Non-Interactive Secure Computation Based on Cut-and-Choose

Arash Afshar[1], Payman Mohassel[1], Benny Pinkas[2⋆], and Ben Riva[2,3⋆⋆]

[1] University of Calgary
[2] Bar-Ilan University
[3] Tel Aviv University

Abstract. In recent years, secure two-party computation (2PC) has been demonstrated to be feasible in practice. However, all efficient general-computation 2PC protocols require multiple rounds of interaction between the two players. This property restricts 2PC to be only relevant to scenarios where both players can be simultaneously online, and where communication latency is not an issue.

This work considers the model of 2PC with a *single* round of interaction, called *Non-Interactive Secure Computation (NISC)*. In addition to the non-interaction property, we also consider a flavor of NISC that allows reusing the first message for many different 2PC invocations, possibly with different players acting as the player who sends the second message, similar to a public-key encryption where a single public-key can be used to encrypt many different messages.

We present a NISC protocol that is based on the cut-and-choose paradigm of Lindell and Pinkas (Eurocrypt 2007). This protocol achieves concrete efficiency similar to that of best multi-round 2PC protocols based on the cut-and-choose paradigm. The protocol requires only t garbled circuits for achieving cheating probability of 2^{-t}, similar to the recent result of Lindell (Crypto 2013), but only needs a single round of interaction.

To validate the efficiency of our protocol, we provide a prototype implementation of it and show experiments that confirm its competitiveness with that of the best multi-round 2PC protocols. This is the *first* prototype implementation of an efficient NISC protocol.

In addition to our NISC protocol, we introduce a new encoding technique that significantly reduces communication in the NISC setting. We further show how our NISC protocol can be improved in the multi-round setting, resulting in a highly efficient constant-round 2PC that is also suitable for pipelined implementation.

1 Introduction

Secure two-party computation (2PC) is a very powerful tool that allows two participants to compute any function of their private inputs without revealing any information about the inputs except for the value of the function. Furthermore, if the execution of the 2PC protocol is completed, it is guaranteed that its output is the correct output. In this work, unless said otherwise, we only discuss 2PC protocols that are secure even against malicious (aka active) participants, who might arbitrarily deviate from the protocol that they should be executing. The investigation of secure two-party protocols began with the seminal work of Yao [Yao86] that showed the feasibility of this concept. In recent years it was shown that the theoretical framework of secure two-party computation can be efficiently implemented and can be run in reasonable time, even under the strongest security guarantees (see, e.g. [PSSW09, SS11, NNOB12, KSS12]).

Non-interactive secure computation (NISC). A major drawback of many 2PC protocols is that they require several rounds of interaction (e.g., [LP07, LP11] with a constant number of rounds, or [NNOB12] with a number of rounds that depends on the function). This paper focuses on efficient constructions of protocols for non-interactive secure computation (NISC) that run in a single round of interaction.

We consider three flavors of NISC. In the first, which we refer to by *One-Sender NISC (OS-NISC)*, there are only two parties, a receiver and a sender. The receiver sends the first message, the sender replies

⋆ Supported by the Israeli Ministry of Science and Technology (grant 3-9094).
⋆⋆ Supported by the Check Point Institute for Information Security and an ISF grant 20006317.

with the second message, and then the receiver outputs the result of the computation. This is essentially a 2PC protocol with the additional restriction of having only one round of interaction. (Following [IKO+11], throughout this work we refer to the party that sends the first message and receives the final output as the *receiver* or as P_1, and refer to the party that sends the second message as the *sender* or P_2.)

The second flavor of NISC, which we call *Multi-Sender NISC (MS-NISC)*, is an extension of OS-NISC where the first message can be used for running secure computation with many different senders. I.e., the receiver broadcasts its first (single) message; each party that wants to participate in a secure computation with the receiver sends a message back to the receiver; then, after receiving second messages from several (possibly different) senders, the receiver outputs the results of its computation with all thee senders (or uses these output values in other protocols). We stress that each sender does not trust other senders, nor the receiver, and wishes to maintain privacy of its input even if everyone else colludes.

A limitation of MS-NISC is that the receiver has to aggregate and output all the secure computation results together. The last flavor of NISC, which we call *Adaptive MS-NISC*, does not have this limitation. Adaptive MS-NISC is essentially like MS-NISC, except that the receiver outputs each of the secure computation results as soon as it gets it (thus, allowing the adversary, who might control some senders, to pick its next inputs based on those results).

In this work we focus on the first two flavors, and only briefly discuss the third flavor where relevant.

Why NISC? Let us begin with a motivating example. Suppose that there is a known algorithm that receives the DNA data of two individuals and decides whether they are related. People would like to use this algorithm to find family relatives, but on the other hand they are not willing to publish their DNA data (which can, e.g., predict their chances of being affected by different diseases). A possible solution is to use a secure computation that implements the algorithm and is run between any pair of people who suspect that they might be related. A multi-round protocol for secure computation requires the participants to coordinate a time where they can both participate in the protocol, and run a secure computation application that exchanges multiple rounds of communication with the application run by the other party. A solution using NISC is much simpler and eliminates the synchronization problem: each interested person can publish, say on his Facebook wall, his first message in the protocol, secretly encoding his DNA data. Those who are interested in finding out whether they are related to that person can send back the second message of the protocol. This message can be sent using Facebook or similar services, or even by email. Then, once in a while, the first person can run the computation with all those who answered him, and find out with whom he is related.

In the previous example, NISC was preferable since a multi-round protocol would have required the parties to synchronize the times in which they participate in the protocol (or incur long delays until the other party is online and sends the next message of the protocol). In general, requiring multiple rounds of interaction is also very limiting in scenarios in which each round of communication is very expensive and/or is slow. E.g., if the communication is done using physical means, for example encoded as a QR code on a brochure sent by snail-mail, or if the other party is a satellite that passes for only a short period above the receiver.

Previous NISC protocols. A NISC protocol (for all three flavors) for general computation can be constructed from Yao's garbled circuit, non-interactive zero-knowledge proofs (NIZK), and fully-secure one-round oblivious transfer (OT): P_1, who is the evaluator of the circuit, sends the first message of the OT protocol. P_2, who is the circuit constructor, returns a garbled circuit, the second message of the OT protocol, and a NIZK proof that its message is correct. (See, for example, [CCKM00, HK07] for such protocols.) Unfortunately, the NIZK proof in this case requires a *non black-box* use of cryptographic primitives (namely, it must prove the correctness of each encryption in each gate of the circuit).

