathsf{msg\_sel}}$ generates l random bits $b_1, \ldots, b_l$ that we call *permutation bits*. Then $\mathsf{Setup}^{\mathsf{msg\_sel}}$ computes an enhanced PoSS of $(b_1, \ldots, b_l)$ for the index i, and an enhanced PoSS of $(1 - b_1, \ldots, 1 - b_l)$ for the index i thus obtaining $(s_{i,1}^0, \ldots, s_{i,n}^0)$ and $(s_{i,1}^1, \ldots, s_{i,n}^1)$ respectively. Intuitively, the party i will receive as a part of ρ_i the permutation bits, and depending on his inputs he will send the corresponding permutation bits. For example, if the first input in the list of p_i is 0 then p_i: 1) takes the permutation bit b_1 (if the input of p_i is 1 then p_1 picks as the permutation bit $1 - b_i$) 2) and sends the permutation bit together with other pieces of information (more details will follow). The output of $\mathsf{Setup}^{\mathsf{msg\_sel}}$ corresponds to $(\rho_0, \rho_1, \ldots, \rho_n)$ where $\rho_i := (s_{j,i}^0, s_{j,i}^1, b_j)_{j \in \{1,\ldots,n\}}$ for all $i \in \{1, \ldots, n\}$ and $\rho_0 := \perp$.

Online messages. The party p_i on input $\rho_i := (s_{j,i}^0, s_{j,i}^1, b_j)_{j \in \{1,\ldots,n\}}$ and the input bits $x_1, \ldots x_l$ computes $d_1 \leftarrow b_1$ if $x_1 = b_1$ and $d_1 \leftarrow 1 - b_1$ otherwise. Repeat the same for $x_2 \ldots x_l$ and sends $m_i := \big((s_{1,i}^0, s_{1,i}^1), \ldots, (s_{n,i}^0, s_{n,i}^1), (d_1, \ldots, d_l)\big)$.

Evaluation. The evaluator p_0, on input $\tilde{\rho}_0, m_{j_1}, \ldots, m_{j_l}$ with $0 \leq j_1 < \cdots < j_l \leq n$, does the following steps. For all $i \in \{1, \ldots, l\}$ compute $y_i^0 \leftarrow \mathsf{Reconstruct}^{\mathsf{PoSS}}(s^0_{j_i,j_1}, \ldots, s^0_{j_i,j_l}, j_i)$, $y_i^1 \leftarrow \mathsf{Reconstruct}^{\mathsf{PoSS}}(s^0_{j_i,j_1}, \ldots, s^0_{j_i,j_l}, j_i)$ and $\tilde{x}_i \leftarrow y_i^{d_{j_i}}$. The output of the evaluator then corresponds to $(\tilde{x}_1, \ldots, \tilde{x}_l)$.

The security of our protocol relies on the security of the enhanced PoSS scheme. Informally, let $X := ((x_{i_1}, i_1), \ldots, (x_{i_N}, i_N))$ with $N \leq l + c$ be the inputs of the parties participating in the protocol (recall that each input represents a list of l bits). The notion of ad hoc PSM guarantees that a malicious evaluator can learn only the output of $f^{\mathsf{msg\_sel}}$ on input any possible set S where $S := ((x_{j_1}, j_1), \ldots, (x_{j_l}, j_l)) \subseteq X$. Hence, the adversary can evaluate $f^{\mathsf{msg\_sel}}$ on up to $\binom{l+c}{l}$ possible sets of inputs. Consider now the input of the party p_{i_α} be x_{i_α} and let $c < l$, then we have the two possible cases (when $c \geq l$ then the evaluator can obtain all the inputs).

- If $\alpha \leq l$ then x_{i_α} can be placed in the α-th input slot of $f^{\mathsf{msg\_sel}}$, or in any other position $i_{\alpha-1}, \ldots, i_{\alpha-\gamma}$ with $\gamma = \min\{c, \alpha - 1\}$.
- If $\alpha > l$ then x_{i_α} can be place in l-th input slot of $f^{\mathsf{msg\_sel}}$, or in any other position $i_{l-1}, \ldots, i_{\alpha-c}$ given that $N = l + c$.

Any other value in the input list x_{i_α} of p_{i_α} has to be protected. We note that this is exactly the security that an ePoSS scheme can guarantee.

Input: $((x_k^{i_1})_{k\in[l]}, i_1), \ldots ((x_k^{i_l})_{k\in[l]}, i_l)$ where $\{i_1, \ldots, i_l\} \subseteq [n]$, $x_k^{i_1}, \ldots, x_k^{i_l} \in \{0,1\}^\lambda$, $l \leq n$ and $n, \lambda \in \mathbb{N}$.
Output: Let $(j_1, \ldots, j_l)$ be a permutation of the values $(i_1, \ldots, i_l)$ such that $0 \leq j_1 < j_2 < \cdots < j_{l-1} < j_l \leq n$, output $x_1^{j_1} || \ldots || x_l^{j_l}$

Fig. 3: $f^{\mathsf{msg\_sel}}$

Theorem 4. *$\Pi^{\mathsf{msg\_sel}}$ is a $(l, l + c)$-secure ad hoc PSM protocol.*

We refer the reader to App. A.4 for the formal proof.

7.1 Ad Hoc PSM for all Functions

In this section we show how to construct a $(l, l + c)$-secure ad hoc PSM for any function f and any constant c, which has a simulator that is successful with probability at least $p = e^{-1}$ (where e is the Euler number). We denote this scheme with $\Pi^{\mathsf{PSM}} := (\mathsf{Setup}^{\mathsf{PSM}}, \mathsf{Msg}^{\mathsf{PSM}}, \mathsf{Eval}^{\mathsf{PSM}})$ and to construct it we make use of the following tools.

- An $(l, l + c)$-secure ad hoc PSM $\Pi^{\mathsf{msg\_sel}} := (\mathsf{Setup}^{\mathsf{msg\_sel}}, \mathsf{Msg}^{\mathsf{msg\_sel}}, \mathsf{Eval}^{\mathsf{msg\_sel}})$ for the message selector function described in the previous section.
- A hash function H with range size $\lambda' = \lambda^{2c+2}$ (this function is defined as the hash function that on input x outputs $x \mod \lambda'$).
- A 2-party 0-robust NIMPC scheme $\Pi^{\mathsf{2PC}} := (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Eval})$ for the funciton g_k (which will be specified later) with the following additional properties:

```
Common input: Input length: $\lambda$, number of parties $n$, threshold $l$ and $c$.
Setup:
 1. For $i = 1, \ldots, n$
    1.1. For each $k = 1, \ldots l$, For each $j = 1, \ldots, \lambda$ Pick $b^k_j \xleftarrow{\$} \{0, 1\}$.
    1.2. Run $\mathsf{PSS}(b^1_1 || \ldots || b^1_\lambda, b^2_1 || \ldots || b^2_\lambda, \ldots, b^l_1 || \ldots || b^l_\lambda, i)$ thus obtaining $(s^0_{i,1}, \ldots, s^0_{i,n})$.
    1.3. Run $\mathsf{PSS}(1 - b^1_1 || \ldots || 1 - b^1_\lambda, 1 - b^2_1 || \ldots || 1 - b^2_\lambda, \ldots, 1 - b^l_1 || \ldots || 1 - b^l_\lambda, i)$ thus obtaining
         $(s^1_{i,1}, \ldots, s^1_{i,n})$.
    1.4. Set $B_i = (b^k_1, \ldots, b^k_\lambda)_{k \in [l]}$.
 2. Output $(\rho_0, \rho_1, \ldots, \rho_n)$ where $\rho_0 := \perp$ and for $i = 1, \ldots, n$, $\rho_i := (B_i, (s^0_{j,i}, s^1_{j,i})_{j \in \{1,\ldots,n\}})$.
Online messages
 1. On input $x^i_1, \ldots, x^i_l \in \{0, 1\}^\lambda$ and $\rho_i$ the party $p_i$ acts as follows.
    1.1. For each $k \in [l]$ parse $x^i_k$ as a $\lambda$ bit string $x_{k,1}, \ldots, x_{k,\lambda}$.
    1.2. For each $k \in [l]$, $j \in [\lambda]$ if $x_{k,j} = b^k_j$ then set $d^k_j = b^k_j$ else set $d^k_j = 1 - b^k_j$.
    1.3. Set $D_i \leftarrow (d^k_1, \ldots, d^k_\lambda)_{k \in [l]}$.
    1.4. Send $m_i := (D_i, (s^0_{1,i}, s^1_{1,i}), \ldots, (s^0_{n,i}, s^1_{n,i}))$.
Evaluation
 1. On input $\rho_0, m_{k_1}, \ldots, m_{k_l}$ with $0 \leq k_1 < \cdots < k_l \leq n$, for $i = 1, \ldots, l$ do the following
    1.1. Compute $y_{1,0} || \ldots || y_{\lambda,0} \leftarrow \mathsf{Reconstruct}^{\mathsf{PoSS}}(s^0_{k_i,k_1}, \ldots, s^0_{k_i,k_l}, k_i)$,
    1.2. Compute $y_{1,1} || \ldots || y_{\lambda,1} \leftarrow \mathsf{Reconstruct}^{\mathsf{PoSS}}(s^1_{k_i,k_1}, \ldots, s^1_{k_i,k_l}, k_i)$
    1.3. For $j = 1, \ldots, \lambda$
          i. $c \leftarrow d^i_j$
         ii. $x_{i,j} \leftarrow y_{j,c}$
 2. Compute and output $x_{1,1} || \ldots || x_{1,\lambda}, \ldots, x_{l,1} || \ldots || x_{l,\lambda}$.
```

Fig. 4: Our $(l, l + c)$-secure ad hoc PSM for the message selector function $f^{\mathsf{msg\_sel}}$.

  1. It accepts inputs of size $\delta = 2\lambda n + n\lambda\lambda'$, where n represents the number of parties and λ is the input size allowed by Π^{PSM} (it also represents the security parameter);[13] and λ' is the range size of H.
  2. The size of the output of Msg depends only on $\mathsf{poly}(\lambda, \delta)$ and it is independent from the function that Π^{2PC} is computing (whereas the output of Setup can grow with the size of the function being computed;
  3. The randomness required to run Setup is $\kappa := \mathsf{poly}(\lambda)$.

- A PRG $\mathsf{PRG} : \{0,1\}^\lambda \rightarrow \{0,1\}^\kappa$ where $\kappa := \mathsf{poly}(\lambda)$ represents the size of the randomness required to run Setup.

We start by giving a high level idea of how our construction works starting from a scheme that does not provide security but contains most of intuitions. Then we gradually modify it until we get our final scheme.

First attempt. Let ρ be the output of the setup phase of $\Pi^{\mathsf{msg\_sel}}$ and consider $(l-1)$ instantiations of Π^{2PC} which we denote with $\Pi_2^{\mathsf{2PC}}, \dots, \Pi_l^{\mathsf{2PC}}$. We denote with R_i, ρ_i^0, ρ_i^1 the output of the setup phase of Π_i^{2PC} for each $i \in \{2, \dots, l\}$.

For each $i \in \{2, \dots, l-1\}$, an instantiation Π_i^{2PC} will be used to evaluate the function g_i. The function g_i takes two inputs $x^0 \in \{0,1\}^\lambda, x^1 \in \{0,1\}^\lambda$ and outputs $\mathsf{Msg}(\rho_{i+1}^0, x^0||x^1)$. That is, g_i outputs an encoding of the message $x^0||x^1$ for Π_{i+1}^{2PC}. The instantiation Π_l^{2PC} is used to evaluate the function g_l, which takes as input $x_1||x_2||\dots||x_{l-1}$ and x_l and outputs $f(x_1, x_2, \dots, x_{l-1}, x_l)$.

Each party p_i on input $x \in \{0,1\}^\lambda$, ρ, $\rho_2^1, \dots \rho_l^1$ and ρ_2^0 does the following.

```
1. Encode the input $x$ for $\Pi_2^{\mathsf{2PC}}$ by running $\mathsf{Msg}(\rho_2^0, x)$ thus obtaining $m_1^0$.
2. For each $j \in \{2, \dots, l\}$
  2.1. Encode the input $x$ for $\Pi_j^{\mathsf{2PC}}$ by running $\mathsf{Msg}(\rho_j^1, x)$ thus obtaining $m_j^1$
3. Run $\mathsf{Msg}^{\mathsf{msg\_sel}}(\rho, m_2^0||m_2^1||m_3^1||m_4^1||\dots||m_l^1)$ thus obtaining $\tilde{m}_i$
4. Output $m_i$.
```

The evaluation algorithm works as follows

```
1. Run $\mathsf{Eval}^{\mathsf{msg\_sel}}$ on input $(\tilde{m}_{k_1}, \dots, \tilde{m}_{k_l})$ thus obtaining $m_1^0, m_2^1, \dots, m_l^1$ (we denote with $k_1, \dots, k_l$
   the indexes of the parties that are participating in the online phase).
2. Run $\mathsf{Eval}(R_2, m_1^0, m_2^1)$ thus obtaining $m_3^0$.
3. For each $j \in \{3, \dots, l-1\}$
  3.1. Run $\mathsf{Eval}(R_j, m_j^0, m_j^1)$ thus obtaining $m_{j+1}^0$.
4. Output $\mathsf{Eval}(R_l, m_l^0, m_l^1)$
```

Despite being correct, the above protocol suffers of a security issue. If more than l parties participate to the protocol, then a corrupted evaluator could be able to obtain the encoding of two different messages with respect to the same ρ_j^1 for some $j \in \{2, \dots, l\}$, and this could harm the security of Π_j^{2PC}.

Second attempt. To solve this problem we give a different ρ_j^1 to each party. In this way, even if two different parties encode different messages we can still rely on the security of Π^{2PC}. This approach requires a more sophisticated function g_j, since now the output of g_j should contain an encoding of the previous inputs under Π^{2PC} which can be combined the with the next party's encoded message, whoever she is. Hence, we modify g_j (for any j) to output multiple encodings, one for each party with index greater than j. Even if this approach never causes the same ρ_j^1 to

[13] Our construction would work for inputs of size $\mathsf{poly}(\lambda)$, but to not overburden the notation we consider only inputs of size λ only.

be used twice on different inputs, now multiple encodings of different inputs under ρ_j^0 might be computed by a malicious evaluator. For example, an evaluator could construct the first input for g_j using two different sequences on inputs (this is possible only if the evaluator has access to more than l messages sent from the honest parties).

Our approach. To mitigate (but not completely solve) the above problem, we modify the above protocol as follows.

1. From the setup phase each party p_i receives $\rho_{j,i}^{\mathsf{sel},0}$ for each $\mathsf{sel} \in [\lambda']$ and each $j \in [l]$ (note that we need to run the setup of Π^{2PC} λ' times more in this protocol).
2. Each party p_i picks a random value v_i, and encodes this value together with its input by running $\mathsf{Msg}(\rho_{j,i}^{\mathsf{sel},0}, x_i||v_i)$ for each $\mathsf{sel} \in \lambda'$ and $j \in \{2, \dots, l\}$.
3. The function g_j now takes as input $v^0||x^0$ and $v^1||x^1$, computes $\mathsf{sel}' \leftarrow \mathsf{H}(v^0 \oplus v^1)$ and outputs $\mathsf{Msg}(\rho_{j+1,i}^{\mathsf{sel}',0}, x^0||x^1||v^0 \oplus v^1)$ for each i where H is an hash function with range size λ'.

