Random Sources in Private Computation

Geoffroy Couteau[1] and Adi Rosén[2]

[1] CNRS, IRIF, Université Paris Cité, Paris, France
couteau@irif.fr
[2] CNRS, FILOFOCS, Tel Aviv, Israel
adiro@irif.fr

Abstract. We consider multi-party information-theoretic private computation. Such computation inherently requires the use of local randomness by the parties, and the question of minimizing the total number of random bits used for given private computations has received considerable attention in the literature, see, e.g., [KR94, KM96, KOR98, BDSPV99, JLR03, DPP16, RU19, KOP$^+$19].
In this work we are interested in another question: given a private computation, we ask how many of the players need to have access to a random source, and how many of them can be deterministic parties. We are further interested in the possible interplay between the number of random sources in the system and the total number of random bits necessary for the computation.
We give a number of results. We first show that, perhaps surprisingly, t players (rather than $t+1$) with access to a random source are sufficient for the information-theoretic t-private computation of any deterministic functionality over n players for any $t < n/2$; by a result of [KM96], this is best possible. This means that, counter intuitively, while private computation is impossible without randomness, it is possible to have a private computation even when the adversary can control *all* parties who can toss coins (and therefore sees all random coins). For randomized functionalities we show that $t+1$ random sources are necessary (and sufficient).
We then turn to the question of the possible interplay between the number of random sources and the necessary number of random bits. Since for only very few settings in private computation meaningful bounds on the number of necessary random bits are known, we consider the AND function, for which some such bounds are known. We give a new protocol to 1-privately compute the n-player AND function, which uses a single random source and 6 random bits tossed by that source. This improves, upon the currently best known results [KOP$^+$21], at the same time the number of sources and the number of random bits ([KOP$^+$21] gives a 2-source, 8-bits protocol). This result gives maybe some evidence that for 1-privacy, using the minimum necessary number of sources one can also achieve the necessary minimum number of random bits. We believe however that our protocol is of independent interest for the study of randomness in private computation.

1 Introduction

A multi-party t-*private* protocol for computing a function f is a distributed protocol that allows $n \geq 3$ players P_i, for $1 \leq i \leq n$, each possessing an individual secret input x_i, to jointly compute the value of $f(x)$ in a way that conceals, with respect to any coalition of at most t players, all information beyond what can be deduced from the value of f and their own inputs. Secure multi-party computation is a fundamental problem in cryptography. It has received intense attention, both in the computational and in the information-theoretic setting, starting with its introduction in the seminal works of Yao [Yao86], and Goldreich, Micali, and Wigderson [GMW87b, GMW87a] (GMW). This is due to both its theoretical interest and its many applications (including, but not limited to, e-voting, auctions, private set intersections, privacy-preserving machine learning, and many more).

Information-theoretic secure computation. In this work, we focus specifically on information-theoretic secure (private) computation, introduced in the seminal works of [BGW88] and [CCD88]. In this model, the protocol proceeds in rounds, where in each round each player sends a message to each other player, over a secure and authenticated point-to-point channel. The privacy property of such a protocol means, informally, that no coalition of at most t players can learn anything (in the information-theoretic sense) from the execution of the protocol, except what is implied by the value of $f(x)$ and the inputs of the player in the coalition. Private computation in this setting was the subject of considerable research, see e.g., [BGW88, CCD88, KR94, AL17, CK91, KM96, KOR98, GR03] and references therein. In addition to its theoretical interest, this setting (and its variants) constitutes the foundation of many cryptographic applications, due (in part) to the existence of (efficient) compilers that can transform a generic representation of any function f (e.g., as a boolean circuit) into a secure protocol computing the same function f [BGW88, CCD88] with complexity proportional to the size of the representation.

Randomness in secure computation. It is a folklore result that the ability to sample random coins is necessary in order to perform private computations involving more than two players (except for the computation of very degenerate functions).[3] That is, the players must have access to (private) sources of unbiased, untemperable, independent random coins. Randomness is typically regarded as a scarce resource in the design of algorithms, and methods for saving random bits in various contexts have been the focus of a wide number of works (see, e.g., [NN93] and its many follow-ups, or [NTS99, Gol09] for surveys). In the context of information theoretic secure computation, a problem of fundamental interest is to understand how much randomness is required to securely compute a function. The design of randomness-efficient private protocols, and the quantification of the amount of randomness necessary to perform private computations of various functions and under various constraints has received considerable attention in the literature, see, e.g., [KR94, KM96, KOR98, BDSPV99, JLR03, DPP16, RU19, KOP$^+$19].

On random sources in secure computation. In this work, we tackle the problem of randomness in secure computation from a new angle, which, to the best of our knowledge, was not investiagted in the past. The main motivation for reducing the randomness complexity of secure computation protocols is that producing high quality, unbiased, untemperable independent random coins is *expensive*: it requires an appropriate, well-calibrated device which can extract this randomness from well-chosen noisy sources. In addition, it is unfortunately common to generate randomness in a poor way, by reusing random strings several times, poorly seeding a pseudorandom generator, using an inappropriate randomness-generating functionalities in some computer languages or softwares etc. This is, of course, an extremely well understood issue in the cryptographic community: poor randomness generations has led to a number of broken implementations of cryptographic primitives, or insecure generations of cryptographic keys (e.g., [NSS$^+$] showed that the bad quality of the randomness used by major manufacturers of cryptographic hardware caused tens of millions of devices to use broken RSA keys. See also [LHA$^+$12] for even more striking examples).

However, once a participant in a cryptographic protocol does have the means to generate randomness properly, then asking this participant to generate a lot of random coins does not necessarily incur a major additional cost. This suggests a natural, different question: rather than looking for bounds on the number of random bits that are necessary to privately compute given functions, or the interplay between the amount of randomness used and other complexity measures, is it possible to bound *how many* of the n players in a cryptographic protocol must have access to private random sources? While this question was, to the best of our knowledge, never studied before, we believe that it is of fundamental interest. From a theoretical point of view, given that secure computation is impossible with deterministic parties alone, it is a very natural question to understand how many of the parties must actually have the ability to generate unbiased random coins (we call such parties *sources*).

From a practical point of view, the question is also well motivated: if a secure computation protocol requires only a small number of (random) sources, then many other (cheaper) individuals that do not have the means or the capacity to produce high quality randomness can still be added to the system and participate in the secure computation.

1.1 Our Contributions

In this work, we seek to characterize the number k of players that must have access to a random source in a system of n players so that one can t-privately compute a (deterministic or randomized)

[3] We remark that the two-party case, $n = 2$, is known to be qualitatively different [CK91].

functionality. Further, we are also interested in the question of the existence of a tradeoff between the number of players that have access to random sources, and the total number of random bits necessary for the private computation. Our main results are twofold.

A full characterization. We precisely characterize how many random sources are necessary and sufficient to t-privately compute an n-party functionality $\mathcal{F}$. For the general case of randomized functionalities, we prove a simple lower bound: $t+1$ sources are necessary. We also provide a matching upper bound, which follows from a simple tweak of the seminal BGW protocol [BGW88]. Then, we turn our attention to the case of deterministic functionalities. Here, it follows from a lower bound of [KM96] that t random sources are necessary in general for t-private computation.

At first glance, it seems natural to believe that the lower bound of [KM96] is not tight in general. Indeed, if there are only t (fixed) parties that can generate randomness among all participants, and if the adversary can corrupt up to t parties, then the adversary can corrupt *all* participants which can generate randomness, and the protocol becomes entirely deterministic from the viewpoint of the adversary. Since secure computation is impossible with deterministic parties, we are tempted to conclude that such a protocol cannot be secure in general.

Our main technical result in this full characterization shows that, surprisingly, this intuition is flawed, and the lower bound of [KM96] is tight. Namely, we prove the following:

Theorem 1 (Informal). *For every deterministic n-party functionality $\mathcal{F}$, and every $t < n/2$, there exists a t-private protocol that securely computes $\mathcal{F}$ between n parties $P_1, \cdots, P_n$, if there is a size-t subset of the parties which have the ability to toss random coins.*

In other words: while secure computation is impossible without randomness, we show that secure computation is always possible using randomness, *even if the adversary is allowed to corrupt all parties that can produce randomness.* The proof of Theorem 1 is non-trivial; it relies on a careful combination of the GMW protocol (used as an outer protocol) and the BGW protocol (used as an inner protocol). At a very high level, the key idea is to *isolate* the t random sources, and to involve them solely in sub-computations that do not involve any actual input, letting the remaining parties perform the bulk of the sensitive computation, using random coins sent by these sources. Intuitively, we achieve the following dichotomy: either the adversary corrupts all sources, but in this case it cannot corrupt any of the parties that actually take part in the "sensitive part" of the computation; or the adversary corrupts at least one deterministic party, but then there is at least one uncorrupted source, which we leverage to generate random coins for all parties. The above intuition is relatively easy to instantiate when $t < n/3$; most of the complexity of our result stems from instantiating it for the optimal bound of $t < n/2$. Our complete characterization is summarized in Table 1.

	Deterministic functionalities	**Randomized functionalities**
Lower Bound	t sources [KM96]	$t+1$ sources (Section 3.1)
Upper Bound	t sources (Section 4)	$t+1$ sources (Section 3.2)

Table 1: Lower and upper bounds on the number of sources necessary for t-private computation of n-party functionalities. See Section 3 for discussions on defining the notion of the necessary numbber random sources.

Extension to UC security and statistical security. For simplicity, all our protocols and lower bounds are discussed in the stand alone model, and with perfect security. However, all our constructions are proven secure using a black-box non-rewinding simulator. By a known result of [KLR06], this implies that our protocols also enjoy perfect universal composability. Second, our lower bounds extend directly to the setting of statistically secure protocols; we actually directly prove our lower bound with respect to statistical security in our proof of Theorem 7, and the proof of Theorem 6 in [KM96] extends also immediately to the setting of statistical privacy.

Extension to Adaptive Security. Our protocols achieve perfect selective security with straight-line black-box simulators. Therefore, it follows by known results [AL17, Section 8] that the protocols also enjoy adaptive simulation with respect to an *inefficient* simulator. Whether we can achieve the stronger notion of adaptive corruption with efficient simulation is an interesting open question.

Randomness complexity of AND. In our second contribution, we turn to the question whether there is a tradeoff between the number of players that have access to a random source and the total number of random bits necessary for the private computation. The motivation for this question is that, in the constructions of our positive results, secure computation using an optimally small number of sources seems to require up to $\Theta(t)$ times more randomness compared to secure computation where all parties can toss coins (this is particularly visible in our simple upper bound for randomized functionalities). It is natural to wonder whether this is inherent: in order to reduce the number of sources, do we have to pay a price in randomness complexity? We put forward a conjecture stating that this is indeed the case for *complex* functionalities, i.e., n-party functionalities that do not have information-theoretic t-private protocols for $t \geq n/2$. Our conjecture states that, for such functionalities, a $\Theta(t)$ blowup in randomness complexity is necessary and sufficient to minimize the number of random sources. We view this conjecture as an interesting open question.

