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, Ramat Gan, Israel
ran.gelles@biu.ac.il
[2] Department of Computer Science, Ariel University, Ariel, Israel
anatpc@ariel.ac.il
[3] School of Informatics, University of Edinburgh, 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 $O(s) + \text{poly}(\kappa \cdot \text{depth}(s))$.

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 [39] 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

The full version of this paper can be found at the Cryptology ePrint Archive [19].

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

D. Catalano and R. De Prisco (Eds.): SCN 2018, LNCS 11035, pp. 445–463, 2018.
https://doi.org/10.1007/978-3-319-98113-0_24

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 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. [8,14]). 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 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 (in short, CC) 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) [37]. 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 polylogarithmic 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.

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

[2] While cryptography typically allows negligible error (in a security parameter larger than the protocol length), here we follow the coding community's approach and insist on obtaining exponentially small error probability; hence, the overhead implied by the naïve approach is in fact linear in the protocol's length, rather than polylogarithmic.

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* [2,30]. On the other hand, the BSC_ε channel can be used as a cryptographic resource/setup [10], implying any function f could have a secure protocol π evaluating it [28]. 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. [25] 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 [26]. 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 [25], 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 6 for discussion on branching program representations of protocols), then we get

[3] Using a recent result by Khurana et al. [27] we are able to extend our result also to other types of noisy channels, such as the elastic channel (cf. [19]).

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 [20], see also Sect. 1) converts BP_0 into a boolean circuit (e.g., by Proposition 8) 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 [32], 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., [12,27,38]). 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. [27] 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.

Theorem 3 (informal). *Let 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 over noiseless channels, and let BP_0 denote a branching program representation of π_0. Then, there's a semi-honest statistically secure protocol π over an (α, β)-elastic channel that computes π_0 with simulation error $2^{-\kappa^c}$ for some constant c. The communication complexity of the obtained protocol is $\mathtt{CC}(\pi) = \tilde{O}(\text{width}(BP_0)) \cdot \mathtt{CC}(\pi_0) + O(\kappa)$.*

Most of the proofs of our theorems are deferred to the full version [19].

Overview of Our 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 $|C| = |BP_0|$ polylog(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 Sect. 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 (Sect. 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. [10–12,27,38]. 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 [3]. 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. [23]. 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* [4,5,18,35,36] (see [15] for a survey). In particular, we use an interactive coding schemes by Haeupler [22] with linear overhead and exponentially small error probability, assuming BSC channels. In a nutshell, the scheme of [22] 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.

Related Literature. In his seminal paper, Yao [39] 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 [33]. This result was later extended to the information-theoretic (IT) setting by Goldreich, Micali, and Wigderson [20]. Their so called GMW protocol for the semi-honest case also assumes that parties have ideal access to an OT functionality (cf. Sect. 2.3).[4] Kilian [28] proved that OT is in-fact a complete primitive even against malicious adversaries, a result made more efficient by Ishai et al. [26]. Crépeau and Kilian [10] proved that OT can be implemented by an information-theoretic protocol using different types of channels, including the BSC_ε. Beaver [3] showed how OT can be precomputed, i.e., how parties can, in an offline phase, compute correlated randomness that allows, during the online phase, to implement OT by simply communicating two messages (cf. Sect. 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., [23–25,31]. Furthermore, [23] showed how to

[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, it is proved to achieve IT security when given ideal access to an OT functionality [21].

semi-honestly evaluate ℓ-parallel OT's from noisy channels with linear communication complexity $O(\ell)$ and exponentially small error in ℓ.

Closer in spirit to our work, Naor and Nissim [32] 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 work, their compiler takes as an input a branching-program BP_0 for π_0, rather than an arithmetic circuit for f. 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 assumes noisy channels, the machinery of [32] assumes reliable channels and the existence of OT.

Secure Computation Over Noisy Channels. Some functions f can be securely computed without any of the above cryptographic tools (assuming reliable channels). Indeed, Kushilevitz [30] (also, Beaver [2]) 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 [8,14]. 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).

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 satisfies $\Pr(x = 0) = 1 - \varepsilon$ 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,

[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 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 $view_B = (y_B, r_B, T_B)$.

The *communication complexity* of π, denoted $\mathtt{CC}(\pi)$, is maximal number of bits exchanges throughout the protocol. The *length* of π, denoted $|\pi|$, is the number of rounds in the longest instance. For simplicity, we assume a single bit is sent at each round, hence, $|\pi| = \mathtt{CC}(\pi)$.

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 4 (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 $negl(\cdot)$.*

Correctness without privacy is easy to achieve over reliable networks: send all inputs to Alice who conducts the computation. With unreliable communication this is no longer a straightforward task. Standard error-correction technique would produce a correct protocol despite the noise, however, the cost in communication complexity will be substantial. Achieving a correct protocol *while keeping its total communication complexity low* is typically a challenging task.

Semi-honest Security. Our protocols are proven secure via the standard simulation-based security notion against semi-honest adversaries. We will use the formulation of [6] which follows the real-world/ideal-world paradigm, but, as we are only considering semi-honest security, our results can be adapted to work in the universal composition framework of Canetti [7].