This protocol remains secure as long the adversary is not able to find a combination of the messages received from the honest parties that yields to a collision in the hash function. This means that an execution of this protocol could be insecure with probability $1/\mathsf{poly}(\lambda)$ given that λ' is a polynomial. On the other hand, we can prove that a run of the protocol is secure with probability e^{-1}. Intuitively, this holds because each hash function can be evaluated at most on $\binom{l+c}{l}$ different random values. Give that c is a constant value we obtain that the number of possible inputs of H is at most n^c. Hence, for a suitable choice of λ' we can show that our protocol is secure with probability e^{-1}. In the next section we show how to amplify the security of this protocol to obtain a secure ad hoc PSM . Note that to amplify the security of the protocol we cannot just simply take a hash function with an exponentially large range. We propose a formal description of the function g_k in Fig. 5 and a formal description of our protocol Π^{PSM} in Fig. 6. For the formal proof of our scheme we refer to App. A.5.

```
$g_k(x||v_1, j||y||v_2||\{\mathsf{r}_{k+1,i>j}^{\mathsf{sel}}\}_{j\in[n],\mathsf{sel}\in[\lambda']})$
    $v \leftarrow v_1 \oplus v_2$, $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
    For each $i \in \{j+1, \dots, n\}$ compute
        $r \leftarrow \mathsf{PRG}(\mathsf{r}_{k+1,i}^{\mathsf{sel}})$, $(R_{k+1,i}^{\mathsf{sel}'}, \rho_{k+1,i}^{\mathsf{sel}',0}, \rho_{k+1,i}^{\mathsf{sel}',1}) \leftarrow \mathsf{Setup}(1^n, 1^\lambda, g_{k+1}; r)$.
        $\mu_{k+1,i}^{\mathsf{sel}',0} \leftarrow \mathsf{Msg}(\rho_{k+1,i}^{\mathsf{sel}',0}, x||y||v)$.
    Return $\{\mu_{k+1,i}^{\mathsf{sel}',0}\}_{i\in\{j+1,\dots,n\}}$
$g_l(x, y)$
    Parse $x$ as $l$ bit-strings of $\lambda$ bits $x_1, \dots, x_{l-1}$.
    Compute and output $f(x_1, \dots, x_{l-1}, y)$.
```

Fig. 5: Specification of the functions g_k with $k \in \{2, \dots, l-1\}$ and g_l.

Theorem 5. *There exists a simulator that successfully satisfies the definition of $(l, l+c)$-secure ad hoc PSM with probability at least e^{-1}, for any constant c.*

How to instantiate the 2-party 0-robust NIMPC scheme Π^{2PC}. Our compiler requires non-standard requirement on the size of the messages of the protocol Π^{2PC}. As also noted in [BKR17], 0-robust

```
Common parameters: Security parameter $\lambda$, range-size of $\mathsf{H}$ $\lambda' = \lambda^{2c+1}$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.
Setup:
 1. For each $i, j \in [n]$ with $i \neq j$ do the following.
  - Run $\mathsf{Setup}(1^2, g_2, 1^\lambda)$ thus obtaining $(R^j_{2,i}, \rho^{j,0}_{2,i}, \rho^{j,1}_{2,i})$.
 2. For each $k \in \{3, \dots, l-1\}$, $i \in [n]$, $\mathsf{sel} \in [\lambda']$ do the following.
  - Pick $\mathsf{r}^{\mathsf{sel}}_{k,i} \overset{\$}{\leftarrow} \{0,1\}^\lambda$.
  - Compute $\mathsf{PRG}(\mathsf{r}^{\mathsf{sel}}_{k,i})$ thus obtaining $r$.
  - Run $\mathsf{Setup}(1^2, g_k, 1^\lambda; r)$ thus obtaining $(R^{\mathsf{sel}}_{k,i}, \rho^{\mathsf{sel},0}_{k,i}, \rho^{\mathsf{sel},1}_{k,i})$.
 3. For each $\mathsf{sel} \in [\lambda']$ $i \in [n]$ run $\mathsf{Setup}(1^2, g_l, 1^\lambda)$ thus obtaining $(R^{\mathsf{sel}}_{l,i}, \rho^{\mathsf{sel},0}_{l,i}, \rho^{\mathsf{sel},1}_{l,i})$
 4. Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^\lambda, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \dots, \rho^{\mathsf{th}}_n)$.
 5. For $i \leftarrow 1, \dots, n$ pick $v_i \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and set
    $\rho_i := (v_i, (\mathsf{r}^{\mathsf{sel}}_{k,j>i})_{j \in [n] \mathsf{sel} \in [\lambda'], k \in \{3,\dots,l\}}, (\rho^{\mathsf{sel},1}_{k,i})_{\mathsf{sel} \in [\lambda'], k \in \{3,\dots,l\}}, (\rho^{i,0}_{2,j}, \rho^{j,1}_{2,i})_{j \in [n]-\{i\}}, \rho^{\mathsf{th}}_i)$ and
    $\rho_0 := \rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel} \in [\lambda'], i \in [n], k \in [l]}$
Online messages
 1. On input $x_i \in \{0,1\}^\lambda$ and $\rho_i$ the party $p_i$ does the following.
  - For each $j \in [n] - \{i\}$ compute $m^{i,0}_{1,j} \leftarrow \mathsf{Msg}(\rho^{i,0}_{2,j}, (x_i, v_i))$.
  - For each $j \in [n] - \{i\}$ compute $m^{j,1}_{2,i} \leftarrow \mathsf{Msg}(\rho^{j,1}_{2,i}, i||x_i||v_i||\{\mathsf{r}^{\mathsf{sel},0}_{3,c>i}\}_{c \in [n]})$.
  - For each $k \in \{3, \dots, l-2\}, \mathsf{sel} \in [\lambda']$ compute
          $m^{\mathsf{sel},1}_{k,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{k,i}, i||x_i||v_i||\{\mathsf{r}^{\mathsf{sel},0}_{k+1,j>i}\}_{j \in [n], \mathsf{sel} \in [\lambda']})$
  - For each $\mathsf{sel} \in [\lambda']$ compute $m^{\mathsf{sel},1}_{l,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{l,i}, x_i)$
  - Compute and send
          $m_i \leftarrow \mathsf{Msg}^{\mathsf{msg\_sel}}(\rho^{\mathsf{th}}_i, (\{m^{i,0}_{1,j}\}_{j \in [n]-\{i\}}, \{m^{j,1}_{2,i}\}_{j \in [n]-\{i\}}, \dots, \{m^{\mathsf{sel},1}_{l,i}\}_{\mathsf{sel} \in [\lambda']}, i))$
Evaluation
 1. On input $\rho_0, m_{k_1}, \dots, m_{k_l}$ with $0 \leq k_1 < \cdots < k_l \leq n$ the evaluator does the following.
  - Run $\mathsf{Eval}(\rho^{\mathsf{th}}_0, m_{k_1}, \dots, m_{k_l})$ thus obtaining $\{m^{k_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel} \in [n]-\{k_1\}}, \{m^{\mathsf{sel},1}_{2,k_2}\}_{\mathsf{sel} \in [n]-\{k_2\}}, \dots,$
      $\{m^{\mathsf{sel},1}_{l-1,k_{l-1}}\}_{\mathsf{sel} \in [\lambda']}, \{m^{\mathsf{sel},1}_{l,k_l}\}_{\mathsf{sel} \in [\lambda']}$.
  - Run $\mathsf{Eval}(R^{k_1}_{2,k_2}, m^{k_1,0}_{1,k_2}, m^{k_1,1}_{2,k_2})$ thus obtaining $\{\mu^{\mathsf{sel}',0}_{3,i}\}_{i \in [n]}$.
  - For $j \leftarrow 3, \dots, l-1$
     - Run $\mathsf{Eval}(R^{\mathsf{sel}'}_{j,k_j}, \mu^{\mathsf{sel}',0}_{j,k_j}, m^{\mathsf{sel}',1}_{j,k_j})$ thus obtaining $\{\mu^{\mathsf{sel}'',0}_{j+1,i}\}_{i \in [n]}$.
     - Set $\mathsf{sel}' \leftarrow \mathsf{sel}''$.
  - Compute $y \leftarrow \mathsf{Eval}(R^{\mathsf{sel}'}_{l,k_l}, \mu^{\mathsf{sel}',0}_{l,k_l}, m^{\mathsf{sel}',1}_{l,k_l})$ and output $y$.
```

Fig. 6: Our ad hoc PSM for all functions that is secure with probability e^{-1}.

NIMPC protocol can be constructed from garbled circuits. And this construction would have all the properties that we need. At a high level the construction works as follows. Let g be a two-input function where each input is of size M. In the setup phase a garbled circuit $\tilde{C}$ for the function g and the corresponding wire keys $L_{0,1}, L_{1,1}, \dots L_{0,M}, L_{1,M}, R_{0,1}, R_{1,1}, \dots R_{0,M}, R_{1,M}$ are computed. Then $\rho = \tilde{C}$ is given to the evaluator, the keys $\rho_0 = L_{0,1}, L_{1,1}, \dots L_{0,M}, L_{1,M}$ are given to to the party p_0 and the keys $\rho_1 = R_{0,1}, R_{1,1}, \dots R_{0,M}, R_{1,M}$ are given to the party p_1. For the evaluation, the party p_0 on input $x \in \{0,1\}^M$ parses it as a bit string $x_1, \dots, x_M$ and sends to the evaluator $L_{x_1,1}, \dots L_{x_M,M}$. The party p_1 does the same for its input y but using the keys $\rho_1 = R_{0,1}, R_{1,1}, \dots R_{0,M}, R_{1,M}$. The evaluator then uses the received keys and $\tilde{C}$ to compute $g(x,y)$. This construction is provided in [FKN94], the only difference is that in their protocol the $\tilde{C}$ is sent by one of the parties instead in our case we assume that $\tilde{C}$ is already given to the evaluator from the setup phase. This construction has the property that we need since the size of the keys of the garbled circuit depends only on the security parameter and on the size of the inputs and *does not* depend on the size of the function g [AIK11]. Then we have the following corollary.

Corollary 1. *If one-way functions exists then there exists a simulator that successfully satisfies the definition of $(l, l+c)$-secure ad hoc PSM with probability at least e^{-1}, for any constant c.*

7.2 Fully Secure Ad Hoc PSM

We are now ready to provide a fully-secure ad hoc PSM $\mathit{\Pi}^{\mathsf{APSM}} := (\mathsf{Setup}^{\mathsf{APSM}}, \mathsf{Msg}^{\mathsf{APSM}}, \mathsf{Eval}^{\mathsf{APSM}})$ that realizes any function f. To construct our protocol we use the following tools.

- An $(l, l+c)$-secure ad hoc PSM protocol $\mathit{\Pi}^{\mathsf{PSM}} := (\mathsf{Setup}^{\mathsf{PSM}}, \mathsf{Msg}^{\mathsf{PSM}}, \mathsf{Eval}^{\mathsf{PSM}})$ that supports up to a n parties and that is simulatable with probability $\frac{1}{p}$ with $p \leq e$ (where e is the Euler number).
- An additive $(l, \mathsf{m}, \mathsf{m}-1)$-HSS Scheme for the function f $\mathsf{HSS} := (\mathsf{Share}^{\mathsf{HSS}}, \mathsf{Eval}^{\mathsf{HSS}}, \mathsf{Dec}^{\mathsf{HSS}})$ where $\mathsf{m} := p\lambda$.

At a very high level our protocol consists of m instantiations of the $\mathit{\Pi}^{\mathsf{PSM}}$ where the j-th instantiation evaluates the function G_j with $j \in [\mathsf{m}]$. The Function G_j takes as input l shares of the HSS scheme, and uses them as input of $\mathsf{Eval}^{\mathsf{HSS}}$ together with the *server index* j (see Fig. 7 for a formal specification of G_j). Each party p_i that wants to participate in the protocol computes a secret sharing of his input thus obtaining m shares. Then p_i encodes each share by running $\mathsf{Msg}^{\mathsf{PSM}}$ (one execution of $\mathsf{Msg}^{\mathsf{PSM}}$ per share). The evaluator runs the evaluation algorithm of the j-th instantiation of $\mathit{\Pi}^{\mathsf{PSM}}$ thus obtaining y_j (which corresponds to the output of $\mathsf{Eval}^{\mathsf{HSS}}$) for each $j \in [\mathsf{m}]$. The output of the evaluation phase then corresponds to $y_1 \oplus \cdots \oplus y_{\mathsf{m}}$. We show that this protocol is secure as long as there is at least one execution of $\mathit{\Pi}^{\mathsf{PSM}}$ that simulatable. Moreover, by choosing m opportunely we can prove that at least for one instantiation of $\mathit{\Pi}^{\mathsf{PSM}}$ the simulator is successful with overwhelming probability. Hence, at least one share of each of the inputs of the honest parties will be protected. Therefore, because of the security offered by the HSS, also the entire input of the parties will be protected.

We refer to Fig. 8 for the formal description of $\mathit{\Pi}^{\mathsf{APSM}}$.

Input: $((x^k_{i_1})_{k\in[l]}, i_1), \ldots ((x^k_{i_l})_{k\in[l]}, i_l)$ where $\{i_1, \ldots, i_l\} \subseteq [n]$, $x^k_{i_1}, \ldots, x^k_{i_l} \in \{0,1\}^\lambda$, $l \leq n$ and $n, \lambda \in \mathbb{N}$.
Output: Let $(j_1, \ldots, j_l)$ be a permutation of the values $(i_1, \ldots, i_l)$ such that $0 \leq j_1 < j_2 < \cdots < j_{l-1} < j_l \leq n$, output $\mathsf{Eval}^{\mathsf{HSS}}(j, x^1_{j_1}, \ldots, x^l_{j_l})$

Fig. 7: G_j with $j \in [\mathsf{m}]$.

```
Common parameters: Security parameter $\lambda$, maximum number of parties $n$, threshold $l$, and $c$ where
$l + c$ denotes the maximum number of active parties supported by the protocol and $\mathsf{m} = p\lambda$.
Setup:
  1. For each $j \in \mathsf{m}$
     1.1. Run $\mathsf{Setup}^{\mathsf{PSM}}(1^n, 1^l, 1^\lambda, G_j)$ thus obtaining $\rho_0^j, \rho_1^j, \dots, \rho_n^j$.
  2. Output $\rho_0, \rho_1, \dots, \rho_n$ where $\rho_0 := (\rho_0^j)_{j\in[\mathsf{m}]}, \rho_1 := (\rho_1^j)_{j\in[\mathsf{m}]}, \dots \rho_n := (\rho_n^j)_{j\in[\mathsf{m}]}$
Online messages
  1. On input $x_i \in \{0,1\}^\lambda$ and $\rho_i$ the party $p_i$ does the following.
     1.1. For each $k \in [l]$
          i. Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, x)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
     1.2. For each $j \in \mathsf{m}$
          i. Run $\mathsf{Msg}^{\mathsf{PSM}}(\rho_i^j, ((x_i^{j,k})_{k\in[l]}, i))$ thus obtaining $m_i^j$.
     1.3. Send $m_i := (m_i^j)_{j\in[\mathsf{m}]}$
Evaluation
  1. On input $\rho_0, m_{k_1} := (m_{k_1}^j)_{j\in[\mathsf{m}]}, \dots, m_{k_l} := (m_{k_l}^j)_{j\in[\mathsf{m}]}$ with $0 \leq k_1 < \dots < k_l \leq n$ the evaluator does
     the following.
  2. For each $j \in \mathsf{m}$
     2.1. Run $\mathsf{Eval}^{\mathsf{PSM}}(\rho_0^j, m_{k_1}^j, \dots, m_{k_l}^j)$ thus obtaining $y^j$.
  3. Output $y^1 \oplus \dots \oplus y^{\mathsf{m}}$
```

Fig. 8: Our fully secure ad hoc PSM for all functions

Theorem 6. *Π^{APSM} is a $(l, l+c)$-secure ad hoc PSM protocol for any constant c.*

We refer to App. A.6 for the formal proof. Since Π^{PSM} can be constructed from OWFs and since the HSS scheme that we need can be instantiated from the LWEs assumption [BGI$^+$18,DHRW16] we have the following corollary.