Then, in the course of getting a better understanding of the relation between randomness complexity and random sources, we turn our attention to a simple, yet very basic, concrete functionality: the n-party AND (it is very common in the literature on randomness complexity of secure computation to study the case of simple functionalities such as XOR [KR94, KM96, KOR98, GR03] or AND [KOP+19], as they are basic functions). Here, the state of the art is the recent work of [KOP+19], that showed that 8 bits are sufficient to 1-privately compute the n-party AND functionality (the paper also shows that more than 1 bit is necessary). The upper bound of [KOP+19] uses two sources, and our question is whether we can match this upper bound using a *single* source or whether a private protocol with a single source will require more random bits.

Here, we again achieve a somewhat surprising result: we *improve* over the result of [KOP+19] in the two aspects at the time, i.e., we reduce both the number of sources and the number of random bits. Using a completely different protocol, we show that 6 bits tossed by a single source are sufficient to 1-privately compute the n-party AND functionality, for any $n \geq 3$.

2 Preliminaries

In this work, we consider perfectly secure protocols in the presence of semi-honest adversaries. More precisely, we focus on the *stand-alone* setting (security is argued for a single execution of the protocol in isolation), with *semi-honest* (perfect) security (the adversary sees the view of all corrupted parties, but all parties follow the specifications of the protocol) in a *static* corruption model (the adversary specifies the set of corrupted parties ahead of time).

Network Model. The parties interact over a synchronous network: the computation takes place in clearly defined rounds. All pairs of parties are connected via perfectly private and authenticated channels.

Notations. We let n denote the number of parties, and t denote the (maximum) number of corrupted parties. Let $[n]$ denote the set $\{1, \cdots, n\}$. We use the following notation for vectors, e.g., $\mathbf{x} = (x_1, \cdots, x_n)$; for any subset $C \subseteq [n]$, we write $\mathbf{x}_C$ for $(x_i)_{i \in C}$. Given a set S, we write $s \overset{\$}{\leftarrow} S$ to denote that s is sampled uniformly at random from S. Given a vector $\mathbf{x}$, we let $|\mathbf{x}|$ denote its length. An n-party deterministic functionality is a function $f : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$; we write $f_i(x_1, \cdots, x_n)$ to denote the i-th output of f on inputs $(x_1, \cdots, x_n)$, and $f_C(x_1, \cdots, x_n)$ to denote $(f(x_1, \cdots, x_n))_C$ for any $C \subseteq [n]$. For randomized functionality, every input vector $(x_1, \cdots, x_n)$ defines a *distribution* $f(x_1, \cdots, x_n)$ over the output space $(\{0,1\}^*)^n$. We say that the protocol computes the deterministic functionality $f : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ (with perfect correctness) if, for every input $\mathbf{x} = (x_1, \cdots, x_n) \in (\{0,1\}^*)^n$, and for any outcome of all coin tosses, the output produced by each party P_i is always $f_i(\mathbf{x})$. When f is a randomized functionality, we say that a protocol computes f (with perfect correctness) if for every input $\mathbf{x} = (x_1, \cdots, x_n) \in (\{0,1\}^*)^n$, the distribution (over the randomness of the parties) of the joint outputs of the parties is exactly $f(x_1, \cdots, x_n)$. We write $\mathcal{D} \equiv \mathcal{D}'$ to denote that two distributions $\mathcal{D}$ and $\mathcal{D}'$ are identical. We sometime write distributions as $\{(a,b)\ :\ \text{sampling process}\}$ to denote a distribution over pairs (a,b) sampled according to the given sampling process. Given a probabilistic algorithm A, we slightly abuse notation and usually view $A(x)$ as the distribution corresponding to the output of A on input x.

2.1 Perfect Privacy

We first define the notion of a *view* of a player. The sets, functions, and random variables in the following definition are implicitly parametrized by a protocol π.

Definition 2. (View) *The view of party P_i (on a joint input $\mathbf{x}$ from all parties) at round $r \geq 1$, denoted $V_i^r(\mathbf{x})$, is the (joint) distribution of the sequence of messages received by P_i in rounds 1 to $r-1$, and the sequence of the results of the coin tosses performed by P_i in rounds 1 to r.*

All the protocols we consider in this paper have deterministic upper bounds on the number of rounds. Hence we can also define the "final view" of the players after that upper bound is attained. We denote those without superscripts, i.e., $V_i(\mathbf{x})$.

Definition 3. (Output Distribution) *We let $O_i(\mathbf{x})$ denote the distribution of the output of P_i after an execution of the protocol with a joint input $\mathbf{x}$.*

Given a subset C of $[n]$, we write $V_C(\mathbf{x})$ to denote $(V_i(\mathbf{x}))_{i\in C}$ and $O_C(\mathbf{x})$ to denote $(O_i(\mathbf{x}))_{i\in C}$; we use $O(\mathbf{x})$ as a shorthand for $O_{[n]}(\mathbf{x})$.

Definition 4 (t-Privacy for deterministic functionalities [AL17]). *Let $f : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be an n-party deterministic functionality and let π be a protocol. We say that π is (perfectly) t*-private *if (1) π computes f with perfect correctness, and (2) there exists a probabilistic polynomial-time algorithm* Sim *such that for every $C \subset [n]$ of cardinality at most t and every $\mathbf{x} \in (\{0,1\}^*)^n$ where $|x_1| = \cdots = |x_n|$, it holds that $\mathsf{Sim}(C, \mathbf{x}_C, f_C(\mathbf{x})) \equiv V_C(\mathbf{x})$.*

While the above definition considers separately (with (1) and (2)) the issues of correctness and privacy, in the general case of randomized functionalities, the two notions are intertwined:

Definition 5 (t-Privacy for randomized functionalities ([AL17], def. 2.2)). *Let $f : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ be an n-party randomized functionality and let π be a protocol. We say that π is (perfectly) t*-private *if (1) π computes f with perfect correctness, and (2) there exists a probabilistic polynomial-time algorithm* Sim *such that for every $C \subset [n]$ of cardinality at most t and every $\mathbf{x} \in (\{0,1\}^*)^n$ where $|x_1| = \cdots = |x_n|$, it holds that*

$$\{(v,y)\ :\ y \leftarrow f(\mathbf{x}), v \leftarrow \mathsf{Sim}(C, \mathbf{x}_C, y_C)\} \equiv (V_C(\mathbf{x}), O(\mathbf{x})),$$

where $(V_C(\mathbf{x}), O(\mathbf{x}))$ denotes the joint *distribution of the corrupted parties' (final) views and the outputs of all parties in a run of the protocol on common input $\mathbf{x}$.*

A simulator according to the above definitions is simply a machine that produces emulated views for all corrupted parties. However, it will be convenient when analyzing the security of our protocols to view the simulator Sim as an *interactive* machine, which pretends to play the role of the honest parties during an execution of the protocol, and interacts with the corrupted parties. Under this viewpoint, Sim receives as input $(C, \mathbf{x}_C, f_C(\mathbf{x}))$, but also the random tapes of all corrupted parties; this is w.l.o.g. since in the above definition, Sim will anyway sample these random coins itself when emulating the views.

3 On the Number of Random Sources in Private Computation

Parties participating in a private computation protocol may or may not have access to a random source. We call a party that has access to a random source a *source*, and assume that this party is given access to an arbitrarily long tape of independent unbiased random bits. Note that what we call a source is exactly the standard definition of a player in standard secure computation protocols. In contrast, players which do not have access to such a tape are called *deterministic parties*: the behavior of deterministic parties at any given time is entirely determined by a deterministic function of their input and the messages they received up to that time.

Static versus dynamic measures. Note that our notions of source and deterministic parties is *static*: which parties can or cannot sample random bits is a priori fixed before the start of the protocol. In other words, if in at least one execution of a protocol π a given party has to sample a random coin, then that party is a source. Even if (say) only 10 distinct parties have to sample coins during any given execution of the protocol, but *which* of the parties sample coins vary over different executions, then we cannot say that π uses only 10 sources – it might be that *all* parties have to be sources. This static notion captures in a more realistic way the setting where some parties in a system have access to a high-quality random number generator, while others do not; the real-world meaning of the dynamic variant of the notion is less clear.

When one considers the static measure, another distinction is called for,

Universal versus non-universal setting. Before we establish our main theorems, we formally define what we mean by a statement of the form "all n-party functionalities can be privately computed with s sources." This can be interpreted in two ways:

1. Fix s sources $(P_1, \cdots, P_s)$, and $n - s$ deterministic parties $(P_{s+1}, \cdots, P_n)$. For any n-party functionality $\mathcal{F}$ there is a protocol that uses the above parties and privately computes $\mathcal{F}$.
2. Fix an n-party functionality $\mathcal{F}$. Then there exists a choice of a subset S, $s = |S|$, of parties among $(P_1, \cdots, P_n)$ such that if $(P_i)_{i \in S}$ are sources, and the other parties are deterministic, then there exists a protocol that uses the above parties and privately computes $\mathcal{F}$.

We call a protocol of the first type *universal*, and a protocol of the second type *non-universal.* Universal protocols are more desirable, since we would like to capture settings where sources are defined by the availability of a good random number generator, and then one can privately compute *any* functionality in that system; we typically do not want the choice of the sources to depend on the specific functionality at hand. Looking ahead, our results will consider the static measure (as it better captures the situation in "realistic" systems), and will be proved with the best flavor: our upper bounds will be *universal* protocols, while our lower bounds hold even for *non-universal* protocols.

3.1 Lower Bounds

Theorem 6 (deterministic functionalities, lower bound). *For any number of parties n, there exists a deterministic n-party functionality $\mathcal{F}$ such that for any $t \leq n-2$, any t-private n-party protocol computing $\mathcal{F}$ must have at least $s \geq t$ sources.*

The theorem follows directly from a result of Mansour and Kushilevitz [KM96] who showed that in order to t-privately compute the function `xor`, $t \leq n-2$, at least t players with access to a local random source are necessary. Note that Mansour and Kushilevitz did not focus on the number of random sources in their work: their goal was to show that the *randomness complexity* of t-private computation of `xor` is at least t (that is, in any given execution of a t-private protocol computing `xor`, at least t random coins must be sampled). However, their proof proceeds by showing that in any given execution *at least t different parties* must sample at least one random bit, hence, as they note, their proof proves also that at least t players with access to a local random source are necessary to t-privately compute `xor`.