Efficient NISC protocols that do not require such non black-box constructions are presented in [IKO+11] based on the MPC-in-the-head technique of [IPS08]. The complexity of the OS-NISC protocol of [IKO+11] is $|C| \cdot \mathsf{poly}(\log(|C|), \log(t)) + \mathrm{depth}(C) \cdot \mathsf{poly}(\log(|C|), t)$ invocations of a Pseudo-Random Generator (PRG), where C is a boolean circuit that computes the function of interest, and t is a statistical security parameter. (Another protocol presented in that work uses only $\mathcal{O}(|C|)$ PRG invocations, but is based on a relaxed security notion.) [IKO+11] also shows an adaptive MS-NISC protocol for a bounded number of corrupted

senders. The complexity of that protocol is $\mathcal{O}((t+Q)|C|)$ PRG invocations, where Q is the bound on the number of corrupted senders.

Although the protocols in [IKO+11] are very efficient asymptotically, their practicality is unclear and left as an open question in [IKO+11]. For instance, the protocols combine several techniques that are very efficient asymptotically, such as scalable MPC and using expanders in a non black-box way, each of which contributes large constant factors to the concrete complexity.

Cut and Choose based 2PC. A very efficient approach for constructing 2PC with security against malicious parties is based on the *cut-and-choose* paradigm. (We refer here to protocols that use cut-and-choose for checking garbled circuits, as in [LP07], and not to protocols that use cut-and-choose in a different way, such as the protocols in [IKO+11].) [MF06, LP07, LP11, SS11, MR13, Lin13, sS13] give constructions that use this paradigm and require $\mathcal{O}(t|C|)$ PRG invocations, and some additional overhead that does not depend on $|C|$. Indeed, for a fixed circuit, this asymptotic overhead is larger than that of [IKO+11], which requires only a poly-logarithmic number of PRG calls per gate of the circuit. However, the concrete constants in the cut-and-choose based protocols are rather small (whereas for [IKO+11] the constants seem fairly large, e.g., the $\mathsf{poly}(\log(|C|))$ factor) making the cut and choose approach of high practical interest as shown in several implementations (e.g., [PSSW09, SS11, KSS12]). However, all current cut-and-choose based 2PC constructions require more than one round of interaction.

1.1 Our Contributions

In this paper, we take a major step beyond feasibility results for NISC. Our main contribution is a new OS-NISC/MS-NISC protocol that we believe to be conceptually simpler than previous NISC protocols, and extremely practical. The complexity of this protocol is similar or better than those of the best multi-round 2PC protocols based on cut-and-choose. We also describe an implementation and evaluation of our NISC protocol, that demonstrate its practicality.

We now discuss our contributions in more detail.

Revisiting the NISC setting. In Section 3 we formalize the informal description of the MS-NISC model by using the ideal/real-model paradigm, defining an ideal functionality that receives an input from the receiver and inputs from many other senders, and returns to the receiver the outputs of the different evaluations.

Intuitively, one would expect that any OS-NISC protocol can also be a MS-NISC protocol with soundness that decreases at most polynomially in the number of senders. In the full version of this paper we show that this intuition is false by describing an attack on the technique of [LP07] for protecting against selective-OT attacks, which results in an exponential (in the number of senders) decrease in the soundness of the protocol.[4]

Our protocols. As discussed earlier, the cut-and-choose technique requires several rounds of interaction since the player who generates the garbled circuits must first send them, and only then see the "cut" and send the circuit openings. We introduce techniques that allow us to *squash this interaction to a single round* in the common random string model (CRS). Until recently, all cut-and-choose based 2PC protocols (e.g., [LP07, LP11, SS11] required at least $\sim 3t$ garbled circuits for achieving soundness of 2^{-t} (ignoring computational soundness). These techniques are sufficient to turn such protocols into NISCs that also use roughly $3t$ garbled circuits.

Reducing the number of circuits. Lindell [Lin13] recently introduced a cut-and-choose based 2PC that requires only t garbled circuits for the same soundness, reducing the number of garbled circuits by (at least) a factor of three. However, this protocol is inherently interactive since it executes two 2PC protocols, one after the other, where the second 2PC is used to recover from potential cheating, with no obvious way of making the protocol non-interactive. We show a new approach that allows working non-interactively with only t garbled circuits (for soundness 2^{-t}). We believe that our approach has significance also in the multi-round setting with several advantages over the techniques of [Lin13] such as (1) suitability for pipelining; and (2) an (arguably) conceptually simpler description.

[4] We note that in the OS-NISC protocol of [IKO+11], a variant of the [LP07] technique is used for protecting against the selective-OT attack. As far as we can tell, our "attack" can be applied to that construction as well, if used for MS-NISC.

Section 1.2 provides a high-level description of the protocol. This protocol is secure under the DDH assumption in the CRS model. We believe that this protocol is easier to understand than previous NISC protocols, and because of that, more approachable for people from outside the crypto community. Hopefully, NISC could gain interest as a model for practical protocols and applications.

We remark that we achieve only the OS-NISC/MS-NISC security notions. The same first message *can* be used for many executions of secure computation with many different senders. The only restriction to achieve adaptive MS-NISC is that once the receiver's outputs are revealed to the other parties, the receiver must refresh its first message, which requires computing only t OT queries.

In the full version of this paper we describe how the efficiency of the protocol can be improved if one permits more than one round of interaction. The resulting 2PC protocol requires only t garbled circuits (for statistical security of 2^{-t}), $\mathcal{O}(tn_1)$ symmetric-key operations, and $\mathcal{O}(tn_2 + t^2)$ exponentiations, where n_i is P_i's input length (and ignoring a small number of seed-OTs).

Reducing communication. In addition to the main protocol, we show how to reduce communication significantly using a new non-interactive adaptation of the method of Goyal et al. [GMS08] to the NISC environment (Section 5). This method, based on the usage of erasure codes (specifically, of polynomials), reduces the communication size to be only slightly higher than the communication required for sending the garbled circuits that are evaluated (as opposed to sending also the garbled circuits that are checked). For example, for soundness 2^{-40}, this protocol requires using 44 garbled circuits, and communicating only 19 garbled circuits.

Implementation and experiments. We describe a prototype implementation of our main protocol, implemented in C for a Linux environment. It is the *first* working implementation (that we are aware of) of a NISC protocol, and it allows using our protocol in all the scenarios described above. Additionally, this is also the first working implementation (that we are aware of) of a 2PC protocol that uses only t garbled circuits for security of 2^{-t}.