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* Ch. *We say that π is a statistically secure protocol for computing $f(x,y)$ 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*

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

Observe that the definition above captures both privacy and correctness, 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.

2.2 Noisy Networks and Coding Schemes

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., [9,34]).

The notion of a protocol needs to be augmented to the above noisy-communication model, keeping 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 from Bob to 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* of a given instance of the protocol consists of all messages sent and received during that given instance $T = (T_A, T_B)$. (For notational simplicity we will refer to the joint transcript simply as the transcript.) 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 assume wlog 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).

Coding Schemes for Interactive Protocols. An interactive *coding scheme* C [15] 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, $1 - 2^{-\Omega(|\pi_0|)}$).

2.3 Primitives, Boolean Circuits, and Branching Programs

Oblivious Transfer. Oblivious Transfer (OT) [33] 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. We use a variant of Branching Programs (BPs) that is convenient for representing 2-party protocols, defined as follows.

Definition 6. *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 $w_i=|V_i|\le w$, 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 $\mathrm{width}(BP)$. It is also easy to verify that the communication of π is connected to the branching program by $\mathtt{CC}(\pi)=\sum_{1<i\le t}\lceil\log w_i\rceil$, hence,

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

Boolean Circuits. We use standard Boolean circuits consisting only of NAND gates[6] with fan-in 2 and unbounded fan-out [1]. We assume all literals depend on the input, i.e., we don't allow constant inputs.[7] We denote by $|C|$ the size of C, i.e, the number of its nodes/gates, and by $\mathrm{depth}(C)$ its depth.

3 Deterministic 2PC over BSC_ε with Linear Rate

In this section we prove our main results, Theorems 1 and 2, and show how to simulate any (possibly non-private) protocol that assumes reliable communication over a BSC_ε.

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

[7] This is wlog 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 7. *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 [19] we show how to extend our compiler to randomized protocols. The theorem is proved in two steps. First, in Sect. 3.1 we argue one can 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 Eq. (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 Sect. 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 8. *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 $\tau_A : \{0,1\}^n \to \{0,1\}^{n_A}$, $\tau_B : \{0,1\}^n \to \{0,1\}^{n_B}$, and $\tau_{out} : \{0,1\}^{\nu_{AB}} \to \{0,1\}^\nu$, such that for all $x, y \in \{0,1\}^n$ it holds that $\tau_{out}\left(C_0\left(\tau_A(x), \tau_B(y)\right)\right) = f(x,y)$.

3.2 Secure Evaluation of Circuits over a BSC_ε

We proceed to the second part of the proof of Theorem 7 and describe a protocol for secure evaluation of circuits over a BSC_ε with communication complexity $O(|C| + \kappa)$. Formally,

Proposition 9. *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 7 follows as a corollary of Propositions 8 and 9. The remainder of the section is dedicated to proving Proposition 9.

3.2.1 Building Blocks

Towards proving Proposition 9, 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.

***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 10 ([23, 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 [3], 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 10) 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.

The protocol Π-OT_ε, described in Fig. 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 11) and its proof use somewhat different properties. Namely, we use the notion of *weak security* and of *channel-transparent security*. 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

Protocol Π-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$_\varepsilon$ protocol

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 11. *For any $\varepsilon < 1/2$, the protocol Π-OT$_\varepsilon$ 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. [The complete protocol in the OT-hybrid setting is given in Fig. 2.] However, in our implementation we will not have an ideal OT, but instead we utilize the protocol Π-OT$_\varepsilon$ 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$_\varepsilon$.

Protocol 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: $\mathcal{F}_{OT}(b_1, (a_2 b_2 + r_2, a_2 b_2 + a_2 + r_2))$. Denote Alice's OT output by o_2.
  4. Alice sets her output to $c_1 = 1 + a_1 b_1 + o_1 + o_2$.

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

Lemma 12. *For any $\varepsilon < 1/2$, the protocol $NAND^{\Pi\text{-}OT_\varepsilon}$ is weakly, channel-transparently, statistically secure in the semi-honest setting, assuming all communication is done over a BSC_ε.*

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 [15]), 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., [4,13,16,17,22,29,35]. We will use one by Haeupler:

Theorem 13 ([22, 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 [22]

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 [22]

The outline of Alice's behaviour in the resulting protocol π is given in Fig. 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) 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 [20]. 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). Concretely, 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 a standard error correction code of length $\Theta_\varepsilon(m+\kappa)$ that decode correctly over BSC_ε except with

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 a standard error-correction code of length $O(|C|)$ with decoding error $\exp(-|C|)$. 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 10) 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.

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_ε (Figure 1).
  After evaluating the last gate, π_0 is assumed to keep sending zeros indefinitely.
- Apply the coding scheme of Theorem 13 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 $z = so_A + so'_B$.

[a] We add these gates because the correctness guarantee in Theorem 13 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 13 for $O(\kappa)$ more rounds.