Theorem 7 (randomized functionalities, lower bound). *For any number of parties n, there exists a randomized n-party functionality $\mathcal{F}$ such that for any $t \leq n-1$, any t-private n-party protocol computing $\mathcal{F}$ must have at least $s \geq t+1$ sources.*

The proof follows the (natural) intuition that if all sources can be corrupted and the outputs of the honest parties depend on independent random coins, these random coins will not be independent of the view of the sources. In fact, the proof below also rules out any *statistically private* protocol for such functionalities, by showing that the statistical distance between the ideal distribution and the simulated distribution cannot be sub-constant.

Proof. Consider the following simple randomized functionality $\mathcal{F}$: on joint input $\mathbf{x}$, the output of each player is $\mathbf{x}$ together with a single bit chosen uniformly and independently at random. Assume that the number s of sources is at most t; since the functionality is symmetrical, let us, without loss of generality, call $P_1, \cdots, P_s$ the s sources, and $P_{s+1}, \cdots, P_n$ the remaining deterministic parties.

Let π be an arbitrary protocol computing $\mathcal{F}$ in the above setting, and Sim be any simulator. Let $C \leftarrow \{1, \cdots, s\}$; that is, the adversary corrupts exactly all the sources. Since all honest parties are deterministic, the only coins tossed during the entire protocol are tossed by corrupted parties. Since the entire joint input $\mathbf{x}$ is part of the output of each party (hence part of the view of the corrupted parties), there necessarily exists a *deterministic* function g such that $\{g(V) : V \leftarrow V_C(\mathbf{x})\} \equiv O(\mathbf{x})$. Now, given $y \leftarrow f(\mathbf{x})$, the input $(C, \mathbf{x}_C, y_C)$ to Sim depends solely on $\mathbf{x}$ and the independent random output bits defining the outputs of the corrupted parties, as defined by the functionality. Writing $((\mathbf{x}, b_1), \cdots, (\mathbf{x}, b_n)) \leftarrow f(\mathbf{x})$, the bits $(b_{s+1}, \cdots, b_n)$ are $n-s$ random bits independent of $(C, \mathbf{x}, y_C)$. Therefore,

$$\Pr[g(\mathsf{Sim}(C, \mathbf{x}_C, y_C)) = f(\mathbf{x})] \leq \frac{1}{2^{n-s}} \leq \frac{1}{2},$$

for any choice of randomness done by Sim. Therefore, the function g provides a simple distinguisher showing that the distributions $\{(v, y) : y \leftarrow f(\mathbf{x}), v \leftarrow \mathsf{Sim}(C, \mathbf{x}_C, y_C)\}$ and $(V_C(\mathbf{x}), O(\mathbf{x}))$ have (at least) a constant statistical distance. this concludes the proof.

3.2 Upper Bounds

In this section, we give the matching upper bounds for both deterministic and randomized functionalities. The upper bound for randomized functionalities follows from a simple and natural protocol; the protocol for deterministic functionalities is considerably more involved, and is the focus of Section 4.

Theorem 8 (deterministic functionalities, upper bound). *For any number of parties n and any $t < n/2$, there is a choice of t sources among the n players such that for any deterministic n-party functionality $\mathcal{F}$ there is a t-private protocol for $\mathcal{F}$ that works with those parties.*

Section 4 below is dedicated to the proof of Theorem 8.

Theorem 9 (randomized functionalities, upper bound). *For any number of parties n and any $t < n/2$, there is a choice of $t+1$ sources among the n players such that for any (randomized) n-party functionality $\mathcal{F}$ there is a t-private protocol for $\mathcal{F}$ that works with those parties.*

The protocol for randomized functionalities captures the (correct, straightforward) intuition that if not all sources can be corrupted, then there is always at least one uncorrupted source which can distribute random coins to the deterministic parties. To formalize this intuition, we recall the seminal result of Ben-Or, Goldwasser, and Wigderson [BGW88]:

Theorem 10 (BGW). *For any $t < n/2$ and any n-party (possibly randomized) functionality $\mathcal{F}$, there is a perfect t-private protocol for $\mathcal{F}$ (with communication and randomness proportional to the circuit size of $\mathcal{F}$).*

In the protocol guaranteed by the above theorem, all parties have access to their own random tape, and the size of the random tape is bounded by an a priori known polynomial in the circuit size of $\mathcal{F}$. Note that the BGW protocol also extends to securely computing randomized functionalities, and functionalities that provide different outputs to all parties [AL17]. In our case we have a fixed set of $t+1$ parties which can toss random coins: they are called the *sources*, and are denoted $S_1, \cdots, S_{t+1}$; the remaining $n-t-1$ parties, denoted $P_{t+2}, \cdots, P_n$, are deterministic. Given the functionality $\mathcal{F}$, let $B_{\mathcal{F}}$ be an upper bound on the maximum number of random coins tossed by any single party in the BGW protocol for computing $\mathcal{F}$ t-privately. The protocol to t-privately compute $\mathcal{F}$ as guaranteed by the theorem works as follows:

1. Each source S_i samples, for $j = t+2$ to n, a random string $r_{i,j} \overset{\$}{\leftarrow} \{0,1\}^{B_{\mathcal{F}}}$ and sends it to P_j. Each party P_j sets $r_j = \bigoplus_{i=1}^{t+1} r_{i,j}$. In addition, each source S_i samples a random string $r_i \overset{\$}{\leftarrow} \{0,1\}^{B_{\mathcal{F}}}$.
2. All n parties run the BGW protocol for t-privately computing $\mathcal{F}$, where each player P_i, $i=1$ to n, uses the tape r_i as their random source.
3. All parties output their output from the BGW protocol.

Correctness is straightforward, and t-privacy follows directly from the fact that from the viewpoint of any subset of t parties (random sources or deterministic), the random tape r_j used by any honest party P_j is perfectly distributed as a real random tape, since it is the XOR of $t+1$ strings, at least one of which is guaranteed to be (random and) unknown to the corrupted parties.

4 Private Computation of Deterministic Functionalities

In this section, we prove that for every *deterministic* functionality $\mathcal{F} : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$, there exists a t-private n-party protocol which requires exactly t sources; these sources can be arbitrary players. Hence we show that for any deterministic functionality $\mathcal{F}$ there is a t-source *uniform* t-private protocol.

4.1 The GMW Protocol with Beaver Triples

To start we recall the seminal GMW protocol of Goldreich, Micali, and Wigderson [GMW87a], which we use in our construction. For our purpose, it will be more convenient to view GMW as an information-theoretic protocol in the *correlated randomness model* of Beaver [Bea92, Bea95], in which the parties have access to a trusted source of correlated random coins, in the form of random *Beaver triples*. Below, whenever we refer to *random (n out of n) shares* of a bit x, we mean the following: a string of n bits $x_1, \ldots, x_n$ sampled randomly conditioned on $\bigoplus_{i=1}^n x_i = x$. Given a value x, we write $\langle x \rangle$ as a more compact representation of the n-tuple $(x_1, \ldots, x_n)$ of shares of x.

Definition 11 (Beaver triple). *We say that n parties $(P_1, \ldots, P_n)$ receive a (random, n-party)* Beaver triple *if the parties receive random n out of n shares of a, b, and $a \cdot b$, where a and b are two independent random bits. That is, each party P_i receives a triple (a_i, b_i, c_i), where all triples are jointly sampled at random conditioned on $(\bigoplus_{i=1}^n a_i) \cdot (\bigoplus_{i=1}^n b_i) = \bigoplus_{i=1}^n c_i$.*

Theorem 12 (GMW + Beaver). *For any $t < n$, there exists a t-private information-theoretic secure n-party protocol in the* correlated randomness model *for computing any deterministic functionality $\mathcal{F} : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ where, prior to the execution of the protocol, the parties can ask a trusted dealer for any number of random Beaver triples. Furthermore, no party needs to toss any additional random coin.*

The GMW Protocol. Let $P_1, \ldots, P_n$ be n parties with respective inputs $(x_1, \ldots, x_n)$, and let $C_{\mathcal{F}}$ be a boolean circuit with XOR and AND gates of fan-in 2 computing the functionality $\mathcal{F}$.[4] The parties will evaluate the circuit gate by gate, starting from the inputs and computing the value of a gate when the values of its two parent nodes (which are either input nodes or gates

[4] To be completely formal, since $\mathcal{F}$ can take inputs from $(\{0,1\}^*)^n$, the circuit $C_{\mathcal{F}}$ must also depend on the input sizes $|x_1|, \ldots, |x_n|$, which means the parties have to reveal their input sizes to each other before the actual protocol starts. This does not contradict security, as privacy is only required to hold when $|x_1| = \ldots = |x_n|$; we ignore this technicality in the remainder of this paper.

themselves) have been computed. The protocol maintains the following invariant: after evaluating a gate, each party P_i will hold an additive share v_i of the value v on this gate (i.e., $v = \bigoplus_{i=1}^{n} v_i$). Without loss of generality, we assume that the parties always hold shares of the inputs to a gate when evaluating it: if an input to the gate is an input bit b belonging to party P_i, we define the shares of $(P_1, \ldots, P_i, \ldots, P_n)$ to be $(0, \ldots, b, \ldots, 0)$.

- Evaluating an XOR gate $\mathsf{XOR}(u, v)$: the parties locally XOR their shares of u and v. No communication is required.
- Evaluating an AND gate $\mathsf{AND}(u, v)$: given $\langle u \rangle$ and $\langle v \rangle$ the parties must compute additive shares $\langle uv \rangle$ of $u \cdot v$. This is done using one invocation of the *secure multiplication* protocol defined below.
- Output: after evaluating an output gate whose output is assigned to a party P_i, all parties send their share of the output to P_i, who reconstructs the output.

Secure Multiplication. Let u, v be the inputs to the AND gate, and let (u_i, v_i) be P_i's share of the inputs, $1 \leq i \leq n$.

- *Beaver triple request.* All parties receive from the trusted dealer a random Beaver triple. Let (a_i, b_i, c_i) denote P_i's share of the triple.
- *Broadcast.* Each party P_i broadcasts $\alpha_i = u_i \oplus a_i$ and $\beta_i = v_i \oplus b_i$; all parties reconstruct $\alpha = \bigoplus_{i=1}^{n} \alpha_i = u \oplus a$ and $\beta = \bigoplus_{i=1}^{n} \beta = v \oplus b$.
- *Output.* Each party P_i outputs $\alpha \cdot v_i \oplus \beta \cdot a_i \oplus c_i$.