We evaluate the prototype with a circuit that computes an AES encryption and a circuit that computes SHA256. The resulting performance is significantly better than that of previous cut-and-choose based protocols. For example, a maliciously secure computation of AES circuit requires about 7 seconds , where the time needed for generating the first message is very small (e.g., much less than a second).

1.2 High Level Description of the Protocol

Step One: Squashing Cut-and-Choose 2PC to One Round. The starting point for the protocol is the most straightforward approach based on the cut-and-choose method with $3t$ garbled circuits. (The constant 3 is chosen for simplifying the description. The exact constants are analysed in [LP11, SS11].) The receiver's first message in this case is an OT query of its input using a two-message OT protocol (e.g., [PVW08]). Namely, if the receiver has n_1 input bits it sends the corresponding n_1 OT queries. The sender garbles $3t$ circuits $gc_1, \ldots, gc_{3t}$ and sends back a message that includes: (1) The $3t$ garbled circuits; (2) The OT answers for the receiver's query, using the input-wire labels that were used for garbling the receiver's inputs; (3) The input-wire labels that correspond to the sender's own input. The receiver is now able to retrieve the labels of its input-wires and evaluate the $3t$ garbled circuits by itself. It then takes the majority result to be its output. This protocol is obviously insufficient. There are three issues that need to be verified: (1) Were the garbled circuits garbled correctly? (2) Did the sender use the right input-wire labels in the OT? (i.e., consistent with the garbled circuits) (3) Was the sender's input consistent in all $3t$ circuits? The goal of our work is to present non-interactive and efficient solutions for these issues.

The standard solution for the first issue, of verifying the garbled circuits, is the cut-and-choose method [LP07] where the sender proves that a random subset of $c \cdot 3t$ circuits (where c is fixed and publicly known, e.g. $c = 1/2$, or $c = 3/5$ to optimize the success probability) were garbled correctly by revealing the randomness that was used to garble them. Normally, the cut-and-choose method requires more than one round of interaction. We solve this problem by using OT in the following way (similar to the technique used in [KSS12, KMR12] for the different purpose of reducing latency). The protocol includes additional $3t$ OTs, denoted as the *circuit-OTs*. In each of these OTs the receiver can choose to either check or evaluate the

corresponding circuit: The receiver chooses a random subset of circuits of size $c \cdot 3t$ that it wants to check, and for each of these circuit it sends an OT query for the 1-bit. For the rest of the circuits it sends an OT query for the 0-bit. The sender picks $3t$ keys $seed_1, \ldots, seed_{3t}$ for a pseudo-random function (PRF) and uses key $seed_i$ to generate all the randomness needed for garbling gc_i. The sender also picks additional $3t$ keys $k_1, \ldots, k_{3t}$, and encrypts, under the key k_i, the labels of the sender's input-wires for circuit gc_i. Now, the sender answers the circuit-OT queries using the $3t$ pairs $(k_i, seed_i)$ as inputs. Observe that if the receiver wants to check gc_i it learns the PRF key $seed_i$ that allows to reconstruct that circuit (using the same circuit construction algorithm used by the sender), but it is not able to decrypt the sender's input-wires labels. If the receiver wishes to evaluate circuit gc_i it learns the key k_i that enables to decrypt the input-wires labels of that circuit, but not the seed $seed_i$. In that case the receiver is able to evaluate the circuit but not to check it. Of course, the sender does not know which circuits are chosen to be checked, due to the security of the OT protocol.

As for the second issue, how to check that the sender uses consistent labels in the OTs for the receiver's input wires, we modify a technique of [KS06, SS11] to work in the NISC setting. Instead of using a regular OT protocol, we work with an OT in which the second OT message *commits* the sender to specific inputs. (I.e., given the second OT message, the sender cannot later claim that it used different inputs than the ones it actually used.) In practice, the highly efficient OT of [PVW08] is sufficient for our purpose. Since we have only one round of interaction, we require that all the randomness used for the second message of the OT queries for circuit gc_i, is also derived from the PRF key $seed_i$. In case the receiver does not ask to check gc_i, this OT is as secure as a regular OT by the security of the PRF. If the receiver chose to check gc_i, it learns $seed_i$, and since it knows both the input labels of the circuit and the randomness that should have been used in the OT it is able to recompute the second OT message by itself and compare it with the message sent by the receiver. If there is a difference, the receiver aborts, since this means that the sender tried to cheat in the OT for gc_i.

For the third issue, i.e. the consistency of the sender's inputs, we modify a technique of [MF06] for the NISC setting. We use a commitment scheme that allows proving, very efficiently, that two commitments are commitments to the same value. (Pedersen's commitment [Ped92] or an ElGamal based commitment suffice.) Instead of using random labels for the sender's input-wires, the sender uses commitments to zero as labels for the 0-bit inputs and commitments to one as labels for 1-bit inputs. In an interactive setting the sender decommits all input-wire labels of the checked circuits and proves that it used correct commitments.

In order to execute the protocol in a single round of interaction, we require that the randomness used for the commitments for the input wires of circuit gc_i is also generated using the seed $seed_i$. This allows the receiver to regenerate the commitments by itself in case it chose to check gc_i. In addition, the sender sends what we call *input commitments*, which are a set of commitments of its actual input bits that is not part of any garbled circuit. The protocol includes commitment equality proofs which prove that each input value in an evaluated circuit is equal to the value committed in the corresponding input commitment. (These proofs are secure since the input commitments are never decommitted, as opposed to the other commitments which are opened in checked circuits). The sender encrypts the commitment equality proofs using k_i in order to hide them from the receiver in the checked circuits. (Otherwise, the receiver could determine the sender's input.)

Note that so far our protocol requires $3t$ garbled circuits and relies on the cut-and-choose guarantee that the majority of the evaluated garbled circuits are correct.

Before we discuss how to reduce the number of garbled circuits, we note that although our protocol is not vulnerable to selective-OT attacks, namely attacks where the sender sets incorrect inputs in the OTs used by the receiver to learn its input labels, we still require the receiver to refresh its first message in case its outputs are revealed to the sender (or are used in other protocols, which can potentially leak them). Technically, this happens since a corrupted sender can use an invalid seed for garbled circuit gc_1, and valid circuits otherwise. This sender could then learn the receiver's first input bit in the circuit-OTs, based on whether the receiver aborted its execution with this sender. In the adaptive MS-NISC setting, this attack could be repeated by several corrupted senders, letting the adversary learn secret information about other bits of the cut-and-choose challenge. As a result, soundness is gone, since the adversary could set the input of

the last sender based on the bits of the cut-and-choose challenge. In order to mitigate this attack, we require the receiver to refresh its first message once its outputs are revealed. Note, however, that some information about the receiver's choices in the circuit-OTs is indeed revealed even if the receiver does refresh its first message. However, these bits are revealed only *after* the execution of the protocol, thus do not undermine security. (In fact, in most cut-and-choose 2PC protocols the challenge is always public. E.g., [LP11, SS11].)

