Secure Two-Party Computation over Unreliable Channels

Secure Two-Party Computation over Unreliable Channels

Ran Gelles[1,⋆], Anat Paskin-Cherniavsky[2], and Vassilis Zikas[3]

[1] Faculty of Engineering, Bar-Ilan University, Israel, ran.gelles@biu.ac.il
[2] Department of Computer Science, Ariel University, Israel, anatpc@ariel.ac.il
[3] School of Informatics, University of Edinburgh, Scotland, UK, vzikas@inf.ed.ac.uk

Abstract. We consider information-theoretic secure two-party computation in the plain model where no reliable channels are assumed, and all communication is performed over the binary symmetric channel (BSC) that flips each bit with fixed probability. In this reality-driven setting we investigate feasibility of communication-optimal noise-resilient semi-honest two-party computation i.e., efficient computation which is both private and correct despite channel noise.

We devise an information-theoretic technique that converts any correct, but not necessarily private, two-party protocol that assumes reliable channels, into a protocol which is both correct *and* private against semi-honest adversaries, assuming BSC channels alone. Our results also apply to other types of noisy-channels such as the elastic-channel.

Our construction combines tools from the cryptographic literature with tools from the literature on interactive coding, and achieves, to our knowledge, the best known communication overhead. Specifically, if f is given as a circuit of size s, our scheme communicates $O(s+\kappa)$ bits for κ a security parameter. This improves the state of the art (Ishai et al., CRYPTO' 11) where the communication is $O(s)+\mathrm{poly}(\kappa\cdot\mathrm{depth}(s))$.

1 Introduction

Secure two-party computation (2PC) allows two parties, Alice and Bob, to securely evaluate any given function on their private inputs. Informally, security corresponds to satisfying two properties: (*correctness*) every party should compute its correct output of the function; (*privacy*) any adversary corrupting a party should learn nothing more than the input and output of the party it corrupts.

The problem of secure 2PC in its full generality, as well as first solutions, were introduced by Yao [Yao82] and has since received a lot of attention in the cryptographic literature. Typically, one considers either a *malicious* adversary, who has full control over the corrupted parties, or a *semi-honest* one, who allows the parties to faithfully execute their protocol on their actual inputs but might try to extract information from their protocol view. Another distinction considers computationally *bounded* adversaries—that are limited to efficient computation—vs. computationally *unbounded* adversaries. The security in the former case is usually referred to as *computational* or *cryptographic*, while the latter is known as the *unconditional* or *statistical* or *information-theoretic*.[1] In this work we focus on semi-honest, information-theoretic security.

Despite the massive attention that 2PC has attracted, most of the existing literature assumes that the parties communicate using reliable (noiseless) channels: when Alice sends a message m to Bob, he receives exactly the information m. However, since modern communication networks might be affected by environmental (or even adversarial) interference, a more realistic case is that Bob actually receives a message $m' \neq m$, subject to some bounded type of noise. A natural question then is what happens when we execute 2PC protocols assuming such unreliable (noisy) communication channels.

Clearly, given a protocol π_0 that is designed to work (and proven secure) over reliable channels, the execution of π_0 over noisy channels may no longer be private, nor correct (see, e.g. [CPT13, GSW15]). One may naïvly believe that if π_0 is secure against a malicious adversary over reliable channels, then it would be at least semi-honest secure over (simple) noisy channels, because the "noise" in the latter setting can be

⋆ Supported in part by the Israel Science Foundation (grant No. 1078/17).

[1] Statistical security allows for some small (negligible) error probability; when this error is 0 we speak of *perfect* security.

reduced to the malicious activity of the adversary in the first setting. However, not even this is the case. Intuitively, the reason is that security against a malicious adversary does not guarantee that the protocol outputs the correct $f(x, y)$ to a deviating corrupted party. In contrast, when the party is just semi-honest, then it should receive the correct output even when the channel is noisy.

In this work we put forth the question of devising secure two-party computation protocols over *unreliable* communication channels, while keeping the communication complexity of such protocols to a minimum. We note that a natural approach to cope with the channels' interference is to wrap every message in π_0 with a good error-correcting code (ECC) [Sha48]. This has the effect of reducing the noisy channel into a channel that is essentially noiseless (i.e., it delivers the correct m with overwhelming probability per channel's instance), thus the execution of π_0 should preserve its security guarantees. Unfortunately, as simple and elegant as the above solution might be, it typically incurs a heavy overhead on the communication-complexity. In the worst case, every message m is very small compared to the length of the protocol (i.e., to its round-complexity), and the blowup the ECC imposes would be at least poly-logarithmic in the protocol's length.[2] Our goal is to devise secure protocols with only a constant multiplicative overhead, independent of the protocol's length.

The "overhead" discussed in the above paragraph compares the communication of the secure protocol π_0 that assumes reliable channels with the communication of π that assumes a binary symmetric channel (BSC_ε) where each bit is flipped with independent probability ε, yet it ignores a fundamental issue: without additional cryptographic assumptions, most functions f *don't have any secure protocol* π_0 *that evaluates them* [Kus89, Bea91]. On the other hand, the BSC_ε channel can be used as a cryptographic resource/setup [CK88], implying any function f could have a secure protocol π evaluating it [Kil88]. In that case, it is not even clear how to define the "overhead" of π with respect to π_0, as for many functions f, no secure π_0 even exists.

Our main result is a compiler that takes any boolean circuit C for some function $f(x, y)$ and outputs a semi-honest secure two-party protocol π for $f()$ that assumes that all the communication is sent over BSC_ε channels.[3] The protocol π has a "small" communication overhead, namely, linear in the size of the circuit C

Theorem 1 (main, informal). *Let $\varepsilon \in (0, 1/2)$ be a given constant and let κ be a security parameter. For any circuit $C : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m$ there exists a two-party (semi-honest) statistically secure protocol π_C that evaluates $C(x, y)$ over BSC_ε. Furthermore, it holds that $\mathtt{CC}(\pi_C) = O_\varepsilon(|C| + \kappa)$.*

When considering previous work for secure 2PC protocols over noisy channels, the state of the art is a compiler by Ishai et al. [IKO+11] that converts a circuit of size $|C|$ into a two-party protocol that communicates only $O(|C|) + \mathrm{poly}(\kappa \cdot \mathrm{depth}(|C|))$ bits assuming all communication is performed over BSC channels, where κ is the security parameter. Their protocol works in the malicious setting (with abort) and achieves statistical security by utilizing the strong machinery of the IPS compiler [IPS08]. In contrast, our result takes a completely different approach (namely, using techniques from interactive coding, which are fairly more simple), and achieves a reduced communication overhead, namely, $O(|C| + \kappa)$. On the other hand, our result applies only to the semi-honest setting, however contrast to [IKO+11], we do not allow the parties to abort—they must complete the protocol while maintaining its security.

Converting (noiseless, non-private) protocols into noise-resilient secure protocols. At times, the computation to be conducted is given as an interactive protocol, rather than an optimal circuit that implements the same functionality. Via relatively standard techniques we can extend our results so that they apply to any protocol π_0 which is *correct* over reliable channels (but *not* necessarily secure!), and convert it into a semi-honest statistically-secure protocol π over BSC_ε.

Specifically, assume π_0 is given as a branching program BP_0 (see Definition 7 for discussion on branching program representations of protocols), then we get

[2] While in the crypto community it is common to allow the error to be negligible (in a security parameter, typically taken to be at least as large as the protocol length), we will follow the standards of the coding community and insist on obtaining exponentially small error probability. In this case, the overhead implied by the naïve approach is in fact linear in the protocol's length, rather than poly-logarithmic.

[3] Using a recent result by Khurana et al. [KMS16] we are able to extend our result also to other types of noisy channels, such as the elastic channel. We defer the proof to the full version of this paper.

Theorem 2 (informal). *Let π_0 be a protocol that is not necessarily private over noiseless channels, and let BP_0 denote a branching program representation of π_0. There exists a compiler mapping π_0 into a semi-honest statistically secure protocol π over BSC_ε channels. The communication complexity of the obtained protocol is $\mathtt{CC}(\pi) = \tilde{O}(\text{width}(BP_0)) \cdot \mathtt{CC}(\pi_0) + O(\kappa)$, where κ is a security parameter.*

While it is unknown whether such a overhead of $\tilde{O}(\text{width}(BP_0))$ is optimal or even required, to our knowledge, the above factor is present in the state-of-the-art work and may be an inherent property of the conversion from protocols to circuits. Indeed, the trivial conversion (GMW [GMW87], see also Section 1.2) converts BP_0 into a boolean circuit (e.g., by Proposition 10) with $|BP_0|\text{polylog}(\text{width}(BP_0))$ gates. A different approach which directly (securely) evaluates each step of BP_0 without converting it first into a boolean circuit [NN01], yields an overhead of

$$\tilde{O}(\text{width}(BP_0)) \cdot \text{len}(BP_0) \approx \tilde{O}(\text{width}(BP_0)) \cdot \mathtt{CC}(\pi_0),$$

which is similar to the overhead we obtain in Theorem 2.

Notably, our result is asymptotically optimal when the protocol has an efficient, i.e., constant-width, branching program representation.

Extensions to other unreliable channels. We furthermore extend our results (Theorems 1 and 2) to other types of unreliable channels, namely, *elastic channels* (see, e.g., [DFMS04, Wul09, KMS16]). The (α, β)-elastic channel resemble to the binary symmetric channel in the sense that every bit is flipped with some independent probability α. However, one of the parties, either the receiver or the sender, but not both, can increase their knowledge of the other party's inputs and outputs to the channel. This is modelled by reducing the flipping probability of each bit received by that party to $\beta < \alpha$.

The work of Khurana et al. [KMS16] fully parametrize the conditions for which an (α, β)-elastic channel can be used in order to perform secure computations. Combining their result into our coding scheme allows secure computation over (α, β)-elastic channel with linear overhead, extending our results to this setting as well.

1.1 Overview of Techniques

As mentioned above, our result is two-folded: (i) secure simulation of circuits over noisy channels; (ii) secure simulation of (insecure, non-resilient) protocols over noisy channels.

The second result consists of converting the input protocol (specified as a branching program, BP) into a boolean circuit of size $|C| = |BP_0|\text{polylog}(\text{width}(BP_0))$ that contains NAND gates and computes the same function as the protocol. The conversion is quite straightforward: every node of the branching program can be implemented as a multiplexer where one party's input selects the next node to transition to. Additionally, some preprocessing of the inputs and the outputs is required, however these can be done locally and requires no communication. See Section 3.1 for further details. Once we obtain a circuit, we simply apply the simulation for circuits described below.

The more technically involved part is a secure simulation of boolean circuits over BSC channels (Section 3.2). Here, we are given a circuit $C(x, y)$ and the goal is to construct a two-party protocol π that evaluates C on the parties inputs (x, y) in a semi-honest, information-theoretic secure way, assuming only BSC channels and no other cryptographic assumption.

The immediate approach is to perform GMW—i.e., compute the circuit gate-by-gate where each gate is securely evaluated via a query to an OT oracle—yet replacing each OT oracle call with an OT implementation from noisy channel, e.g. [CK88, Cré97, DFMS04, Wul09, KMS16]. However, this still falls short of reaching our goal, as the above works treat the noisy channel as a resource rather than as the main communication channel; in particular, all the above works assume the parties share a reliable channel in addition to the noisy channel. Again we stress that simulating a reliable channel over a BSC_ε by wrapping each message with a standard ECC incurs a high communication overhead. A possible remedy would be to "group" many instances of OT together and encode their communication as a single message. For instance, group together each layer in the evaluated circuit. This approach potentially allows a constant blowup, however the blowup is higher for various circuit families, e.g., when the width of each layer in the circuit is smaller than the security parameter.

Our solution to this conundrum is to employ a technique of precomputed OT, first suggested by Beaver [Bea95]. This method allows the parties to "perform" OT before its inputs are known: in a pre-computation step the parties perform OT on random bits and end up with correlated randomness which later allows them to simulate an OT functionality on their real inputs by exchanging messages. Following this idea our protocol begins by performing many OT instances on *random inputs*, generating a large string of correlated randomness, where all these instances are grouped and encoded together using standard ECC. We keep the communication of this step low (i.e., with a constant blowup): ℓ OT instances can be computed with communication $O(\ell)$ using a result by Harnik et al. [HIKN08]. Then, our protocol "consumes" parts of the correlated randomness for each OT simulation used by the GMW procedure.

The last step takes care of channel-errors that may happen at the second part of each precomputed OT instantiation, i.e., when the parties exchange messages in order to simulate OT on the real input. Luckily, we prove that each such noise causes a very specific leakage. When simulating $\mathrm{OT}(b, x_0, x_1)$ the receiver might learn the incorrect input, x_{1-b}, but if that happens, the receiver learns nothing about x_b. Intuitively, this may compromise the correctness of the computation, but not its privacy (recall that all computations in GMW are performed on inputs that are secretly shared by the parties. The above error in the OT translates to learning one share of a (wrong) gate output).

Then, in order to solve this breach in the correctness, we employ techniques from the literature of *interactive coding* (see [Gel17] for a survey). In particular, we use an interactive coding schemes by Haeupler [Hae14] with linear overhead and exponentially small error probability, assuming BSC channels. In a nutshell, the scheme of [Hae14] works by executing a constant number of rounds from of the input protocol π_0 without any coding, after which the parties exchange information that allows them to reveal inconsistencies, specifically, the parties exchange hash values of their observed transcripts. Based on these exchanges, the parties decide whether to continue with running π_0 (if everything seems correct), or delete a certain amount of rounds (if some error is observed), hopefully, reverting the protocol into a state where both the observed transcripts are consistent. Repeating the above enough times guarantees that both parties end up with a correct transcript of π_0 with overwhelming probability while communicating only $O_\varepsilon(\mathtt{CC}(\pi_0))$ bits over a BSC_ε.

Finally, we show how to tweak the above coding so it doesn't compromise the privacy of the computation. The main issue here is back-tracking: the noise may cause the coding scheme to progress in one way, then back-track to a previous round and progress in a different way—this is usually a source for privacy leakage. We avoid such leakage and make the scheme secure via the common technique of re-sharing intermediate values with fresh randomness every time the simulation reverts to a previous point.