Corollary 2. *Assuming the hardness LWE, then Π^{APSM} is a $(l, l+c)$-secure ad hoc PSM protocol for any constant c.*

8 Our adaptive-ad-hoc PSM protocol

As we have anticipated in the introduction, it is straightforward to construct a (l, t)-secure adaptive-ad-hoc PSM from a (l, t)-secure Ad Hoc PSM protocol. We now provide a forma description of the protocol.

Let $\mathcal{F} = (\mathcal{F}_n)_{n\in\mathbb{N}}$ be an ensemble of sets $\mathcal{F}_n$ of functions $f : \mathcal{X}^n \to \mathcal{Y}$, in this section we want to construct an adaptive-ad-hoc PSM for $\overline{f} := f_l, \dots f_{l+c}$ where $f_\alpha \in \mathcal{F}_\alpha$. It is straightforward to construct an adaptive-ad-hoc PSM having an $(l, l+c)$-secure ad hoc PSMscheme $\Pi^{\mathsf{APSM}} := (\mathsf{Setup}^{\mathsf{APSM}}, \mathsf{Msg}^{\mathsf{APSM}}, \mathsf{Eval}^{\mathsf{APSM}})$. Indeed, we just need to run c instantiation of Π^{APSM}, where each instantiation computes a function f_α with $\alpha \in [l+c]$. For sake of completeness we propose a formal description of the protocol in Fig. 9.

```
**Common parameters:** Security parameter $\lambda$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c$ denotes the maximum number of active parties supported by the protocol.
**Setup:**
 1. For each $\alpha \in \{l, \ldots, l + c\}$
    1.1. Run $\mathsf{Setup}^{\mathsf{APSM}}(1^n, 1^l, 1^c, 1^\lambda, f_\alpha)$ thus obtaining $\rho_0^\alpha, \rho_1^\alpha, \ldots, \rho_n^\alpha$.
 2. Output $\rho_0, \rho_1, \ldots, \rho_n$ where $\rho_0 := (\rho_0^j)_{j \in \{l,\ldots,l+c\}}, \rho_1 := (\rho_1^\alpha)_{\alpha \in \{l,\ldots,l+c\}}, \ldots \rho_n := (\rho_n^\alpha)_{\alpha \in \{l,\ldots,l+c\}}$
**Online messages**
 1. On input $x_i \in \{0, 1\}$ and $\rho_i$ the party $p_i$ does the following.
    1.1. For each $\alpha \in \{l, \ldots, l + c\}$
          i. Run $\mathsf{Msg}^{\mathsf{APSM}}(\rho_i^\alpha, (x_i, i))$ thus obtaining $m_i^\alpha$.
    1.2. Send $m_i := (m_i^\alpha)_{\alpha \in \{l,\ldots,l+c\}}$
**Evaluation**
 1. On input $\rho_0, m_{k_1}, \ldots, m_{k_N}$ with $0 \leq k_1 < \cdots < k_N \leq n$ the evaluator does the following.
    1.1. If $l \leq N \leq l + c$ then run $y \leftarrow \mathsf{Eval}^{\mathsf{APSM}}(\rho_0^N, m_{k_1}, \ldots, m_{k_N})$ and output $y$, output $\perp$ otherwise.
```

Fig. 9: Our adaptive-ad-hoc PSM for all functions

Acknowledgments

Vipul Goyal is 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. Rafail Ostrovsky is supported in part by DARPA under Cooperative Agreement No: HR0011-20-2-0025, NSF Grant CNS-2001096, US-Israel BSF grant 2015782, Google Faculty Award, JP Morgan Faculty Award, IBM Faculty Research Award, Xerox Faculty Research Award, OKAWA Foundation Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, the Department of Defense, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein. Michele Ciampi is supported by H2020 project PRIVILEDGE #780477 and the work is done in part while consulting for Stealth Software Technologies, Inc.

References

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

App17. Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:67, 2017.

BGI$^+$01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

BGI$^+$14. Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 387–404, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

BGI$^+$18. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94, pages 21:1–21:21, Cambridge, MA, USA, January 11–14, 2018. LIPIcs.

BGIK16. Amos Beimel, Ariel Gabizon, Yuval Ishai, and Eyal Kushilevitz. Distribution design. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 81–92, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery.

BIK17. Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. Ad hoc PSM protocols: Secure computation without coordination. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 580–608, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

BKN18. Amos Beimel, Eyal Kushilevitz, and Pnina Nissim. The complexity of multiparty PSM protocols and related models. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 287–318, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

BKR17. Fabrice Benhamouda, Hugo Krawczyk, and Tal Rabin. Robust non-interactive multiparty computation against constant-size collusion. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 391–419, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

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, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

CGOS07. Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multi-party computation. In *48th Annual Symposium on Foundations of Computer Science*, pages 238–248, Providence, RI, USA, October 20–23, 2007. IEEE Computer Society Press.

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

FKN94. Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th Annual ACM Symposium on Theory of Computing*, pages 554–563, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.

GKP$^+$13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

HIJ$^+$16. Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 157–168. ACM, 2016.

HIJ$^+$17. Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 181–211, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

HJO^{+}16. Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 149–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

HLP11. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

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, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

JSW17. Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 40–71, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

JW16. Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 433–458, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

Kol05. Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 136–155, Chennai, India, December 4–8, 2005. Springer, Heidelberg, Germany.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

KW93. Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of Structures in Complexity Theory*, pages 102–111, 1993.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Security Proofs

A.1 Proof of Theorem 1

Proof. Correctness. The correctness of the protocol follows from the correctness of the standard secret sharing schemes used in the protocol.
Standard security. In this case the proof relies on the observation that, to reconstruct a value $x_i = x_i^0 \oplus x_i^1 \oplus \tilde{x}_i$ with $i \in \{1, \dots, l\}$ are required at least $(i-1)$ shares of the $(i-1)$-out-of-$(j-1)$ secret sharing scheme (to compute x_i^0), $l-i$ shares of the $(l-i)$-out-of-$(n-j)$ secret sharing scheme (to compute x_i^1) and the share $\tilde{x}_i$. If only $l' < l$ shares of the PoSS scheme are available then either the number of shares to reconstruct x_i^0 is insufficient, or the number of shares to reconstruct x_i^1 is insufficient or the share $\tilde{x}_i$ is missing. More formally, if by contradiction our scheme is not secure then we can construct a reduction that, for some $i \in \{1, \dots, l\}$, breaks the security of either the $(i-1)$-out-of-$(j-1)$ or the $(l-i)$-out-of-$(n-j)$ secret sharing or the 3-out-of-3 secret sharing scheme used to compute $(x_i^0, x_i^1, \tilde{x}_i)$.
Positional security. We first prove that the first condition of Definition 10 holds and then we show that also the second condition holds. The proof proceeds via hybrid arguments. Let $S = \{j_1, \dots, j_l\} \subseteq \{1, \dots, n\}$ with $0 \leq j_1 < j_2 < \dots < j_{l-1} < j_l \leq n$ with $j_\alpha = j$ be an arbitrarily chosen set.

We denote with $\mathcal{H}^0_\ell$, with $\ell \in \{0, \dots, \alpha-1\}$, the hybrid experiment that acts as follows.

Compute $(s_1, \dots, s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{PoSS}^\star}((x'_1, \dots, x'_\ell, x_{\ell+1}, \dots, x_\alpha, \dots, x_l), j)$.
Output $(s_{j_1}, \dots, s_{j_l})$.
We denote with $\mathcal{H}^1_\ell$, with $\ell \in \{\alpha+1, \dots, l\}$, the hybrid experiment that acts as follows.
Compute $(s_1, \dots, s_n) \xleftarrow{\$} \mathsf{Share}^{\mathsf{PoSS}^\star}((x'_1, \dots, x'_{\alpha-1}, x_\alpha, x'_{\alpha+1}, \dots, x'_\ell, x_{\ell+1}, \dots, x_l), j)$.
Output $(s_{j_1}, \dots, s_{j_l})$.

We now show that $\mathcal{H}^0_0 \equiv \dots \equiv \mathcal{H}^0_{\alpha-1} \equiv \mathcal{H}^1_{\alpha+1} \equiv \dots \equiv \mathcal{H}^1_l$.

Lemma 1. *The output distribution of $\mathcal{H}^0_{i-1}$ is identical to the output distribution of $\mathcal{H}^0_i$ for all $i \in \{1, \dots, \alpha-1\}$.*

Proof. We assume by contradiction that the lemma does not hold and we show how to do a reduction to the security of the underlying secret sharing scheme.

In the output of $\mathcal{H}^0_i$ (and $\mathcal{H}^0_{i-1}$) there are shares of a secret sharing scheme run x_i^0 and x_i^1 where $x_i^0 \oplus x_i^1 = x'_i \oplus \tilde{x}_i$ ($x_i^0 \oplus x_i^1 = x_i \oplus \tilde{x}_i$ in the case of $\mathcal{H}^0_{i-1}$). In more details we have:
an $(i-1)$-out-of-$(j-1)$ secret sharing of x_i^0: $(s_{i,1}, \dots, s_{i,j-1})$;
a $(l-i)$-out-of-$(n-j)$ secret sharing of x_i^1: $(s_{i,j+1}, \dots, s_{i,n})$;
and $s_{i,j} = \tilde{x}_i$.

Given the output of $\mathcal{H}^0_i$ (and $\mathcal{H}^0_{i-1}$) be $(s_{j_1}, \dots, s_{j_l})$ with $s_{j_1} = (s_{1,j_1}, \dots, s_{l,j_1}), \dots, s_{j_l} = (s_{1,j_l}, \dots, s_{l,j_l})$, $j_\alpha = j$ and $i < \alpha$ then it is possible to reconstruct x_0^i (since the output of the experiment contains $(\alpha-1)$ shares of a $(i-1)$-out-of-$(j-1)$ secret sharing scheme run on input x_i^0). However, given that $i < \alpha$, then there are $(l-\alpha) < (l-i)$ shares of the $(l-i)$-out-of$(n-j)$ secret sharing scheme run on input x_i^1 in the output of $\mathcal{H}^0_i$ (and $\mathcal{H}^0_{i-1}$), which are insufficient to reconstruct x_1^i. More formally, in this case we can make a reduction to the security of the $(l-i)$-out-of-$(n-j)$ secret sharing scheme. The reduction acts exactly as $\mathcal{H}^0_i$ (and $\mathcal{H}^0_{i-1}$) with the exception that the shares for the $(l-i)$-out-of-$(n-j)$ are computed by an external challenger. The external challenger, on input (y_0, y_1) such that $x_i^0 \oplus \tilde{x}_i \oplus y_0 = x_i$ and $x_i^0 \oplus \tilde{x}_i \oplus y_1 = x_i'$, tosses a coin $b \in \{0,1\}$, computes a secret sharing of y_b and sends $(l-i-1)$ shares to the reduction. The reduction then pick a subset of size $(l-j)$ of the received shares and uses them to complete the experiment accordingly to $\mathcal{H}_i^0$ (and $\mathcal{H}_{i-1}^0$). Note that if $b = 0$ then the output of the experiment corresponds to the output of $\mathcal{H}_{i-1}^0$ and to the output of $\mathcal{H}_i^0$ otherwise.

An argument similar to the one used in the proof of the Lemma 1 can be used to show prove the following lemma (the main difference is that in this case we rely on the security of the $(i-1)$-out-of-$(j-1)$ secret sharing sheme).

Lemma 2. *The output distribution of $\mathcal{H}_{i-1}^1$ is identical to the output distribution of $\mathcal{H}_i^1$ for all $i \in \{\alpha+1, \ldots, l\}$.*

To prove the second property we just need to observe that when $\nexists \alpha \in \{1, \ldots, l\}$ such that $j_\alpha = j$ then the share $s_{i,j} = \tilde{x}_i$ does not appear in the output of the experiment for all $i \in \{1, \ldots, l\}$. Therefore we can make a reduction to the 3-out-of-3 secret sharing scheme.

A.2 Proof of Theorem 2

Proof. We have already shown in Theorem 1 that $\mathsf{Share}^{\mathsf{PoSS}\star}, \mathsf{Reconstruct}^{\mathsf{PoSS}\star}$ is an ePoSS scheme, here we just prove that the scheme satisfies the property of Enhanced positional security.
Enhanced positional security. The proof that the second property holds follows from the same arguments showed above. We prove that the security holds for the case $c \le l$, the proof for the case where $c > l$ follows by similar arguments. The proof proceeds via hybrid arguments. Let $S = \{j_1, \ldots, j_{l+c}\} \subseteq \{1, \ldots, n\}$ be an arbitrarily chosen set with $0 \le j_1 < j_2 < \cdots < j_{l+c-1} < j_{l+c} \le n$ and $j_\alpha = j$.

We denote with $\mathcal{H}_\ell^0$, with $\ell \in \{0, \ldots, \alpha-\gamma-1\}$, the hybrid experiment that acts as follows.
Compute $(s_1, \ldots, s_n)$
$\overset{\$}{\leftarrow} \mathsf{Share}^{\mathsf{PoSS}\star}((x_1', \ldots, x_\ell', x_{\ell+1}, \ldots, x_{\alpha-\gamma}, \ldots, x_\alpha, x_{\alpha+1}, \ldots, x_l), j)\}$
Output $(s_{j_1}, \ldots, s_{j_l})$.
We denote with $\mathcal{H}_\ell^1$, with $\ell \in \{1, \ldots, l-\alpha\}$, the hybrid experiment that acts as follows.
Compute $(s_1, \ldots, s_n) \overset{\$}{\leftarrow}$
$\mathsf{Share}^{\mathsf{PoSS}\star}((x_1', \ldots, x_{\alpha-\gamma-1}', x_{\alpha-\gamma}, \ldots, x_\alpha, x_{\alpha+1}', \ldots, x_{\alpha+\ell}', x_{\alpha+\ell+1}, \ldots, x_l), j)\}$
Output $(s_{j_1}, \ldots, s_{j_{l+c}})$.

We now show that $\mathcal{H}_0^0 \equiv \cdots \equiv \mathcal{H}_{\alpha-\gamma-1}^0 \equiv \mathcal{H}_{\alpha+1}^1 \equiv \cdots \equiv \mathcal{H}_l^1$.

Lemma 3. *The output distribution of $\mathcal{H}_{i-1}^0$ is identical to the output distribution of $\mathcal{H}_i^0$ for all $i \in \{1, \ldots, \alpha-\gamma-1\}$.*

Proof. We assume by contradiction that the lemma does not hold and we show how to do a reduction to the security of the underlying secret sharing scheme.

In the output of $\mathcal{H}_i^0$ (and $\mathcal{H}_{i-1}^0$) there are shares of a secret sharing scheme run on input x_i^0 and x_i^1 where $x_i^0 \oplus x_i^1 = x_i' \oplus \tilde{x}_i$ ($x_i^0 \oplus x_i^1 = x_i \oplus \tilde{x}_i$ in the case of $\mathcal{H}_{i-1}^0$). In more detail, we have:
an $(i-1)$-out-of-$(j-1)$ secret sharing of x_i^0: $(s_{i,1}, \ldots, s_{i,j-1})$;
a $(l-i)$-out-of-$(n-j)$ secret sharing of x_i^1: $(s_{i,j+1}, \ldots, s_{i,n})$;
and $s_{i,j} = \tilde{x}_i$.