Security follows from the fact that the pairs u_i, v_i are uniformly random; correctness follows from the relation $\langle u \cdot v \rangle = (u \oplus a) \cdot \langle v \rangle + \langle a \rangle \cdot (v \oplus b) + \langle ab \rangle$.

In our main protocol, we will rely on the above version of the GMW protocol. It will be convenient to view it as follows: the GMW protocol involves *deterministic* parties in a model where all parties can request, upon need, a random Beaver triple in order to execute a secure multiplication.

4.2 Intuition behind our Protocol: Two Complementary Scenarios

We consider n parties wishing to t-privately compute a deterministic n-party functionality $\mathcal{F} : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)^n$ on their private inputs, with exactly t sources $(S_1, \ldots, S_t)$, and $n - t$ deterministic parties, $(P_{t+1}, \ldots, P_n)$. We assume $t < n/2$. The first important observation is that the following two scenarios are mutually exclusive and cover all possible situations:

1. There is an honest majority among the $n - t$ the deterministic parties.
2. A majority of the $n - t$ deterministic parties is corrupted (*i.e.*, there are at least $\lceil (n-t)/2 \rceil$ corrupted deterministic parties). *But* in that case there is an honest majority among the t sources and there is at least one honest deterministic party.

Obviously the two scenarios are mutually exclusive and cover all possible situations. To see that in the second scenario there is still at least one honest deterministic party and there is an honest majority among the t sources, observe that $2t < n$ and hence $(n - t) - t > 0$. At the same time the number of corrupted sources is at most $t - (n - t)/2 < t/2$, hence there is an honest majority among the t sources.

Now, assume for a moment that the parties could somehow *know* in which scenario they are. In what follows we assume for simplicity that the functionality delivers the same output to all parties (the general case can be handled with standard techniques, as we will show in our full construction).

A Protocol for Scenario 1. If the parties know that the corruption pattern follows scenario 1 above, then we can *isolate the sources* and let the deterministic parties alone perform the computation as they have an honest majority among them. For simplicity, assume for now that the functionality output the same value to everyone. Since there is an honest majority among the deterministic parties, they can execute an appropriate BGW protocol among themselves, using random tapes sent by the sources (after the sources share their own inputs among the deterministic parties). During the entire protocol, the deterministic parties never send *anything* to the sources, except for the final output. Then, security follows from a simple case disjunction:

- If at least one source is honest, then the random tapes of the deterministic parties (obtained by XORing independent tapes received from each of the sources) are guaranteed to be uncorrupted, and security follows via the same argument as for the simple protocol from Section 3.2.
- Else, if all t sources are corrupted, then *all deterministic parties are honest.* In this case, security follows trivially from the fact that no corrupted party ever receives any message whatsoever, except the output itself.

More formally,

1. Each source S_i (for $i = 1$ to t) with input x_i computes $n-t$ random additive shares $(y_{i,j})_{t+1 \leq j \leq n}$ of x_i, and sends $y_{i,j}$ to party P_j for $j = t+1$ to n.
   Let $\mathcal{F}'$ be the following $(n-t)$-party functionality: on input $(x_j, y_{1,j}, \ldots, y_{t,j})$ of each party P_j, $\mathcal{F}'$ outputs $\mathcal{F}(\bigoplus_{j=1}^{n-t} x_{1,j}, \ldots, \bigoplus_{j=1}^{n-t} y_{t,j}, x_{t+1}, \ldots, x_n)$.
2. Each source S_i samples a random string $r_{i,j} \overset{\$}{\leftarrow} \{0,1\}^{B_{\mathcal{F}'}}$ and sends it to P_j for $i = 1$ to t and $j = t+1$ to n; Each party P_j sets $r_j \leftarrow \bigoplus_{i=1}^{t+1} r_{i,j}$.
3. The $n-t$ deterministic parties run the BGW protocol for securely computing $\mathcal{F}'$, where P_j uses r_j as its random tape to emulate the coin tosses in the BGW protocol.
4. The deterministic parties send the output of the BGW protocol to the t sources. All parties output this result.

Correctness follows from the fact that $\bigoplus_{j=1}^{n-t} y_{i,j} = x_i$, hence the output of $\mathcal{F}'$ is indeed $\mathcal{F}(x_1, \ldots, x_n)$. t-privacy follows immediately from the simple case disjunction outlined above. The general case, where different parties can receive different outputs, is easily handled with the standard reduction of multi-output secure computation to single-output secure computation: each party samples a random mask to mask its own output, and the parties jointly evaluate the single-output functionality that outputs (to everyone) the string of all outputs, each masked by the random mask of the corresponding party. Then, each party gets their own output (and nothing more) by removing their own mask.

A Protocol for Scenario 2. In scenario 2, we do not have anymore an honest majority among the $n-t$ deterministic parties (but there is still at least one honest party among the deterministic parties). However, this guarantees that there is now an honest majority among the *sources.* There are several solutions in this setting. We sketch one such solution: the deterministic parties can use the GMW protocol from Section 4.1, which tolerates up to all-but-one corruptions. The GMW protocol, in its version as described in Section 4.1, is deterministic, but the parties must request *random Beaver triples* from a trusted dealer. Here, we let the *sources* jointly emulate the trusted dealer: each time the deterministic parties ask for a random Beaver triple, the sources distributively generate and send to the parties (shares of) this triple. This distributed generation is performed among the sources using the BGW protocol. Since in this scenario a majority of the sources are honest, the Beaver triples are guaranteed to remain uncorrupted, and security follows from the security of the GMW protocol.

Our Goal: a Best of Both Worlds Protocol. In each of the two scenarios above, there is a secure protocol for computing $\mathcal{F}$; the trouble is, of course, that the parties do not know *in which* scenario they are. To solve this issue, our aim is, at a high level, to combine the two protocols above into a single *best of both worlds* protocol: a combined protocol which is guaranteed to be secure if at least *one* of the protocols is secure, which is always the case (as the two scenarios cover all possible situations). The idea of reconciling protocols with different security guarantees is not new: it originates in the work of Chaum [Cha90]. Chaum's original motivation was the following: some protocols achieve *computational* security against $n-1$ corruptions, while others achieve *unconditional* security against $t < n/2$ corruptions (assuming secure point-to-point channels); however, no protocol achieves both at the same time. To overcome this limitation, Chaum introduced the idea of combining an *outer protocol*, computing the target functionality and an *inner protocol*, used by the parties to emulate some key sub-functionality required by the outer protocol. Crucially, the inner protocol is never invoked directly on private values held by the parties.

4.3 The Protocol

Outer Protocol $\Pi_{\mathcal{F}}$

Fix n participants $(S_1, \ldots, S_t, P_{t+1}, \ldots, P_n)$, where only the players S_i, $1 \leq i \leq t$ are sources.

Fix an n-party deterministic functionality $\mathcal{F} : (\{0,1\}^\ell)^n \mapsto (\{0,1\}^\kappa)^n$, and let the joint input to the players be $(x_1, \ldots, x_n) \in (\{0,1\}^\ell)^n$. Let κ be the length, known to everyone, of each participant's output.

Initialization. Each source S_i (for $i = 1$ to t) with input x_i computes $n-t$ random additive shares $(y_{i,j})_{t+1 \leq j \leq n}$ of x_i, samples $n-t$ random masks $m_{i,j} \xleftarrow{\$} \{0,1\}^\kappa$, $t+1 \leq j \leq n$ and sends $(y_{i,j}, m_{i,j})$ to party P_j for $j = t+1$ to n.
We call $m_i = \bigoplus_{j=t+1}^n m_{i,j}$ the *output mask* of S_i. We let $z_j = (x_j, y_{1,j}, \ldots, y_{t,j}, m_{1,j}, \ldots, m_{t,j})$ denote the *outer input* of party P_j for $j = t+1$ to n.

Execution. The outer protocol will first run an $(n-t)$-party "GMW + Beaver" protocol (as defined in Section 4.1), run on players $P_{t+1}, \ldots P_n$, and computing the $(n-t)$-party deterministic functionality $\mathcal{F}'$ defined below.
Each time that the "GMW + Beaver" protocol is supposed to request a Beaver triple from the trusted party, all parties run instead the inner protocol represented in Figure 2, which results in players $P_{t+1}, \ldots, P_n$ having what would have the trusted party given them. We note that in the GMW protocol the sequence of requests of the beaver triples is a function of the functionality being computed only (and not of the inputs or of previous random choices).
Functionality. The functionality $\mathcal{F}'$ is defined as follows: on input $(z_{t+1}, \ldots, z_n)$, where z_i is defined as above, let

$$(w_1, \ldots, w_n) = \mathcal{F}\left(\bigoplus_{j=t+1}^n y_{1,j}, \ldots, \bigoplus_{j=t+1}^n y_{t,j}, x_{t+1}, \ldots, x_n\right),$$

where all w_i are seen as elements in $\{0,1\}^\kappa$. Let

$$(m_1, \ldots, m_t) = (\bigoplus_{j=t+1}^n m_{1,j}, \ldots, \bigoplus_{j=t+1}^n m_{t,j}).$$

The functionality $\mathcal{F}'$ outputs w_i to each party P_i for $i \geq t+2$, and $(w_{t+1}, w_1 \oplus m_1, \ldots, w_t \oplus m_t)$ to P_{t+1}.

Output Phase. Once the execution of the protocol is completed, the deterministic parties first execute the output phase of the GMW protocol: for all $(i,j) \in [t+1, n]^2$, P_i sends its share of P_j's output to P_j. Then, P_{t+1} sends $w_i \oplus m_i$ to each source S_i, which unmasks w_i using their output mask m_i. Each deterministic party P_j outputs w_j and each source S_i outputs w_i.

Fig. 1: A t-private, t-sources, n-party protocol $\Pi_{\mathcal{F}}$ for any n-party deterministic functionality $\mathcal{F}$

We represent in Figure 1 and Figure 2 a protocol which allows n players $(S_1, \ldots, S_t, P_{t+1}, \ldots, P_n)$, where only $(S_1, \ldots, S_t)$ can toss coins, to jointly and t-privately compute an arbitrary n-party deterministic functionality of their joint input. The protocol combines a GMW-based *outer protocol* with a BGW-based *inner protocol*. More precisely, *The Outer Protocol (Figure 1).* The outer protocol is essentially the protocol for *scenario 2* above: the sources $(S_1, \ldots, S_t)$ additively share their inputs among the deterministic parties $(P_{t+1}, \ldots, P_n)$, and the latter jointly run an instance of the GMW protocol to evaluate the original functionality to be computed, where the inputs are the inputs of the deterministic parties, and the shares of the inputs of the sources (now known to the deterministic parties). Each time the deterministic players need to receive a random Beaver triple, all players (sources and deterministic parties together) run the *inner protocol* in order to compute the shares of this triple to be given to the deterministic parties.