1.2 Related Literature

In his seminal paper, Yao [Yao82] provided a semi-honest computationally secure protocol, which can efficiently evaluate any given boolean circuit in a constant number of rounds. Yao's protocol assumes that the parties can access an Oblivious-Transfer (OT) functionality [Rab81] (see Appendix 2.3 for the definition of OT). This result was later extended to the information-theoretic (IT) setting by Goldreich, Micali, and Wigderson [GMW87]. Their so called GMW protocol for the semi-honest case also assumes that parties have ideal access to an OT functionality (cf. Section 2.3).[4] In a nutshell, GMW works as follows given the representation of the function f as an arithmetic circuit over a finite field. The parties compute the given circuit for f in a gate-by-gate fashion, from leaves to root. First, the inputs for each gate are secret-shared between the parties. Then, using OT the parties can compute a secret-share of the output of the gate. This progresses along the circuit until the parties hold a secret-sharing of the last gate, i.e., of the output.

Kilian [Kil88] proved that OT is in-fact a complete primitive even against malicious adversaries, a result made more efficient by Ishai et al. [IPS08]. Crépeau and Kilian [CK88] proved that OT can be implemented by an information-theoretic protocol using different types of channels, including the BSC_ε. Beaver [Bea95] showed how OT can be precomputed, i.e., how parties can, in an offline phase, compute

[4] In fact, the original GMW paper claims only computational security, even for the semi-honest case, as it uses a computational instantiation of OT; however, its is proved to achieve IT security when given ideal access to an OT functionality [Gol04].

correlated randomness that allows, during the online phase, to implement OT by simply communicating two messages (cf. Section 3.2.1). A fair amount of work has been devoted to so-called *OT combiners* namely protocols that can access several OT protocols out of which ℓ might be insecure, and combine them into a secure OT protocol, e.g., [HKN+05, MPW07, HIKN08, IKO+11]. Furthermore, [HIKN08] showed how to evaluate ℓ-parallel OT's (denoted OT^{ℓ}) in the semi-honest setting with linear communication complexity $O(\ell)$ and exponentially small error in ℓ.

Closer in spirit to our work, Naor and Nissim [NN01] considered the task of converting a (correct) protocol π_0 into a secure (both correct and private) protocol π (over noiseless channels), with minimal overhead. Similar to our result, their compiler takes as an input a branching-program BP_0 that represents π_0, rather than an arithmetic circuit for f. Also similar to our result, their obtained overhead is dominated by $\tilde{O}(\text{width}(BP_0))$; for the computational setting their obtained overhead is polylogarithmic in $\text{width}(BP_0)$. On the other hand, while our protocol works over noisy channels, the machinery of [NN01] assumes reliable channels. Furthermore, their compiler uses OT while ours does not assume any cryptographic setup assumptions (indeed, the BSC resource implies OT, so no other assumptions are needed).

Secure Computation Over Noisy Channels. For some functions f, none of the above cryptographic tools is needed in order to obtain a secure protocol for f (assuming reliable channels). Indeed, Kushilevitz [Kus89] (also, Beaver [Bea91]) gave a complete specification of the class G of two-party functions that can be unconditionally securely computed by a semi-honest 2PC protocol over reliable channels. More recently, the question of secure 2PC over *noisy* channels was addressed, for noisy all-powerful *adversarial* channels. In this case, a strong impossibility was shown [CPT13, GSW15]. Specifically, it was shown that for any $\mu > 0$, there exists $f \in G$, for which there exists an adversarial channel that corrupts up to μ fraction of the transmissions, over which f does not have a statistically secure protocol (despite the fact that $f \in G$, so it can be privately computed over noiseless channels). In addition to the above impossibility, Chung et al. [CPT13] give a positive result for the computational setting. Specifically, they demonstrate a tradeoff between the communication of a secure protocol and the noise level μ of the computationally bounded channel.

2 Model and Preliminaries

Throughout this paper we use (standard) asymptotical notations, in particular, for functions $f, g : \mathbb{R} \to \mathbb{R}^+$, we say that $f = \tilde{O}(g)$ if $f = O(g \cdot \log^c(g))$ for some constant $c > 0$. We say that a function is negligible if it is sub-inverse-polynomial, i.e., $negl(x) = o(1/poly(x))$. We denote $x \sim Ber(\varepsilon)$ for a random variable x that is distrebuted according to the Bernoulli distribution with parameter $0 \leq \varepsilon \leq 1$, i.e., $\Pr(x = 0) = 1 - \varepsilon$ and $\Pr(x = 1) = \varepsilon$. Addition and multiplication of bits are always to be interpreted as addition and multiplication over $GF(2)$.

2.1 Protocols, Correctness and Security.

We consider interactive computations between two parties, Alice and Bob with inputs $x_A \in \{0,1\}^n$ and $x_B \in \{0,1\}^n$, respectively. The parties wish to compute a given (deterministic) function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{\nu}$.[5] For simplicity, we assume $|x_A| = |x_B|$ throughout this work; however our results trivially apply to $|x_A| \neq |x_B|$ as well. To compute the function f, the parties execute a (potentially randomized) protocol $\pi = (\pi_A, \pi_B)$ which defines, for each party, the next message to send as a function of the party's input, the party's private randomness, and all the messages received so far. The protocol, also determines the output of each party (again, as a function of the party's input and received messages), denoted by out_A, and out_B for Alice and Bob, respectively. We will denote by r_A and r_B the random coins of Alice and Bob, respectively, in π. The *view* of Alice, $view_A = (x_A, r_A, T_A)$ consists of her input x_A, randomness, r_A, and transcript T_A; similarly, the view of Bob is $view_B = (y, r_B, T_B)$.

[5] As usual in the MPC literature, we restrict our handling to deterministic functions; the more general case of randomized functions can be easily treated by standard techniques (each of Alice and Bob inputs, in addition to their input x_A and x_B, a random string and their sum is used as the random coins.).

The total number of bits that are sent throughout the protocol π, i.e., the number of bits that Alice sends to Bob plus the bits that Bob sends to Alice, is the protocol's *communication complexity*, denoted as $\mathtt{CC}(\pi)$. The *length* of a protocol π, denoted $|\pi|$, is the number of rounds in the longest instance of the protocol. For simplicity, each round is assumed to be a transmission of a single bit, therefore the length of the protocol equals its communication complexity, $|\pi| = \mathtt{CC}(\pi)$.

We consider two types of protocols. Protocols that are only correct, i.e., compute the correct input (but not necessarily private), and protocols that are secure against a semi-honest adversary (i.e., both correct and private). The correctness definition is rather straightforward:

Definition 3 (Correctness). *A (randomized) protocol π for evaluating $f(x,y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\nu$ is δ-correct if at the end of π both parties output $f(x,y)$ with probability $\geq 1-\delta$. The protocol is* statistically correct *(in a given security parameter κ) if is $negl(\kappa)$-correct for some negligible function $negl(\cdot)$.*

Clearly, correctness without privacy is easy to achieve in the reliable network setting: a simple protocol which is always correct is to have Alice send her input to Bob who performs the computation and returns the output. But this is not always the case when the network is unreliable, especially when communication complexity is an issue. Indeed, although the above simple protocol might be easily transformed to be correct when executed over an unreliable network (e.g., by employing standard error correction), doing so for an arbitrary protocol *while keeping its total communication complexity low* is typically a challenging task.

Semi-Honest Security. Our final protocols are proved secure according to the standard simulation-based security notion against semi-honest adversaries. We will use the formulation of [Can00] which follows the real-world/ideal-world paradigm.[6] In a nutshell, the protocol execution in the real-world is compared to an ideal evaluation of the function the protocol is supposed to compute. In this ideal evaluation, a trusted party, usually called an *ideal functionality*, receives the inputs from the parties, performs the computation, and hands the outputs to the parties as well as the adversary.

Definition 4 (Closeness of Distributions). *For two distributions U and V we say that they are ε-*close*, and denote $U \approx_\varepsilon V$ if,*

$$\frac{1}{2}\sum_{\omega} |U(\omega) - V(\omega)| \leq \varepsilon.$$

The notion of ε-closeness is easily extendible to probability ensembles (i.e., families of distributions) $U = \{U_\kappa\}_\kappa$ and $V = \{V_\kappa\}_\kappa$ parameterized by a security parameter κ.

Informally, we say that a protocol π is ε-secure (in the semi-honest setting), where ε can be a function of the security parameter κ, if every party outputs the correct value of the function evaluated on the given inputs, and no (adversarial) party learns more than his intended output. The latter property is captured by requiring that for every adversary attacking the real-world protocol execution, there exists a simulator, also referred to as an *ideal adversary*, and the view of every party in the real world can be simulated in the ideal world with statistical distance ε.

Definition 5 (statistical, semi-honest security). *Let $\pi = (\pi_A, \pi_B)$ denote a protocol for evaluating a function $f(x,y) = (f_A(x,y), f_B(x,y))$. For a given x, y let $VIEW_A$, $VIEW_B$, OUT_A, OUT_B be the distribution of $view_A$, $view_B$, out_A, out_B in π given those inputs (over the randomness of the parties and the noise), when running over* $\mathtt{Ch}$.

We say that π is a statistically secure protocol for computing $f(x,y)$ over $\mathtt{Ch}$ *against semi-honest adversaries if there exist (possibly inefficient) simulators Sim_A, Sim_B for Alice and Bob, respectively, such that for all x, y, and κ a security parameter*

$$(Sim_A(1^\kappa, x, f_A(x,y)), f_B(x,y)) \approx_{\exp(-\kappa)} (VIEW_A, OUT_B), \text{ and}$$
$$(Sim_B(1^\kappa, y, f_B(x,y)), f_A(x,y)) \approx_{\exp(-\kappa)} (VIEW_B, OUT_A).$$

[6] Since we are only considering semi-honest security, our results can be easily adapted to work in the universal composition framework of Canetti [Can01].

Observe that the definition above captures both privacy and correctness. This is so since the ideal functionality's output to the honest party in the ideal world is indeed $f(x, y)$. We require a simulation error of $\exp(-\kappa)$ (as opposed to the traditional $negl(\kappa)$ for some negligible function $negl(\cdot)$). This is because lowering the error (even if it remains negligible) may affect the rate, so we want to carefully control this parameter (setting it to $\exp(-\kappa)$ is sufficiently low for most applications). As common in the setting of coding for interactive communication, κ will typically equal ℓ, the number of rounds in the protocol, but can be set higher, if needed. Another difference between our definition and the MPC definition is that the simulator, as well as π_0 and the encoding scheme, do not need to be efficient.

In Appendix A We give several further notions of security used in our proofs.

2.2 Noisy Networks, Coding Schemes, and Error Correction Codes

Protocols over Noisy Channels. We assume the communication channel connecting the parties is private—i.e., the adversary might only read messages transferred through the channel by corrupting the sender or the receiver and observing the corrupted party's channel interface—but is not reliable and might modify arbitrary many of the transmitted bits but *without reordering*. Concretely, the channel we assume stochastically flips each transmitted bit with a given constant probability ε, independent of other bits. This corresponds to the multi-use extension of the well-known, *binary symmetric channel* BSC_ε (see, e.g., [CT06, Rot06]).

The notion of a protocol needs to be augmented to the above noisy-communication model. One must keep in mind that in this case Alice and Bob might have inconsistent views of the transmitted messages, which depend on the noise. For instance, if Alice inputs to the channel a sequence $m^{(A)}_{A,1}, \ldots, m^{(A)}_{A,\ell}$ of messages to send to Bob, then the sequence $m^{(B)}_{A,1}, \ldots, m^{(B)}_{A,\ell}$ which Bob receives might be different than the original sequence, and vice versa for messages sent by Bob and received by Alice. Hence, Alice's view of the transcript corresponds to a sequence $T_A = (m^{(A)}_{pid_1,1}, \ldots, m^{(A)}_{pid_\ell,\ell})$, where each pid_i is A or B depending on whether the i-th bit $m_{pid_i,i}$ was sent from Alice or Bob, respectively; Bob's (view of the) transcript $T_B = (m^{(B)}_{pid_1,1}, \ldots, m^{(B)}_{pid_\ell,\ell})$ is defined analogously and may be different. The (noisy) *joint transcript*[7] of a given instance of the protocol consists of all messages sent and received during that given instance $T = (T_A, T_B)$. We denote a prefix of Alice's transcript of length ℓ, by $T_A[1, \ell]$ (resp., Bob's by $T_B[1, \ell]$). Throughout this work we will assume without loss of generality that the length of the protocol and the order of speaking is fixed, and in particular that Alice and Bob sends messages in alternating rounds, where Alice is the first to speak (in Round 1).

Error Correcting Codes for BSC channels. We use standard error correction codes implied by the work of Shannon (see, e.g., [CT06, Rot06]). Formally,

Lemma 6 (Shannon Coding Theorem [Sha48]). *For any discrete memoryless channel* $\mathtt{Ch}$ *with capacity* $\mathcal{C}$ *and any* k*, there exists a code* $\mathsf{ECC} : \{0,1\}^k \to \{0,1\}^n$ *and* $\mathsf{ECC}^{-1} : \{0,1\}^n \to \{0,1\}^k$ *with* $n = O(\frac{1}{\mathcal{C}}k)$ *such that for any* $m \in \{0,1\}^k$ *it holds that,*

$$\Pr\left[\mathsf{ECC}^{-1}(\mathtt{Ch}(\mathsf{ECC}(m))) \neq m\right] < 2^{-\Omega(n)}.$$

Recall that the capacity of the BSC_ε channel is $\mathcal{C}_{\mathsf{BSC}_\varepsilon} = 1 + \varepsilon \log \varepsilon + (1-\varepsilon)\log(1-\varepsilon) = O_\varepsilon(1)$. Also note that efficient constructions of such codes are well known (e.g., [Spi95, GI05]).

Coding Schemes for Interactive Protocols. An interactive *coding scheme* C [Gel17] for a given unreliable channel $\mathtt{Ch}$, e.g., over BSC_ε, transforms any correct protocol π_0 over noiseless channels, into a correct protocol $\pi = C(\pi_0)$ over the channel $\mathtt{Ch}$, that computes the same functionality as π_0 with high probability (usually, $1 - 2^{-\Omega(|\pi_0|)}$). The rate of a coding scheme C is defined as

$$\mathsf{rate}(C) = \liminf_{n\to\infty} \inf_{\substack{\pi_0 \\ \text{s.t. } |\pi_0|=n}} \frac{\mathtt{CC}(\pi_0)}{\mathtt{CC}(C(\pi_0))}.$$

[7] For notational simplicity we will refer to the joint transcript simply as the transcript.