Step Two: Reducing the Number of Garbled Circuits. Assume for simplicity that the circuit the players use has only one output wire, and that the sender has only one input bit. We use the protocol from the previous section, but with only t garbled circuits, and let P_1 pick a random subset of them for verification (instead of a constant fraction c, as described above). Obviously, if all evaluated circuits output the same bit, then this bit is the correct output with probability $1 - 2^{-t}$ (since in order to cheat, the sender must guess *all* the checked circuits and *all* the evaluated ones). However, if some of the evaluated circuits output different bits, then the receiver knows that the sender is trying to cheat and needs to determine the right output. Following [Lin13], we would like to provide the receiver in this case with a "trapdoor" that allows it to recover the sender's input in case the sender behaves maliciously (but, of course, not in case it behaves honestly). Then, the receiver can simply use the sender's input in order to compute the function by itself, and output the correct result.

As described earlier, the sender's input-wire labels are commitments to their actual values. Let $\mathsf{EGCommit}(h; b, r) = (g^r, h^r g^b)$ be an ElGamal based commitment for a bit b, given a group $\mathcal{G}$ in which DDH is hard, and a generator g. This is a perfectly-binding commitment, even if the party that commits knows $\log_g(h)$. However, knowing $\log_g(h)$ allows "decrypting" g^b, which otherwise is hidden because of the DDH assumption.

In the protocol, the sender picks w, sends $h = g^w$ to the receiver, and sets the labels of its input wire in gc_i to be $\mathsf{EGCommit}(h; 0, r_{i,0})$ and $\mathsf{EGCommit}(h; 1, r_{i,1})$. Next, the sender picks at random w_0, w_1 such that $w = w_0 + w_1$, and sends $h_0 = g^{w_0}$ and $h_1 = g^{w_1}$. (P_1 verifies that $h = h_0 \cdot h_1$.) For gc_i, the sender sends *output recovery commitments* $h_0 g^{l_{i,0}}$ and $h_1 g^{l_{i,1}}$, where $l_{i,0}, l_{i,1}$ are chosen at random.[5] Then, it sets the output wire labels of this circuit to be $l_{i,0}$ and $l_{i,1}$, corresponding to 0 and 1, respectively.

As part of the cut-and-choose stage, if the receiver chooses to check gc_i, then it learns $seed_i$ and can recover the output wire labels and verify both the input-wire labels and the output recovery commitments. However, if the receiver chooses to evaluate gc_i, then the sender also sends it the values $w_0 + l_{i,0}$ and $w_1 + l_{i,1}$. (These values are sent encrypted under k_i, so the receiver only gets them in case it chose to evaluate gc_i.) The receiver verifies that these values are consistent with the output recovery commitments by computing g to the power of these two values (if this verification fails then the receiver aborts). In addition, the receiver checks that the $l_{i,b}$ it received from the evaluation of gc_i is a valid decommitment of $h_0 g^{l_{i,b}}$. If this check pass, the receiver marks gc_i as a *semi-trusted* circuit. (Note that the probability of marking no circuit as semi-trusted is 2^{-t}, as it requires the sender to guess the set of evaluated circuits.)

After the receiver evaluates all the circuits chosen for evaluation, it is left with either a single output from all semi-trusted circuits, or with two outputs from at least two semi-trusted circuits. In the first case, since with probability 2^{-t} there is at least one good evaluated garbled circuit, that single output is the correct one. In case there are two different outputs, the receiver initiates the *cheating recovery process*: Say that gc_i's output is 0 and $gc_{i'}$'s output is 1 (and both are semi-trusted). From evaluating gc_i, the receiver learns $l_{i,0}$, and from the sender's message, it learns $w_0 + l_{i,0}$. Thus, it can recover w_0. Similarly, from $gc_{i'}$ it recovers w_1. Having $w = w_0 + w_1$ allows the receiver to decrypt the input-commitments, and recover the sender's input as needed. Note that in case the sender is honest, the receiver would get the same output from all evaluated circuits, and thus would learn only one of w_0 and w_1.

When there are more than one output wire, different w_0, w_1 are chosen for each output wire, thus the receiver learns one value from each pair. See Section 4 for a detailed description of the protocol.

[5] Clearly, since P_2 knows w_0, w_1, $h_0 g^{l_{i,0}}$ does not bind P_2 to h_0. Rather, it binds P_2 to $w_0 + l_{i,0}$.

2 Preliminaries: Notations and Primitives

Let $\mathsf{Hash}(\cdot)$ be a collision resistant hash function, $\mathsf{REHash}(\cdot)$ be a collision-resistant hash function that is a suitable randomness extractor (e.g., see [DGH+04]), $\mathsf{Commit}(\cdot)$ be a commitment scheme, and let $\mathsf{Enc}(k, m)$ be the symmetric encryption of message m under key k.

Garbled Circuits. Our protocol is based on the garbled circuit protocol of Yao [Yao86] and can work with any garbling scheme (see [LP09, BHR12] and the full version of this paper more details). We only require that the labels of the output-wires reveal the actual outputs of the circuit (but still consist of random strings). We use the notation $\mathsf{label}(gc, j, b)$ to denote the label of wire j corresponding to bit value b in the garbled circuit

Commitments with efficient proof-of-equality and trapdoor. We use a commitment scheme that allows one to efficiently prove that two commitments are for the same bit, without revealing any information about the committed bit. Also, we require the commitment scheme to have a "trapdoor" that allows extracting the committed value.

A commitment that satisfies our requirement can be based on ElGamal. Given finite group $\mathcal{G}$ and a generator g, the committer picks a random element $h \in \mathcal{G}$, and sends $\mathsf{EGCommit}(h, m, r) = (g^r, h^r g^m)$. This commitment is computationally-hiding (under the DDH assumption) and perfectly-binding. Given $\mathsf{EGCommit}(h, m, r)$ and $\mathsf{EGCommit}(h, m, r')$, the commiter can prove equality by giving $r - r'$. Last, given the "trapdoor" $\log_g(h)$, one can decrypt the commitment, $\mathsf{EGCommit}(h, m, r)$, and recover m.