Given the output of $\mathcal{H}_i^0$ (and $\mathcal{H}_{i-1}^0$) be $(s_{j_1}, \ldots, s_{j_{l+c}})$ with $s_{j_1} = (s_{1,j_1}, \ldots, s_{l+c,j_1}), \ldots, s_{j_{l+c}} = (s_{1,j_l}, \ldots, s_{l+c,j_{l+c}})$, $j_\alpha = j$ and $i < \alpha - \gamma$ then it is possible to reconstruct x_0^i (since the output of the experiment contains $(\alpha - \gamma)$ shares of a $(i-1)$-out-of-$(j-1)$ secret sharing scheme run on input x_i^0). However, given that $i < \alpha - \gamma$, then there are at most $(l - \alpha + \gamma)$ shares of a the $(l-i)$-out-of$(n-j)$ secret sharing scheme run on input x_i^1 in the output of $\mathcal{H}_i^0$ (and $\mathcal{H}_{i-1}^0$). Given that $(l-i) > (l - \alpha + \gamma)$ then there is an insufficient to reconstruct x_1^i. More formally, in this case we can make a reduction to the security of the $(l-i)$-out-of-$(n-j)$ secret sharing scheme.

The reduction acts exactly as $\mathcal{H}_i^0$ (and $\mathcal{H}_{i-1}^0$) with the exception that the shares for the $(l-i)$-out-of-$(n-j)$ are computed by an external challenger. The external challenger, on input (y_0, y_1) such that $x_i^0 \oplus \tilde{x}_i \oplus y_0 = x_i$ and $x_i^0 \oplus \tilde{x}_i \oplus y_1 = x_i'$, tosses a coin $b \in \{0,1\}$, computes a secret sharing of y_b and sends $(l-i-1)$ shares to the reduction. The reduction then pick a subset of size $(l - \alpha + \gamma)$ of the received shares and uses them to complete the experiment accordingly to $\mathcal{H}_i^0$ (and $\mathcal{H}_{i-1}^0$). Note that if $b = 0$ then the output of the experiment corresponds to the output of $\mathcal{H}_{i-1}^0$ and to the output of $\mathcal{H}_i^0$ otherwise.

An argument similar to the one used in the proof of the Lemma 3 can be used to show prove the following lemma (the main difference is that in this case we rely on the security of the $(i-1)$-out-of-$(j-1)$ secret sharing sheme).

Lemma 4. *The output distribution of $\mathcal{H}_{i-1}^1$ is identical to the output distribution of $\mathcal{H}_i^1$ for all $i \in \{1, \ldots, l-\alpha\}$*

A.3 Proof of Theorem 3

Proof. To prove the theorem we need to show a simulator Sim such that for any $f \in \mathcal{F}_l$ and $x_{\overline{T}} \in \mathcal{X}_{\overline{T}}^l$, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:

$$\{\mathsf{Sim}^{f|_{\overline{T},x_{\overline{T}}}}(1^n, 1^l, 1^\lambda, T, K)\}, \{\mathsf{View}(1^n, 1^l, 1^\lambda, f, T, K, x_{\overline{T}})\}$$

where$\{\mathsf{View}(1^n, 1^l, 1^\lambda, f, T, K, x_{\overline{T}})\}$ is the view of the evaluator p_0 and of the colluding parties p_i (for $i \in T$) from running $\mathsf{NIMPC}^{\mathsf{th}}$ on input $x_{\overline{T}}$ for the honest parties.

By assumption we know that there exists a simulator Sim_1 such that for any $f \in \mathcal{F}_n$ and $x_{\overline{T}} \in \mathcal{X}_{\overline{T}}$, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:

$$\{\mathsf{Sim}_1^{f|_{\overline{T},x_{\overline{T}}}}(1^n, 1^\lambda, T)\}, \{\mathsf{View}(1^n, 1^\lambda, f, T, x_{\overline{T}})\}$$

where $\{\mathsf{View}(1^n, 1^\lambda, f, T, x_{\overline{T}})\}$ is the view of the evaluator p_0 and of the colluding parties p_i (for $i \in T$) from running NIMPC.

Let $T = \{c_1, \ldots, c_t\}$ be the set containing the indexes of the corrupted parties and $\overline{T} = \{1, \ldots, n\} - T = \{h_1, \ldots, h_{l-t}\}$ be the indexes of the honest parties participating in the execution of $\mathsf{NIMPC}^{\mathsf{th}}$. Sim works as follows.

Run Sim_1, and when Sim_1 queries $f|_{\overline{T},x_{\overline{T}}}$ forward the query to $f|_{\overline{T},x_{\overline{T}}}$. When an answer is received from $f|_{\overline{T},x_{\overline{T}}}$ forward it to Sim_1. Let $\tilde{\rho}_0, \tilde{m}_{h_1}, \ldots, \tilde{m}_{h_{l-t}}, \tilde{\rho}_{c_1}, \ldots \tilde{\rho}_{c_t}$ be the output do Sim_1

For each $i \in [t]$ compute $\tilde{m}_{c_i}^0 \leftarrow \mathsf{Msg}(\tilde{\rho}_{c_i}, 0)$ and $\tilde{m}_{c_i}^1 \leftarrow \mathsf{Msg}(\tilde{\rho}_{c_i}, 1)$

For $i = 1, \ldots, n$ pick $b_i \xleftarrow{\$} \{0,1\}$

If $i \in \overline{T}$ then run $\mathsf{PSS}(0^\lambda, \ldots, 0^\lambda, \tilde{m}_i, 0^\lambda, \ldots, 0^\lambda, i)$ thus obtaining $(s_{i,1}^{b_i}, \ldots, s_{i,n}^{b_i})$ and run $\mathsf{PSS}(0^\lambda, \ldots, 0^\lambda, 0^\lambda, 0^\lambda, \ldots, 0^\lambda, i)$ obtaining $(s_{i,1}^{1-b_i}, \ldots, s_{i,n}^{1-b_i})$.

If $i \in T$ then run $\mathsf{PSS}(0^\lambda, \ldots, 0^\lambda, \tilde{m}_i^0, 0^\lambda, \ldots, 0^\lambda, i)$ obtaining $(s_{i,1}^{b_i}, \ldots, s_{i,n}^{b_i})$ and run $\mathsf{PSS}(0^\lambda, \ldots, 0^\lambda, \tilde{m}_i^1, 0^\lambda, \ldots, 0^\lambda, i)$ thus obtaining $(s_{i,1}^{1-b_i}, \ldots, s_{i,n}^{1-b_i})$.

If $i \notin T \cup \overline{T}$ then run $\mathsf{PSS}(0^\lambda, \ldots, 0^\lambda, 0^\lambda, 0^\lambda, \ldots, 0^\lambda, i)$ obtaining $(s_{i,1}^{b_i}, \ldots, s_{i,n}^{b_i})$ and run $\mathsf{PSS}(0^\lambda, \ldots, 0^\lambda, 0^\lambda, 0^\lambda, \ldots, 0^\lambda, i)$ thus obtaining $(s_{i,1}^{1-b_i}, \ldots, s_{i,n}^{1-b_i})$.

Define $(\rho_0, \rho_1, \ldots, \rho_n)$ where $\rho_0 := \tilde{\rho}_0$ and for $i = 1, \ldots, n$, $\rho_i := (b_i, (s_{j,i}^0, s_{j,i}^1)_{j \in \{1,\ldots,n\}})$.

For each $i \in \overline{T}$

Set $s_{i,i} \leftarrow s_{i,i}^{b_i}$ and $d_i \leftarrow b_i$.

Set $m_i := ((s_{1,i}^0, s_{1,i}^1), \ldots, s_{i,i}, \ldots, (s_{n,i}^0, s_{n,i}^1), d_i)$.

Output $(m_i)_{i \in \overline{T}}, \tilde{\rho}_0, (\rho_i)_{i \in T}$.

Let $i_1, \ldots, i_l$ be the index of the parties participating in the execution of $\mathsf{NIMPC}^{\mathsf{th}}$. To show that the output distribution of the simulator is indistinguishable from the real world experiment we can consider a sequence of hybrid experiments $\mathcal{H}_1, \ldots, \mathcal{H}_l$ where $\mathcal{H}_1$ corresponds to the real world experiment. The hybrid experiment $\mathcal{H}_\ell$ with $\ell \in [l-1]$ is identical to $\mathcal{H}_{\ell-1}$ with the difference that the inputs of the PoSS scheme in position ℓ becomes 0 for all the executions of PoSS with respect to the index $i \neq i_\ell$.

If there exists $\ell' \in [l-1]$ such that the output distribution of $\mathcal{H}_\ell$ and $\mathcal{H}_{\ell+1}$ are distinguishable, then we can make a reduction to the positional security property of the PoSS scheme.

We now consider the sequence of hybrids experiments $\mathcal{H}'_1, \ldots, \mathcal{H}'_{l-t}$ where $\mathcal{H}'_1$ corresponds to $\mathcal{H}_\ell$. The hybrid experiment $\mathcal{H}'_\ell$ with $\ell \in [l-t]$ is identical to $\mathcal{H}'_{\ell-1}$ with the difference that the inputs of the PoSS scheme in position ℓ becomes 0 for all the executions of PoSS of step 3.1 with respect to the index $i = h_\ell$ of Fig. 2.

If there exists $\ell' \in [l-t-1]$ such that the output output distributions of $\mathcal{H}'_\ell$ and $\mathcal{H}'_{\ell+1}$ are distinguishable, then we can make a reduction to the standard security property of the PoSS scheme.

The last hybrid that we consider $\mathcal{H}^\star$ is equal to $\mathcal{H}'_{l-t}$ with the only difference that the messages of NIMPC are replaced with the output of the simulator Sim_1

The proof ends with the observation that $\mathcal{H}^\star$ corresponds to Sim.

A.4 Proof of Theorem 4

Proof. Correctness follows by inspection. We start the proof of security by observing that if less than l parties are active then the adversary learns nothing due to the property of *standard security* offered by the PoSS scheme. We now prove our theorem for the case where $c = 1$. The proof then can be generalized for any constant c. That is, we assume that the number of active parties is just $l + 1$.

For simplicity, let us assume that $1, \ldots, l+1$ are the indexes of the active parties. Let $S_1, \ldots, S_{l+1}$ be all the possible subsets of size l of $\{1, \ldots, l+1\}$. Then the possible output that can be computed are $\mathsf{OUT}_1, \ldots, \mathsf{OUT}_{l+1}$ where for each $i \in \{1, \ldots, l+1\}$ $\mathsf{OUT}_i \leftarrow f^{\mathsf{msg\_sel}}\big((x_{j_1}, j_1), \ldots, (x_{j_l}, j_l)\big)$ with $S_i := \{j_1, \ldots, j_l\}$.

This means that each output of $f^{\mathsf{msg\_sel}}$ is obtained by constructing the concatenation of l values sorted in non-decreasing order with respect to their indexes taken from $A = (x_1^1, x_1^2, x_2^2, x_2^3, x_3^3, \ldots, x_{l-1}^l, x_l^l, x_l^{l+1})$. The simulator can compute all these values by querying $\mathcal{O}_f$ using the subsets $S_1, \ldots, S_{l+1}$. In the description of the simulator we assume that the simulator knows A (obtained by querying $\mathcal{O}_f$ as described above).

Simulation for the setup

For each $j = 1, \ldots, \lambda$ *//Simulation for the party* p_1

```
    Pick $b^1_j \xleftarrow{\$} \{0,1\}$.
    Set $y^1_{j,0} := 0$ and $y^1_{j,1} := 1$.
For each $k = 2, \dots l$
    For each $j = 1, \dots, \lambda$
        Pick $b^k_j \xleftarrow{\$} \{0,1\}$.
        Set $y^k_{j,0} := 0$ and $y^k_{j,1} := 0$.
Run $\mathsf{PSS}(y^1_{1,b^1_1}||\dots||y^1_{\lambda,b^1_\lambda}, y^2_{1,b^2_{\lambda+1}}||\dots||y^2_{\lambda,b^2_\lambda}, \dots, y^l_{1,b^l_1}||\dots||y^l_{\lambda,b^l_\lambda}, 1)$ thus obtaining $(s^0_{i,1}, \dots, s^0_{i,n})$.
Run $\mathsf{PSS}(y^1_{1,1-b^1_1}||\dots||y^1_{\lambda,1-b^1_\lambda}, y^2_{1,1-b^2_{\lambda+1}}||\dots||y^2_{\lambda,1-b^2_\lambda}, \dots, y^l_{1,1-b^l_1}||\dots||y^l_{\lambda,1-b^l_\lambda}, 1)$ thus obtaining $(s^1_{i,1}, \dots, s^1_{i,n})$.
Set $B_1 = (b^k_1, \dots, b^k_\lambda)_{k \in [l]}$.
For each $j = 1, \dots, \lambda$   //Simulation for the party $p_{l+1}$
    Pick $b^1_j \xleftarrow{\$} \{0,1\}$.
    Set $y^l_{j,0} := 0$ and $y^l_{j,1} := 1$.
For each $k = 1 \dots l-1$
    For each $j = 1, \dots, \lambda$
        Pick $b^k_j \xleftarrow{\$} \{0,1\}$.
        Set $y^k_{j,0} := 0$ and $y^k_{j,1} := 0$.
Run $\mathsf{PSS}(y^1_{1,b^1_1}||\dots||y^1_{\lambda,b^1_\lambda}, y^2_{1,b^2_{\lambda+1}}||\dots||y^2_{\lambda,b^2_\lambda}, \dots, y^l_{1,b^l_1}||\dots||y^l_{\lambda,b^l_\lambda}, 1)$ thus obtaining $(s^0_{i,1}, \dots, s^0_{i,n})$.
Run $\mathsf{PSS}(y^1_{1,1-b^1_1}||\dots||y^1_{\lambda,1-b^1_\lambda}, y^2_{1,1-b^2_{\lambda+1}}||\dots||y^2_{\lambda,1-b^2_\lambda}, \dots, y^l_{1,1-b^l_1}||\dots||y^l_{\lambda,1-b^l_\lambda}, 1)$ thus obtaining $(s^1_{i,1}, \dots, s^1_{i,n})$.
Set $B_1 = (b^k_1, \dots, b^k_\lambda)_{k \in [l]}$.
For $i = 2, \dots, l$ //Simulation for the parties $p_2, \dots, p_l$
    For each $k = 1, \dots i-2, i+1, \dots l$
        For each $j = 1, \dots, \lambda$
            Pick $b^k_j \xleftarrow{\$} \{0,1\}$.
            Set $y^k_{j,0} := 0$ and $y^k_{j,1} := 0$.
    For each $k = i-1, \dots i$
        For each $j = 1, \dots, \lambda$
            Pick $b^k_j \xleftarrow{\$} \{0,1\}$.
            Set $y^k_{j,0} := 0$ and $y^k_{j,1} := 1$.
    Run $\mathsf{PSS}(y^1_{1,b^1_1}||\dots||y^1_{\lambda,b^1_\lambda}, y^2_{1,b^2_{\lambda+1}}||\dots||y^2_{\lambda,b^2_\lambda}, \dots, y^l_{1,b^l_1}||\dots||y^l_{\lambda,b^l_\lambda}, i)$ thus obtaining $(s^0_{i,1}, \dots, s^0_{i,n})$.
    Run $\mathsf{PSS}(y^1_{1,1-b^1_1}||\dots||y^1_{\lambda,1-b^1_\lambda}, y^2_{1,1-b^2_{\lambda+1}}||\dots||y^2_{\lambda,1-b^2_\lambda}, \dots, y^l_{1,1-b^l_1}||\dots||y^l_{\lambda,1-b^l_\lambda}, i)$ thus obtaining $(s^1_{i,1}, \dots, s^1_{i,n})$.
    Set $B_i = (b^k_1, \dots, b^k_\lambda)_{k \in [l]}$.
```