2.3 Cryptographic Primitives, Boolean Circuits, and Branching Programs

Oblivious Transfer. Oblivious Transfer (OT) [Rab81] is a two-party functionality $\mathcal{F}_{OT}(b,(x_0,x_1))$ taking a pair of bits x_0,x_1 from Bob, and a bit $b\in\{0,1\}$ from Alice. It outputs x_b to Alice and nothing to Bob. A String-OT with string length s (shortly s-OT), is a functionality similar to OT, with the difference that x_0,x_1 are s-bit strings rather than bits. OT^ℓ is a functionality evaluating ℓ instances of OT on independent inputs.

We say that a protocol π operates in the *OT-hybrid model*, and denote $\pi^{\mathcal{F}_{OT}}$ if it is augmented to have (fixed) rounds where both parties query an ideal OT functionality $\mathcal{F}_{OT}$ and receive the corresponding outputs at the end of the same round.

Branching Programs. In this paper we use a specific variant of Branching Programs (BPs) that are particularly convenient for representing 2-party protocols, defined as follows.

Definition 7. *A (layered) BP on inputs (x,y) with depth t and width w is represented as a directed acyclic graph in which the vertices are partitioned into t disjoint sets $V_1,V_2,\ldots,V_t$ and edges go only from V_i to V_{i+1}. For any i, it holds that $w_i=|V_i|\leq w$, and for the initial layer, $V_1=\{\mathsf{start}\}$.*

Every node $v\in V_i$ in $i<t$ is assigned to either Alice or Bob, and has a transition function *$f_v:\{0,1\}^n\to V_{i+1}$. The nodes of the last layer V_t are labeled using some alphabet Σ. Without loss of generality, we assume $|V_t|=|\Sigma|$.*

The output, $BP(x,y)$, is evaluated by starting at $v=\mathsf{start}$ and following the path induced by applying $f_v(\cdot)$'s on either x or y according to the party that owns the current node, until reaching the last layer. The output is the label of the node in V_t reached by the above process.

Using standard notation, we denote by $|BP|$ the *size* of the BP, i.e., the number of nodes in the BP graph. We also refer to $w=\max_i w_i$ as the *width* of the BP, and denote it as $\mathrm{width}(BP)$. We note in passing that a BP representation of a protocol π is a generalization of π's protocol tree. This representation may be more compact in certain cases, for instance, when parties store only limited amount of information at any point of the protocol (in this case, the amount of information corresponds to the logarithm of the BP's width).

Importantly, a BP representation of some protocol π allows an efficient emulation of π, i.e., we can efficiently generate an execution of π given its BP representation as above. However, for a given protocol, computing such a BP might not necessarily be efficient. It is also easy to verify that the communication of π is connected to the branching program by $\mathtt{CC}(\pi)=\sum_{1<i\leq t}\lceil\log w_i\rceil$, hence,

$$\mathrm{depth}(BP)\leq\mathtt{CC}(\pi)\leq\mathrm{depth}(BP)\cdot\lceil\log(\mathrm{width}(BP))\rceil \tag{1}$$

Boolean Circuits. We use standard Boolean circuits consisting only of NAND gates[8] with fan-in 2 and unbounded fan-out [AB09]. We assume all literals depend on the input, i.e., we don't allow constant inputs.[9] We denote by $|C|$ the size of C, i.e, the number of its nodes/gates, and by $\mathrm{depth}(C)$ its depth.

Definition 8 (Boolean Circuits). *For every $n,\nu\in\mathbb{N}$ a boolean circuit $C(x)$ with n inputs and ν outputs is a directed acyclic graph. It contains n nodes with no incoming edges; called the input nodes, and ν nodes with no outgoing edges, called the output nodes. Each input node is labeled by an input bit x_i. All other nodes are called NAND gates. Each gate node has fan-in (indegree) 2 and unbounded fan-out (outdegree). The size of C, denoted by $|C|$, is the number of nodes in it,* $\mathrm{depth}(C)$ *is the length of the longest directed path in C.*

3 Deterministic 2PC over BSC_ε with linear rate

In this section we prove our main results, Theorem 1 and Theorem 2, and show how to simulate any (possibly non-private) protocol that assumes reliable communication over a BSC_ε. Let us first re-state Theorem 2 in a more formal manner.

[8] Recall that NAND gates are universal logic gates, i.e., functionally complete.

[9] This is without loss of generality since we only consider semi-honest security (any of the two parties can be requested to contribute any needed constants as part of its input.)

Theorem 9. *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\nu$, κ be a security parameter, and $\varepsilon \in (0, 1/2)$. Let π_0 be a deterministic correct protocol for evaluating f over noiseless channels, and let BP_0 denote a branching program representation of π_0. Then, there exists a compiler mapping π_0 into a (semi-honest) statistically secure protocol π over BSC_ε channels. The communication complexity of the obtained protocol is $\mathtt{CC}(\pi) = \tilde{O}(\text{width}(BP_0)) \cdot \mathtt{CC}(\pi_0) + O(\kappa)$.*

Note that the above theorem considers only deterministic protocols. In Section 4 we show how our compiler can be extended to randomized protocols (Theorem 19).

Theorem 9 is proved in two steps. First, in Section 3.1 we show how to convert a protocol π_0 for which we know a branching-program representation BP_0, into a Boolean circuit C_0 of size $|BP_0| \cdot \text{polylog}(\text{width}(BP_0))$. From Equation (1), we conclude that

$$\begin{aligned} |C_0| &\leq \text{width}(BP_0)\text{depth}(BP_0)\text{polylog}(\text{width}(BP_0)) \\ &\leq \text{width}(BP_0)\mathtt{CC}(\pi_0)\text{polylog}(\text{width}(BP_0)) \\ &= \mathtt{CC}(\pi_0)\tilde{O}(\text{width}(BP_0)). \end{aligned}$$

Second, in Section 3.2 we show how to securely evaluate C_0 over (only) a BSC_ε channel. Our circuit-evaluation method has communication $O(|C_0|) + O(\kappa)$.

3.1 Reducing protocols to circuit evaluation

Our first step is converting a protocol π_0 given as the branching program BP_0, into a boolean circuit C_0 of size $|C_0| = |BP_0|\text{polylog}(\text{width}(BP_0))$, that implements the same functionality.

Proposition 10. *Let $f(x, y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\nu$ be a function, and let π_0 be a deterministic protocol for f over noiseless channels. The protocol π_0 is assumed to have perfect correctness (i.e., $\pi_0(x, y) = f(x, y)$ for all $x, y \in \{0,1\}^n$) but no privacy guarantees. Furthermore, let BP_0 be a branching program representation of π_0.*

Then, for some n_A, n_B, ν_{AB} there exists a circuit $C_0 : \{0,1\}^{n_A+n_B} \to \{0,1\}^{\nu_{AB}}$ of size $|C_0| = |BP_0|\text{polylog}(\text{width}(BP_0))$, and "translation" functions $\tau_A : \{0,1\}^n \to \{0,1\}^{n_A}$, $\tau_B : \{0,1\}^n \to \{0,1\}^{n_A}$, and $\tau_{out} : \{0,1\}^{\nu_{AB}} \to \{0,1\}^\nu$, such that for all $x, y \in \{0,1\}^n$ it holds that

$$\tau_{out}\left(C_0\left(\tau_A(x), \tau_B(y)\right)\right) = f(x, y).$$

We defer the proof to Appendix B.1.

3.2 Secure evaluation of circuits over a BSC_ε

We proceed to the second part of the proof of theorem 9 and describe a protocol for secure evaluation of circuits over a BSC_ε with communication complexity $O(|C| + \kappa)$. Formally,

Proposition 11. *Let $\varepsilon \in (0, 1/2)$ be a given constant and let κ be a security parameter. For any circuit $C : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^\nu$ there exists a two-party (semi-honest) statistically secure protocol π_C that evaluates $C(x, y)$ over BSC_ε. Furthermore, it holds that $\mathtt{CC}(\pi_C) = O_\varepsilon(|C| + \kappa)$.*

The above is the formal version of our main theorem (Theorem 1 from the introduction). Note that Theorem 9 (i.e., Theorem 2) follows as a corollary of Proposition 10 and Proposition 11 (see Appendix B.2 for details). The remainder of the section is dedicated to proving Proposition 11.

3.2.1 Building blocks. Towards proving Proposition 11, we start with a description of the tools that we will combine into our final construction. Some of these tools come from the MPC literature, while other come from the field of coding for interactive communication. Missing proofs of this section appear in Appendix B.

OT^ℓ over BSC_ε with linear communication overhead. To facilitate the privacy of our construction we rely on the following implementation of ℓ parallel OT's over BSC_ε with communication linear in ℓ.

Theorem 12 ([HIKN08, Theorem 9]). *For any constant $\varepsilon \in (0, 1/2)$, and any ℓ, there exists a two-party protocol π^{OT^ℓ} that assumes the parties are connected (only) by an BSC_ε channel, which implements OT^ℓ. The protocol is statistically secure against semi-honest parties with error $2^{-\Omega(\ell)}$, and has a communication complexity of $O_\varepsilon(\ell)$ bits.*

OT over a BSC with limited leakage, provided precomputed OT. Another tool we will need, is a way to implement a specific type of "buggy OT" over a BSC. In this "buggy" version of the OT protocol on input (b, x_0, x_1), with constant probability p it may happen that the Alice (the receiver) learns the wrong input x_{1-b} instead of the correct value x_b. Otherwise, the protocol works as a standard OT, i.e., Alice learns x_b. In both cases Bob (the sender) learns nothing. The key property here is that in either case Alice learns exactly one of the values x_0, x_1, and can never learn both.

Our OT implementation builds on a scheme by Beaver [Bea95], and requires the parties to already share correlated bits of special form: their correlation corresponds to outputs of OT on random inputs. In hindsight, those correlations will be obtained by performing OT^ℓ (by Theorem 12) on random inputs in a precomputation step. This precomputation step is instrumental to keep communication low assuming BSC channels. Indeed, it is more efficient to encode over a noisy channel a large amount of OT instances, rather than encode them one by one. On the other hand, most MPC protocols make sequential call to OT, one-by-one, as the protocol progresses. Performing OT based on precomputed bits allows us to benefit both worlds: the precomputation step creates a bulk of correlated bits in a communication efficient way; then, each instantiation of OT consumes bits from that bulk, without having large communication overhead, and while keeping the privacy guarantees.

Protocol Π-$\mathbf{OT}_\varepsilon$

- **Inputs:** Alice's input is a bit b; Bob's input is a pair of bits (x_0, x_1).
- Pre-computation step: The parties are assumed to have (trusted) preshared bits sampled as follows: Let (b', x'_0, x'_1) be random independent bits. Bob gets x'_0, x'_1, while Alice gets b' and $x'_{b'}$, that is, she either gets x'_0 or x'_1 according to the value of b'.
- Alice and Bob perform as follows
  1. Alice sends $c = b + b'$; Assume Bob receives c'
  2. Bob sends $(x_0 + x'_{c'}, x_1 + x'_{1-c'})$.
  3. Let (y_0, y_1) denote the bits received by Alice in the second round. Alice outputs $y_b + x'_{b'}$ as her output.

Fig. 1: The Π-OT_ε Protocol

The protocol Π-OT_ε, described in Figure 1, is such a "buggy-OT" where all communication is done over BSC_ε. Our above buggy-OT discussion provides the high level intuition for the usefulness of protocol Π-OT_ε as a building block for our protocol. The formal statement (Lemma 13) and its proof (see Appendix B.3) use somewhat different properties. Namely, we use the notion of *weak security* and of *channel-transparent security* (see Appendix A their formal definition). The meaning of these new notions is roughy as follows:

weak security against semi-honest adversaries relaxes standard semi-honest security by requiring that the views of the parties are consistent with an execution where the corrupted party's input z is replaced by some z' (depending only on z), rather than with the original input z.

Channel-transparent security strengthens the standard notion of security, by requiring that even if the adversary could see the messages received by the honest party, it would not learn anything it was not supposed to learn.

Lemma 13. *For any $\varepsilon < 1/2$, the protocol Π-OT_ε over BSC_ε is weakly, channel-transparently, statistically secure in the semi-honest setting over BSC_ε channels.*

Computing NAND gates via OT. Assume we wish to compute a NAND gate over the inputs (a, b) where the parties secret-share the inputs, i.e., Alice holds a_1, b_1 and Bob holds a_2, b_2 where a_1, b_2 are uniform independent random bits and $a = a_1 + a_2$, $b = b_1 + b_2$. We wish to compute the bit $c = \text{NAND}(a, b)$ so that at the end of the computation the parties will hold a secret-sharing of c, i.e., Alice will hold a random bit c_1, and Bob will hold c_2 so that $c = c_1 + c_2$.

This task can easily be done assuming we can utilize two instances of an ideal OT functionality. Note that $c_1 + c_2 = 1 + (a_1 + a_2)(b_1 + b_2) = 1 + a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2$. The parties utilize two OT instances in which Alice, using her a_1 and b_1, retrieves one of $(c_2, c_2 + b_2)$ and one of $(a_2b_2, a_2 + a_2b_2)$, respectively, up to some randomness added by Bob that keeps his values a_2, b_2, c_2 private. (The complete protocol in the OT-hybrid setting is given in Figure 2.) However, in our implementation we will not have an ideal OT, but instead we utilize the protocol Π-OT_ε assuming pre-comupted correlated randomness. The following lemma provides the security of the NAND computation protocol when each OT is realized via the above Π-OT_ε.

Lemma 14. *For any $\varepsilon < 1/2$, the protocol $NAND^{\Pi\text{-}OT_\varepsilon}$ (i.e., the algorithm in Figure 2, where each OT-instance is replaced with an execution of Π-OT_ε (Figure 1)) is weakly, channel-transparently, statistically secure in the semi-honest setting, assuming all communication is done over a BSC_ε.*

Protocol $\text{NAND}^{\mathcal{F}_{OT}}$

- **Inputs:** Alice holds $a_1, b_1 \in \{0, 1\}$, Bob holds $a_2, b_2 \in \{0, 1\}$.
- **Outputs:** Alice gets c_1 and Bob gets c_2 so that $c_1 + c_2 = 1 - (a_1 + a_2)(b_1 + b_2)$. I.e., if $a = a_1 + a_2$, $b = b_1 + b_2$, and $c = c_1 + c_2$ then $c = \text{NAND}(a, b)$.
- Protocol's Description:
  1. Bob picks random bits r_1, r_2, and sets $c_2 = r_1 + r_2$.
  2. The parties query the OT oracle: $\mathcal{F}_{OT}(a_1, (r_1, b_2 + r_1))$. Denote Alice's OT output by o_1.
  3. The parties query the OT oracle: $\mathcal{F}_{OT}(b_1, (a_2b_2 + r_2, a_2b_2 + a_2 + r_2))$. Denote Alice's OT output by o_2.
  4. Alice sets her output to $c_1 = 1 + a_1b_1 + o_1 + o_2$.