The Inner Protocol (Figure 2). The inner protocol is the simple protocol of Section 3.2 (which is, in spirit, essentially the same as the protocol of scenario 1), applied to the specific functionality that distributes random shares of a random Beaver triple to the deterministic parties.

A Note on Input and Output Size. In the protocol, we assume that the length of the output of any participant is a priori known to all parties. This is without loss of generality since in the semi-honest model, we can always add a pre-initialization phase where all parties announce their input length to each other. Since t-privacy is only required to hold when all inputs are of the same length, this does not harm privacy. Then, all parties can compute an upper bound on the length of any output by computing $\kappa = \max_{i \in [n]} \max_{\mathbf{x}} |\mathcal{F}_i(\mathbf{x})|$, where the second maximum is taken over all possible inputs of the appropriate length. Finally, all outputs can be interpreted as elements of $\{0,1\}^\kappa$, by padding them with zeroes.

Inner Protocol Π_{in}

Fix n participants $(S_1, \ldots, S_t, P_{t+1}, \ldots, P_n)$, where only the players S_i, $1 \le i \le t$ are sources.

Let $\mathcal{F}_{\mathsf{BT}}$ be the n-party randomized functionality which, on any input to all parties, (ignores the input and) samples two independent unbiased random bits $(a, b) \overset{\$}{\leftarrow} \{0,1\}^2$, and sets $(a_i, b_i, c_i)_{t+1 \le i \le n}$ to be uniformly random $(n-t)$ out of $(n-t)$ shares of $(a, b, a \cdot b)$. It outputs $\perp$ to each source S_i (for $i = 1$ to t) and (a_j, b_j, c_j) to each party P_j (for $j = t+1$ to n). Let B_{BT} be an upper bound on the maximum number of coins tossed by any party during the t-private computation of $\mathcal{F}_{\mathsf{BT}}$ by the BGW protocol.

1. **Tape sharing phase.** Each source S_k for $k = 1$ to t samples $n-t$ random strings $r_{k,\ell} \overset{\$}{\leftarrow} \{0,1\}^{B_{\mathsf{BT}}}$, $t+1 \le \ell \le n$ and sends $r_{k,\ell}$ to P_ℓ. Each deterministic party P_ℓ sets $r_\ell \leftarrow \bigoplus_{k=1}^{t} r_{k,\ell}$. In addition each source S_k, for $k = 1$ to t, samples a random string $r_k \overset{\$}{\leftarrow} \{0,1\}^{B_{\mathsf{BT}}}$
2. **BGW phase** All n parties jointly run the BGW protocol for t-privately computing the n-party randomized functionality $\mathcal{F}_{\mathsf{BT}}$, where they use the strings r_i, $1 \le i \le n$ as their random sources. All players use, e.g., 0 as their input (since $\mathcal{F}_{\mathsf{BT}}$ ignores its input, any fixed input would do).
3. **Output** The output of each party is its output in the BGW protocol.

Fig. 2: The inner protocol Π_{in}

Security Analysis. The intuitive idea behind the security guarantee of our protocol is as follows. Observe that only the deterministic players receive messages that depend on the input of the functionality to be computed. Therefore, if the adversary does not corrupt any deterministic party then security is guaranteed. If, however, there are corrupted deterministic parties, then at least one source is uncorrupted. It follows that from the point of view of the adversary the randomness used in the BGW, inner, protocol is real randomness (provided by the at least one uncorrupted source), and the inner BGW protocol that is run on all players (to compute the Beaver triples) is secure since there is always an honest majority among all players. Since there is always at least one uncorrupted deterministic player (given the relation between n and t) we have that in this case (i.e., when there are corrupted deterministic players) the GMW + Beaver protocol is secure too.

We note in passing that the correctness of our protocol is guaranteed since both the outer and the inner protocol are run on "honest-but-curious" players.

We now formally prove the security property of the protocol. Fix a functionality $\mathcal{F} : (\{0,1\}^\ell)^n \mapsto (\{0,1\}^\kappa)^n$ ($\mathcal{F}$ is deterministic). Let $C_S \subset [t]$ and $C_P \subset [n-t]$ denote the subsets of corrupted sources and corrupted deterministic parties, respectively. Since we consider a static corruption model, $C = (C_S, C_P)$ is known to the simulator. Let $\mathbf{x}$ be the joint input vector of the players. We describe a simulator Sim for the protocol defined above.

If no party is corrupted (i.e., $C_S = C_P = \{\emptyset\}$), the simulation is trivial. We assume now that at least one party is corrupted. On input $(C = (C_S, C_P), \mathbf{x}_C, \mathcal{F}_C(\mathbf{x}))$, Sim distinguishes between two cases: either $C_S \neq [t]$ (there is at least one uncorrupted source, some deterministic parties can be corrupted) or $C_S = [t]$ and $C_P = \{\emptyset\}$ (all sources are corrupted, and only them).

Case 1: $C_S \neq [t]$. Let $t' = |C_S| + |C_P| \leq t$ denote the number of corrupted parties. In the initialization phase, for every $i \in [t] \setminus C_S$ and every $j \in C_P$, Sim sends a uniformly random ℓ-bit string $y_{i,j}$ and a uniformly random κ-bit string $m_{i,j}$ to P_j on behalf of S_i. Sim also stores the input shares $(y_{i,j})_{t+1 \leq j \leq n}$ and the outputs masks m_i of each corrupted source S_i (they can be computed from their inputs and random tapes, which Sim knows). Then, for every $j \in C_P$, Sim computes and stores the outer input $z_j = (x_{1,j}, \ldots, x_{t,j}, m_{1,j}, \ldots, m_{t,j})$ of P_j. Throughout the protocol, Sim maintains a local simulation of the shares held by each corrupted deterministic participant P_j for $j \in C_P$ of the value carried by each wire in the circuit.

When the parties evaluate an XOR gate, Sim simply updates its local simulation of the output wire shares of the corrupted parties, by locally XORing their shares of the input wires. Each time the $n-t$ deterministic parties evaluate an AND gate, Sim behaves as follows:

- *(Tape sharing phase)* For every $k \in [t] \setminus C_S$ and any $\ell \in C_P$, Sim samples a random string $r_{k,\ell}$ on behalf of S_k and sends it to P_ℓ.
- *Emulation of $\mathcal{F}_{\mathsf{BT}}$.* Sim locally emulates $\mathcal{F}_{\mathsf{BT}}$ as follows: it samples uniformly random triples (a_j, b_j, c_j) for every $j \in C_P$, and sets (a_j, b_j, c_j) to be the emulated output of $\mathcal{F}_{\mathsf{BT}}$ to P_j (note that the (a_j, b_j, c_j) are perfectly distributed as in the real execution from the viewpoint of all corrupted P_j, because $|C_P| \leq t < n-t$: there is at least one uncorrupted deterministic party). Then, Sim runs $\mathsf{SimBGW}(C, \mathbf{0}^{t'}, \mathsf{out}_C)$, where SimBGW is the perfect simulator for the BGW protocol, and out_C is equal to $\perp$ for all indices in C_S, and to (a_j, b_j, c_j) for each $j \in C_P$.
- *Secure multiplication.* Sim emulates the broadcast message of each honest deterministic party P_j by broadcasting two uniformly random bits (α_j, β_j) on their behalf.

Output Phase. At the end of the outer protocol, each party P_i, $t+1 \leq i \leq n$ holds a share $s_{i,j}$ of the output of P_j, and all shares held by corrupted deterministic parties are known to Sim. For every $j \in C_P \setminus \{t+1\}$, Sim sends $n - t - |C_P|$ uniformly random shares of $\mathcal{F}_j(\mathbf{x}) \oplus \bigoplus_{i \in C_p} s_{i,j}$ to P_j on behalf of each uncorrupted party P_k. Finally,

- If P_{t+1} is corrupted, Sim sends $n - t - |C_P|$ uniformly random shares of
$$(\mathcal{F}_{t+1}(\mathbf{x}), \mathcal{F}_1(\mathbf{x}) \oplus m_1, \ldots, \mathcal{F}_t(\mathbf{x}) \oplus m_t) \oplus \bigoplus_{i \in C_p} s_{i,1}$$
to P_1, on behalf of each uncorrupted party P_k.
- If P_{t+1} is honest, Sim sends $\mathcal{F}_i(\mathbf{x}) \oplus m_i$ to each corrupted source S_i on P_{t+1} behalf.

We now argue that Sim's emulation is perfect in Case 1. First, the simulation of the input sharing, mask sharing, and tape sharing phases are perfectly identical to an honest execution of the protocol. Second, since $C_S \neq [t]$, there exists at least one uncorrupted source. This guarantees that the string r_ℓ, used by each party P_ℓ as its random tape in (any given instance of) the BGW protocol, is a uniformly random string unknown to any corrupted party. Therefore, we can treat each participant in the BGW protocol as a probabilistic player in the analysis. Now, let (u, v) be the inputs to an AND gate, of which the deterministic parties hold shares; the shares of the corrupted parties are known to Sim. Let (u_j, v_j) denote P_j's share, for $j = t+1$ to n. Recall that the (a_j, b_j) part of the output of $\mathcal{F}_{\mathsf{BT}}$ are uniformly random independent bits. Since Sim's messages (α_j, β_j) are uniformly random, so are $\alpha_j \oplus u_j$ and $\beta_j \oplus v_j$. By the t-privacy of BGW for randomized functionalities, the joint distribution of $\mathsf{SimBGW}(C, \mathbf{0}^{t'}, \mathsf{out}_C)$ and all pairs $(\alpha_j \oplus u_j, \beta_j \oplus v_j)$ for each uncorrupted P_j are distributed perfectly as the joint distribution of the views of the corrupted parties in a real execution, together with the random outputs (a_j, b_j) of $\mathcal{F}_{\mathsf{BT}}$ to all honest deterministic parties. Therefore, $\mathsf{SimBGW}(C, \mathbf{0}^{t'}, \mathsf{out}_C)$ together with the simulated messages (α_j, β_j) is perfectly distributed as the corrupted parties' views together with the real messages $(\alpha_j, \beta_j) = (u_j \oplus a_j, v_j \oplus b_j)$.