Simulation for the online messages

```
Simulation for the party $p_1$
Parse $x^1_1$ as a $\lambda$ bit string $x_{1,1}, \dots, x_{1,\lambda}$.
For each $j \in [\lambda]$ if $x_{1,j} = b^1_j$ then set $d^1_j = b^1_j$ else set $d^1_j = 1 - b^1_j$.
For each $k \in \{2, \dots, l\}$, $j \in [\lambda]$ set $d^k_j \xleftarrow{\$} \{0,1\}$.
Set $D_i \leftarrow (d^k_1, \dots, d^k_\lambda)_{k \in [l]}$.
Set $m_i := (D_i, (s^0_{1,i}, s^1_{1,i}), \dots, (s^0_{n,i}, s^1_{n,i}))$.
Simulation for the party $p_{l+1}$
Parse $x^{l+1}_l$ as a $\lambda$ bit string $x_{l,1}, \dots, x_{l,\lambda}$.
For each $j \in [\lambda]$ if $x_{l+1,j} = b^1_j$ then set $d^{l+1}_j = b^{l+1}_j$ else set $d^{l+1}_j = 1 - b^{l+1}_j$.
For each $k \in \{1, \dots, l-1\}$, $j \in [\lambda]$ set $d^k_j \xleftarrow{\$} \{0,1\}$.
Set $D_i \leftarrow (d^k_1, \dots, d^k_\lambda)_{k \in [l]}$.
Set $m_i := (D_i, W_i, (s^0_{1,i}, s^1_{1,i}), \dots, (s^0_{n,i}, s^1_{n,i}))$.
For $i = 2, \dots, l$ //Simulation for the parties $p_2, \dots, p_l$
    For each $k \in [l]$ parse $x^i_k$ as a $\lambda$ bit string $x_{k,1}, \dots, x_{k,\lambda}$.
    For each $k \in \{i-1, i\}$, $j \in [\lambda]$ if $x_{k,j} = b^k_j$ then set $d^k_j = b^k_j$ else set $d^k_j = 1 - b^k_j$.
    For each $k \in \{1, \dots, i-2, i+1, \dots, l\}$, $j \in [\lambda]$ set $d^k_j \xleftarrow{\$} \{0,1\}$.
    Set $D_i \leftarrow (d^k_1, \dots, d^k_\lambda)$.
    Set $m_i := (D_i, (s^0_{1,i}, s^1_{1,i}), \dots, (s^0_{n,i}, s^1_{n,i}))$.
```

The enhanced security of the PoSS scheme guarantees that for each active party p_{k_i} the only messages that can be reconstructed are the messages given as input by p_{k_i} in the positions $i - \gamma, \dots, i$ with $\gamma := \min\{\alpha - 1, c\}$ if $i \leq l$ or the messages in positions $i - c, \dots l$ if $i > l$. In the specific case that we are describing we have that for p_1 only x^1_1 can be reconstructed, for p_i only x^i_{i-1} and x^i_i (with $i \in \{2, \dots, l\}$) can be reconstructed and for p_{l+1} only x^{l+1}_l.

Hence, we can replace the input of the honest parties in any other position with 0^λ and rely on the enhanced security of the ePoSS scheme.

A.5 Proof of Theorem 5

Proof. Let $N \leq l + c$ be the number of parties participating in the execution of the protocol. Let $K = (k_1, \dots, k_N)$ be the indexes of the the active parties, $X = (x_{k_1}, \dots, x_{k_N})$ be the respective inputs and $V = (v_{k_1}, \dots, v_{k_N})$ be the random values chosen by the parties in the step 1.1. of the online phase. Let $S_1, \dots, S_{\mathsf{inputs}}$ be all the possible subset of K of size l where $\mathsf{inputs} = \binom{N}{l}$. We denote with OUT_i the output of f evaluated on the inputs of the parties indexed by the elements of S_i for all $i \in [\mathsf{inputs}]$.

We say that in an execution of Π^{PSM} there is a collision if and only there are four (or more) indexes $\alpha, \beta, \gamma, a, b$ with $\alpha \neq \beta$, $\alpha, \beta \in [\mathsf{inputs}]$, $a, b \in [l]$ such that $\mathsf{H}(A) = \mathsf{H}(B)$ where $S_\alpha = \{v_{\alpha_1}, \dots, v_{\alpha_l}\}$, $S_\beta = \{v_{\beta_1}, \dots, v_{\beta_l}\}$ and $A \leftarrow v_{\alpha_1} \oplus \dots \oplus v_{\alpha_a}$, $B \leftarrow v_{\beta_1} \oplus \dots \oplus v_{\beta_b}$.

We now prove that if there are no collisions then Π^{PSM} is secure. In the case that there is a collision then Π^{PSM} becomes insecure. This come from the fact that an adversary can obtain two (or more) outputs of the function $g_{k'}$ for some k' for the same value sel' but with respect to two different couple of inputs $(x, y) \neq (x', y')$. Hence, the adversary gets $\mu^{\mathsf{sel}',0}_{k+1,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel}',0}_{k'+1,i}, x||y||\cdot)$ and $\mu^{\mathsf{sel}',0}_{k+1,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel}',0}_{k'+1,i}, x'||y'||\cdot)$ for all $i \in [n]$ where $\mathsf{sel}' = \mathsf{H}(A) = \mathsf{H}(B)$ (i.e., the adversary gets two evaluations of Msg on two different inputs using the same randomness). The protocol Π^{2PC} gives no security guarantees in such a situation and any attempts of simulation would fail.

We denote the event of collision in an execution with $\mathsf{Exp\_collision}$ and prove the following lemma

Lemma 5. Prob $[\ \mathsf{Exp\_collision}\ ] \leq 1 - e^{-1}$.

Proof. For the Birthday Paradox we have that, given a set Q of q element chosen independently and uniformly at random the probability that there exists a couple $(y_1, y_2) \in Q$ such that $y_1 \neq y_2$ and $\mathsf{H}(y_1) = \mathsf{H}(y_2)$ is at most $p_{\mathsf{Hcoll}} = \frac{q^2}{2\lambda'}$.

In our case we consider $q = \binom{l+c}{l} \leq \binom{\lambda+c}{\lambda} \leq \lambda^c$ since in g_k, for each $k \in [l]$, the hash function H can be evaluated on at most $N \leq \binom{l+c}{l}$ values. We note that we are guaranteed that the elements on which the hash function is evaluated are independent and uniformly random because of the step 4 of the setup phase. Therefore, we have that $p_{\mathsf{Hcoll}} = \frac{\lambda^{2c}}{2\lambda'} = \frac{\lambda^{2c}}{2\lambda^{2c+2}} \leq \frac{1}{\lambda^2}$.

Then we have that Prob $[\ \mathsf{Exp\_collision}\ ] = 1 - (1 - p_{\mathsf{Hcoll}})^l \leq 1 - (1 - \frac{1}{\lambda^2})^l \leq 1 - (1 - \frac{1}{l})^l \leq 1 - e^{-1}$ since $\lambda^2 > l$.

To complete the proof we need to show that in the case that the event $\mathsf{Exp\_collision}$ does no occur then our protocol is secure. More formally, we prove the following lemma.

Lemma 6. *If the event* $\mathsf{Exp\_collision}$ *does not occur, the* Π^{PSM} *is a c-secure ad hoc PSM protocol for any constant c.*

Since Π^{2PC} is 0-robust, by assumption we know that there exists a randomized algorithm Sim such that for any g_k with $k \in [l]$ and $x_{\overline{T}} \in \mathcal{X}_{\overline{T}}$, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:

$$\{\mathsf{Sim}_1^{g_k}(1^n, 1^\lambda, T)\}, \{\mathsf{View}(1^n, 1^\lambda, f, T, x_{\overline{T}})\}$$

where T is an empy set and $\{\mathsf{View}(1^n, 1^\lambda, g_j, K, \overline{x})\}$ is the view of the evaluator from running Π^{2PC}.

Moreover, by assumption we have that for any $K = \{j_1, \ldots, j_N\}$ there exists a randomized algorithm Sim (called simulator) such that for any $\overline{x} = x_{j_1}, \ldots, x_{j_N} \in \{\{0,1\}^\lambda\}^N$, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:

$$\{\mathsf{Sim}_2^{\mathcal{O}_{f^{\mathsf{msg\_sel}}}}(1^n, 1^\lambda, K)\}, \{\mathsf{View}(1^n, 1^\lambda, f^{\mathsf{msg\_sel}}, K, \overline{x})\}$$

$\{\mathsf{View}(1^n, 1^\lambda, f^{\mathsf{msg\_sel}}, K, \overline{x})\}$ is the view of the evaluator from running $\Pi^{\mathsf{msg\_sel}}$.

The simulator Sim works as described in Fig. 10.

```
**Parameters:** Security parameter $\lambda$, range-size of $\mathsf{H}$ $\lambda' = \lambda^{2c+1}$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.
**Setup simulation:**
  1. Pick $V = (v_{k_1}, \ldots, v_{k_N})$ such that there are no indexes $\alpha, \beta, \gamma, a, b$ with $\alpha \neq \beta$, $\alpha, \beta \in [\mathsf{inputs}]$, $a, b \in [l]$ such that $\mathsf{H}(A) = \mathsf{H}(B)$ where $S_\alpha = \{v_{\alpha_1}, \ldots, v_{\alpha_l}\}$, $S_\beta = \{v_{\beta_1}, \ldots, v_{\beta_l}\}$ and $A \leftarrow v_{\alpha_1} \oplus \cdots \oplus v_{\alpha_a}$, $B \leftarrow v_{\beta_1} \oplus \cdots \oplus v_{\beta_b}$.
  - For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        $v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_l}$.
        $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
        $w \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_{l-1}}$.
        $\mathsf{sel}'' \leftarrow \mathsf{H}(w)$
        Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_l$ answer with $\mathsf{OUT}_i$ thus obtaining $(R_{l,s_l}^{\mathsf{sel}''}, \mu_{l,s_l}^{\mathsf{sel}'',0}, m_{l,s_l}^{\mathsf{sel}'',1})$.
  - For each $c \in [n]$
        If $(R_{l,c}^{\mathsf{sel}''}, \mu_{l,c}^{\mathsf{sel}'',0}, m_{l,c}^{\mathsf{sel}'',1})$ has not been defined then run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $0^\lambda$ obtaining $(R_{k,c}^{\mathsf{sel}''}, \mu_{k,c}^{\mathsf{sel}'',0}, m_{k,c}^{\mathsf{sel}'',1})$.
  - For each $k \in \{l-1, \ldots, 3\}$
        For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
            $v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_k}$.
            $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
            $w \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_{k-1}}$.
            $\mathsf{sel}'' \leftarrow \mathsf{H}(w)$
            Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $\{\mu_{k+1,c}^{\mathsf{sel}',0}\}_{c \in \{k+2,\ldots,n\}}$ thus obtaining $(R_{k,s_k}^{\mathsf{sel}''}, \mu_{k,s_k}^{\mathsf{sel}'',0}, m_{k,s_k}^{\mathsf{sel}'',1})$.
        For each $c \in [n]$
            If $(R_{k,c}^{\mathsf{sel}''}, \mu_{k,c}^{\mathsf{sel}'',0}, m_{k,c}^{\mathsf{sel}'',1})$ has not been defined then run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $0^\lambda$ obtaining $(R_{k,c}^{\mathsf{sel}''}, \mu_{k,c}^{\mathsf{sel}'',0}, m_{k,c}^{\mathsf{sel}'',1})$.
  - Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_2$ answer with $\{\mu_{3,i}^{\mathsf{sel}',0}\}_{i \in \{3,\ldots,n\}}$ thus obtaining $(R_{2,s_1}^{s_2}, m_{1,s_2}^{s_1,0}, m_{2,s_1}^{s_2,1})$.
  - For each $i, j \in [n]$ with $i \neq j$, if $(R_{2,i}^{j}, m_{1,j}^{i,0}, m_{2,i}^{j,1})$ has not been defined then run $\mathsf{Sim}_1$, and when $\mathsf{Sim}_1$ queries $g_2$ answer with $0^\lambda$ thus obtaining $(R_{2,i}^{j}, m_{1,j}^{i,0}, m_{2,i}^{j,1})$.
  - Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^\lambda, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho_0^{\mathsf{th}}, \rho_1^{\mathsf{th}}, \ldots, \rho_n^{\mathsf{th}})$.
**Simulation of the online messages**
    Run $\mathsf{Sim}_2$, and when $\mathsf{Sim}_2$ queries $f^{\mathsf{msg\_sel}}$ with $S_i$ answer with

        $\{m_{1,\mathsf{sel}}^{s_1,0}\}_{\mathsf{sel} \in [n]-\{s_1\}}, \{m_{2,s_2}^{\mathsf{sel},1}\}_{\mathsf{sel} \in [\lambda']}, \ldots, \{m_{l-1,s_{l-1}}^{\mathsf{sel},1}\}_{\mathsf{sel} \in [\lambda']}, \{m_{l,s_l}^{\mathsf{sel},1}\}_{\mathsf{sel} \in [\lambda']}$.

    Let $\rho_0^{\mathsf{th}}, (m_{k_1}, \ldots, m_{k_N})$ be the output of $\mathsf{Sim}_2$.
    Output $\rho_0 := (\rho_0^{\mathsf{th}}, \{R_{k,i}^{\mathsf{sel}}\}_{\mathsf{sel} \in [\lambda'], i \in [n], k \in [l]}), (m_{k_1}, \ldots, m_{k_N})$.
```

Fig. 10: Our simulator Sim.

To prove our theorem we need to show that a set $K = \{j_1, \ldots, j_N\}$ and the inputs $\overline{x} = x_{j_1}, \ldots, x_{j_N} \in \{\{0,1\}^\lambda\}^N$, the following distributions are perfectly (resp., statistically, computationally) indistinguishable:

$$\{\mathsf{Sim}^f(1^n, 1^\ell, 1^\lambda, K)\}, \{\mathsf{View}(1^n, 1^\ell, 1^\lambda, f, K, \overline{x})\}$$

The only difference between the two distributions is that in the first experiment all the all the messages of Π^{2PC} and the messages of $\Pi^{\mathsf{msg\_sel}}$ are computed using, respectively, the simulators

Sim_1 and Sim_2, whereas in the second experiment the messages of Π^{2PC} and $\Pi^{\mathsf{msg\_sel}}$ are computed using honest algorithms defined by their respective setup and the online procedures.

More formally, we show that the following holds.

$\mathsf{View}(1^n, 1^\ell, 1^\lambda, f, K, \overline{x}) \approx \mathcal{H}_0 \approx \mathcal{H}_1 \approx \mathcal{H}_2 \approx \mathcal{H}_a^3 \approx \mathcal{H}_b^3 \approx \mathcal{H}_a^4 \approx \mathcal{H}_b^4 \approx \cdots \approx \mathcal{H}_a^{l-2} \approx \mathcal{H}_b^{l-2} \approx \mathcal{H}_4 \approx \mathsf{Sim}^f(1^n, 1^\ell, 1^\lambda, K)$, where the formal description of $\mathcal{H}_0$ is provided in Fig. 11, the description of $\mathcal{H}_1$ in Fig. 12, the description of $\mathcal{H}_2$ in Fig. 13, the description of $\mathcal{H}_a^i$ with $i \in \{3, \ldots l-1\}$ in Fig. 14, the description of $\mathcal{H}_b^i$ with $i \in \{3, \ldots l-1\}$ in Fig. 15 and the description of $\mathcal{H}_4$ in Fig. 16.

To prove that $\mathsf{View}(1^n, 1^\ell, 1^\lambda, f, K, \overline{x}) \approx \mathcal{H}_0$ we rely on the security of Π^{PSM} since the only difference between the real world experiment and $\mathcal{H}_0$ is that in $\mathcal{H}_0$ we use Sim_2 instead of running the algorithms of Π^{PSM}.

The difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that the messages for the instantiations of Π^{2PC} used to evaluate the function g_2 are simulated. In this case we can rely on the security of Π^{2PC} to argue that $\mathcal{H}_0 \approx \mathcal{H}_1$.

The difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is that the randomness used to run the setup for the Π^{2PC} instantiations used to evaluate g_3 is randomly sampled from $\{0,1\}^\kappa$ instead of being computed using the PRG. So, in this case we can rely on the security of the PRG to prove that $\mathcal{H}_1 \approx \mathcal{H}_2$.

The difference between $\mathcal{H}_2$ and $\mathcal{H}_a^3$ is that the messages for the execution of Π^{2PC} used to evaluate the function g_3 are simulated. In this case we can rely on the security of Π^{2PC} to argue that $\mathcal{H}_3 \approx \mathcal{H}_a^3$.

For each $\ell \in \{4, \ldots l-2\}$, the difference between $\mathcal{H}_a^\ell$ and $\mathcal{H}_b^\ell$ is that the randomnesses used to run the setup for the Π^{2PC} instantiations used to evaluate g_ℓ is randomly sampled from $\{0,1\}^\kappa$ instead of being computed using the PRG. So, in this case we can rely on the security of the PRG to prove that $\mathcal{H}_a^\ell \approx \mathcal{H}_b^\ell$ for each $\ell \in \{4, \ldots l-2\}$.

For each $\ell \in \{4, \ldots l-2\}$, the difference between $\mathcal{H}_b^{\ell-1}$ and $\mathcal{H}_a^\ell$ is that the messages for the executions of Π^{2PC} used to evaluate the function g_ℓ are simulated. In this case we can rely on the security of Π^{2PC} to argue that $\mathcal{H}_a^\ell \approx \mathcal{H}_b^\ell$ for each $\ell \in \{4, \ldots l-2\}$.

The difference between $\mathcal{H}_b^{\ell-2}$ and $\mathcal{H}_4$ is that the messages for the executions of Π^{2PC} used to evaluate the function g_l are simulated. In this case we can rely on the security of Π^{2PC} to argue that $\mathcal{H}_b^{\ell-2} \approx \mathcal{H}_4$

The proof ends with the observation that $\mathcal{H}_4$ corresponds to Sim.

```
Common parameters: Security parameter $\lambda$, range-size of $\mathsf{H}$ $\lambda' = \lambda^{2c+1}$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.
Setup:
    For each $i, j \in [n]$ with $i \neq j$ do the following.
        Run $\mathsf{Setup}(1^2, g_2, 1^\lambda)$ thus obtaining $(R^j_{2,i}, \rho^{j,0}_{2,i}, \rho^{j,1}_{2,i})$.
    For each $k \in \{3, \dots, l-1\}$, $i \in [n]$, $\mathsf{sel} \in [\lambda']$ do the following.
        Pick $\mathsf{r}^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^\lambda$.
        Compute $\mathsf{PRG}(\mathsf{r}^{\mathsf{sel}}_{k,i})$ thus obtaining $r$.
        Run $\mathsf{Setup}(1^2, g_k, 1^\lambda; r)$ thus obtaining $(R^{\mathsf{sel}}_{k,i}, \rho^{\mathsf{sel},0}_{k,i}, \rho^{\mathsf{sel},1}_{k,i})$.
    For each $\mathsf{sel} \in [\lambda']$ $i \in [n]$ run $\mathsf{Setup}(1^2, g_l, 1^\lambda)$ thus obtaining $(R^{\mathsf{sel}}_{l,i}, \rho^{\mathsf{sel},0}_{l,i}, \rho^{\mathsf{sel},1}_{l,i})$
    Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^\lambda, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \dots, \rho^{\mathsf{th}}_n)$.
    For $i \leftarrow 1, \dots, n$ pick $v_i \xleftarrow{\$} \{0,1\}^\lambda$ and set
    $\rho_i := (v_i, (\mathsf{r}^{\mathsf{sel}}_{k,j>i})_{j\in[n]\mathsf{sel}\in[\lambda'],k\in\{3,\dots,l\}}, (\rho^{\mathsf{sel},1}_{k,i})_{\mathsf{sel}\in[\lambda'],k\in\{3,\dots,l\}}, (\rho^{i,0}_{2,j}, \rho^{j,1}_{2,i})_{j\in[n]-\{i\}}, \rho^{\mathsf{th}}_i)$ and
    $\rho_0 := \rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel}\in[\lambda'],i\in[n],k\in[l]}$
Online messages
    On input $x_i \in \{0,1\}^\lambda$ and $\rho_i$ the party $p_i$ does the following.
        For each $j \in [n] - \{i\}$ compute $m^{i,0}_{1,j} \leftarrow \mathsf{Msg}(\rho^{i,0}_{2,j}, (x_i, v_i))$.
        For each $j \in [n] - \{i\}$ compute $m^{j,1}_{2,i} \leftarrow \mathsf{Msg}(\rho^{j,1}_{2,i}, x_i||v_i||\{\mathsf{r}^{\mathsf{sel},0}_{3,c>i}\}_{c\in[n]})$.
        For each $k \in \{3, \dots, l-2\}, \mathsf{sel} \in [\lambda']$ compute
        $m^{\mathsf{sel},1}_{k,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{k,i}, x_i||v_i||\{\mathsf{r}^{\mathsf{sel},0}_{k+1,j>i}\}_{j\in[n],\mathsf{sel}\in[\lambda']})$
        For each $\mathsf{sel} \in [\lambda']$ compute $m^{\mathsf{sel},1}_{l,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{l,i}, x_i||v_i)$
        $\underline{\text{Run } \mathsf{Sim}_2\text{, and when } \mathsf{Sim}_2 \text{ queries } f^{\mathsf{msg\_sel}} \text{ with } S_i \text{ answer with}}$

            $$\underline{\{m^{s_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel}\in[n]-\{s_1\}}, \{m^{\mathsf{sel},1}_{2,s_2}\}_{\mathsf{sel}\in[\lambda']}, \dots, \{m^{\mathsf{sel},1}_{l-1,s_{l-1}}\}_{\mathsf{sel}\in[\lambda']}, \{m^{\mathsf{sel},1}_{l,s_l}\}_{\mathsf{sel}\in[\lambda']}.}$$

        $\underline{\text{Let } \rho^{\mathsf{th}}_0, (m_{k_1}, \dots, m_{k_N}) \text{ be the output of } \mathsf{Sim}_2}$.
        Output $\rho_0 := (\rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel}\in[\lambda'],i\in[n],k\in[l]}), (m_{k_1}, \dots, m_{k_N})$.
```

Fig. 11: $\mathcal{H}_0$. The parts that differs from the previous hybrid are underlined. We follow the same approach also in the description of the next hybrid.

```
**Common parameters:** Security parameter $\lambda$, range-size of $\mathsf{H}$ $\lambda' = \lambda^{2c+1}$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.
**Setup:**
    For each $i \in [n]$, $\mathsf{sel} \in [\lambda']$, $k \in \{3, \ldots, l\}$
        Pick $\mathsf{r}^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\lambda}$.
        Compute $\mathsf{PRG}(\mathsf{r}^{\mathsf{sel}}_{k,i})$ thus obtaining $r^{\mathsf{sel}}_{k,i}$.
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$.
        $v \leftarrow v_{s_1} \oplus v_{s_2}$.
        $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
        For each $i \in \{4, \ldots, n\}$ compute
            $(R^{\mathsf{sel}'}_{3,i}, \rho^{\mathsf{sel}',0}_{3,i}, \rho^{\mathsf{sel}',1}_{3,i}) \leftarrow \mathsf{Setup}(1^2, 1^{\lambda}, g_3; r^{\mathsf{sel}'}_{3,i})$.
            $\mu^{\mathsf{sel}',0}_{3,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel}',0}_{3,i}, x_{s_1}||x_{s_2}||v)$.
        Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_2$ answer with $\{\mu^{\mathsf{sel}',0}_{3,i}\}_{i\in\{3,\ldots,n\}}$
        thus obtaining $(R^{s_2}_{2,s_1}, m^{s_1,0}_{1,s_2}, m^{s_2,1}_{2,s_1})$.
    For each $i, j \in [n]$ with $i \neq j$, if $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$ has not ben defined then run
    $\mathsf{Sim}_1$, and when $\mathsf{Sim}_1$ queries $g_2$ answer with $0^{\lambda}$ thus obtaining $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$.
    For each $i \in [n]$ with and $\mathsf{sel} \in [\lambda']$, if $\mu^{\mathsf{sel},1}_{3,i}$ has not ben defined then run
    $\mathsf{Setup}(1^2, g_3, 1^{\lambda}; r^{\mathsf{sel}}_{3,i})$ thus obtaining $(R^{\mathsf{sel}}_{3,i}, \rho^{\mathsf{sel},0}_{3,i}, \rho^{\mathsf{sel},1}_{3,i})$
    For each $k \in \{4, \ldots, l-1\}$, $i \in [n]$, $\mathsf{sel} \in [\lambda']$ do the following.
        Run $\mathsf{Setup}(1^2, g_k, 1^{\lambda}; r^{\mathsf{sel}}_{k,i})$ thus obtaining $(R^{\mathsf{sel}}_{k,i}, \rho^{\mathsf{sel},0}_{k,i}, \rho^{\mathsf{sel},1}_{k,i})$.
    For each $\mathsf{sel} \in [\lambda']$ $i \in [n]$ run $\mathsf{Setup}(1^2, g_l, 1^{\lambda}; r^{\mathsf{sel}}_{l,i})$ thus obtaining $(R^{\mathsf{sel}}_{l,i}, \rho^{\mathsf{sel},0}_{l,i}, \rho^{\mathsf{sel},1}_{l,i})$
    Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^{\lambda}, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \ldots, \rho^{\mathsf{th}}_n)$.
**Online messages**
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        For each $k \in \{3, \ldots, l-2\}, \mathsf{sel} \in [\lambda'] - \{\mathsf{sel}'\}$ compute
        $m^{\mathsf{sel},1}_{k,s_k} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{k,s_k}, x_{s_k}||v_{s_k}||\{\mathsf{r}^{\mathsf{sel},0}_{k+1,j>i}\}_{j\in[n],\mathsf{sel}\in[\lambda']})$
        For each $\mathsf{sel} \in [\lambda']$ compute $m^{\mathsf{sel},1}_{l,s_l} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{l,s_l}, x_{s_l}||v_{s_l})$
        Run $\mathsf{Sim}_2$, and when $\mathsf{Sim}_2$ queries $f^{\mathsf{msg\_sel}}$ with $S_i$ answer with

            $$\{m^{s_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel}\in[n]-\{s_1\}}, \{m^{\mathsf{sel},1}_{2,s_2}\}_{\mathsf{sel}\in[\lambda']}, \ldots, \{m^{\mathsf{sel},1}_{l-1,s_{l-1}}\}_{\mathsf{sel}\in[\lambda']}, \{m^{\mathsf{sel},1}_{l,s_l}\}_{\mathsf{sel}\in[\lambda']}.$$

        Let $\rho^{\mathsf{th}}_0, (m_{k_1}, \ldots, m_{k_N})$ be the output of $\mathsf{Sim}_2$.
        Output $\rho_0 := (\rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel}\in[\lambda'],i\in[n],k\in[l]}), (m_{k_1}, \ldots, m_{k_N})$.
```

Fig. 12: $\mathcal{H}_1$.

```
**Common parameters:** Security parameter $\lambda$, range-size of $\mathsf{H}$ $\lambda' = \lambda^{2c+1}$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.
**Setup:**
    $\underline{\text{For each } i \in [n],\ \mathsf{sel} \in [\lambda],\ k = 3}$
        $\underline{\text{Pick } r^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\kappa}.}$
    For each $i \in [n]$, $\mathsf{sel} \in [\lambda]$, $k \in \{4, \ldots, l\}$
        Pick $\mathsf{r}^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\lambda}$.
        Compute $\mathsf{PRG}(\mathsf{r}^{\mathsf{sel}}_{k,i})$ thus obtaining $r^{\mathsf{sel}}_{k,i}$.
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$.
        $v \leftarrow v_{s_1} \oplus v_{s_2}$.
        $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
        For each $i \in \{4, \ldots, n\}$ compute
            $(R^{\mathsf{sel}'}_{3,i}, \rho^{\mathsf{sel}',0}_{3,i}, \rho^{\mathsf{sel}',1}_{3,i}) \leftarrow \mathsf{Setup}(1^2, 1^{\lambda}, g_3; r^{\mathsf{sel}'}_{3,i})$.
            $\mu^{\mathsf{sel}',0}_{3,i} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel}',0}_{3,i}, x_{s_1}||x_{s_2}||v)$.
        Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_2$ answer with $\{\mu^{\mathsf{sel}',0}_{3,i}\}_{i \in \{3,\ldots,n\}}$ thus obtaining $(R^{s_2}_{2,s_1}, m^{s_1,0}_{1,s_2}, m^{s_2,1}_{2,s_1})$.
    For each $i, j \in [n]$ with $i \neq j$, if $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$ has not ben defined then run $\mathsf{Sim}_1$ queries $g_2$ answer with $0^{\lambda}$ thus obtaining $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$.
    For each $i \in [n]$ with and $\mathsf{sel} \in [\lambda']$, if $\mu^{\mathsf{sel},1}_{3,i}$ has not ben defined then run run $\mathsf{Setup}(1^2, g_3, 1^{\lambda})$ thus obtaining $(R^{\mathsf{sel}}_{3,i}, \rho^{\mathsf{sel},0}_{3,i}, \rho^{\mathsf{sel},1}_{3,i})$
    For each $k \in \{4, \ldots, l-1\}$, $i \in [n]$, $\mathsf{sel} \in [\lambda']$ do the following.
        Run $\mathsf{Setup}(1^2, g_k, 1^{\lambda}; r)$ thus obtaining $(R^{\mathsf{sel}}_{k,i}, \rho^{\mathsf{sel},0}_{k,i}, \rho^{\mathsf{sel},1}_{k,i})$.
    For each $\mathsf{sel} \in [\lambda']$ $i \in [n]$ run $\mathsf{Setup}(1^2, g_l, 1^{\lambda})$ thus obtaining $(R^{\mathsf{sel}}_{l,i}, \rho^{\mathsf{sel},0}_{l,i}, \rho^{\mathsf{sel},1}_{l,i})$
    Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^{\lambda}, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \ldots, \rho^{\mathsf{th}}_n)$.
**Online messages**
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        For each $k \in \{3, \ldots, l-2\}, \mathsf{sel} \in [\lambda'] - \{\mathsf{sel}'\}$ compute
        $m^{\mathsf{sel},1}_{k,s_k} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{k,s_k}, x_{s_k}||v_{s_k}||\{\mathsf{r}^{\mathsf{sel},0}_{k+1,j>i}\}_{j \in [n], \mathsf{sel} \in [\lambda']})$
        For each $\mathsf{sel} \in [\lambda']$ compute $m^{\mathsf{sel},1}_{l,s_l} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{l,s_l}, x_{s_l}||v_{s_l})$
        Run $\mathsf{Sim}_2$, and when $\mathsf{Sim}_2$ queries $f^{\mathsf{msg\_sel}}$ with $S_i$ answer with

            $$\{m^{s_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel} \in [n] - \{s_1\}}, \{m^{\mathsf{sel},1}_{2,s_2}\}_{\mathsf{sel} \in [\lambda']}, \ldots, \{m^{\mathsf{sel},1}_{l-1,s_{l-1}}\}_{\mathsf{sel} \in [\lambda']}, \{m^{\mathsf{sel},1}_{l,s_l}\}_{\mathsf{sel} \in [\lambda']}.$$

        Let $\rho^{\mathsf{th}}_0, (m_{k_1}, \ldots, m_{k_N})$ be the output of $\mathsf{Sim}_2$.
        Output $\rho_0 := (\rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel} \in [\lambda'], i \in [n], k \in [l]}), (m_{k_1}, \ldots, m_{k_N})$.
```

Fig. 13: $\mathcal{H}_2$.