Fig. 2: Shared-input shared-output NAND computation in the OT-hybrid setting

A coding scheme for interactive communication with linear rate. The last tool we need is taken from the literature of coding for interactive communication and provides a way to fortify a given protocol π_0 (that assumes noiseless channels), resulting in a noise-resilient protocol π so that the output π equals that of π_0 with probability $1 - \exp_\varepsilon(-|\pi_0|)$ assuming BSC_ε channels.

The general idea, often referred to as *the rewind-if-error paradigm* (see [Gel17]), is to run π_0 as-is for several rounds, after which the coding scheme communicates some consistency information to verify that both parties agree on the transcript. In case the parties detect that they agree, they continue in running π_0 for another several rounds; Otherwise, they backtrack to some point in the past were they are (hopefully) in agreement.

Several coding schemes follow this paradigm and achieve efficient schemes with good communication rate, e.g., [Sch92, KR13, Hae14, BKN14, GHK+16, EGH16, GH17]. We will use one by Haeupler:

Theorem 15 ([Hae14, Algorithm 3]). *Given any $\varepsilon < 1/2$, any deterministic protocol π_0 can be efficiently transformed into a randomized protocol π that communicates over BSC_ε, with $\mathtt{CC}(\pi) = O_\varepsilon(\mathtt{CC}(\pi_0))$. For any (x, y), it holds that $\pi(x, y) = \pi_0(x, y)$ with probability at least $1 - \exp_\varepsilon(-|\pi_0|)$.*

Interactive Coding Scheme for BSC [Hae14]

1. Let π_0 be a deterministic ℓ-round protocol, and $\varepsilon < 1/2$ the BSC error probability. Let $v = \Omega_\varepsilon(1)$, $\ell' = O_\varepsilon(\ell)$.
2. Run an initialization step (independent of π_0), setting up a shared randomness resource sr.
3. Initialize the transcript (prefix) $T_A \leftarrow \phi$ of the execution of π_0 seen so far, and initialize some additional variables tracking statistics V_A. The state of the protocol is $S_A = (T_A, V_A)$.
4. For each iteration $i \in [\ell'/v]$
   (a) Exchange verification information $h_A = H_i(S_A, sr)$
   (b) Receive Bob's possibly noisy verification information h'_B.
   (c) As a function of S_A, h_A, h'_B, decide whether to:
      i. Continue running the protocol: starting from T_A for v steps (both sending and receiving messages, as prescribed by π_0). Append them to the transcript T_A
      ii. Backtrack: run the protocol as in the previous item, but send random bits instead of the real protocol messages, and do not advance T_A.[a]
   (d) If backtracking, additionally truncate the suffix of T_A by $g \cdot v$ steps, where g is an integer determined by S_A, h'_B.
   (e) Update the statistics V_A based on the current T_A and h_A, h'_B.
5. Output the value output by π_0, based on $T_A[1, \ell]$.

[a] The concrete dummy values are different in [H14], but are immaterial for its correctness, and these values are slightly more convenient in our case. Also, for correctness to hold, the original protocol is padded to length ℓ' by appending dummy moves, say, exchanging random bits.

Fig. 3: A Simplified Outline of Algorithm 3 in [Hae14]

The outline of Alice's behaviour in the resulting protocol π is given in Figure 3. Bob's program is symmetric. In a nutshell, the parties in the above scheme execute π_0 but occasionally compare (hashes of) prefixes of their observed transcripts. A hash mismatch is an indication for a possible inconsistency in π_0's execution due to channel errors, and the party that observes such a mismatch may decide to backtrack. A careful choice of the protocol's parameters—including the number of steps to retract and the hash range—yields a constant rate.

Observe that the local transcripts have different lengths (e.g., if one party backtracks while the other party does not), or may contain different information (due to noise). The simulation makes real progress when the local transcripts of both parties, T_A, T_B have the same length and content, and the parties perform Step 4(c)i in the algorithm. All the effort in the construction (and its correctness proof) goes into making sure that $\ell = |\pi_0|$ many such progress steps are made (and not undone by backtracking) with overwhelming probability at the end of the $\ell' = O(\ell)$ rounds of π's execution.

3.2.2 Circuit simulation over a BSC Our starting point toward devising a secure protocol for evaluating circuits, is the classical GMW protocol [GMW87]. GMW performs a secure evaluation of a given circuit C_0 on the parties' (private) inputs, assuming the parties are connected through a noiseless channel in the OT-hybrid setting (i.e., assuming they have access to a perfect OT functionality).

GMW evaluates the circuit C_0 gate by gate according to a predetermined topological ordering of the circuit graph. The inputs for each gate are secret-shared between the parties, and the evaluation of the gate yields a secret-sharing of it's output value. More precisely, the activity of GMW can be described using the following three phases.

- **Initialization:** Alice shares every bit x_i of her input into a simple $(2,2)$-additive sharing of x_i $(s_{i,1}, s_{i,2}) = (r, x_i + r)$ where r is a uniform bit. Alice keeps $s_{i,1}$ as her share of x_i, and sends Bob $s_{i,2}$ as his share. Bob does the same thing on his input bits y_i.
- **Evaluation:** The parties evaluate each NAND gate on the shared inputs, obtaining a randomly shared output (giving each party a share). The evaluation of NAND gates is implemented using two calls to the OT oracle, where Bob always plays the sender and Alice plays the receiver.
- **Output Delivery:** At the end of the evaluation phase, the parties hold random shares of each output bit. The parties then send their share vectors to each other, thereby each party learns exactly the values of the outputs.

We now discuss how to augment each one of the above phases, when the communication channels are assumed to be BSC_ε, and argue that this augmentation is statistically close to the original GMW, thus, it is statistically secure.

Initialization and Output Delivery. The initialization part consists of two "rounds" (where in one round Alice communicates many bits, and then in the second round Bob communicates many bits). Thus we can use the simple approach of encoding each of the messages with a short ECC (Lemma 6) with codewords of length $\Theta_\varepsilon(m+\kappa)$ (i.e., adding at least $O(\max\{m,\kappa\})$ redundant bits, where m is the length of the encoded message), that decode correctly over BSC_ε except with probability $\exp_\varepsilon(-m-\kappa)$. The same holds for the output delivery phase. The size of each such encoded message is $O_\varepsilon(|C_0|+\kappa)$, so asymptotic communication complexity does not change.

The decoding of these messages fails with probability at most $\exp_\varepsilon(-|C_0|-\kappa)$ and this value dominates the simulation error with respect to the original GMW.

The Evaluation phase. As in GMW, the input of this phase is a random secret-sharing of the input bits of C_0 as produced by the initialization phase. The output of this phase is a random secret-sharing of C_0's output value.

Following the GMW approach, this phase computes the NAND gates of C one by one. However, this approach hits two immediate obstacles: (1) each NAND computation requires two OT instantiations, each of which may take $O(\kappa)$ communication leading to a global communication of $O(\kappa|C_0|)$, rather than our aimed communication of $O(|C_0|+\kappa)$. (2) Due to channel noise, some of the NAND gates (as well as the OT evaluations) will be computed incorrectly. This may lead to information leak or to correctness deficiency.

Our solution to the above hurdles is achieved by employing Beaver's method of precomputed OT in conjunction with Haeupler's interactive coding scheme. Since all the OTs are precomputed, constant overhead can be achieved. Correctness is obtained due to the coding scheme and security is obtained by carfuly analyzing the possible leakage in case a certain NAND gate evaluation fails due to noise.

The construction follows these high-level steps:

1. The parties execute π^{OT^ℓ} on random inputs. The output of this step is $\ell = O(|C_0|+\kappa)$ tuples $(x'_0, x'_1, b', x'_{b'})$ where x'_0, x'_1, b' are independent uniform bits distributed; Alice receives $(b', x'_{b'})$ and Bob receives (x'_0, x'_1). These ℓ pairs serve as correlated randomness for later OT instantiations (see below).
2. Define C to be a circuit that computes the same function as C_0, yet contains $O(\kappa)$ more gates. (This step increases the success probability, and is equivalent to performing a stronger coding in Step 4 below on the original C_0.)
3. Each NAND gate in C is being evaluated using the protocol $\text{NAND}^{\Pi\text{-OT}_\varepsilon}$: the parties execute the NAND protocol (Figure 2) where each OT execution tis replaced with an execution of Π-OT_ε (Figure 1).
4. The above evaluations of NAND gates are coded via the interactive coding scheme of Theorem 15, with linear communication overhead and success probability of $1-\exp(-|C|) = 1-\exp(-|C_0|-\kappa)$.

The complete construction, Π_{2pc}, is depicted in Figures 4–5.

Theorem 16. *The protocol Π_{2PC} depicted in Figures 4 and 5 satisfies Proposition 11.*

We now give high level and intuitive arguments that explain why the above construction is both private and correct, and has a linear communication overhead. The detailed proof appears in Appendix B.5.

Protocol Π_{2PC}

Inputs: A public input circuit C_0 and private inputs x and y held by Alice and Bob, respectively.

Initialization:

- Augment C_0 by adding $O(\kappa)$ dummy gates evaluating the length-κ vector $\overline{0}$. The output of these added dummy gates is to be ignored by the parties. From here and on we assume C is the augmented circuit.[a]
- Alice sends her encoded shares of her inputs x for C using an ECC of length $O(|C|)$ with decoding error $\exp(-|C|)$ (Lemma 6). She also receives and decodes the (encoded) shares of the y's. Alice stores the resulting shares as the values of the corresponding circuit wires.
- Alice and Bob run $\pi^{OT^{\ell'}}$ (Theorem 12) on uniformly random inputs (we set ℓ' shortly). The output is ℓ' pairs (b, x_0, x_1) where for each such pair Bob holds x_0, x_1 and Alice holds b, x_b. Denote these as precomputed correlations vectors $\overline{v_A}, \overline{v_B}$, respectively.

[a] We add these gates because the correctness guarantee in Theorem 15 behaves like $1-\exp(-|C_0|)$, which is insufficient for small circuits. To improve this probability to a magnitude of $\exp(-|C|) = \exp(-|C_0| - \kappa)$ we increase the circuit size by adding κ dummy gates. This is equivalent to running the coding scheme of Theorem 15 for $O(\kappa)$ more rounds.

Fig. 4: Secure Circuit Evaluation protocol Π_{2PC} (Input and Initialization)

Protocol Π_{2PC} (cont.)

Evaluation:

- Let π_0 denote the protocol induced by running GMW on the augmented circuit C (recall section 3.2.2). Namely, the parties evaluate each of the NAND gates on their input shares (Figure 2) in a gate-by-gate fashion according to a predetermined topological ordering. Each call for $\mathcal{F}_{OT}$ in the implementation of Figure 2 is replaced with an execution of Π-OT$_\varepsilon$ (Figure 1).
  After evaluating the last gate, π_0 is assumed to keep sending zeros indefinitely.
- Apply the coding scheme of Theorem 15 onto the protocol π_0 with the following augmentations: each iteration of the coding scheme works in chunks that are aligned with a complete evaluation of NAND gates; this way, backtracking is always aligned with a beginning of evaluating a NAND gate.
  Let π denote the resulting protocol. Let ℓ', v denote the parameters of π as defined in Figure 3.
- When evaluating the j-th NAND gate ($1 \le j \le v$) of the i-th iteration ($1 \le i \le \ell'/v$), the following applies:
  (1) First note that Alice does not use any randomness during the NAND evaluation. Also recall that Bob's randomness is r_B and that $\overline{v}_A, \overline{v}_B$ denote the pre-computed OT pairs obtain in the initialization phase.
  (2) Each NAND evaluation (Figure 2) requires 2 OT instantiation. The k'th OT instantiation ($k \in \{1, 2\}$) uses the randomness $r_B[i][j][k]$ and the pre-comupted pairs $\overline{v}_A[i][j][k], \overline{v}_B[i][j][k]$.
  (3) The inputs used by the parties to evaluate a given NAND gates are either those stored at its input wires, or random values in case the coding scheme (Figure 3) performs Step 4(c)ii and requires sending dummy value.

Output Delivery:

- If $|T_A| < \ell$, output $\perp$.
- Alice extracts her share vector so_A of the output wires from her stored values. She sends Bob $\mathsf{ECC}(so_A)$ using a code with length $O(so_A + \kappa)$.
- Alice receives (a noisy version of) Bob's encoded share vector $\mathsf{ECC}(so_B)$, and decodes it to obtain so'_B. Alice outputs $z = so_A + so'_B$.

Fig. 5: Secure Circuit Evaluation protocol Π_{2PC} (Evaluation and Output)

First, let us consider correctness. Again, due to channel noise it is possible that some NAND computations provide an incorrect output. In fact, a constant fraction of the NAND computations are incorrect. We address the correctness issue by employing the interactive coding scheme of Theorem 15. Namely, we plug the GMW evaluation protocol into the coding scheme of Figure 3, with the following main (but insignificant) difference: instead of simulating the protocol bit-by-bit, the smallest unit in our adaptation is a NAND gate, i.e., the number of bits it takes to evaluate a NAND gate. This way, if the coding scheme rewinds the GMW evaluation, it always rewinds to the beginning of an evaluation of a NAND gate. Then, if the noise corrupted some transmissions (which may cause an incorrect NAND evaluation), the coding scheme will indicate this event and allow the parties to re-evaluate any incorrect NAND evaluations. The correctness of the coding scheme implies that at the end of the scheme, the parties share the correct value (except with exponentially small probability).

Next, let us consider the privacy of the NAND computations assuming noisy channels. Recall that computing NAND is immediate given a secure OT protocol (Figure 2). The straightforward implementation of each OT would be using the BSC_ε channel as a resource, e.g., via [CK88, KMS16] (see also [HIKN08] for a simple description assuming random inputs in the semi-honest setting). However, those implementations assume a noiseless channel in addition to the noisy BSC channel, which we don't have in our setting. If we simply replace the noiseless channel by a BSC channel (without any other adaptations) the resulting protocol may leak both inputs to the OT-receiver. Such a leaky instantiation of OT is unacceptable for the GMW evalutation: it fully reveals the value of internal wires in the circuits which is not allowed as it leaks information on the other party's input. Another privacy issue stems from using a coding scheme: rewinding the evaluation to a previous gate and re-evaluating a gate may cause privacy leakage unless performed correctly.