Batch committing-OT. Batch committing-OT protocol is an OT protocol where the sender has two tuples of inputs $[K_1^0, K_2^0, \ldots, K_t^0]$, $[K_1^1, K_2^1, \ldots, K_t^1]$. The receiver has a bit b and wishes to learn the tuple $[K_1^b, K_2^b, \ldots, K_t^b]$.

We use a variant of the batch committing-OT protocol of [SS11] (which is based on the highly efficient one-round, UC-secure OT of [PVW08]). The protocol is secure under the DDH assumption. Let $\mathcal{G}$ be a group of prime order p in which the DDH assumption is assumed to hold, and let (g_0, g_1, h_0, h_1) be a common reference string (CRS) where g_0, g_1, h_0, h_1 are random elements in $\mathcal{G}$. The receiver picks $r \in \mathbb{Z}_p$ at random and sends $g = (g_b)^r, h = (h_b)^r$ to the sender. For $i = 1 \ldots t$ and $b' \in \{0, 1\}$, the sender picks at random $r_{i,b'}, s_{i,b'} \in \mathbb{Z}_p$ and sends $X_{i,b'} = g_{b'}^{r_{i,b'}} h_{b'}^{s_{i,b'}}$ and $Y_{i,b'} = g^{r_{i,b'}} h^{s_{i,b'}} K_i^{b'}$. For $i = 1 \ldots t$, the receiver retrieves $K_i^b = Y_{i,b} / X_{i,b}^r$.

After executing the above protocol, if the receiver asks the sender to reveal both its inputs K_i^0, K_i^1 for some i, the sender returns the values K_i^0, K_i^1, $r_{i,0}, s_{i,0}$, $r_{i,1}, s_{i,1}$ and the receiver verifies that the values $X_{i,0}, Y_{i,0}$, $X_{i,1}, Y_{i,1}$ that it received were properly constructed using these values.

For simplicity and generality, in our NISC protocols we denote by $\mathsf{COT}_1(b)$ the first message that is sent (from the receiver to the sender) in an invocation of the committing-OT protocol for the receiver's input bit b, and similarly denote the second message (that is sent from the sender to the receiver) by $\mathsf{COT}_2([K_1^0, K_2^0, \ldots, K_t^0], [K_1^1, K_2^1, \ldots, K_t^1], \mathsf{COT}_1(b))$.

In the full version of this paper we give further details about the security of this protocol, and discuss the CRS in case there are many invocations of MS-NISC with different senders.

3 The NISC Model

The OS-NISC notion is essentially like 2PC with one round of interaction, thus the security definition is exactly as for multi-round 2PC (e.g., [Gol04]), with the additional restriction on the number of rounds in the real execution.

For defining MS-NISC, we use the ideal/real paradigm (in the standalone setting), and use the ideal functionality from Figure 1. See the full version of this paper for a formal definition. Throughout this work we assume that senders cannot see or tamper with other senders' messages to avoid malleability concerns. In the full version of this paper we discuss how to correctly encrypt those messages if this is not the case. (Note that in many applications there is only one sender and then malleability is not an issue. E.g., if the sender is a satellite that sends messages periodically. In this case there is only one sender that sends many messages, and no malleability issues occur.)

Assume that $f(\perp,\cdot) = f(\cdot,\perp) = \perp$.

- Initialize a list L of pairs of strings.
- Upon receiving a message (input, x) from P_1, store x and continue as following:
  - Upon receiving a message (input, y) from P_i, insert the pair (P_i, y) to L. If P_1 is corrupted, send $(P_i, f(x,y))$ to the adversary. Else, send $(\text{messageReceived}, P_i)$ to P_1.
  - Upon receiving a message (getOutputs) from P_1, send $\left(\left\{(P_i, f(x,y))\right\}_{(P_i,y)\in L}\right)$ to P_1, and halt.

Fig. 1. MS-NISC functionality $\mathcal{F}$.

4 An OS-NISC/MS-NISC Protocol

The protocol is in the CRS (common reference string) model, which is a necessary requirement for the one-round OT protocol that we use [PVW08]. (Unlike other results that are presented in the OT-hybrid model, we use this specific OT protocol which is currently the most efficient fully-secure, simulation-proven OT. We preferred to use a concrete instantiation of OT in order to be able to use a committing variant of OT, in which the OT sender is committed to its OT inputs after it sends its OT message. Still, our techniques can be used with any committing-OT protocol that is proved secure using simulation and that can be executed concurrently without sacrificing security.) Since the nature of NISC is mostly for indirect communication (e.g, using a Facebook wall), we favor a solution that has a minimal communication overhead.

For high level description of the protocol, we refer the reader to Section 1.2. The detailed protocol is described in Figures 2-4. Its concrete efficiency analysis and proof of the following theorem are in the full version of this paper.

Theorem 1. *Assume that the Decisional Diffie-Hellman problem is hard in the group $\mathcal{G}$ and that* PRF*,* REHash*,* Commit *and* Enc *are secure. Then, the protocol of Figures 2-4 is a multi-sender non-interactive secure computation for any function $f : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m$ computable in polynomial time. The complexity of the protocol is $\mathcal{O}(t(n_1+n_2+m))$ expensive operations and $\mathcal{O}(t(n_1+n_2+m+|C|))$ inexpensive operations.*

The protocol is described for a single sender. When there are more senders (or one with several inputs), each sender executes the steps that are described below for P_2.

Preliminaries: As defined in Section 2, we denote by $\mathsf{COT}_1()$ the first message sent in an invocation of the committing-OT protocol, and denote the second message of that protocol as $\mathsf{COT}_2()$. Also, denote by $\mathsf{EGCommit}(h; b, r)$ the ElGamal commitment (which supports an efficient proof-of-equality) to bit b. Let $\mathcal{G}$ be a group of size p with generator g.

Inputs: P_1 has input x and P_2 has input y. Let $f : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m$ be the function of interest and let $C(x,y)$ be a circuit that computes f. The input wires of P_1 and P_2 are denoted by the sets $\text{IN}_c(1)$ and $\text{IN}_c(2)$, respectively. The output wires are denoted by the set OUT_c.