Common parameters: Security parameter λ, range-size of H $\lambda' = \lambda^{2c+1}$, maximum number of parties n, threshold l, and c where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.

```
Setup:
    For each $i \in [n]$, $\mathsf{sel} \in [\lambda]$, $\underline{k \in \{3, \ldots, \ell\}}$
        Pick $r^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\kappa}$.
  - For each $i \in [n]$, $\mathsf{sel} \in [\lambda]$, $\underline{k \in \{\ell+1, \ldots, l\}}$
        Pick $\mathsf{r}^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\lambda}$.
        Compute $\mathsf{PRG}(\mathsf{r}^{\mathsf{sel}}_{k,i})$ thus obtaining $r^{\mathsf{sel}}_{k,i}$.
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        $\underline{v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_\ell}}$ $\underline{\mathsf{sel}' \leftarrow \mathsf{H}(v)}$.
        $\underline{\text{For each } c \in \{\ell+1, \ldots, n\} \text{ compute}}$
            $\underline{(R^{\mathsf{sel}'}_{\ell+1,c}, \rho^{\mathsf{sel}',0}_{\ell+1,c}, \rho^{\mathsf{sel}',1}_{\ell+1,c}) \leftarrow \mathsf{Setup}(1^2, 1^{\lambda}, g_{\ell+1}; r^{\mathsf{sel}'}_{\ell+1,c}).}$
            $\underline{\mu^{\mathsf{sel}',0}_{\ell+1,c} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel}',0}_{\ell+1,c}, x_{s_1}||x_{s_2}|| \ldots ||x_{s_\ell}||v).}$
    $\underline{\text{For each } k \in \{\ell, \ldots, 3\}}$
        $\underline{\text{For each } i \in [\mathsf{inputs}] \text{, let } s_1, \ldots, s_l \text{ be the elements of } S_i}$
            $\underline{v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_k}, \mathsf{sel}' \leftarrow \mathsf{H}(v), w \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_{k-1}}, \mathsf{sel}'' \leftarrow \mathsf{H}(w)}$
            $\underline{\text{Run } \mathsf{Sim}_1 \text{ and when } \mathsf{Sim}_1 \text{ queries } g_k \text{ answer with } \{\mu^{\mathsf{sel}',0}_{k+1,c}\}_{c \in \{k+2, \ldots, n\}}}$
            $\underline{\text{thus obtaining } (R^{\mathsf{sel}''}_{k,s_k}, \mu^{\mathsf{sel}'',0}_{k,s_k}, m^{\mathsf{sel}'',1}_{k,s_k}).}$
        $\underline{\text{For each } c \in [n]}$
            $\underline{\text{If } (R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c}) \text{ has not been defined then run } \mathsf{Sim}_1 \text{ and when}}$
            $\underline{\mathsf{Sim}_1 \text{ queries } g_k \text{ answer with } 0^{\lambda} \text{ obtaining } (R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c}).}$
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        $v \leftarrow v_{s_1} \oplus v_{s_2}$, $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
        Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_2$ answer with $\{\mu^{\mathsf{sel}',0}_{3,i}\}_{i \in \{3, \ldots, n\}}$ thus obtaining
        $(R^{s_2}_{2,s_1}, m^{s_1,0}_{1,s_2}, m^{s_2,1}_{2,s_1})$.
  - For each $i, j \in [n]$ with $i \neq j$, if $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$ has not been defined then run $\mathsf{Sim}_1$, and when $\mathsf{Sim}_1$
    queries $g_2$ answer with $0^{\lambda}$ thus obtaining $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$.
  - For each $k \in \{\ell+1, \ldots, l-1\}$, $i \in [n]$, $\mathsf{sel} \in [\lambda']$ do the following.
        Run $\mathsf{Setup}(1^2, g_k, 1^{\lambda}; r^{\mathsf{sel}}_{k,i})$ thus obtaining $(R^{\mathsf{sel}}_{k,i}, \rho^{\mathsf{sel},0}_{k,i}, \rho^{\mathsf{sel},1}_{k,i})$.
  - For each $i \in [n]$ $\mathsf{sel} \in [\lambda']$ run $\mathsf{Setup}(1^2, g_l, 1^{\lambda})$ thus obtaining $(R^{\mathsf{sel}}_{l,i}, \rho^{\mathsf{sel},0}_{l,i}, \rho^{\mathsf{sel},1}_{l,i})$
  - Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^{\lambda}, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \ldots, \rho^{\mathsf{th}}_n)$.
Online messages
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        $\underline{\text{For each } k \in \{\ell+1, \ldots, l-2\}, \mathsf{sel} \in [\lambda'] - \{\mathsf{sel}'\} \text{ compute}}$
        $\underline{m^{\mathsf{sel},1}_{k,s_k} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{k,s_k}, x_{s_k}||v_{s_k}||\{\mathsf{r}^{\mathsf{sel}}_{k+1,j>s_k}\}_{j \in [n], \mathsf{sel} \in [\lambda']})}$
        For each $\mathsf{sel} \in [\lambda']$ compute $m^{\mathsf{sel},1}_{l,s_l} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{l,s_l}, x_{s_l}||v_{s_l})$
        Run $\mathsf{Sim}_2$, and when $\mathsf{Sim}_2$ queries $f^{\mathsf{msg\_sel}}$ with $S_i$ answer with
        $\{m^{s_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel} \in [n] - \{s_1\}}, \{m^{\mathsf{sel},1}_{2,s_2}\}_{\mathsf{sel} \in [\lambda']}, \ldots, \{m^{\mathsf{sel},1}_{l-1,s_{l-1}}\}_{\mathsf{sel} \in [\lambda']}, \{m^{\mathsf{sel},1}_{l,s_l}\}_{\mathsf{sel} \in [\lambda']}$. Let $\rho^{\mathsf{th}}_0, (m_{k_1}, \ldots, m_{k_N})$ be
        the output of $\mathsf{Sim}_2$.
        Output $\rho_0 := (\rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel} \in [\lambda'], i \in [n], k \in [l]}), (m_{k_1}, \ldots, m_{k_N})$.
```

Fig. 14: $\mathcal{H}^{\ell}_{a}$.

Common parameters: Security parameter λ, range-size of H $\lambda' = \lambda^{2c+1}$, maximum number of parties n, threshold l, and c where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.

```
Setup:
    For each $i \in [n]$, $\mathsf{sel} \in [\lambda]$, $\underline{k \in \{3, \ldots, \ell + 1\}}$
        Pick $r^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\kappa}$.
  - For each $i \in [n]$, $\mathsf{sel} \in [\lambda]$, $\underline{k \in \{\ell + 2, \ldots, l\}}$
        Pick $\mathsf{r}^{\mathsf{sel}}_{k,i} \xleftarrow{\$} \{0,1\}^{\lambda}$.
        Compute $\mathsf{PRG}(\mathsf{r}^{\mathsf{sel}}_{k,i})$ thus obtaining $r^{\mathsf{sel}}_{k,i}$.
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        $v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_\ell}$.
        $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
        For each $c \in \{\ell + 1, \ldots, n\}$ compute
            $(R^{\mathsf{sel}'}_{\ell+1,c}, \rho^{\mathsf{sel}',0}_{\ell+1,c}, \rho^{\mathsf{sel}',1}_{\ell+1,c}) \leftarrow \mathsf{Setup}(1^2, 1^{\lambda}, g_{\ell+1}; r^{\mathsf{sel}'}_{\ell+1,c})$.
            $\mu^{\mathsf{sel}',0}_{\ell+1,c} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel}',0}_{\ell+1,c}, x_{s_1}||x_{s_2}|| \ldots ||x_{s_\ell}||v)$.
    For each $k \in \{\ell, \ldots, 3\}$
        For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
            $v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_k}$, $\mathsf{sel}' \leftarrow \mathsf{H}(v)$, $w \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_{k-1}}$, $\mathsf{sel}'' \leftarrow \mathsf{H}(w)$
            Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $\{\mu^{\mathsf{sel}',0}_{k+1,c}\}_{c \in \{k+2,\ldots,n\}}$ thus obtaining
            $(R^{\mathsf{sel}''}_{k,s_k}, \mu^{\mathsf{sel}'',0}_{k,s_k}, m^{\mathsf{sel}'',1}_{k,s_k})$.
        For each $c \in [n]$
            If $(R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c})$ has not been defined then run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer
            with $0^{\lambda}$ obtaining $(R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c})$.
    Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_2$ answer with $\{\mu^{\mathsf{sel}',0}_{3,i}\}_{i \in \{3,\ldots,n\}}$ thus obtaining $(R^{s_2}_{2,s_1}, m^{s_1,0}_{1,s_2}, m^{s_2,1}_{2,s_1})$.
  - For each $i, j \in [n]$ with $i \neq j$, if $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$ has not been defined then run $\mathsf{Sim}_1$, and when $\mathsf{Sim}_1$
    queries $g_2$ answer with $0^{\lambda}$ thus obtaining $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$.
  - For each $k \in \{\ell + 1, \ldots, l - 1\}$, $i \in [n]$, $\mathsf{sel} \in [\lambda']$ do the following.
        Run $\mathsf{Setup}(1^2, g_k, 1^{\lambda}; r^{\mathsf{sel}}_{k,i})$ thus obtaining $(R^{\mathsf{sel}}_{k,i}, \rho^{\mathsf{sel},0}_{k,i}, \rho^{\mathsf{sel},1}_{k,i})$.
  - For each $i \in [n]$ $\mathsf{sel} \in [\lambda']$ run $\mathsf{Setup}(1^2, g_l, 1^{\lambda}; r^{\mathsf{sel}}_{l,i})$ thus obtaining $(R^{\mathsf{sel}}_{l,i}, \rho^{\mathsf{sel},0}_{l,i}, \rho^{\mathsf{sel},1}_{l,i})$
  - Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^{\lambda}, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \ldots, \rho^{\mathsf{th}}_n)$.
Online messages
    For each $i \in [\mathsf{inputs}]$, let $s_1, \ldots, s_l$ be the elements of $S_i$
        For each $k \in \{\ell + 1, \ldots, l - 2\}, \mathsf{sel} \in [\lambda'] - \{\mathsf{sel}'\}$ compute
        $m^{\mathsf{sel},1}_{k,s_k} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{k,s_k}, x_{s_k}||v_{s_k}||\{\mathsf{r}^{\mathsf{sel}}_{k+1,j>s_k}\}_{j \in [n], \mathsf{sel} \in [\lambda']})$
        For each $\mathsf{sel} \in [\lambda']$ compute $m^{\mathsf{sel},1}_{l,s_l} \leftarrow \mathsf{Msg}(\rho^{\mathsf{sel},1}_{l,s_l}, x_{s_l}||v_{s_l})$
        Run $\mathsf{Sim}_2$, and when $\mathsf{Sim}_2$ queries $f^{\mathsf{msg\_sel}}$ with $S_i$ answer with

            $$\{m^{s_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel} \in [n]-\{s_1\}}, \{m^{\mathsf{sel},1}_{2,s_2}\}_{\mathsf{sel} \in [\lambda']}, \ldots, \{m^{\mathsf{sel},1}_{l-1,s_{l-1}}\}_{\mathsf{sel} \in [\lambda']}, \{m^{\mathsf{sel},1}_{l,s_l}\}_{\mathsf{sel} \in [\lambda']}.$$

        Let $\rho^{\mathsf{th}}_0, (m_{k_1}, \ldots, m_{k_N})$ be the output of $\mathsf{Sim}_2$.
        Output $\rho_0 := (\rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel} \in [\lambda'], i \in [n], k \in [l]}), (m_{k_1}, \ldots, m_{k_N})$.
```

Fig. 15: $\mathcal{H}^{\ell}_{b}$.

```
**Common parameters:** Security parameter $\lambda$, range-size of $\mathsf{H}$ $\lambda' = \lambda^{2c+1}$, maximum number of parties $n$, threshold $l$, and $c$ where $l + c \leq n$ denotes the maximum number of active parties supported by the protocol.
**Setup:**
  - For each $i \in [\mathsf{inputs}]$, let $s_1, \dots, s_l$ be the elements of $S_i$
        $v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_l}$.
        $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
        $w \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_{l-1}}$.
        $\mathsf{sel}'' \leftarrow \mathsf{H}(w)$
        Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_l$ answer with $\mathsf{OUT}_i$ thus obtaining $(R^{\mathsf{sel}''}_{l,s_l}, \mu^{\mathsf{sel}'',0}_{l,s_l}, m^{\mathsf{sel}'',1}_{l,s_l})$.
  - For each $c \in [n]$
        If $(R^{\mathsf{sel}''}_{l,c}, \mu^{\mathsf{sel}'',0}_{l,c}, m^{\mathsf{sel}'',1}_{l,c})$ has not been defined then run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $0^\lambda$ obtaining $(R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c})$.
  - For each $k \in \{l-1, \dots, 3\}$
        For each $i \in [\mathsf{inputs}]$, let $s_1, \dots, s_l$ be the elements of $S_i$
            $v \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_k}$.
            $\mathsf{sel}' \leftarrow \mathsf{H}(v)$
            $w \leftarrow v_{s_1} \oplus \cdots \oplus v_{s_{k-1}}$.
            $\mathsf{sel}'' \leftarrow \mathsf{H}(w)$
            Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $\{\mu^{\mathsf{sel}',0}_{k+1,c}\}_{c \in \{k+2,\dots,n\}}$ thus obtaining $(R^{\mathsf{sel}''}_{k,s_k}, \mu^{\mathsf{sel}'',0}_{k,s_k}, m^{\mathsf{sel}'',1}_{k,s_k})$.
        For each $c \in [n]$
            If $(R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c})$ has not been defined then run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_k$ answer with $0^\lambda$ obtaining $(R^{\mathsf{sel}''}_{k,c}, \mu^{\mathsf{sel}'',0}_{k,c}, m^{\mathsf{sel}'',1}_{k,c})$.
  - Run $\mathsf{Sim}_1$ and when $\mathsf{Sim}_1$ queries $g_2$ answer with $\{\mu^{\mathsf{sel}',0}_{3,i}\}_{i \in \{3,\dots,n\}}$ thus obtaining $(R^{s_2}_{2,s_1}, m^{s_1,0}_{1,s_2}, m^{s_2,1}_{2,s_1})$.
  - For each $i, j \in [n]$ with $i \neq j$, if $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$ has not been defined then run $\mathsf{Sim}_1$, and when $\mathsf{Sim}_1$ queries $g_2$ answer with $0^\lambda$ thus obtaining $(R^{j}_{2,i}, m^{i,0}_{1,j}, m^{j,1}_{2,i})$.
  - Run $\mathsf{Setup}^{\mathsf{msg\_sel}}(1^n, 1^l, 1^\lambda, f^{\mathsf{msg\_sel}})$ thus obtaining $(\rho^{\mathsf{th}}_0, \rho^{\mathsf{th}}_1, \dots, \rho^{\mathsf{th}}_n)$.
**Online messages**
    Run $\mathsf{Sim}_2$, and when $\mathsf{Sim}_2$ queries $f^{\mathsf{msg\_sel}}$ with $S_i$ answer with

        $\{m^{s_1,0}_{1,\mathsf{sel}}\}_{\mathsf{sel} \in [n]-\{s_1\}}, \{m^{\mathsf{sel},1}_{2,s_2}\}_{\mathsf{sel} \in [\lambda']}, \dots, \{m^{\mathsf{sel},1}_{l-1,s_{l-1}}\}_{\mathsf{sel} \in [\lambda']}, \{m^{\mathsf{sel},1}_{l,s_l}\}_{\mathsf{sel} \in [\lambda']}.$

    Let $\rho^{\mathsf{th}}_0, (m_{k_1}, \dots, m_{k_N})$ be the output of $\mathsf{Sim}_2$.
    Output $\rho_0 := (\rho^{\mathsf{th}}_0, \{R^{\mathsf{sel}}_{k,i}\}_{\mathsf{sel} \in [\lambda'], i \in [n], k \in [l]}), (m_{k_1}, \dots, m_{k_N})$.
```

Fig. 16: $\mathcal{H}_4$.