To get around these privacy issues, our approach is two-fold: (a) we utilize the Π-OT_ε protocol (Figure 1, Section 3.2.1) that limits the leakage—Alice may only learn one of Bob's input (but maybe the wrong one); and (b) we execute each OT on an independent set of inputs, that is, we always re-share intermediate values so that information leaked in a previous execution is meaningless to the new execution.

Lemma 13 and 14 guarantee the security for a *single-shot* instance NAND execution using the Π-OT_ε protocol. However, in the GMW protocol we execute these protocols multiple times: once per each gate in C. Yet, these activations happen *sequentially* and they operate on independent inputs.

Moreover, privacy is preserved even when the coding scheme rewinds the execution to a previous round. This follows since Bob re-shares the output at every NAND execution using fresh randomness. That is, if there was an error, then Alice learns a single share of an incorrect output, however, she does *not* learn this output. Next time the same gate is evaluated, the output is re-shared again using fresh and independent randomness, and Alice learns one share of this new value. Clearly, the old share she learnt in a previous execution is independent of the share she learn in the repeated evaluation.

Finally, we discuss the communication overhead of the evaluation phase. The tool we need is a way to implement a large number of OT's over BSC_ε with constant communication overhead while allowing the inputs of each OT depend on the outputs of previous OT; this construction must never leak both sender's inputs, however it may leak to the receiver the other, non-chosen, input. To our knowledge, no existing scheme achieves the above without preprocessing. For example, if each OT is performed individually (say, via the secure construction of [CK88]) then the overhead will not be constant in the security parameter.

Instead, our OT instantiation through the Π-OT_ε scheme makes use of shared correlated randomness (that the parties pre-compute). This enables sequential evaluation of OTs with the above security requirements in the online phase, while keeping the overhead constant both in the online and the pre-processing steps. Specifically, we use the π^{OT^ℓ} protocol (Theorem 12) *on random inputs*. This supplies the parties with the required correlated randomness needed for the online OT evaluations throughout the computation. Each OT evaluation in the online phase takes a constant number of transmissions (given the correlated randomness); additionally, the preprocessing step has a linear communication overhead, i.e., $O_\varepsilon(\ell) = O(|C_0| + \kappa)$. Note that Theorem 12 assumes all communication happens over a BSC_ε as happens in our setting. It is then easy to verify that the evaluation phase can be done with linear communication overhead in the size of the (augmented) circuit.

4 Randomized 2PC over BSC_ε with linear rate

The main result of Theorem 9 applies only to deterministic protocols. Here we show this result can be extended to arbitrary, potentially randomized, protocols by incurring only an *additive* factor of $O(\kappa + \nu + \log n)$.

The proof follows as a corollary of Theorem 9 and the following Lemma 18 that proves that one can replace a randomized protocol π_0 with a distribution over a small set of deterministic protocols, causing only a little loss in the correctness (both over noiseless channels).

Definition 17. *An interactive randomized (two-party) protocol for f is said to be* $\exp(-\kappa)$-correct *if for any given input vector (x, y) the protocol's output is identical to the value of $f(x, y)$, except with negligible probability.*

Lemma 18 ([NN01], Lemma 4.4). *Let π_0 denote a δ-correct randomized protocol for $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\nu$. Then, there exists a protocol π that needs only $\Delta = O(\kappa + \log(n) + \nu)$ random bits and has communication complexity $\mathtt{CC}(\pi) = \mathtt{CC}(\pi_0) + \Delta$, which is $\delta + \exp(-\kappa)$-correct.*

Furthermore, π can be constructed as follows: Alice randomly picks a random string r of length Δ and sends r to Bob. Then, the parties expand r into randomness strings R_A, R_B and run π_0 assuming that randomness.

See Lemma 4.4 in [NN01], for a proof. We remark that the expansion of r into R_A, R_B in the above lemma is done in a non-uniform way: we assume the parties hold some function g that maps each r to some R_A, R_B. This non-uniformness assumption could be avoided if the parties assume to preshare a large amount of randomness (e.g., of size $2^\Delta \cdot (|R_A| + |R_B|)$; then g simply maps each one of the 2^Δ possible r's to some random pair (R_A, R_B)), or when the parties are allowed to communicate a large amount of information in a preprocessing step (in order to create this large bank of shared randomness).

As a corollary of the above Lemma and Theorem 9, our main result (Theorem 2) holds also for randomized protocols.

Theorem 19. *Given $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\nu$, κ a security parameter, and $\varepsilon \in (0, 1/2)$, let π_0 be a randomized* $\exp(-\kappa)$*-correct protocol for f assuming noiseless channels. There exists a compiler mapping π_0 into a (semi-honest) statistically secure π that computes the same f over BSC_ε channels.*

It holds that $\mathtt{CC}(\pi) = \max_r \tilde{O}(\mathrm{width}(BP_r))\mathtt{CC}(\pi_0) + O(\kappa + \nu + \log(n))$, where BP_r denotes the branching program of the deterministic protocol $\pi_r = \pi_0(\cdot\,; r)$ obtained by setting the randomness of π_0 to $r = (r_A, r_B)$.

Proof (sketch). Let π_0 denote a randomized protocol that $\exp(-\kappa)$-correctly computing some f over noiseless channels. Let $\tilde{\pi}_0$ denote the protocol obtained by applying Lemma 18 on the input protocol π_0. Specifically, Let $\Delta = O(\kappa + \log(n) + \nu)$, and assume the parties share a function $g : \{0,1\}^\Delta \to R_A \times R_B$ where R_A and R_B are the domain of Alice and Bob randomness in π_0, respectively. We can describe $\tilde{\pi}_0$ as in the following way.

The protocol $\tilde{\pi}_0$:

1. Alice samples a random string r of length Δ.
2. Alice sends r to Bob.
3. Execute π_0 using the randomness $g(r) \in R_A \times R_B$.

Note that by Lemma 18, $\tilde{\pi}_0$ is $\exp(-\kappa)$-correct assuming noiseless channels.

We now augment $\tilde{\pi}_0$ into a protocol π that assumes BSC_ε channels.

The protocol π:

1. Alice samples a random string r of length Δ.
2. Alice encodes r via a standard error correcting code (Lemma 6) and communicates $\mathsf{ECC}(r)$ to Bob.
3. Bob decodes the received message and obtains r'.
4. Let $\pi_r = \pi_0(\cdot\,; g(r))$ denote the deterministic protocol obtained from π_0 executed with randomness $g(r) \in R_A \times R_B$. Let π'_r denote the protocol obtained from π_r by applying Theorem 9. Similarly define $\pi'_{r'}$. Alice executes π'_r and Bob executes $\pi'_{r'}$.

It is easy to see that the above protocol is statistically secure using the following simulator(s). The simulator Sim_A for Alice in π_0 is defined as follows.

- Submit input x to the ideal functionality f, and let o denote the received output.
- Pick r as in Lemma 18. Run Sim^r_A simulating A's view in π'_r, with o as the output of f. Let $View^r_A$ denote the output of the simulator.
- Output $((\mathsf{ECC}(r), View^r_A), o)$.

The simulator for Bob is identical (except that r' is a received message, rather than part of its randomness). We sketch the proof that the distribution $Sim_A(1^k, x, f(x,y) = o)$ produced by the above simulator is $\exp(-\kappa)$-close to $(VIEW_A, OUT_B)$.

First note that note that that $|\mathsf{ECC}(r)| = O_\varepsilon(\kappa + \log(n) + \nu)$ so the probability that Bob fails to decode $r' = r$ is at most $\exp(-\kappa)$. Hence, with probability at least $1 - \exp(-\kappa)$ both parties execute the same protocol π'_r.

Next, observe that for at least a $1 - \exp(-\kappa)$ fraction of r's, the protocol π_r is correct. The reason is that any specific π_r is deterministic and that $\tilde{\pi}_0$, which can be seen as a distribution over these deterministic protocols, is $\exp(-\kappa)$-correct. The latter holds by Lemma 18 since π_0 is $\exp(-\kappa)$-correct to begin with, which implies that $\tilde{\pi}_0$ is $\exp(-\kappa) + \exp(-\kappa) = \exp(-\kappa)$ – correct.

For any such "good" selection of r, we have $out_B = f(x,y)$, and thus by the properties of π'_r and its simulator Sim^r_A, we get that $VIEW_A$ is $\exp(-\kappa)$-close to $View^r_A$ of the simulation.

Since all failure events are of magnitude of $\exp(-\kappa)$ and since otherwise the views are $\exp(-\kappa)$-close, we obtain a maximal overall distance of at most

$$1 \cdot \exp(-\kappa) + \exp(-\kappa)(1 - \exp(-\kappa)) = \exp(-\kappa).$$

5 Extension to other unreliable channels

In this section we argue that our results extend to other types of unreliable channels. Specifically, we consider elastic-channels.

Definition 20. *For any $0 \le \beta \le \alpha \le 1/2$ the (α,β)-elastic channel $\mathtt{Ch}$ is the channel obtained by the concatenation of two BSC channels with parameters β and γ such that $\beta(1-\gamma) + (1-\beta)\gamma = \alpha$,*

$$\mathtt{Ch}(b) = \mathsf{BSC}_\gamma(\underbrace{\mathsf{BSC}_\beta(b)}_{y}).$$

For any bit $b \in \{0,1\}$ the receiver learns $\mathtt{Ch}(b)$ unless the receiver is adversarial, in which case it learns y. Note that if the receiver is honest, the channel behaves equivalently to a BSC_α.

Our Coding results apply to a large family of (α,β)-elastic channels. Formally,

Theorem 21. *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\nu$, κ be a security parameter, and $0 < \beta < \alpha < 1/2$ such that $\alpha < \left(1 + (4\beta(1-\beta))^{-1/2}\right)^{-1}$. Let π_0 be a deterministic correct protocol for evaluating f over noiseless channels, and let BP_0 denote a branching program representation of π_0. Then, there exists a compiler mapping π_0 into a (semi-honest) statistically secure protocol π over an (α,β)-elastic channel with simulation error $2^{-\kappa^c}$ for some constant c. The communication complexity of the obtained protocol is $\mathtt{CC}(\pi) = \tilde{O}(\mathrm{width}(BP_0)) \cdot \mathtt{CC}(\pi_0) + O(\kappa)$.*

Proof. (sketch) Our coding scheme construction (Figures 4 and 5) is composed of two main parts: (a) generating OT-triplets, and (b) simulating π_0 over a noisy channel. Note that part (b) remains the same regardless of the channel being elastic — compared to a BSC_α, the error probability of each bit only decreases. This can only help the coding scheme to perform better. In fact, part (b) works perfectly even if the channel is completely noiseless.

However, we need to argue how to generate OT-triplets in a secure way with linear overhead over elastic channels. This is a more difficult task than generating OT-triplets from a BSC. Indeed, when the channel is elastic, the adversary possibly learns more information than it would have learned if the channel was a BSC. This extra information may jeopardize the security of the OT implementation. Luckily, the following theorem proves that if $0 < \beta < \alpha < 1/2$ such that $\alpha < \left(1 + (4\beta(1-\beta))^{-1/2}\right)^{-1}$, then ℓ instances of OT can be computed from an (α, β)-elastic channel with communication $O(\ell)$.

Theorem 22 ([KMS16]). *For $0 < \beta < \alpha < 1/2$ if $\alpha < \left(1 + (4\beta(1-\beta))^{-1/2}\right)^{-1}$, then, there exists a protocol that securely realizes OT^ℓ by accessing an (α, β)-elastic channel for $O(\ell)$ times. The protocol has simulation error of at most $2^{-\ell^c}$, for some constant c.*

The protocol guaranteed by the above theorem can be plugged directly into our construction to complete the proof of the theorem.

A similar result holds for randomized protocol using the same argument of the above and Section 4.

References

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edn., 2009.

[Bea91] D. Beaver. Perfect privacy for two-party protocols. *Proceedings of DIMACS Workshop on Distributed Computing and Cryptography*, vol. 2, pp. 65–77, 1991.

[Bea95] D. Beaver. Precomputing oblivious transfer. D. Coppersmith, ed., *Advances in Cryptology — CRYPTO' 95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings*, pp. 97–109, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[BKN14] Z. Brakerski, Y. T. Kalai, and M. Naor. Fast interactive coding against adversarial noise. *J. ACM*, 61(6):35:1–35:30, 2014.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *42nd FOCS*, pp. 136–145, IEEE Computer Society Press, 2001.

[CPT13] K.-M. Chung, R. Pass, and S. Telang. Knowledge-preserving interactive coding. *Proceedings of the 54th annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 449–458, 2013.

[CT06] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2nd edn., 2006.

[Cré97] C. Crépeau. Efficient cryptographic protocols based on noisy channels. W. Fumy, ed., *Advances in Cryptology — EUROCRYPT '97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings*, pp. 306–317, 1997.

[CK88] C. Crépeau and J. Kilian. Achieving oblivious transfer using weakened security assumptions. *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pp. 42–52, 1988.

[DFMS04] I. Damgård, S. Fehr, K. Morozov, and L. Salvail. Unfair noisy channels and oblivious transfer. M. Naor, ed., *Theory of Cryptography, Lecture Notes in Computer Science*, vol. 2951, pp. 355–373, 2004.

[EGH16] K. Efremenko, R. Gelles, and B. Haeupler. Maximal noise in interactive communication over erasure channels and channels with feedback. *IEEE Transactions on Information Theory*, 62(8):4575–4588, 2016.

[Gel17] R. Gelles. Coding for interactive communication: A survey. *Foundations and Trends® in Theoretical Computer Science*, 13(1–2):1–157, 2017.

[GH17] R. Gelles and B. Haeupler. Capacity of interactive communication over erasure channels and channels with feedback. *SIAM Journal on Computing*, 46(4):1449–1472, 2017.

[GHK+16] R. Gelles, B. Haeupler, G. Kol, N. Ron-Zewi, and A. Wigderson. Towards optimal deterministic coding for interactive communication. *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1922–1936, 2016.