P_1's message:

- Picks a random t-bit string where t_i denotes the i-th bit of this string. We define T such that $i \in T$ if and only if $t_i = 1$.
- For all circuits $i \in [t]$ publishes $\mathsf{COT}_1(t_i)$. Denote these as the *circuit-OT* queries.
- For all inputs $j \in \text{IN}_c(1)$ publishes $\mathsf{COT}_1(x_j)$, where x_j is P_1's input bit for the j-th input wire. Denote these as the *input-OT* queries.

Fig. 2. The OS-NISC/MS-NISC Protocol: Preliminaries and P_1's message.

– Picks $w \in_R \mathbb{Z}_p$ and sends $h = g^w$. Here, w would be the "trapdoor" to P_2's inputs.
– Sends $\mathsf{EGCommit}(h; y_j, r_j)$, for all $j \in \mathrm{IN_c}(2)$, where y_j is its input bit for input-wire j, and r_j is chosen randomly. We call these the *input-commitments*.
– Sends $h_{j,0} = g^{w_{j,0}}$ and $h_{j,1} = g^{w_{j,1}}$, where $w_{j,0} \in_R \mathbb{Z}_p$ and $w_{j,1} = w - w_{j,0}$, for all output wires $j \in \mathrm{OUT_c}$. We call these the *output-commitments*.

For all $i \in [t]$,

Generate garbled circuit:

- Picks a random value $seed_i$.
- Computes $u_{i,j,b} = \mathsf{EGCommit}(h; b, r_{i,j,b})$ for all $j \in \mathrm{IN_c}(2)$ and $b \in \{0,1\}$, where $r_{i,j,b} = \mathsf{PRF}_{seed_i}(\text{“}EGCommitment\text{”} \circ j \circ b)$.
- Sends the garbled circuit gc_i, which is generated using a pseudo-random function PRF_{seed_i} in the following way:
  - For all $j \in \mathrm{IN_c}(2)$ and $b \in \{0,1\}$, let $\mathsf{label}(gc_i, j, b) = \mathsf{REHash}(u_{i,j,b})$. Namely, the label for bit b of the jth wire is associated with the value of $\mathsf{EGCommit}(h; b, \cdot)$ computed with randomness that is the output of a PRF keyed by $seed_i$. Note that given $u_{i,j,b}$, P_1 can compute $\mathsf{REHash}(u_{i,j,b})$ by itself and get the corresponding label.
  - The garbled circuit is constructed in a standard way, where all other labels in the circuit are generated by a PRF keyed by $seed_i$. (E.g., the 0-label of wire j is $\mathsf{PRF}_{seed_i}(\text{“}label\text{”} \circ j \circ 0)$.)
- Sends the set of commitments $\left\{[\mathsf{Commit}(u_{i,j,\pi_{i,j}}), \mathsf{Commit}(u_{i,j,1-\pi_{i,j}})] \mid \pi_{i,j} \in_R \{0,1\}\right\}_{j \in \mathrm{IN_c}(2)}$. The randomness of the commitments is derived from a PRF keyed by $seed_i$ as well. Denote by $du_{i,j,b}$ the decommitment of $u_{i,j,b}$.

Preparing and sending the cheating recovery box:

Sends the cheating recovery box, for all output wires $j \in \mathrm{OUT_c}$, which includes:

- Two *output recovery commitments* $h_{j,0}g^{K_{i,j,0}}, h_{j,1}g^{K_{i,j,1}}$, where $K_{i,j,0}, K_{i,j,1} \in_R \mathbb{Z}_p$.
- Two encryptions $\mathsf{Enc}(\mathsf{label}(gc_i, j, 0), K_{i,j,0}), \mathsf{Enc}(\mathsf{label}(gc_i, j, 1), K_{i,j,1})$. (Note that given $\mathsf{label}(gc_i, j, b)$, one can recompute $h_{j,0}g^{K_{i,j,b}}$.)

Preparing and sending proofs of consistency:

- Let inputs_i be the set $\left\{u_{i,j,y_j}, du_{i,j,y_j}\right\}_{j \in \mathrm{IN_c}(2)}$, and let $\mathsf{inputsEquality}_i$ be the set $\left\{r_j - r_{i,j,y_j}\right\}_{j \in \mathrm{IN_c}(2)}$ (namely, P_2's input labels and their proof of equality with the input-commitments).
- Let $\mathsf{outputDecom}_i$ be the set $\left\{([w_{j,0} + K_{i,j,0}], [w_{j,1} + K_{i,j,1}])\right\}_{j \in \mathrm{OUT_c}}$ (namely, the discrete logarithms of $h_{j,0}g^{K_{i,j,0}}$ and $h_{j,0}g^{K_{i,j,1}}$).
- Picks a random key k_i and sends the encryption $\mathsf{Enc}(k_i, \mathsf{inputs}_i \circ \mathsf{inputsEquality}_i \circ \mathsf{outputDecom}_i)$.

Sending the garbled values of P_1's inputs:

Let $inp\text{-}q_j$ be the input-OT query for input-wire j of P_1. P_2 sends the OT answer, which includes the garbled values of either the 0 or 1 labels for the corresponding input wire. Namely, it sends the value $\mathsf{COT}_2([\mathsf{label}(gc_1, j, 0), \ldots, \mathsf{label}(gc_t, j, 0)], [\mathsf{label}(gc_1, j, 1), \ldots, \mathsf{label}(gc_t, j, 1)], inp\text{-}q_j)$. Moreover, we require that all the randomness used in the OT for the answers of the i-th circuit is generated from PRF_{seed_i}. (E.g., set $r_{i,1}$ of the j-th wire of the i-th circuit to be $\mathsf{PRF}_{seed_i}(\text{“}OT\text{”} \circ 1 \circ \text{“}r\text{”} \circ i \circ j)$.)

Circuits cut-and-choose:

Let $circ\text{-}q_i$ be the circuit-OT query for circuit i, P_2 sends $\mathsf{COT}_2([k_i], [seed_i], circ\text{-}q_i)$. Namely P_1 receives $seed_i$ if it asked to open this circuit, and k_i if it is about to evaluate the circuit.

Fig. 3. The OS-NISC/MS-NISC Protocol: P_2's response.