It remains to argue that the simulation is perfect for the output phase as well. Consider a given output wire where P_j should receive the output, and assume without loss of generality that this wire goes out of an AND gate (since the parties can always add a dummy multiplication by 1 to any output gate). For simplicity, let us assume for now that $j \neq t+1$ (the case $j = t+1$ can be handled similarly). Let u, v be the inputs to this AND gate: it holds that $u \cdot v = \mathcal{F}_j(\mathbf{x}, \mathbf{y})$ (the case where $j = t+1$ is identical, except that $\mathcal{F}_{t+1}(\mathbf{x}, \mathbf{y})$ must be replaced by the longer output of P_{t+1}, which also includes the masked output of other players). Therefore, by the definition of $\mathcal{F}_{\mathsf{BT}}$, the triples (a_k, b_k, c_k) are uniformly random triples conditioned on the c_k being random shares of $\mathcal{F}_{j+t}(\mathbf{x}, \mathbf{y}) \oplus \alpha \cdot v \oplus \beta \cdot u$, where α, β are the XOR of the messages α_k, β_k broadcast by all deterministic parties.

For each $k \in [t+1, n] \setminus C_P$, let us call γ_k the simulated random share of $\mathcal{F}_j(\mathbf{x}) \oplus \bigoplus_{i \in C_P} s_{i,j}$ sent by Sim on behalf of P_k. By construction, the sequence $(\gamma_k)_k$ and the sequence $(s_{i,j})_i$ jointly constitute uniformly random shares of $\mathcal{F}_j(\mathbf{x}) = u \cdot v$. Let us rewrite w_k the share of each party P_k for notational convenience (each w_k is either $s_{k,j}$ or γ_k, depending on whether P_k is corrupted or not). Then the (α_k, β_k, w_k) and the input shares (u_k, v_k) virtually define triples (a_k, b_k, c_k) as $(\alpha_k \oplus u_k, \beta_k \oplus v_k, w_k \oplus \alpha \cdot v_k \oplus \beta \cdot (\alpha_k \oplus u_k))$ which, by construction, are uniformly distributed as random Beaver triples. By the t-privacy of BGW for randomized functionalities, the joint distribution of $\mathsf{SimBGW}(C, \mathbf{0}^{t'}, \mathsf{out}_C)$ together with all triples $(\alpha_k \oplus u_k, \beta_k \oplus v_k, w_k \oplus \alpha \cdot v_k \oplus \beta \cdot (\alpha_k \oplus u_k))$ is perfectly indistinguishable from the views of the corrupted parties in a real execution together with random Beaver triples (a_k, b_k, c_k). This implies that the joint distribution of $\mathsf{SimBGW}(C, \mathbf{0}^{t'}, \mathsf{out}_C)$ together with the simulated messages $(\alpha_k, \beta_k, \gamma_k)$ are perfectly distributed as in a real execution (with random Beaver triples (a_k, b_k, c_k) equal to $(\alpha_k \oplus u_k, \beta_k \oplus v_k, w_k \oplus \alpha \cdot v_k \oplus \beta \cdot (\alpha_k \oplus u_k))$). This concludes the analysis for Case 1.

Case 2: $C_S = [t]$. In this case, the adversary corrupted all sources, and only them. Observe that all messages seen by the sources during the entire execution of the protocol are of two types:

- Messages exchanged during an execution of a BGW protocol for a secure multiplication. In all such instances, the inputs of the honest parties are never involved, since the BGW instances evaluate an input-independent randomized functionality.
- The output message, where each source S_i receives $w_i \oplus m_i$ from P_1, where m_i is a mask chosen by S_i and w_i is S_i's output in the protocol.

The simulation is therefore straightforward: during the entire execution of the protocol, Sim will just play honestly on behalf of the (deterministic) P_i's, and using inputs 0. Since the P_i only interact with the sources in input-independent protocols, this simulation is perfectly indistinguishable from the honest execution. Finally, Sim sends $w_i \oplus m_i$ to each source S_i at the end of the protocol on behalf of P_1, where w_i is $\mathcal{F}_i(\mathbf{x}, \mathbf{y})$ and m_i can be deterministically computed from the random tape of S_i (hence both are known to Sim). This concludes the analysis for Case 2.

5 On the Relation Between Randomness Complexity and the Number of Random Sources

In this section, we are interested in the relation between the randomness complexity of a perfect secure protocol for a given functionality, and the number of random sources that this protocol uses. The main question we ask is:

Does keeping the number of random sources to a minimum come at the cost of increasing the the number of random bits necessary for the perfect secure computation?

To illustrate the question, consider the t-source protocol we described in Section 4. There is a natural variant of the protocol that uses n sources, but $\Omega(t)$ times less randomness: each time the parties jointly generate a Beaver triple using BGW (which are essentially the only steps that use randomness, besides the input sharing and output masking), the parties use their own random tape, instead of letting each of the $n-t$ deterministic parties aggregate (*i.e.*, XOR) t random tapes sent by each of the t sources. Ignoring the input sharing and output masking (for large enough functions, the Beaver triple generation dominates the cost), if the circuit has n_{AND} AND gates, this reduces the randomness complexity from $t \cdot (n-t+1) \cdot B_{\mathsf{BT}} \cdot n_{\mathsf{AND}}$ bits to $n \cdot B_{\mathsf{BT}} \cdot n_{\mathsf{AND}}$ bits: a $\Omega(t)$ reduction. In the following, we put forth a conjecture stating that this factor t cost is essentially inherent.

5.1 A Preliminary Investigation

The randomness complexity of t-private n-party computation for the XOR functionality was studied in the work of [KM96]. Interestingly, this work achieves in particular the best known upper bound on the randomness complexity of t-private n-party XOR, and this upper bound is achieved using a minimal number of sources: exactly t. This seemingly contradicts the intuition that t-source private computation should require more randomness than private computation without limitations on the number of sources.

A conjecture on sources versus randomness. We warn the reader that what follows is a purely intuitive reasoning: our goal here is to develop an intuition about which conjecture can reasonably be expected. Intuitively, all known private computation protocols proceed , one way or another, by operating on random shares – this is how multiple parties can jointly manipulate private values. These shares do typically enjoy linear homomorphism: all linear functions can be computed on the shares, without any communication. This creates a crucial distinction between linear functions and nonlinear functions in secure computation: in the former, following an input-sharing phase, the protocol involves only local computation followed by a reconstruction of the final output. In contrast, for nonlinear function, there will necessarily be interactions where *intermediate values of the computation* are jointly manipulated and used to communicate.

Now, consider any t-source protocol for t-privately computing a nonlinear function. In this protocol, there necessarily exist parties that will be involved (*i.e.*, receive messages) in exchanges involving intermediate values of the computation – we call them *sensitive parties*, in the sense that they see messages where random coins are used to hide sensitive intermediate values. Assume that the adversary simultaneously corrupts some sources *and* one (or more) of the sensitive parties. Then, when a deterministic party sends a sensitive message to one of the sensitive parties, the randomness used to generate this message must be uncorrupted, and can only come from the sources. But the deterministic party cannot possibly know *which* of the sources are uncorrupted: it appears unavoidable, then, that the party must aggregate randomness for *all* t sources to obtain uncorrupted coins in this situation. Therefore, we expect that there should exists *nonlinear functions* where the $n-t$ deterministic parties will necessarily have to receive t coins from the sources for each coin they would have tossed themselves if they had the ability to. The above (informal) discussion clearly breaks down when we consider solely linear functions; for sufficiently complex functions, however, the t factor appears inherent.

Above, we did not precisely define what are *linear* and *nonlinear* functionalities. The informal conditions we stated, however, corresponds to the feature that partitions all Boolean functions into two classes: those that can be n-privately computed, and those that can only be t-privately computed for $t < n/2$ [CK91]. It also corresponds to the feature that typically distinguishes functionalities that admit n-party t-private protocols for $t \geq n/2$. For example, the subprotocol computing the AND gates in a circuit is the only component in the BGW protocol [BGW88] that requires an honest majority, i.e., $t < n/2$; all other components handle linear parts of the circuit, and can have $t \geq n/2$ (this is discussed in details in [AL17]). Similarly, the (BGW-based) protocol used in our work to handle the AND gates is also precisely what requires a t-times randomness blowup. Hence, it seems plausible to expect that the functionalities for which the factor-t randomness blowup is inherent are the same functionalities for which private computation requires an honest majority. We state this in Conjecture 13. We view the proof of Conjecture 13 an interesting open question.

Conjecture 13. Let $\mathcal{F}$ be an n-party functionality that cannot be t-privately computed for $t \geq n/2$. For any $t < n/2$, let R_t be the randomness complexity of t-privately computing $\mathcal{F}$ (with any number of sources). Then the t-source randomness complexity of $\mathcal{F}$ is $\Theta(t \cdot R_t)$.

Results for the AND functionality. Characterizing the minimal amount of randomness required for securely computing a functionality is non-trivial in general, and indeed, no such general characterization is known. We expect that relating the randomness complexity to the number of sources might be of comparable difficulty in general. Most previous works on randomness complexity focused on simple functionalities such as n-party XOR and n-party AND, as these are simple building blocks in other computations, and in order to make the problem tractable. Even for these simple functionalities randomness lower bounds are difficult to obtain, and in almost all known cases (with the exception of the 1-private computation of XOR, which can be done using a single random bit, and which is tight since private computation of XOR without randomness is impossible), the known upper bounds do not match the known lower bounds. Therefore, we focus on the following simpler question:

When t is a constant, is it possible to match the best known upper bound on the randomness complexity of simple functionalities such as n-party AND, using an optimally small number of random sources?

Note that t being a constant captures a setting where matching the best known randomness complexity would not contradict Conjecture 13. In particular, when $t = 1$, we ask: is it possible to 1-privately compute a functionality, using a single source, with a randomness complexity matching the best known multi-source randomness complexity for this functionality? The randomness complexity of the 1-private computation of the n-party AND functionality was investigated in a recent work [KOP$^+$19]. Their upper bound uses 8 random bits to 1-privately compute AND (for any number of parties), and uses two random sources; they also give a lower bound stating that more than a single random bit is necessary to 1-privately compute AND. We obtain a non-trivial, and perhaps surprising result: we give a 6-bit 1-private protocol to compute AND (for any number of parties), which uses a single random source, thus improving the result of [KOP$^+$19] both in the number of random bits and the number of sources. While this result is interesting in the context of understanding the relation between the number of sources and the randomness complexity, we view it as being mainly a result of independent interest on the randomness complexity of AND.