[GSW15] R. Gelles, A. Sahai, and A. Wadia. Private interactive communication across an adversarial channel. *IEEE Transactions on Information Theory*, 61(12):6860–6875, 2015.

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

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pp. 218–229, ACM, New York, NY, USA, 1987.

[GI05] V. Guruswami and P. Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. on Information Theory*, 51(10):3393–3400, 2005.

[Hae14] B. Haeupler. Interactive Channel Capacity Revisited. *Proceedings of the IEEE Symposium on Foundations of Computer Science*, FOCS '14, pp. 226–235, 2014.

[HIKN08] D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen. OT-combiners via secure computation. R. Canetti, ed., *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings*, pp. 393–411, 2008.

[HKN+05] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On robust combiners for oblivious transfer and other primitives. R. Cramer, ed., *EUROCRYPT 2005*, *LNCS*, vol. 3494, pp. 96–113, Springer, 2005.

[IKM+13] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky. On the power of correlated randomness in secure computation. *TCC*, pp. 600–620, 2013.

[IKO+11] Y. Ishai, E. Kushilevitz, R. Ostrovsky, M. Prabhakaran, A. Sahai, and J. Wullschleger. Constant-rate oblivious transfer from noisy channels. P. Rogaway, ed., *Advances in Cryptology – CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pp. 667–684, 2011.

[IPS08] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer – efficiently. D. Wagner, ed., *Advances in Cryptology – CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pp. 572–591, 2008.

[KMS16] D. Khurana, H. K. Maji, and A. Sahai. Secure computation from elastic noisy channels. M. Fischlin and J.-S. Coron, eds., *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pp. 184–212, 2016.

[Kil88] J. Kilian. Founding crytpography on oblivious transfer. *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 20–31, ACM, New York, NY, USA, 1988.

[KR13] G. Kol and R. Raz. Interactive channel capacity. *STOC '13: Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pp. 715–724, 2013.

[Kus89] E. Kushilevitz. Privacy and communication complexity. *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 416–421, IEEE Computer Society, 1989.

[MPW07] R. Meier, B. Przydatek, and J. Wullschleger. Robuster combiners for oblivious transfer. S. P. Vadhan, ed., *TCC 2007*, *LNCS*, vol. 4392, pp. 404–418, Springer, 2007.

[NN01] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pp. 590–599, ACM, New York, NY, USA, 2001.

[Rab81] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[Rot06] R. Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[Sch92] L. J. Schulman. Communication on noisy channels: a coding theorem for computation. *Foundations of Computer Science, Annual IEEE Symposium on*, pp. 724–733, 1992.

[Sha48] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. Originally appeared in *Bell System Tech. J.* 27:379–423, 623–656, 1948.

[Spi95] D. A. Spielman. Linear-time encodable and decodable error-correcting codes. *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pp. 388–397, ACM, New York, NY, USA, 1995.

[Wul09] J. Wullschleger. Oblivious transfer from weak noisy channels. O. Reingold, ed., *Theory of Cryptography: 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pp. 332–349, 2009.

[Yao82] A. C.-C. Yao. Protocols for secure computations (extended abstract). *23rd FOCS*, pp. 160–164, IEEE Computer Society Press, 1982.

Appendix

A Extended Notions of Security

Weak security. Here we define a relaxed form of security, where the simulator is allowed to modify the input of the honest party that it sends to the ideal functionality. In a sense, the definition allows to protocol to output an incorrect value due to channel noise. This definition is similar to that of malicious security, but is a relaxation of Definition 5, since it only concerns real-world adversaries that follow the protocol (no guarantees are provided for stronger adversaries).

Definition 23 (weak statistical security over Ch). *Let $\pi = (\pi_A, \pi_B)$ denote a protocol for evaluating a function $f(x, y) = (f_A(x, y), f_B(x, y))$.*

We say that π is a weakly statistically secure protocol for computing $f(x, y)$ when running over Ch *against semi-honest adversaries if there exist (possibly inefficient) simulators Sim_A, Sim_B for Alice and Bob, respectively, such that for all x, y, and κ a security parameter there exist distributions X', Y' over $\{0,1\}^n$ so that*

$$(Sim_A(1^\kappa, x, f_A(x', y)), f_B(x', y)) \approx_{\exp(-\kappa)} (VIEW_A, OUT_B), \text{ and}$$
$$(Sim_B(1^\kappa, y, f_B(x, y')), f_A(x, y')) \approx_{\exp(-\kappa)} (VIEW_B, OUT_A).$$

In the above x', y' are sampled from X', Y' respectively, and the two instance of x' (y') are the same one.

Channel-transparent security. Here we define the notion of channel-transparent security. This notion is an augmentation of either Definition 5 or Definition 23. For a secure protocol over Ch (according to one of the above definitions) we say it is secure with *channel transparency*, if $VIEW_A = (x_A, r_A, T_A)$, $VIEW_B = (x_B, r_B, T_B)$ in the respective security definition are augmented into $VIEW'_A, VIEW'_B$ to include *both* T_A and T_B. That is, $VIEW'_A = (x_A, r_A, T_A, T_B), VIEW'_A = (x_A, r_A, T_B, T_A)$.

Conceptually, this definition implies that the security holds even if we assume that a party could see also the messages received by the other party, i.e., it learns the noise introduced by the channel. This means, in particular, that the protocol does not *rely* on the channel's noise for its own security. Note that assuming only BSC channels, such a notion of security cannot be achieved for many functionalities, e.g., OT [CK88], and additional assumptions are needed (in our case, correlated randomness).

Secure computation with correlated randomness. We also consider a variant of secure function evaluation, where the parties have access to a randomized ideal functionality g with no inputs *before* they get their inputs. More precisely, there is a preprocessing phase (before they receive their inputs to f), where the parties make a single call to the functionality g, which samples a random variable (cr_A, cr_B), and outputs cr_A to Alice, and cr_B to Bob. See, e.g. [IKM+13, Bea95] for more details. We define $view_A$ ($view_B$) as before, except that we append p_A (p_B) to it.

We say that a protocol π with correlated randomness specified by g, computes f with statistical security, if it satisfies Definition 5 where the definition of a party's view is augmented to include cr_A, cr_B as described above.

B Detailed Proofs

B.1 Proof of Proposition 10

In this section we complete the proof of Proposition 10. Let us recall its statement:

Before proving the proposition, let us define a multiplexer gate (mux). The gate is parametrized by w denoting the number of inputs and ℓ denoting the length of each input. The gate also has a selector input (of length $\log w$ bits). The output of the gate is the ℓ-bit input whose index number is given by the selector inputs. Formally,

Definition 24. *A multiplexer* $MUX^{w,l} : \{1,\ldots,w\} \times (\{0,1\}^\ell)^w \to \{0,1\}^\ell$ *takes an array* A *of* w ℓ*-bit strings as its second argument, and an index* i *as its first argument, and outputs* $MUX^{w,l}(i,A) = A[i]$.

Proof. (**Proposition 10**) Assume BP_0 contains t layers and let $w = \text{width}(BP_0)$; assume for simplicity of notation that each layer V_i for $i \geq 2$ has width exactly w, and denote the nodes in the i-th layer by $V_i = \{v_i^1, \ldots, v_i^w\}$ (this is wlog., as we can add unreachable nodes to each layer). For any node v assigned to Alice (resp., Bob) we let $next(v) = f_v(x)$ (resp., $next(v) = f_v(y)$). Recall that f_v is the transition function; cf. Definition 7.

In order to prove the claim we need to show that there exist a circuit C_0 and translation functions τ_A, τ_B, and τ_{out} with the following properties:

1. τ_A and τ_B encode the inputs x and y (to π_0) of Alice and Bob, respectively, into inputs $\tau_A(x)$ and $\tau_B(y)$ for C_0.
2. If $v_{out} = C_0\left(\tau_A(x), \tau_B(y)\right)$—i.e., v_{out} is the output of C_0 on the encoded values—then $\tau_{out}(v_{out}) = f(x,y)$. In other words, τ_{out} "decodes" the output of C_0 back to the corresponding output of π_0.

The circuit C_0 sequentially emulates the path taken by BP_0 on (x,y), transition-by-transition. That is, the node in the first layer V_1 is by definition $u_1 = \mathsf{start}$. The node u_2 at the second layer V_2, is $u_2 = next(\mathsf{start})$. The node u_3 reached at the third layer $V_3 = \{v_3^1, \ldots, v_3^w\}$ equals $next(u_2)$, and so on. Finally, the protocol's output is the name of the layer-t node reached, which uniquely determines π_0's output on x, y (by our definition of protocol BP). Formally, $\tau_{out}(v_t^i) = i$

Let us show how the sequence $u_1, \ldots, u_t$ can be computed from the inputs (x,y). Taking u_2 as an example: it can be evaluated using u_1 and $NEXT_1 = \{next(v_2^1), \ldots, next(v_2^w)\}$ by computing the gate $\text{MUX}^{w,\log w}(u_2, NEXT_1)$. Generally, for any $1 < i \leq t$, the value u_i is the output of the gate $\text{MUX}^{w,\log w}(u, NEXT_{i-1})$, where $NEXT_{i-1} = \{next(v_i^1), \ldots, next(v_i^w)\}$. From the above we get that the input to the circuit C_0 are the values $NEXT_i$ for all $i < t$. Formally, let $\tau_A(x) = (next(v_i^1), \ldots, next(v_i^w))_{1 \leq i \leq t, \text{i is odd}}$, and $\tau_B(x) = (next(v_i^1), \ldots, next(v_i^w))_{1 \leq i \leq t, \text{i is even}}$.

It is easy to verify that $C(x,y) = \tau_{out}\left(C_0\left(\tau_A(x), \tau_B(y)\right)\right) = f(x,y)$ as required.

Last, we consider the size of the resulting circuit C_0. Clearly, an instance of $\text{MUX}^{w,\log w}$ can be implemented by $w \cdot \text{poly}(\log w)$ NAND gates, and C_0 contains t such gates. Hence the size of C_0 is given by

$$|C_0| = t \cdot w \cdot \text{poly}(\log w) = |BP_0| \cdot \text{polylog}(\text{width}(BP_0)).$$

B.2 Proof of Theorem 9

Proof. (**Theorem 9**) Let BP_0 be the branching program to be simulated. Proposition 10 asserts we can convert BP_0 into a circuit C_0 of size $|C_0| = |BP_0|\text{polylog}(\text{width}(BP_0))$, and that there are functions τ_A, τ_B, τ_{out} such that for any x, y we have $BP_0(x,y) = \tau_{out}(C_0(\tau_A(x), \tau_B(y))$.

Proposition 11 asserts that the above circuit C_0 can by evaluated on inputs $\tau_A(x)$ and $\tau_B(y)$ respectively, over BSC_ε channel by communicating $O(|C_0| + \kappa) = O(|BP_0|\text{polylog}(\text{width}(BP_0)) + \kappa)$ bits, in a statistically secure manner (assuming semi-honest adversaries). Note that Alice and Bob can compute $\tau_A(x)$ and $\tau_B(y)$ from their inputs independently without communication; Similarly, both can decode C_0's output via τ_{out} to obtain an output for the evaluating BP_0.

Finally, via Eq. (1) we get the claimed communication complexity

$$\begin{aligned} O(|BP_0|\text{polylog}(\text{width}(BP_0))) &= O\big(\text{depth}(BP_0)\text{width}(BP_0) \cdot \text{polylog}(\text{width}(BP_0))\big) \\ &= CC(\pi_0)\tilde{O}(\text{width}(BP_0)). \end{aligned}$$

B.3 Proof of Lemma 13

We devise a suitable simulator for Alice, Sim_A. It is easy to verify that the simulator proves weak statistical channel-independent security of the protocol. Since the protocol is defined in the model where trusted

correlated randomness is preshared, that randomness (cr_A, cr_B respectively) is also part of the parties' view, see, e.g., [IKM+13] for more precise definitions and discussion on security in the presence of preshared correlated randomness. For convenience of presentation of our main protocol later on, we slightly strengthen the definition discussed above, and fix the corrupted party's correlated randomness as part of the simulator's input. We require that the simulator works conditioned on any value of the corrupted party's correlated randomness.

Generating the distribution $Sim_A(1^\kappa, b, o)$, given the input bit b and assuming the ideal functionality $\mathcal{F}_{OT}$

1. Sample $b', x'_{b'} \in \{0,1\}$ uniformly and independently, as Alice's correlated randomness cr_A.
2. Pick $\delta_1, \delta_2, \delta_3 \sim Ber(\varepsilon)$ independently at random; The δ_i's represent the channel's added bits by order of delivery.
3. Query the ideal functionality on the input $x' = b + \delta_1$, to obtain the output bit $o = \mathcal{F}_{OT}(x', (x_0, x_1))$.
4. Pick a random bit r. Let $m_{in} = (o + x'_{b'} + \delta_2, r + \delta_3)$ denote Bob's messages if $x' = 0$, and $m_{in} = (r + \delta_3, o + x'_{b'} + \delta_2)$ otherwise.
5. Output $(\text{input}, cr_A, r_A, T_A, T_B) = (b, (b', x'_{b'}), \emptyset, m_{in}, x' + b')$ as Alice's view.[10]

We proceed with a simulator for Bob, Sim_B, which is even simpler.

Generating the distribution $Sim_B(1^\kappa, (x_0, x_1))$ given inputs (x_0, x_1) and assuming the ideal functionality $\mathcal{F}_{OT}$

- Sample $x'_0, x'_1 \in \{0,1\}$ uniformly and independently for cr_B.
- Sample $\delta_1, \delta_2, \delta_3 \sim Ber(\varepsilon)$ independently at random. The δ_i's represent the channel's added bits by order delivery.
- Sample a random bit $m_{in} \in \{0,1\}$, to serve as Alice's sent message, and set $c' = m_{in} + \delta_1$ as Bob's received message.
- Query the ideal functionality using the inputs $(x_0 + \delta_1(x'_0 + x'_1) + \delta_2, x_1 + \delta_1(x'_0 + x'_1) + \delta_3)$.
- Output $\big(\text{input}, cr_B, r_B, T_B, T_A\big) = \big((x_0, x_1), (x'_0, x'_1), \emptyset, c', (x_0 + x'_{c'} + \delta_2, x_1 + x'_{1-c'} + \delta_3)\big)$ as Bob's view.[11]

Analysis. Let us write the real-world distribution $(VIEW_A, OUT_B)$. Note that $OUT_B = \perp$. Since we require channel-transparency in the correlated-randomness setting, we have

$$\begin{aligned} VIEW_A &= (\text{input}, cr_A, r_A, T_A, T_B) \\ &= (b, (b', x'_{b'}), \emptyset, (x_0 + x'_{c'} + \delta_2, x_1 + x'_{1-c'} + \delta_3), c' = b + b' + \delta_1) \end{aligned}$$

where $\delta_1, \delta_2, \delta_3 \sim Ber(\varepsilon)$. It is quite straightforward to check that this distribution is identical to $Sim_A(1^\kappa, b, o)$ with $o = \mathcal{F}_{OT}(x', (x_0, x_1))$ as described above: the input, and randomness are identically distributed, as well as the message Bob receive (up to adding a noise δ_1 which is distributed according to the BSC_ε noise).