After receiving responses from all senders, P_1 processes all of them together and outputs a vector of outputs. For each response it does the following:

- Decrypts all OT answers.
- Verifies that $h_{j,0} \cdot h_{j,1} = h$ for all $j \in \mathrm{OUT_c}$.
- For all opened circuits $i \in T$, checks that $seed_i$ indeed correctly generates gc_i (with its commitments), and the answers of the input-OT queries. (Otherwise, it aborts processing this response.) It also checks the cheating recovery boxes and aborts if there is a problem.
- For all circuits $i \in [t] \setminus T$, decrypts $\mathsf{inputs}_i, \mathsf{inputsEquality}_i, \mathsf{outputDecom}_i$.
  - Checks that inputs_i and $\mathsf{inputsEquality}_i$ are consistent with the input-commitments. (I.e., checks that $u_{i,j,y_j} \cdot (g^{r_i - r_{i,j,y_j}}, h^{r_i - r_{i,j,y_j}}) = \mathsf{EGCommit}(h; y_j, r_j)$). Also, verifies the decommitments du_{i,j,y_j}. (Otherwise, it aborts.)
  - Checks that $\mathsf{outputDecom}_i$ are correct discrete-logs of the elements of the set $\left\{h_{j,b} g^{K_{i,j,b}}\right\}_{j \in \mathrm{OUT_c}, b \in \{0,1\}}$. (Otherwise, it aborts.)
  - Evaluates circuit gc_i. Say that it learns the labels $\{l_{i,j}\}_{j \in \mathrm{OUT_c}}$. P_1 tries to use these labels to decrypt the corresponding encryptions $\mathsf{Enc}(\mathsf{label}(gc_i, j, b), K_{i,j,b})$ from the cheating recovery box. Then, it checks if the result is a correct "decommitment" of the output recovery commitment $h_{j,b} g^{K_{i,j,b}}$ (where the b values are the actual output bits it received from gc_i). If all these steps pass correctly for all output wires, we say that circuit gc_i is *semi-trusted*.
- If the outputs of all semi-trusted circuits are the same, P_1 outputs that output.
  Otherwise,
  - Let $gc_i, gc_{i'}$ be two semi-trusted circuits that have different output in the jth output wire, and let $l_{i,j}$ and $l_{i',j}$ be their output labels. From one of $l_{i,j}$ and $l_{i',j}$, P_1 learns $w_{j,0}$ and from the other value it learns $w_{j,1}$ (since it learns $K_{i,j,b}, K_{i',j,1-b}$ from the cheating recovery boxes, and $w_{j,b} + K_{i,j,b}, w_{j,1-b} + K_{i',j,1-b}$ from $\mathsf{outputDecom}_i, \mathsf{outputDecom}_{i'}$).
  - P_1 computes $w = w_{j,0} + w_{j,1}$ and decrypts P_2's input-commitments. Let y be the decrypted value of P_2's input.
  - P_1 outputs $f(x, y)$.

Fig. 4. The OS-NISC/MS-NISC Protocol: P_1's computation.

5 Reducing the Communication Overhead

Goyal et al. [GMS08] suggest a method that significantly reduces the communication overhead of 2PC protocols based on cut-and-choose. In their protocol, as in ours, P_2 picks a different seed for each garbled circuit and uses a pseudo-random function, keyed with that seed, to generate all the randomness needed for garbling that circuit. P_2 does not send the circuits to P_1 but only "commits" to them by sending the hash of each circuit. Then, when P_2 is asked to open a subset of the circuits, it sends to P_1 the seeds used for constructing these circuits, as well as the actual garbled tables of the evaluated circuits. P_1 uses the seeds to reconstruct the checked circuits and verify that they agree with the desired functionality and with the hashes that were sent in the initial step (the hashes are computed with a collision resistant hash function $\mathsf{Hash}(\cdot)$ and therefore prevent a circuit from being changed after its hash is received).

Trying to apply this modification in the NISC setting encounters a major obstacle: In order for P_2 to send only the gates of the evaluated circuits, it must learn, based on P_1's first message, which circuits are evaluated. Since P_2 learns this information before it sends any message to P_1, it is able to set its evaluated and checked circuits in a way that fools P_1's checks.

When communication is through a third-party service. A simple solution can be based on the observation that in many applications of NISC the communication channel is actually implemented through a third-party service, e.g., a Facebook wall. In those cases, P_2 could upload *all* circuits to the service, along with their hash values. Then, P_1 downloads only the circuits for evaluation and the hashes of all circuits. Assuming that the service hides from P_2 which circuits were actually downloaded by P_1, the result is secure, and the communication of P_1 and of the service (but not of P_2) depends only on the number of evaluated circuits.

A more general solution. We describe a solution that does not depend on any third party. The solution requires that the number of evaluated circuits is known to P_2 (e.g., for soundness 2^{-40} the players can use 44 circuits and evaluate 19 of them. Communication would roughly be the size of 19 garbled circuits.)

The protocol is based on the usage of erasure codes, and in particular of polynomials. Say that P_2 garbles t circuits and that P_1 evaluates ct of them for some known constant $c < 1$. Also, let b be some convenient block length and denote the number of blocks in the description of a garbled circuit by l. P_2 garbles the t circuits, and then computes l polynomials $p_1(\cdot), p_2(\cdot), \ldots, p_l(\cdot)$ such that $p_j(i)$ equals to the j-th block of garbled circuit gc_i. The degree of each polynomial is $t-1$. Then, for each polynomial p_i, P_2 sends to P_1 ct values, $\langle p_i(t+1), p_i(t+2), \ldots, p_i(t+ct)\rangle$. It also sends to P_1 hashes of all garbled circuits. P_1 then picks the $(1-c)t$ circuits that it wishes to check, and receives from P_2 the PRF seeds that were used for generating them. Using these seeds, P_1 reconstructs the checked garbled circuits, checks that they agree with the hash values and validates their structure. Afterwards, P_1 uses polynomial interpolation to recover the polynomials $p_1(\cdot), p_2(\cdot), \ldots, p_l(\cdot)$. Using these polynomials it retrieves the garbled circuits that it chose to evaluate, verifies that they agree with the hash values that P_1 has sent, and continues with the protocol.

The main advantage of this technique is that it enables to reduce communication even without knowing P_1's challenge. The overall communication overhead of this method is as the size of the ct evaluated circuits, which matches the communication overhead of [GMS08], but allows us to use this technique in the NISC setting. The proof of security of the resulting protocol is almost identical to the proof of Theorem 1 (except that the hash is also checked by the simulator) and therefore omitted.