Fig. 4. Secure circuit evaluation protocol Π_{2PC}

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 Evaluation Phase. 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 complete construction, Π_{2pc}, is depicted in Fig. 4.

Theorem 14. *The protocol Π_{2PC} satisfies Proposition 9.*

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach, 1st edn. Cambridge University Press, New York (2009)
2. Beaver, D.: Perfect privacy for two-party protocols. In: Proceedings of DIMACS Workshop on Distributed Computing and Cryptography, vol. 2, pp. 65–77 (1991)
3. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_8
4. Brakerski, Z., Kalai, Y.T., Naor, M.: Fast interactive coding against adversarial noise. J. ACM **61**(6), 35:1–35:30 (2014)
5. Braverman, M., Rao, A.: Toward coding for maximum errors in interactive communication. IEEE Trans. Inf. Theory **60**(11), 7248–7255 (2014)
6. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
7. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
8. Chung, K.-M., Pass, R., Telang, S.: Knowledge-preserving interactive coding. In: FOCS 2013, pp. 449–458 (2013)
9. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley, Hoboken (2006)
10. Crépeau, C., Kilian, J.: Achieving oblivious transfer using weakened security assumptions. In: FOCS 1988, pp. 42–52 (1988)
11. Crépeau, C.: Efficient cryptographic protocols based on noisy channels. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 306–317. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_21
12. Damgård, I., Fehr, S., Morozov, K., Salvail, L.: Unfair noisy channels and oblivious transfer. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 355–373. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_20

13. Efremenko, K., Gelles, R., Haeupler, B.: Maximal noise in interactive communication over erasure channels and channels with feedback. IEEE Trans. Inf. Theory **62**(8), 4575–4588 (2016)
14. Gelles, R., Sahai, A., Wadia, A.: Private interactive communication across an adversarial channel. IEEE Trans. Inf. Theory **61**(12), 6860–6875 (2015)
15. Gelles, R.: Coding for interactive communication: a survey. Found. Trends Theor. Comput. Sci. **13**(1–2), 1–157 (2017)
16. Gelles, R., Haeupler, B.: Capacity of interactive communication over erasure channels and channels with feedback. SIAM J. Comput. **46**(4), 1449–1472 (2017)
17. Gelles, R., Haeupler, B., Kol, G., Ron-Zewi, N., Wigderson, A.: Towards optimal deterministic coding for interactive communication. In: SODA 2016, pp. 1922–1936 (2016)
18. Gelles, R., Moitra, A., Sahai, A.: Efficient coding for interactive communication. IEEE Trans. Inf. Theory **60**(3), 1899–1913 (2014)
19. Gelles, R., Paskin-Cherniavsky, A., Zikas, V.: Secure two-party computation over unreliable channels. Cryptology ePrint Archive, Report 2018/506 (2018). https://eprint.iacr.org/2018/506
20. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC 1987, pp. 218–229 (1987)
21. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, New York (2004)
22. Haeupler, B.: Interactive channel capacity revisited. In: FOCS 2014, pp. 226–235 (2014)
23. Harnik, D., Ishai, Y., Kushilevitz, E., Nielsen, J.B.: OT-combiners via secure computation. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 393–411. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_22
24. Harnik, D., Kilian, J., Naor, M., Reingold, O., Rosen, A.: On robust combiners for oblivious transfer and other primitives. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 96–113. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_6
25. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A., Wullschleger, J.: Constant-rate oblivious transfer from noisy channels. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 667–684. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_38
26. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_32
27. Khurana, D., Maji, H.K., Sahai, A.: Secure computation from elastic noisy channels. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 184–212. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_7
28. Kilian, J.: Founding crytpography on oblivious transfer. In: STOC 1988, pp. 20–31. ACM. New York (1988)
29. Kol, G., Raz, R.: Interactive channel capacity. In: STOC 2013, pp. 715–724 (2013)
30. Kushilevitz, E.: Privacy and communication complexity. In: FOCS 1989, pp. 416–421. IEEE Computer Society (1989)
31. Meier, R., Przydatek, B., Wullschleger, J.: Robuster combiners for oblivious transfer. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 404–418. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_22
32. Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: STOC 2001, pp. 590–599 (2001)

33. Rabin, M.O.: How to exchange secrets with oblivious transfer. Technical report TR-81, Aiken Computation Lab, Harvard University (1981)
34. Roth, R.: Introduction to Coding Theory. Cambridge University Press, Cambridge (2006)
35. Schulman, L.J.: Communication on noisy channels: a coding theorem for computation. In: FOCS 1992, pp. 724–733 (1992)
36. Schulman, L.J.: Coding for interactive communication. IEEE Trans. Inf. Theory **42**(6), 1745–1756 (1996)
37. Shannon, C.E.: A mathematical theory of communication. Bell System Tech. J. **27**(379–423), 623–656 (1948)
38. Wullschleger, J.: Oblivious transfer from weak noisy channels. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 332–349. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_20
39. Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd FOCS, pp. 160–164. IEEE Computer Society Press, November 1982