For T_A, first note that the noise δ_2, δ_3 in the simulated distribution behaves according to a BSC_ε and is similar to the real-world noise. Next, note that in the real world if $b + \delta_1 = 0$ then $c' = b'$ and $T_A = (x_0 + x'_{b'} + \delta_2, x_1 + x'_{1-b'} + \delta_3)$. In this case, in the simulated T_A we have $x' = 0$ and the output is $T_A = (o + x'_{b'} + \delta_2, r + \delta_3)$ with $o = \mathcal{F}_{OT}(b + \delta_1, (x_0, x_1)) = x_0$. Note that indeed $x'_{1-b'}$ is uniform conditioned on the view of Alice. Symmetrically, if $b + \delta_1 = 1$ then in the real-world we have $c' = 1 - b'$ thus $T_A = (x_0 + x'_{1-b'} + \delta_3, x_1 + x'_{b'} + \delta_2)$ while the simulated T_A is $(r + \delta_3, o + x'_{b'} + \delta_2)$ with $o = x_1$; again, $x'_{1-b'}$ is uniform conditioned on the view of Alice, while the other coordinate is distributed identically.

[10] We omit Alice's outgoing messages, as they are determined by her input, randomness and incoming messages.
[11] We omit Bob's outgoing messages, as they are determined by his input, randomness and incoming messages.

Next consider Bob's side. We have

$$VIEW_B = \Big(\text{input} = y, cr_B = (x'_0, x'_1), r_B = \emptyset, \\ T_B = c' = b + b' + \delta_1, \\ T_A = (x_0 + x'_{c'} + \delta_2, x_1 + x'_{1-c'} + \delta_3)\Big) \tag{2}$$

and,

$$OUT_A = \begin{cases} x_0 + x'_{c'} + x'_{b'} + \delta_2 & b = 0 \\ x_1 + x'_{1-c'} + x'_{b'} + \delta_3 & b = 1 \end{cases} \tag{3}$$

Note that conditioned on Bob's inputs and randomness, b', x'_0, x'_1 are all uniform independent bits. Then, it is easy to verify that the simulator's output is identically distributed as $VIEW_B$.

As for the output, note that Alice's real-world output is distributed as follows: If $b = 0$ Alice outputs $x_0 + x'_{b'+\delta_1} + x'_{b'} + \delta_2$; if $b = 1$ Alice outputs $x_1 + x'_{b'+\delta_1} + x'_{b'} + \delta_3$. Hence, let $err = \delta_1(x'_0 + x'_1)$ and define $y' = (x_0 + err + \delta_2, x_1 + err + \delta_3)$. It is clear that the ideal-world experiment computes $\mathcal{F}_{OT}(b, y')$ which produces identically distributed output for Alice as the real-world computation.

B.4 Proof of Lemma 14

We give here a simulator for the protocol. Recall that due to the weakly, channel-transparant property, we need to simulate, for both views, both transcripts.

We begin with describing how to generate Alice's view given access to an ideal NAND functionality, $\mathcal{F}_{\text{NAND}}$. Since the protocol basically only calls Π-OT$_\varepsilon$ and doesn't send any additional messages, the simulation is rather straightforward, however, we need to be careful about simulating the correct output distribution, since the Π-OT$_\varepsilon$ protocol over a BSC_ε may give the wrong output.

Specifically, for each OT instance, with probability ε (i.e., when the message from Bob to Alice is flipped) Alice receives a flipped version of requested bit x_b. Moreover, with probability ε (i.e., when the message form Alice to Bob is flipped) Alice "gets" a uniformly random bit as the OT output. In this case, Alice also gets from Bob the value of his other OT input. Alice ignores this value, however the simulator must generate it correctly in order to maintain the same view's distribution.

Corresponding to the above, the simulator can simulate which message gets corrupted and flip the output bit of the simulated OT with probability ε, or generate a random bit as the output, accordingly.

Generating the distribution $Sim_A(1^\kappa, (a_1, b_1), c_1)$, given the inputs a_1, b_1 and assuming an ideal NAND functionality $\mathcal{F}_{\text{NAND}}$

1. Sample $\delta_1^1, \delta_2^1, \delta_3^1 \sim Ber(\varepsilon)$ and $\delta_1^2, \delta_2^2, \delta_3^2 \sim Ber(\varepsilon)$ that will serve as the noise in the implementation of the first and second OT realization, respectively.
2. Query $\mathcal{F}_{\text{NAND}}$ using the inputs (a_1, b_1) to obtain the output bit c_1.
3. Sample $o_1 \in \{0, 1\}$ uniformly at random and set $o_2 = c_1 + o_1 + 1 + a_1 b_1$.
4. Activate Alice's simulator for Π-OT$_\varepsilon$ on the input a_1. In order to generate Bob's messages, assume Bob holds the input (o_1, u^1) with a uniform bit u^1 if a_1, or the input (u^1, o_1) otherwise.[12] Use the above $\delta_1^1, \delta_2^1, \delta_3^1$ for simulating the channel's noise in Π-OT$_\varepsilon$.
5. Activate Alice's simulator for Π-OT$_\varepsilon$ on the input b_1; Similar to the above OT instance, Assume Bob's inputs are (o_2, u^2) if $b_1 = 0$ or (u^2, o_2) otherwise. Here u^2 is a uniform random bit. Use $\delta_1^2, \delta_2^2, \delta_3^2$ for simulating the channel's noise in Π-OT$_\varepsilon$.
6. Output $(\text{input}, cr_A, r_A, T_A, T_B)$ where $input = (a_1, b_1)$, and cr_A, r_A, T_A, T_B as given by the output of the two instances of OT simulation.

The simulation for Bob is as follows.

Generating the distribution $Sim_B(1^\kappa,(a_2,b_2),c_2)$, given the inputs a_2,b_2 and assuming an ideal NAND functionality $\mathcal{F}_{\text{NAND}}$

1. Sample $\delta_1^1,\delta_2^1,\delta_3^1 \sim Ber(\varepsilon)$ and $\delta_1^2,\delta_2^2,\delta_3^2 \sim Ber(\varepsilon)$ that will serve as the noise in the implementation of the first and second OT realization, respectively.
2. Sample $x_0'^1,x_1'^1,x_0'^2,x_1'^2$ uniform bits as Bob's correlated randomness.
3. Set $\rho^1=(\delta_2^2+\delta_3^2)$ and $\rho^2=(\delta_2^1+\delta_3^1)$. Query $\mathcal{F}_{\text{NAND}}$ using the inputs $(a_2+\rho^1,b_2+\rho^2)$. Let c_2 denote the output given to Bob.
4. Sample a random bit r_1 and set $r_2=r_1+c_2+\mathsf{flip}$, where $\mathsf{flip}=\delta_1^1(x_0'^1+x_1'^1)+\delta_1^2(x_0'^2+x_1'^2)+\delta_2^1+\delta_2^2+\rho^1\rho^2+a_2\rho^2+\rho^1 b_2$ describes the flipping of the output due to channel noise.
5. Activate Bob's simulator for Π-OT$_\varepsilon$ on the input (r_1,b_2+r_1); Use $\delta_1^1,\delta_2^1,\delta_3^1$ for simulating the channel's noise in Π-OT$_\varepsilon$, and $x_0'^1,x_1'^1$ as Bob's correlated randomness.
6. Activate Bob's simulator for Π-OT$_\varepsilon$ on the input $(a_2b_2+r_2,a_2b_2+a_2+r_2)$; Use $\delta_1^2,\delta_2^2,\delta_3^2$ for simulating the channel's noise in Π-OT$_\varepsilon$, and $x_0'^2,x_1'^2$ as Bob's correlated randomness.
7. Output $(\text{input},cr_B,r_B,T_A,T_B)$ where $input=(a_2,b_2)$, and cr_B,r_B,T_A,T_B as given by the output of the two instances of the OT simulation.

Analysis. Let us now analyze this simulator and show that along with c_2 (the output of $\mathcal{F}_{\text{NAND}}$ for Bob) it generates a distribution close to $(VIEW_A,OUT_B)$ for the protocol. First, note that $VIEW_A$ is merely two independent activations of Π-OT$_\varepsilon$ on independent inputs. Indeed, Bob's inputs in the first activation are masked with some random bit r_1 while the second with an independent bit r_2. Moreover, the two instances use different pairs of correlated randomness, and their transcripts are also not correlated. Formally, we have

$$\begin{aligned} VIEW_A = \Big(&\text{input}=(a_1,b_1), cr_A=(b'^1,x'^1_{b'^1},b'^2,x'^2_{b'^2}), r_A=\emptyset,\\ &T_A=\big((r_1+x'^1_{c'^1}+\delta_2^1, r_1+b_2+x'^1_{1-c'^1}+\delta_3^1),\\ &\qquad (r_2+a_2b_2+x'^2_{c'^2}+\delta_2^2, r_2+a_2b_2+a_2+x'^2_{1-c'^2}+\delta_3^2)\big)\\ &T_B=(c'^1,c'^2)=(a_1+b'^1+\delta_1^1, b_1+b'^2+\delta_1^2)\Big) \end{aligned} \tag{4}$$

We use superscripts to denote the instance number of the OT. Next, note that OUT_B is a uniform random bit.

$$OUT_B = c_2 = r_1+r_2, \tag{5}$$

however, some care should be taken as this bit is correlated with $VIEW_A$, e.g., it holds that

$$c_1+c_2 = 1+(a_1+a_2)(b_1+b_2)$$

Yet, due to the security of the OT, at each activation Alice may learn only one input of Bob (the other message looks like a random bit to her). The input that Alice can learn (even if she doesn't use it) is determined by $a_1+\delta_1^1$ in the first activation, and by $b_2+\delta_1^2$ in the second.

It is easy to verify that the messages generates by the simulator agree with the transcript part of $VIEW_A$, recalling that Alice always holds only one of $x'_{c'}$ and $x'_{1-c'}$, which makes one message in each pair to be uniformly distributed in an independent way of all the other information.

As for generating the correct output, by querying $\mathcal{F}_{\text{NAND}}$ on the (correct, $x'=x$) inputs (a_1,b_2) we guarantee that if Bob uses his inputs $y=(a_2,b_2)$, then Bob's output v given by $\mathcal{F}_{\text{NAND}}$ (corresponding to

[12] That is, the simulated messages Alice receives from Bob are either the messages $(o_1+x'^1_{b'^1}+\delta_2^1, u^1+\delta_3^1)$ or $(u^1+\delta_2^1, o_1+x'^1_{b'^1}+\delta_3^1)$ for $a_1=0$ and $a_1=1$, respectively. $x'^1_{b'^1}$ denotes the correlated randomness $x'_{b'}$ of this instance.

$f_B(x', y)$ in Definition 23) satisfies $v + c_1 = 1 + (a_1 + a_2)(b_1 + b_2)$, hence

$$v = (o_1 + o_2 + 1 + a_1 b_1) + 1 + (a_1 + a_2)(b_1 + b_2) \tag{6}$$
$$= o_1 + o_2 + a_1 b_2 + a_2 b_1 + a_2 b_2. \tag{7}$$

From the simulator definition it holds that $o_1 = r_1$ if $a_1 = 0$ or $o_1 = r_1 + b_2$ otherwise; Similarly, $o_2 = r_2 + a_2 b_2$ if $b_1 = 0$ or $o_2 = r_2 + a_2 b_2 + a_2$ otherwise. Then, v equals

$$v = \begin{cases} r_1 + r_2 + a_1 b_2 + a_2 b_1 \\ r_1 + r_2 + a_1 b_2 + a_2 b_1 + b_2 \\ r_1 + r_2 + a_1 b_2 + a_2 b_1 + a_2 \\ r_1 + r_2 + a_1 b_2 + a_2 b_1 + a_2 + b_2 \end{cases}$$

corresponding to the cases where (a_1, b_1) is $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$, respectively. Now, it is easy to verify that in all the four cases, $v = r_1 + r_2 = OUT_B$.

We continue to Bob's side, and show that the simulator (along with the output of $\mathcal{F}_{\text{NAND}}$ on a related input) generates a distribution close to $(VIEW_B, OUT_A)$. First we can derive Alice output by recalling that each of o_1, o_2 is distributed according to Eq. (3).

$$\begin{aligned} OUT_A &= 1 + a_1 b_1 + o_1 + o_2 \\ &= 1 + a_1 b_1 + \begin{cases} r_1 + \delta_1^1(x_0'^1 + x_1'^1) + \delta_2^1 & a_1 = 0 \\ r_1 + b_2 + \delta_1^1(x_0'^1 + x_1'^1) + \delta_3^1 & a_1 = 1 \end{cases} \\ &\qquad + \begin{cases} r_2 + a_2 b_2 + \delta_1^2(x_0'^2 + x_1'^2) + \delta_2^2 & b_1 = 0 \\ r_2 + a_2 b_2 + a_2 + \delta_1^2(x_0'^2 + x_1'^2) + \delta_3^2 & b_1 = 1 \end{cases} \end{aligned} \tag{8}$$

The corresponding Bob's view is given by

$$\begin{aligned} VIEW_B = \Big(&\text{input} = (a_2, b_2), \\ &cr_B = (x_0'^1, x_1'^1, x_0'^2, x_1'^2), \\ &r_B = (r_1, r_2), \\ &T_B = (c'^1 = a_1 + b'^1 + \delta_1^1, c'^2 = b_1 + b'^2 + \delta_1^2), \\ &T_A = ((r_1 + x'^1_{c'^1} + \delta_2^1, r_1 + b_2 + x'^1_{1-c'^1} + \delta_3^1), \\ &\qquad (r_2 + a_2 b_2 + x'^2_{c'^2} + \delta_2^2, r_2 + a_2 b_2 + a_2 + x'^2_{1-c'^2} + \delta_3^2))\Big) \end{aligned} \tag{9}$$

Clearly, the randomness and correlated randomness generated by the simulator are correctly distributed, and T_A, T_B are merely two (independent) instantiations of Π-OT$_\varepsilon$, using Eq. (2).