6 Evaluation

Prototype implementation. Our prototype consists of several modules which communicate through files (for making the protocol suitable for asynchronous communication mechanisms like e-mail). It does not use the communication reduction techniques of Section 5. The prototype makes use of several libraries, namely RELIC-Toolkit [AG], JustGarble [BHKR13], and OpenSSL [OPE]. Relic-toolkit is chosen for its fast and efficient implementation of elliptic curve operations and is used to implement our OT and ElGamal based commitments. We use the binary curve B-251, which (roughly) provides 124-bit security. (Computing

a single elliptic curve multiplication, which corresponds to a single exponentiation in our protocol, costs about 120,103 CPU cycles for a fixed base and 217,378 cycles for a general base.) JustGarble is chosen for its fast implementation of garbling and evaluating circuits. ([BHKR13] advocates using fixed-key AES as a cryptographic permutation, and its implementation takes advantage of the AES-NI instruction set.) We modified JustGarble to read the circuit format of [TS], and read/write garbled circuits from/to a file (and not only the circuit structure). Lastly, we use the AES implementation from OpenSSL, to realize a PRF.

The setup. To evaluate our prototype we used two circuits, one for AES with non-expanded key (with 8,492 non-XOR gates and 25,124 XOR gates) and one for SHA256 (with 194,083 non-XOR gates and 42,029 XOR gates). The circuits were taken from [TS] (and slightly modified). In both circuits, each party has a 128 bit input value. The output of the AES circuit is 128 bit long, while SHA256 has a 256 bit output.

The experiments were run on a virtual Linux machine with a 64bit, i7-4650U CPU @ 1.70GHz and 5.4GB of RAM. (For a more accurate comparison, our code utilizes only a single core of the CPU. The average CPU frequency during the experiments was about 2.3GHz.) We measured clock cycles of each module of the system using the `RDTSC` instruction, and used the `clock_gettime()` system function to calculate the running time. Each experiment was run 10 times and the average run time was calculated in both cycles and seconds.

Performance. The experiments were done with statistical security parameter $t = 40$ and label length of 128 bits. Garbling was performed with the Free-XOR [KS08] and Garbled Row Reduction [PSSW09] techniques. (We also tested the protocol without those techniques. The results were slower by at most 10%.)

See Figure 5 for the running times of the main parts of our prototype. (Recall that when interacting with more than one sender, the receiver P_1 has to generate the first message only once. Then, for each sender, its running time will be similar to the time it takes it to process the sender's response in the single sender scenario that we examined.) The values represented in Figure 5 contain all operations, including I/O handling.

Observe that as the circuit size grows, the I/O portion becomes significant. For example, for the AES circuit, where every garbled circuit was stored in a 3 MB file, the total I/O time for the protocol is 0.53 seconds, whereas for the SHA256 circuit, where each circuit is stored in a 31 MB file, the total I/O time is 4.89 seconds. (For AES the I/O time was about 8% of the total time, whereas for SHA256 it was around 38% of the total time. This is expected since both functions have inputs of the same size, while the SHA256 circuit is much bigger.) The costs of garbling and evaluation of a garbled circuit are quite small (e.g., garbling takes less than 100 million cycles for the SHA256 circuit). The more significant overhead comes from I/O operations and the exponentiations done in the protocol.

Module or part name	**#Cycles**	**Time**	**#Cycles**	**Time**
	AES circuit		SHA256 circuit	
Init	42	0.02	44	0.02
P_1's message	71	0.03	73	0.03
P_2's response	8216	3.55	17651	7.59
P_1's computation	6452	2.79	11771	5.10
Cheating recovery	0.7	< 0.01	0.7	< 0.01
Total time	-	6.39	-	12.74
I/O time	-	0.53	-	4.89

Fig. 5. Running times for the prototype with statistical security parameter $t = 40$ and label length = 128. Time is in seconds and cycles are measured in millions of cycles. Running times include file operations.

In addition, we ran the experiment for AES with t=80 and got, as expected, that the costs are roughly twice those of the experiment with t=40. (Specifically, with t=80, it takes P_1 78 million cycles to compute its message, 16,518 million cycles for P_2 to compute its response, and 12,870 million cycles for P_1 to compute the output).

Due to lack of space, a comparison with previous multi-round 2PC implementations appears in the full version of this paper. We note here that although there is no standard benchmark for comparing 2PC implementations, it is clear that our NISC implementation is competitive with the best known interactive implementations.

Acknowledgements. The fourth author would like to thank Ran Canetti for helpful comments about this work, and to Yuval Ishai for introducing him to the work of [IKO+11]. We thank an anonymous EUROCRYPT reviewer for suggesting a simplification of the cheating-recovery commitments.

References

[AG] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. http://code.google.com/p/relic-toolkit/.

[BHKR13] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE S&P*, 2013.

[BHR12] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796. ACM, 2012.

[CCKM00] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. ICALP, pages 512–523. Springer-Verlag, 2000.

[DGH+04] Y. Dodis, R. Gennaro, J. Hstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the cbc, cascade and hmac modes. In *CRYPTO*, pages 494–510. Springer, 2004.

[GMS08] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT*, pages 289–306. Springer, 2008.

[Gol04] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[HK07] O. Horvitz and J. Katz. Universally-composable two-party computation in two rounds. CRYPTO, pages 111–129. Springer-Verlag, 2007.

[IKO+11] Y. Ishai, E. Kushilevitz, R. Ostrovsky, M. Prabhakaran, and A. Sahai. Efficient non-interactive secure computation. In *EUROCRYPT*, pages 406–425. Springer, 2011.

[IPS08] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer — efficiently. In *CRYPTO*, pages 572–591, 2008.

[KMR12] S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In *CCS*, pages 797–808. ACM, 2012.

[KS06] M. S. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of yaos garbled circuit construction. In *In Proceedings of 27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *ICALP*, pages 486–498, 2008.

[KSS12] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, pages 14–14, 2012.

[Lin13] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, pages 1–17, 2013.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78. Springer, 2007.

[LP09] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, April 2009.

[LP11] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346. Springer, 2011.

[MF06] P. Mohassel and M. K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, pages 458–473. Springer, 2006.

[MR13] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *CRYPTO*, pages 36–53, 2013.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700. Springer, 2012.

[OPE] OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org.

[Ped92] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. CRYPTO, pages 129–140. Springer-Verlag, 1992.

[PSSW09] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009.

[PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[SS11] A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405. Springer, 2011.

[sS13] a. shelat and C.-h. Shen. Fast two-party secure computation with minimal assumptions. In *CCS*, pages 523–534. ACM, 2013.

[TS] S. Tillich and N. Smart. Circuits of Basic Functions Suitable For MPC and FHE. http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.

[Yao86] A. C.-C. Yao. How to generate and exchange secrets. In *SFCS*, pages 162–167. IEEE Computer Society, 1986.