5.2 Randomness Upper Bound for Secure Computation of AND

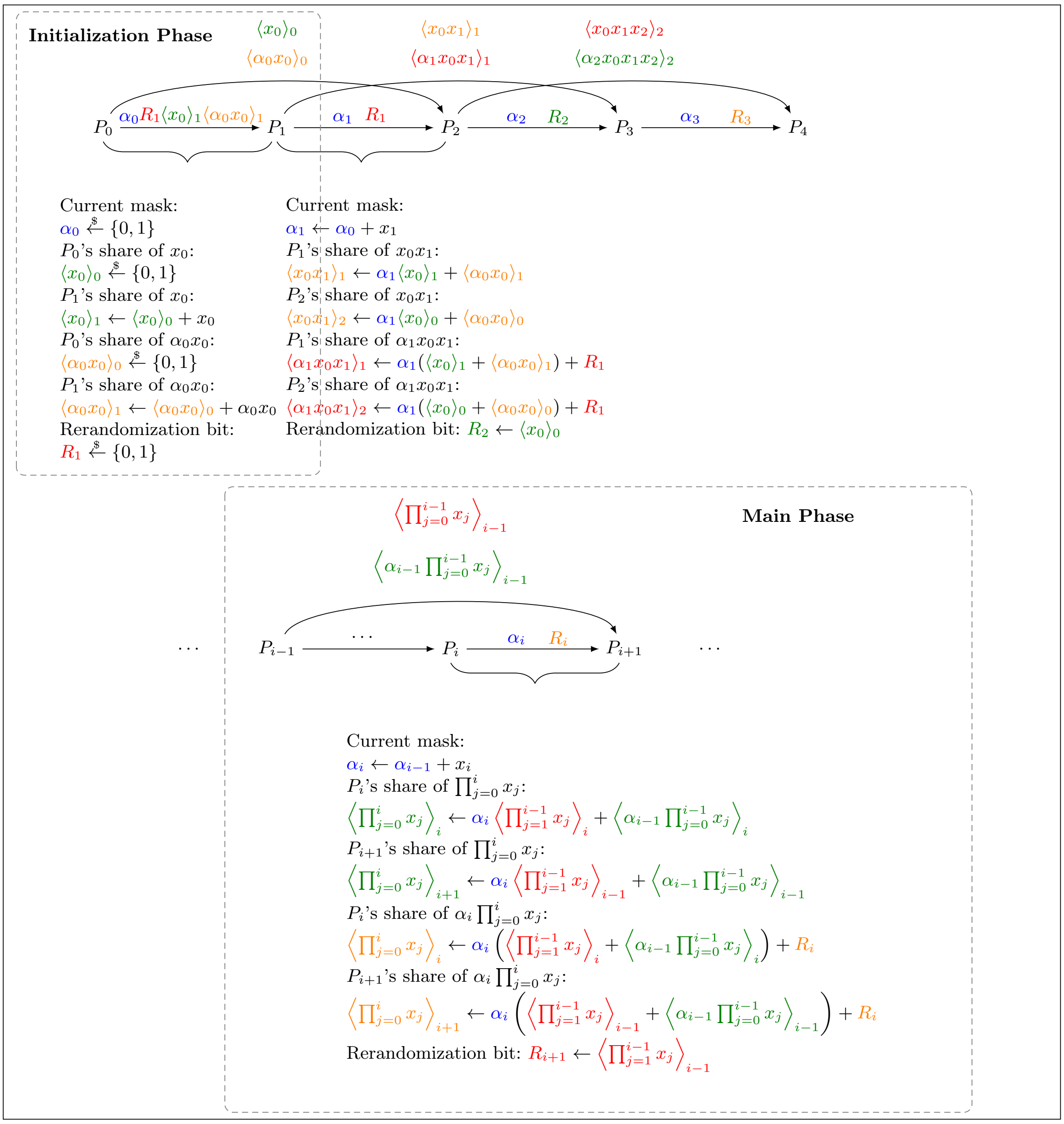
For the n-party AND functionality, not much is known in the general case of t-privacy. A long-standing open question exists even in the case of 1-privacy, where there is still a considerable gap between the best known upper and lower bounds on the number of random bits required for the 1-private computation of AND. Here, the only known non-trivial lower bound was recently proven in [KOP$^+$19]: More than one random bits are required for the 1-private computation of n-party AND. In the same work, the authors also improved the previously best known upper bound from 73 bits (implicit in [KOR99]) to 8 bits.

The protocol of [KOP$^+$19] requires two sources. In contrast, the 73-bit protocol of [KOR99] requires a single source (which is optimal). It is therefore natural to wonder whether two sources are necessary to achieve a very low randomness complexity. Perhaps surprisingly, our result in this section indicates that it might not be the case: We give a 1-private protocol for n-party AND which uses 6 random bits, and a single source. Our result also tightens the gap between the best known lower and upper bounds on the randomness of AND in general, which is of independent interest.

Theorem 14. *For any $n \geq 3$, let $\mathcal{F}_{\mathsf{AND}n}$ be the following n-party functionality: on input of a single bit x_i to each party P_i, $\mathcal{F}_{\mathsf{AND}n}$ outputs $\bigwedge_{i=1}^{n} x_i$ to all parties.*

There exists an n-party perfect 1-private protocol for $\mathcal{F}_{\mathsf{AND}n}$ that uses a single source, and where that source tosses exactly 6 random coins.

A pictural representation of the full protocol is given on Figure 3 (initialization phase and main phase) and Figure 4 (output phase). Before we proceed with the formal proof, we explain the intuition underlying the protocol. At its heart is a "transition protocol" (the main phase) which transitions from parties P_{i-1} and P_i having shares of $\bigwedge_{j=0}^{i-1} x_j$ to P_i and P_{i+1} having shares of

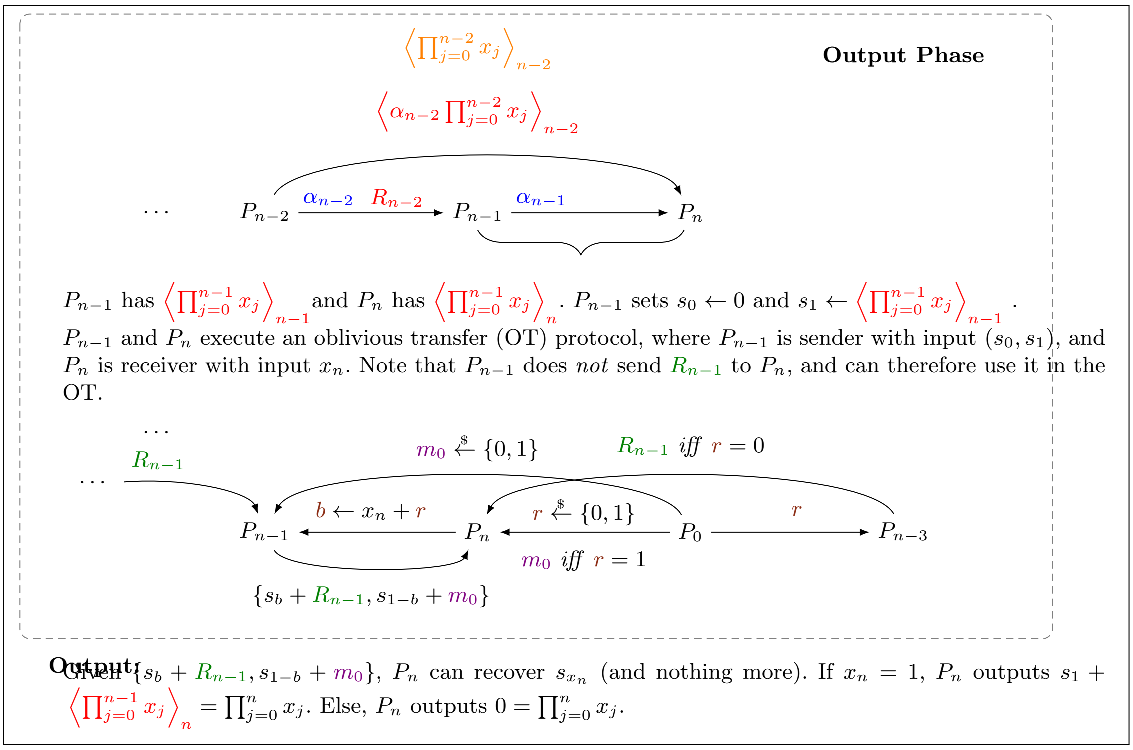


Initialization Phase
$\langle x_0\rangle_0$
$\langle \alpha_0 x_0\rangle_0$
$\langle x_0 x_1\rangle_1$
$\langle \alpha_1 x_0 x_1\rangle_1$
$\langle x_0 x_1 x_2\rangle_2$
$\langle \alpha_2 x_0 x_1 x_2\rangle_2$
P_0
$\alpha_0 R_1 \langle x_0\rangle_1 \langle \alpha_0 x_0\rangle_1$
P_1
α_1 R_1
P_2
α_2 R_2
P_3
α_3 R_3
P_4
Current mask:
$\alpha_0 \overset{\$}{\leftarrow} \{0,1\}$
P_0's share of x_0:
$\langle x_0\rangle_0 \overset{\$}{\leftarrow} \{0,1\}$
P_1's share of x_0:
$\langle x_0\rangle_1 \leftarrow \langle x_0\rangle_0 + x_0$
P_0's share of $\alpha_0 x_0$:
$\langle \alpha_0 x_0\rangle_0 \overset{\$}{\leftarrow} \{0,1\}$
P_1's share of $\alpha_0 x_0$:
$\langle \alpha_0 x_0\rangle_1 \leftarrow \langle \alpha_0 x_0\rangle_0 + \alpha_0 x_0$
Rerandomization bit:
$R_1 \overset{\$}{\leftarrow} \{0,1\}$
Current mask:
$\alpha_1 \leftarrow \alpha_0 + x_1$
P_1's share of $x_0 x_1$:
$\langle x_0 x_1\rangle_1 \leftarrow \alpha_1 \langle x_0\rangle_1 + \langle \alpha_0 x_0\rangle_1$
P_2's share of $x_0 x_1$:
$\langle x_0 x_1\rangle_2 \leftarrow \alpha_1 \langle x_0\rangle_0 + \langle \alpha_0 x_0\rangle_0$
P_1's share of $\alpha_1 x_0 x_1$:
$\langle \alpha_1 x_0 x_1\rangle_1 \leftarrow \alpha_1(\langle x_0\rangle_1 + \langle \alpha_0 x_0\rangle_1) + R_1$
P_2's share of $\alpha_1 x_0 x_1$:
$\langle \alpha_1 x_0 x_1\rangle_2 \leftarrow \alpha_1(\langle x_0\rangle_0 + \langle \alpha_0 x_0\rangle_0) + R_1$
Rerandomization bit: $R_2 \leftarrow \langle x_0\rangle_0$
Main Phase
$\left\langle \prod_{j=0}^{i-1} x_j \right\rangle_{i-1}$
$\left\langle \alpha_{i-1} \prod_{j=0}^{i-1} x_j \right\rangle_{i-1}$
$\cdots$
P_{i-1}
$\cdots$
P_i
α_i R_i
P_{i+1}
$\cdots$
Current mask:
$\alpha_i \leftarrow \alpha_{i-1} + x_i$
P_i's share of $\prod_{j=0}^{i} x_j$:
$\left\langle \prod_{j=0}^{i} x_j \right\rangle_i \leftarrow \alpha_i \left\langle \prod_{j=1}^{i-1} x_j \right\rangle_i + \left\langle \alpha_{i-1} \prod_{j=0}^{i-1} x_j \right\rangle_i$
P_{i+1}'s share of $\prod_{j=0}^{i} x_j$:
$\left\langle \prod_{j=0}^{i} x_j \right\rangle_{i+1} \leftarrow \alpha_i \left\langle \prod_{j=1}^{i-1} x_j \right\rangle_{i-1} + \left\langle \alpha_{i-1} \prod_{j=0}^{i-1} x_j \right\rangle_{i-1}$
P_i's share of $\alpha_i \prod_{j=0}^{i} x_j$:
$\left\langle \prod_{j=0}^{i} x_j \right\rangle_i \leftarrow \alpha_i \left(\left\langle \prod_{j=1}^{i-1} x_j \right\rangle_i + \left\langle \alpha_{i-1} \prod_{j=0}^{i-1} x_j \right\rangle_i \right) + R_i$
P_{i+1}'s share of $\alpha_i \prod_{j=0}^{i} x_j$:
$\left\langle \prod_{j=0}^{i} x_j \right\rangle_{i+1} \leftarrow \alpha_i \left(\left\langle \prod_{j=1}^{i-1} x_j \right\rangle_{i-1} + \left\langle \alpha_{i-1} \prod_{j=0}^{i-1} x_j \right\rangle_{i-1} \right) + R_i$
Rerandomization bit: $R_{i+1} \leftarrow \left\langle \prod_{j=1}^{i-1} x_j \right\rangle_{i-1}$