Finally, we show that output is distributed correctly. The simulator queries $\mathcal{F}_{\text{NAND}}$ on Bob's inputs $y' = (a_2 + \rho^1, b_2 + \rho^2)$ and receives an output c_2. If Alice queries the ideal NAND functionality on her inputs (a_1, b_2) she will receive a value v (this corresponds to $f_A(x, y')$ in Definition 23) such that

$$v + c_2 = 1 + (a_1 + a_2 + \rho^1)(b_1 + b_2 + \rho^2).$$

Recalling that the simulator sets $c_2 = r_1 + r_2 + \mathsf{flip}$, Alice's output is given by

$$\begin{aligned} v &= 1 + (a_1 + a_2 + \rho^1)(b_1 + b_2 + \rho^2) + r_1 + r_2 + \mathsf{flip} \\ &= 1 + a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2 + \rho^1 b_1 + \rho^1 \rho^2 + a_1 \rho^2 + a_2 \rho^2 + \rho^1 b_2 + r_1 + r_2 + \mathsf{flip} \\ &= 1 + a_1 b_1 + a_2 b_2 + r_1 + r_2 \\ &\quad + \begin{cases} 0 & a_1 = 0 \\ b_2 + \rho^2 & a_1 = 1 \end{cases} + \begin{cases} 0 & b_1 = 0 \\ a_2 + \rho^1 & b_1 = 1 \end{cases} \\ &\quad + \rho^1 \rho^2 + a_2 \rho^2 + \rho^1 b_2 + \mathsf{flip} \end{aligned}$$

substituting the values of $\rho^1, \rho^2, \mathsf{flip}$ and re-arranging the terms, we get

$$= 1 + a_1 b_1 + a_2 b_2 + r_1 + r_2 + \delta_1^1(x_0'^1 + x_1'^1) + \delta_1^2(x_0'^2 + x_1'^2) + \begin{cases} \delta_2^1 & a_1 = 0 \\ b_2 + \delta_3^1 & a_1 = 1 \end{cases} + \begin{cases} \delta_2^2 & b_1 = 0 \\ a_2 + \delta_3^2 & b_1 = 1 \end{cases}$$

which is distributed in a similar way to OUT_A as given by Eq. (8).

B.5 Proof of Theorem 16

We sketch a simulator for the combined circuit-evaluation construction. We present only a simulator for Alice. For simplicity, in the following we assume that ε is a sufficiently small constant. Otherwise, we encode each bit with a sufficiently long repetition code; this changes the communication overhead only by a constant.

B.5.1 Simulator for Alice For clarity, we split the simulation into three parts, corresponding to the initialization, evaluation, and output phase of the protocol Π_{2PC} (Figures 4 and 5). We describe the simulator assuming the circuit C has a single output wire (and let 'root' be the output gate); extending the simulation to multiple output wires is straightforward.

Initialization Phase.

1. . Produce a sharing of Alice's input: $s_1 = r_1$, $s_2 = r_1 + x$, where r_1 is a random string of length $|x|$. The share s_1 is kept by Alice as her local share, and s_2 simulating the share to be sent to Bob.
2. Simulate the incoming message from Bob, m_A, which is of the form $\mathsf{ECC}(\tilde{y}) + \mathcal{N}_{in}$, where $\tilde{y}$ is a random string of length $|y|$ and $\mathcal{N}_{in}$ is the noise string composed of i.i.d. $Ber(\varepsilon)$ of the corresponding length.
3. Run the simulator for the $\pi^{OT^{\ell'}}$ protocol (guaranteed by Theorem 12). Let $cr_A = (x'_{b'_1}, b'_1), \ldots, (x'_{b'_{\ell'}}, b'_{\ell'})$ denote the simulator's output of Alices' part of the correlated randomness.

Evaluation Phase. Interpret s_1, m_A as the shares of x, y, respectively, that Alice holds.

1. Initialize the local statistics V_A, V_B and the transcript T_A^π, T_B^π (all empty).[13] Note that these variables are the simulator's variables.
2. Simulate the ℓ'/v iterations of the scheme in Figure 3. In iteration i:
   (a) Simulate the exchange of statistics: Set $S_A = (T_A^\pi, V_A)$ compute $h_A = H_i(S_A, sr)$ and record h_A. Set $S_B = (T_B^\pi, V_B)$ compute $h_B = H_i(S_B, sr)$ and record $h'_B = h_B + \mathcal{N}_i$ where for any i we let $\mathcal{N}_i, \mathcal{N}'_i$ denote strings of i.i.d $Ber(\varepsilon)$ of the corresponding length.
   (b) based on S_A, h_A, h'_B determine whether Alice advances or backtracks during this iteration. Similarly, based on $S_B, h_B, h'_A = h_A + \mathcal{N}'_i$ decide whether Bob advances or backtracks during this iteration.
   (c) Simulate v evaluations of NAND gates that belong to the current iteration:
   Let Sim'_A denote the simulator for NAND$^{\Pi\text{-OT}_\varepsilon}$ for Alice, guaranteed in Lemma 14.
   For the j-th NAND evaluation in that chunk:
      i. Determine the output of the gate:
      if gate $\neq$ root: sample a random bit o_1^j as the output of the gate;
      if gate = root: query the ideal function to obtain C's output $\mathsf{out} = \mathcal{F}_C(x)$. Set o_1^j to be one share of the secret sharing of out, i.e., sample a random bit o_2^j and set $o_1^j = o_2^j + \mathsf{out}$.

[13] We denote the internal variables of π, which include the simulated transmitted and received messages of Alice in the coding scheme π by T_A^π (or T_B^π, respectively, for Bob) in order to distinguish them from the simulated variables T_A, T_B of the protocol Π_{2PC}.

ii. Use the NAND$^{\Pi\text{-OT}_\varepsilon}$ simulator to simulate the gate's evaluation:
Execute Sim'_A assuming the inputs (a_1^j, b_1^j) and the output o^j, where a_1^j, b_1^j is the value stored at the relevant input wires of the j-th gate; if the phase evaluation requires dummy values—use random inputs instead. In this simulation, use (a fresh set of) correlated randomness from cr_A generated in the **Initialization Phase** above (instead of sampling it afresh).

iii. Let $VIEW_{i,j}^{\text{NAND}}$ denote the simulator's output, for j-th NAND evaluation in the i-th iteration.

iv. Propagate the outcome to the next gate:
Let g_1^j be the simulated output of the j-th gate (note: this value may differ from o_1^j due to channel noise. Also note that g_1^j is a deterministic function $VIEW_{i,j}^{\text{NAND}}$). if this iteration of the coding scheme is progress (rather than backtracking), store g_1^j at the output wire of the j-th NAND gate in this iteration.

v. In case all the gates were evaluated yet the coding hasn't terminated yet, simulate sending and receiving zeros instead of evaluating further NAND gates.

(d) Derive the transcripts T_A^π, T_B^π from $VIEW_{i,1}^{\text{NAND}}, \ldots, VIEW_{i,v}^{\text{NAND}}$, and update V_A and V_B according to $T_A^\pi, T_B^\pi, h'_A, h'_B, h_A, h_B$, and whether Alice and Bob progress or rewind in the i-th iteration. Note that although $VIEW_{i,j}^{\text{NAND}}$ contains only the information *received* by Alice and by Bob, the information sent by the parties can be interpolated from the receives messages given knowledge of $\delta_1, \delta_2, \delta_3$ of each OT instance. It follows that T_A^π, T_B^π can be completely simulated.

3. Following the completion of the ℓ'/v prescribed iterations, use the a prefix of T_A^π corresponding to π_0 to extract Alice's output shares (if $|T_A^\pi|$ is too short output $\perp$).

Output Delivery phase. Set Alice's and Bob's output shares according to the value of the output wires[14]

$$so_A = \begin{cases} \perp & |T_A^\pi| < \ell \\ o_1^{root} & \text{otherwise} \end{cases} \qquad so_B = \begin{cases} \perp & |T_B^\pi| < \ell \\ o_2^{root} & \text{otherwise} \end{cases}$$

The simulator sets Bob's simulated message as $\mathsf{ECC}(so_B) + \mathcal{N}_{out}$ of the appropriate length. Again $\mathcal{N}_{out}$ is a noise string composed of i.i.d $Ber(\varepsilon)$ of the corresponding length. Recall that $o_2^{\text{root}} = \mathcal{F}_C(x) + o_1^{root}$. For completion, Alice's simulated sent message is $\mathsf{ECC}(so_A)$ (the simulator doesn't add it to T_A as it is a deterministic function of Alice's simulated view).

B.5.2 Analysis (sketch) We next argue that the above simulator is correct, addressing the more subtle issues. Note that we prove here the standard notion of semi-honest security (i.e., Definition 5). Specifically, since we don't require channel-transparency, the simulator needs to simulate only the messages Alice receives and it doesn't need to simulate messages received by Bob. Yet, the fact that the underlying sub-protocols (e.g., NAND$^{\Pi\text{-OT}_\varepsilon}$) are channel-transparent is pivotal in allowing simulating the coding scheme correctly, as its actions depend on both Alice's and Bob's received transcripts.

Moreover, since we don't settle for weak-security, the simulated outputs must correspond to the correct output, despite channel noise and despite the fact that the underlying sub-protocols may deviate from the correct output due to channel noise. This issue is solved due to the coding scheme which keeps track of the errors and rewinds the simulated protocol.[15] The coding scheme guarantees that the output is correct w.h.p, hence we can obtain (non-weak) security, despite using weak-secure primitives.

As above, we discuss the three stages of the simulation separately.

Initialization. In the real world protocol, messages consist a secret sharing of the inputs, hence, their distribution (conditioned on the view of the other party) is uniformly random. Recalling that each such string is encoded via ECC of appropriate length, and suffer from stochastic noise, independently per bit, it follows

[14] Here we assume a single output wire (namely, a single "root" gate), but the same holds if C has multiple outputs.

[15] Note: the coding scheme π and thus the simulator Sim_A of Π_{2PC} never rewind! It is only the underlying (noiseless) protocol, i.e., π_0, that is being rewound.

that the simulated message received by Alice is identically distributed as the messages received in the real world protocol.

Next, we argue that the proper correlated randomness is used. By the security of $OT^{\ell'}$ protocol (Theorem 12) the correlated randomness produced is $2^{-\Omega(\kappa)}$-close to a properly distributed vector of correlations (resulting in an increase of at most $\exp(-\kappa)$ in the final simulation error).

Evaluation. The simulation in this phase is composed of two sub-simulations: a simulation of the coding scheme, and simulation of the underlying noiseless protocol (i.e., of the GMW). Let us begin with the GMW simulation.

On its surface, the correctness follows from the correctness of the $\text{NAND}^{\Pi\text{-OT}_\varepsilon}$ simulator (Lemma 14). However, Lemma 14 guarantees the correctness of a single instance, while here we activate multiple instances of the protocol. This multiple activation still preserve privacy due to the fact it is being done sequentially gate-by-gate, and due to the following properties of the GMW simulation, that guarantee that each instance is independent of any previous instance:

— *(Property 1)* For each OT call during the evaluation step, Bob's inputs to the OT are random bits.

— *(Property 2)* For each OT call during the evaluation step, Bob's inputs to any specific OT are independent of all of his inputs to previous OT instances, the other OT of the same NAND evaluation, his input y, and of all messages previously received by him.

The above properties of GMW imply independence between different NAND evaluations—if they are not correlated, each evaluation can be generated separately using the $\text{NAND}^{\Pi\text{-OT}_\varepsilon}$ simulator. Indeed, following these properties we get that (i) Bob, as the sender, does not learn anything from the two OT-instances, so Alice's privacy is not compromised regardless. (ii) Alice either learns the correct output or a (fresh!) random bit, independent of all other bits she has seen in any *other* calls of OT. Hence, each gate can be simulated independently of the rest of the transcript so far (since the transcript is generated given that input, using the security of the $\text{NAND}^{\Pi\text{-OT}_\varepsilon}$ protocol, Lemma 14). We can thus employ a simple hybrid argument on the gates of the compiled circuit, where in each hybrid the evaluation of an additional gate is replaced by the simulation of this gate. Following the above analysis, any two neighboring hybrids will be statistically indistinguishable as long as none of the the above events occur. By a union bound (since the circuit is polynomially big and the probability of any of these events occurring is negligible) the total error probability is negligible.

Moreover, due to the rewinding caused by the coding scheme, we often need to simulate the evaluation of some gate multiple times. Using the same argument, any evaluation of a gate is independent not only of evaluations of previous gates, but also of previous evaluations of the same gate.

Next, we discuss simulation issues that stem from following the coding scheme. The main issue is how the simulator decides when the coding scheme needs to progress and when to rewind without simulating Bob's view. To this purpose, we required that the security of NAND evaluation (Lemma 14) as well as its $\Pi\text{-OT}_\varepsilon$ subprotocol (Lemma 13) is *channel-transparant*, i.e., it generates Bob's transcript T_B in addition to Alice's View. Using these two transcript, the simulator can keep track of the progress of the coding schemes, compute the correct statistic and progress in a correct way.

Output. As long as there weren't too many errors (so that the output is not $\perp$), the output shares sh_A, sh_B are a secret share of the correct circuit's output $\mathcal{F}_C(x)$. Similar to the protocol, the information sh_B is then encoded via ECC and the appropriate noise distribution is added, so that the simulated received message is identically distributed to the protocol's received message.

Note that $OUT_B = \mathcal{F}_C(x)$ unless one of the following bad events happen:

1. Alice or Bob decode an incorrect share of y or x, respectively, at the initialization phase
2. The coding scheme fails to simulate the (noiseless) GMW protocol π_0
3. Bob decodes an incorrect output share sh_A

The first and last items occur with probability at most $\exp(-\kappa)$, since the information is sent over a BSC_ε channel encoded via ECC of length $\Omega(\kappa)$ (Lamma 6). The second item occurs with probability at most $\exp(-|C|) = \exp(-|C_0| - \kappa)$ by Theorem 15. Hence, the statistical difference due to correctness is at most $\exp(-\kappa)$.