A.6 Proof of Theorem 6

Proof. By assumption we have that for any $K = \{k_1, \dots, k_N\}$ there exists a randomized algorithm Sim_e such that for any $\overline{x} = x_{k_1}, \dots, x_{k_N} \in \{\{0,1\}^\lambda\}^N$, with probability e^{-1} the following distributions are indistinguishable:

$$\{\mathsf{Sim}_e^{G_e}(1^n, 1^\lambda, 1^l, K)\}, \{\mathsf{View}(1^n, 1^\lambda, 1^l, G_e, K, \overline{x})\}$$

where $\{\mathsf{View}(1^n, 1^\lambda, G_e, K, \overline{x})\}$ is the view of the evaluator from running Π^{PSM}.

By definition then we know that an execution of Π^{PSM} is secure with probability $1/p$ for a constant $p \leq e$. This implies that in one execution of Π^{APSM} there is, with overwhelming probability, at least one execution of Π^{PSM} that is secure. That is, the probability that there exists at least one secure execution of Π^{PSM} is given by $1 - (1 - 1/p)^{\mathsf{m}} \leq 1 - e^{-\lambda}$ since $\mathsf{m} = p\lambda$.

We recall that it is possible to check whether an execution of Π^{PSM} is secure (i.e., simulatable) by having access to the randomness used in the setup (the v values). We use this property to make sure that in the simulated execution of Π^{APSM} the number of insecure executions of Π^{PSM} are the same (in expectation) as in the real world. In more detail, our simulator Sim samples m random coins $r_1, \dots, r_{\mathsf{m}}$ and checks if there exists one randomness r_α (with $\alpha \in [\mathsf{m}]$) such that the setup off Π^{PSM} would yield to a secure execution of Π^{PSM}. If there is no such a randomness then Sim aborts, otherwise it runs the simulator Sim_α using fresh randomness. Above we have showed that with overwhelming probability such r_α is sampled by Sim. In the description of Sim that we propose, we assume that α has been already computed following this strategy. We now provide more details on how the simulation works.

Let $N \leq l + c$ be the number of parties participating in the execution of the protocol. Let $K = (k_1, \dots, k_N)$ be the indexes of the the active parties, $X = (x_{k_1}, \dots, x_{k_N})$ be the respective inputs.

Let $S_1, \dots, S_{\mathsf{inputs}}$ be all the possible subset of K of size l where $\mathsf{inputs} = \binom{N}{l}$. We denote with OUT_i the output of f evaluated on the inputs of the parties indexed by the elements of S_i for all $i \in [\mathsf{inputs}]$.

Our simulator Sim works as follows.

```
Sample the randomnesses $r_1, \dots r_{\mathsf{m}}$ and compute $\alpha$ as described earlier
For each $j \in \mathsf{m} - \{\alpha\}$
    Run $\mathsf{Setup}^{\mathsf{PSM}}(1^n, 1^l, 1^\lambda, G_j)$ on input $r_j$ thus obtaining $\rho_0^j, \rho_1^j, \dots, \rho_n^j$.
Set $\rho_0 := (\rho_0^j)_{j \in [\mathsf{m}] - \{j\}}, \rho_1 := (\rho_1^j)_{j \in [\mathsf{m}] - \{j\}}, \dots \rho_n := (\rho_n^j)_{j \in [\mathsf{m}] - \{j\}}$
For each $k \in l$, $i \in K$
    Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, 0^\lambda)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
For each $j \in [\mathsf{m}] - \{\alpha\}$
    For each $i \in K$
        Run $\mathsf{Msg}^{\mathsf{PSM}}(\rho_i^j, ((x_i^{j,k})_{k \in [l]}, i))$ thus obtaining $m_i^j$.
For each $j \in [\mathsf{m}] - \{\alpha\}$
    For each $i \in [N]$
        Compute $y_i^j = \mathsf{Eval}^{\mathsf{HSS}}(j, x_{s_{i_1}}^{j,1}, \dots, x_{s_{i_l}}^{j,l})$ where $S_i = (s_{i_1}, \dots, s_{i_l})$
Run $\mathsf{Sim}_\alpha$ and when $\mathsf{Sim}_\alpha$ queries $g_\alpha$ with the indexes $S_i$ return $y_i^\alpha \leftarrow \mathsf{OUT}_i \oplus y_i^1 \oplus, \dots, \oplus y_i^{\alpha-1} \oplus y_i^{\alpha+1} \oplus \cdots \oplus y_i^{\mathsf{m}}$.
Parse the output $\mathsf{Sim}_\alpha$ as $\rho_0^\alpha, \{m_i^\alpha\}_{i \in K}$ and add $\rho_0^\alpha$ to the list $\rho_0$
Output $m_i := (m_i^j)_{i \in K, j \in [\mathsf{m}]}$
```

We now show that Sim is a valid simulator via hybrid arguments. The first hybrid that we consider is $\mathcal{H}^\star$ and is described as follows.

```
For each $j \in \mathsf{m} - \{\alpha\}$
    Run $\mathsf{Setup}^{\mathsf{PSM}}(1^n, 1^l, 1^\lambda, G_j)$ thus obtaining $\rho_0^j, \rho_1^j, \dots, \rho_n^j$.
Set $\rho_0 := (\rho_0^j)_{j \in [\mathsf{m}]-\{j\}}, \rho_1 := (\rho_1^j)_{j \in [\mathsf{m}]-\{j\}}, \dots \rho_n := (\rho_n^j)_{j \in [\mathsf{m}]-\{j\}}$
For each $k \in l$, $i \in K$
    Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, x_i)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
For each $j \in [\mathsf{m}] - \{\alpha\}$
    For each $i \in K$
        Run $\mathsf{Msg}^{\mathsf{PSM}}(\rho_i^j, ((x_i^{j,k})_{k \in [l]}, i))$ thus obtaining $m_i^j$.
For each $j \in [\mathsf{m}] - \{\alpha\}$
    For each $i \in [N]$
        Compute $y_i^j = \mathsf{Eval}^{\mathsf{HSS}}(j, x_{s_{i_1}}^{j,1}, \dots, x_{s_{i_l}}^{j,l})$ where $S_i = (s_{i_1}, \dots, s_{i_l})$
Run $\mathsf{Sim}_\alpha$ and when $\mathsf{Sim}_\alpha$ queries $g_\alpha$ with the indexes $S_i$ return $y_i^\alpha \leftarrow \mathsf{OUT}_i \oplus y_i^1 \oplus, \dots, \oplus y_i^{\alpha-1} \oplus y_i^{\alpha+1} \oplus \dots \oplus y_i^{\mathsf{m}}$.
Parse the output $\mathsf{Sim}_\alpha$ as $\rho_0^\alpha, \{m_i^\alpha\}_{i \in K}$ and add $\rho_0^\alpha$ to the list $\rho_0$
Output $m_i := (m_i^j)_{i \in K, j \in [\mathsf{m}]}$
```

The only difference between this hybrid and the real world experiment is that one execution of $\mathit{\Pi}^{\mathsf{PSM}}$ is executed by using the simulator instead of the honest procedure. In the case that there exists and adversary that distinguishes between $\mathcal{H}_1$ and the real world experiment we can construct an adversary $\mathcal{A}'$ that breaks the security of $\mathit{\Pi}^{\mathsf{PSM}}$ with probability q' greater than $1/2$. The adversary $\mathcal{A}'$ works as follows.

In this security game the challenger picks a random bit b and if $b = 0$ it generates a real world transcript for $\mathit{\Pi}^{\mathsf{PSM}}$ and check if the transcript would be simulatable. If it is not then it aborts, otherwise $\mathcal{A}'$ sends the transcript to $\mathcal{A}'$. If $b = 1$ then the challenger generates a simulated transcript and sends it to $\mathcal{A}'$

The adversary $\mathcal{A}'$ works as follows

If the challenger aborts, then $\mathcal{A}'$ outputs 0 (i.e. the transcript would have been a real world transcript).

If the challenger sends $\tilde{\rho}, \{\tilde{m}_i\}_{i \in K}$ then $\mathcal{A}'$ acts exactly as in $\mathcal{H}_1$ but he uses $\tilde{\rho}, \{\tilde{m}_i\}_{i \in K}$ in the place of $\rho_0^\alpha, \{m_i^\alpha\}_{i \in K}$ in the step 7.

$\mathcal{A}'$ then outputs what $\mathcal{A}$ outputs.

Let $q = 1 - p^{-1}$. The success probability of $\mathcal{A}'$ is given by $\frac{q + (1-q)q'}{2} + \frac{q'}{2} \geq \frac{q + 2q'}{2} = q' + \frac{q}{2} > \frac{1+q}{2}$. Given that that the probability of distinguishing between the real and the simulated execution of $\mathit{\Pi}^{\mathsf{PSM}}$ is at most $\frac{1}{2} + \frac{q}{2}$ and that $q' > 1/2$ we have reached a contradiction.

Next we consider a sequence of hybrids $\mathcal{H}_1^\ell$ for each $\ell \in n$. The hybrid $\mathcal{H}^\ell$ is described as follows.

```
Let $L$ be the set containing the first $\ell$ smaller values contained in $K$.
For each $j \in \mathsf{m} - \{\alpha\}$
    Run $\mathsf{Setup}^{\mathsf{PSM}}(1^n, 1^l, 1^\lambda, G_j)$ thus obtaining $\rho_0^j, \rho_1^j, \dots, \rho_n^j$.
Set $\rho_0 := (\rho_0^j)_{j \in [\mathsf{m}]-\{j\}}, \rho_1 := (\rho_1^j)_{j \in [\mathsf{m}]-\{j\}}, \dots \rho_n := (\rho_n^j)_{j \in [\mathsf{m}]-\{j\}}$
For each $k \in l$
    For each $i \in L$
        Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, 0^\lambda)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
    For each $i \in K - L$
        Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, x_i)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
```

```
For each $j \in [\mathsf{m}] - \{\alpha\}$
    For each $i \in K$
        Run $\mathsf{Msg}^{\mathsf{PSM}}(\rho_i^j, ((x_i^{j,k})_{k\in[l]}, i))$ thus obtaining $m_i^j$.
For each $j \in [\mathsf{m}] - \{\alpha\}$
    For each $i \in [N]$
        Compute $y_i^j = \mathsf{Eval}^{\mathsf{HSS}}(j, x_{s_{i_1}}^{j,1}, \dots, x_{s_{i_l}}^{j,l})$ where $S_i = (s_{i_1}, \dots, s_{i_l})$
Run $\mathsf{Sim}_\alpha$ and when $\mathsf{Sim}_\alpha$ queries $g_\alpha$ with the indexes $S_i$ return $y_i^\alpha \leftarrow \mathsf{OUT}_i \oplus y_i^1 \oplus, \dots, \oplus y_i^{\alpha-1} \oplus y_i^{\alpha+1} \oplus \dots \oplus y_i^{\mathsf{m}}$.
Parse the output $\mathsf{Sim}_\alpha$ as $\rho_0^\alpha, \{m_i^\alpha\}_{i\in K}$ and add $\rho_0^\alpha$ to the list $\rho_0$
Output $m_i := (m_i^j)_{i\in K, j\in[\mathsf{m}]}$
```

We now prove the following lemma

Lemma 7. *If* HSS *is a secure HSS scheme then the output distributions of* $\mathcal{H}_\ell$ *is indistinguishable from the output distributions of* $\mathcal{H}_\ell$ *for each* $\ell \in \{0, \dots n-1\}$.

Proof. If by contradiction the lemma does not hold then there exists $\ell' \in \{1, \dots n\}$ such that the output distributions of $\mathcal{H}_{\ell'-1}$ and $\mathcal{H}_{\ell'}$ are distinguishable. Let $\mathcal{A}$ be adversary that distinguishes the two hybrids, then we show an adversary $\mathcal{A}'$ for the HSS scheme. $\mathcal{A}$ works as follows.

```
Let $L$ be the set containing the first $\ell - 1$ smaller elements of $K$, and $x$ be the $\ell$-th smaller values contained in $K$.
Send $(x, 0^\lambda)$ as challenge messages to the challenger of the HSS security game.
Upon receiving the sharers $\{x_{\ell'}^{1,k}, \dots, x_{\ell'}^{\alpha-1,k}, x_{\ell'}^{\alpha+1,k}, \dots x_{\ell'}^{\mathsf{m},k}\}_{k\in[l]}$ does the following[14].
    For each $j \in \mathsf{m} - \{\alpha\}$
        Run $\mathsf{Setup}^{\mathsf{PSM}}(1^n, 1^l, 1^\lambda, G_j)$ thus obtaining $\rho_0^j, \rho_1^j, \dots, \rho_n^j$.
    Set $\rho_0 := (\rho_0^j)_{j\in[\mathsf{m}]-\{j\}}, \rho_1 := (\rho_1^j)_{j\in[\mathsf{m}]-\{j\}}, \dots \rho_n := (\rho_n^j)_{j\in[\mathsf{m}]-\{j\}}$
    For each $k \in l$
        For each $i \in L - \{x\}$
            Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, 0^\lambda)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
        For each $i \in K - L - \{x\}$
            Run $\mathsf{Share}^{\mathsf{HSS}}(1^\lambda, k, x_i)$ thus obtaining $x_i^{1,k}, \dots x_i^{\mathsf{m},k}$.
    For each $j \in [\mathsf{m}] - \{\alpha\}$
        For each $i \in K$
            Run $\mathsf{Msg}^{\mathsf{PSM}}(\rho_i^j, ((x_i^{j,k})_{k\in[l]}, i))$ thus obtaining $m_i^j$.
    For each $j \in [\mathsf{m}] - \{\alpha\}$
        For each $i \in [N]$
            Compute $y_i^j = \mathsf{Eval}^{\mathsf{HSS}}(j, x_{s_{i_1}}^{j,1}, \dots, x_{s_{i_l}}^{j,l})$ where $S_i = (s_{i_1}, \dots, s_{i_l})$
    Run $\mathsf{Sim}_\alpha$ and when $\mathsf{Sim}_\alpha$ queries $g_\alpha$ with the indexes $S_i$ return $y_i^\alpha \leftarrow \mathsf{OUT}_i \oplus y_i^1 \oplus, \dots, \oplus y_i^{\alpha-1} \oplus y_i^{\alpha+1} \oplus \dots \oplus y_i^{\mathsf{m}}$.
    Parse the output $\mathsf{Sim}_\alpha$ as $\rho_0^\alpha, \{m_i^\alpha\}_{i\in K}$ and add $\rho_0^\alpha$ to the list $\rho_0$
    Send $m_i := (m_i^j)_{i\in K, j\in[\mathsf{m}]}$ to $\mathcal{A}$ and output what $\mathcal{A}$ outputs
```

The proof ends with the observation that if the challenger has encrypted 0 then the output of $\mathcal{A}'$ corresponds to the output of $\mathcal{A}$ in $\mathcal{H}_{\ell'}$, and to the output of $\mathcal{A}$ in $\mathcal{H}_{\ell-1}$ otherwise.

[14] Without loss of generality in the reduction assume that the security of the HSS scheme is maintained under parallel composition.

The proof of the theorem instead ends with the observation that the output distribution of $\mathcal{H}_\ell$ is identical to the output distribution of the simulated experiment.