Fig. 3: Initialization phase and main phase of the 1-private protocol for the $(n+1)$-AND functionality. Each party P_i has an input bit x_i. P_n receives the output $\prod_{j=1}^{n} x_j$. The protocol uses six random bits in total $(\alpha_0, R_1, \langle x_0\rangle_0, \langle \alpha_0 x_0\rangle_0, m_0, r)$, which are all generated by the single random source P_0. The color of a value indicates which of the six random bits masks it additively. All operations are over $\mathbb{F}_2$.

$\bigwedge_{j=0}^{i} x_j$, while maintaining a carefully chosen invariant (described in the *main phase* below), and using exactly four random bits. Crucially, these four random bits can be reused for each step, with a cyclic shift of their roles (this will become clearer in the sequel). The output phase requires a final oblivious transfer to reconstruct the output, which requires 3 bits. One of the four bits of the main phase can actually be recycled for this purpose, hence only two "fresh" bits are required, leading to the 6 bits total cost.

The main phase. To simplify the exposition, we consider here $n+1$ (and not n) parties, $(P_0, \cdots, P_n)$ (starting with index 0) each holding an input bit x_i. We will further assign a color code to the four random bits used during the main phase. During the main phase, the parties are placed on a line; for each $i \geq 1$ each party P_{i-1} will send messages solely to P_i and P_{i+1}. Fix three


Output Phase
$\left\langle \prod_{j=0}^{n-2} x_j \right\rangle_{n-2}$
$\left\langle \alpha_{n-2} \prod_{j=0}^{n-2} x_j \right\rangle_{n-2}$
$\dots$
P_{n-2}
α_{n-2} R_{n-2}
P_{n-1}
α_{n-1}
P_n
$\dots$
R_{n-1}
$m_0 \xleftarrow{\$} \{0,1\}$
R_{n-1} *iff* $r = 0$
$b \leftarrow x_n + r$
$r \xleftarrow{\$} \{0,1\}$
r
P_{n-1}
P_n
P_0
P_{n-3}
m_0 *iff* $r = 1$
$\{s_b + R_{n-1}, s_{1-b} + m_0\}$

P_{n-1} has $\left\langle \prod_{j=0}^{n-1} x_j \right\rangle_{n-1}$ and P_n has $\left\langle \prod_{j=0}^{n-1} x_j \right\rangle_{n}$. P_{n-1} sets $s_0 \leftarrow 0$ and $s_1 \leftarrow \left\langle \prod_{j=0}^{n-1} x_j \right\rangle_{n-1}$. P_{n-1} and P_n execute an oblivious transfer (OT) protocol, where P_{n-1} is sender with input (s_0, s_1), and P_n is receiver with input x_n. Note that P_{n-1} does *not* send R_{n-1} to P_n, and can therefore use it in the OT.

Output: Given $\{s_b + R_{n-1}, s_{1-b} + m_0\}$, P_n can recover s_{x_n} (and nothing more). If $x_n = 1$, P_n outputs $s_1 + \left\langle \prod_{j=0}^{n-1} x_j \right\rangle_{n} = \prod_{j=0}^{n} x_j$. Else, P_n outputs $0 = \prod_{j=0}^{n} x_j$.

Fig. 4: Output phase of the 1-private protocol for the $(n+1)$-AND functionality. Each party P_i has an input bit x_i. P_n receives the output $\prod_{j=1}^{n} x_j$. The protocol uses six random bits in total ($\alpha_0, R_1, \langle x_0 \rangle_0, \langle \alpha_0 x_0 \rangle_0, m_0, r$), which are all generated by the single random source P_0. The color of a value indicates which of the six random bits masks it additively. All operations are over $\mathbb{F}_2$.

parties (P_{i-1}, P_i, P_{i+1}). The protocol will have in sequence the following invariant for all i between 1 and $n-2$ (all sums are over $\mathbb{F}_2$):

- P_{i-1} and P_i hold a *mask* α_{i-1}, which is a random bit (from the viewpoint of P_j for $j \geq i$); let us call it the *blue bit*.
- P_i and P_{i+1} hold a *rerandomizer* R_i, which is also a random bit (from the viewpoint of P_j for $j \geq i$), and which we call the *red bit*.
- P_{i-1} and P_i hold random additive shares of $\prod_{j=0}^{i-1} x_j$ (the AND of all inputs up to P_{i-1}). That is, P_{i-1} holds a random bit u_{i-1}, and P_i holds $u'_{i-1} = u_{i-1} + \prod_{j=0}^{i-1} x_j$. We call u_{i-1} and u'_{i-1} the *green bit* of P_{i-1} and P_i respectively.
- In addition, P_{i-1} and P_i hold random additive shares of $\alpha_{i-1} \cdot \prod_{j=0}^{i-1} x_j$. That is, P_{i-1} holds a random bit v_{i-1}, and P_i holds $v'_{i-1} = v_{i-1} + \alpha_{i-1} \cdot \prod_{j=0}^{i-1} x_j$. We call v_{i-1} and v'_{i-1} the *orange bit* of P_{i-1} and P_i respectively. We use these notations and definitions to clarify that each share is a uniformly random bit from the viewpoint of the party that holds it, but the joint distribution (u_{i-1}, v_{i-1}) has a single bit of entropy.

It remains to explain how the parties communicate in order to transform the above situation to the corresponding situation (invariant) relative to P_i, P_{i+1} and P_{i+2}. Throughout this transition, the role of the blue bit remains identical, while the roles of the last three random bits will undergo a cyclic shift. The transition proceeds as follows:

- P_i defines the *new mask* α_i to be $\alpha_{i-1} + x_i$, and sends it to P_{i+1}. Observe that from the viewpoint of all parties $j \geq i+2$ (which we call "remaining parties" with respect to i), α_i is a uniform random bit (since x_i is masked by the uniform random "blue" bit α_{i-1}, about which they have no information). The bit α_i is defined as the new blue bit.
- P_{i-1} sends its two shares (that is, the green random bit u_{i-1} and the orange random bit v_{i-1}) to P_{i+1}.

- P_i computes $u_i = \alpha_i \cdot u'_{i-1} + v'_{i-1}$ and P_{i+1} computes $u'_i = \alpha_i \cdot u_{i-1} + v_{i-1}$. Observe that

$$\begin{aligned} u_i + u'_i &= \alpha_i(u_{i-1} + u'_{i-1}) + v_{i-1} + v'_{i-1} \\ &= (\alpha_{i-1} + x_i) \cdot \prod_{j=0}^{i-1} x_j + \alpha_{i-1} \cdot \prod_{j=0}^{i-1} x_j = \prod_{j=0}^{i} x_j, \end{aligned}$$

  hence u_i and u'_i do indeed form shares of $\prod_{j=0}^{i} x_j$. Furthermore, from the view point of the remaining parties with respect to i, these shares are indeed random, because u_i is additively masked with v'_{i-1} – that is, the *orange bit* of P_i. Therefore, we view the bits u_i and u'_i as the new *orange bit* of P_i and P_{i+1} respectively.
- P_i computes $v_i = \alpha_i \cdot (u'_{i-1} + v'_{i-1}) + R_i$ and P_{i+1} computes $v'_i = \alpha_i \cdot (u_{i-1} + v_{i-1}) + R_i$. Observe that

$$\begin{aligned} v_i + v'_i &= \alpha_i(u_{i-1} + u'_{i-1} + v_{i-1} + v'_{i-1}) \\ &= (\alpha_{i-1} + x_i) \cdot (\prod_{j=0}^{i-1} x_j + \alpha_{i-1} \cdot \prod_{j=0}^{i-1} x_j) \\ &= (\alpha_{i-1} + x_i) \cdot \prod_{j=0}^{i} x_j = \alpha_i \cdot \prod_{j=0}^{i} x_j, \end{aligned}$$

  hence v_i and v'_i do indeed form shares of $\alpha_i \cdot \prod_{j=0}^{i} x_j$. Furthermore, from the view point of the remaining parties with respect to i, these shares are indeed random, because v_i is additively masked with R_i – that is, the *red bit* of P_i (without R_{i-1}, which both parties add to their shares, the shares would be biased towards 0; the purpose of the rerandomizer R_i is precisely to rerandomize these shares). Therefore, we view v_i and v'_i as the new *red bit* of P_i and P_{i+1} respectively.
- It remains to set a new rerandomizer R_{i+1} bit to be used by P_{i+1} and P_{i+2}. Above, the blue, orange, and red bits have already been "used", hence we will recycle the green bit. Recall that P_{i+1} received P_{i-1}'s green random bit u_{i-1}, which is a uniformly random bit from the viewpoint of P_{i+1} and P_{i+2}. Therefore, P_{i+1} sets $R_{i+1} \leftarrow u_{i-1}$ and sends R_{i+1} to P_{i+2}.

The transition, and the invariant it maintains, are represented in Figure 5.

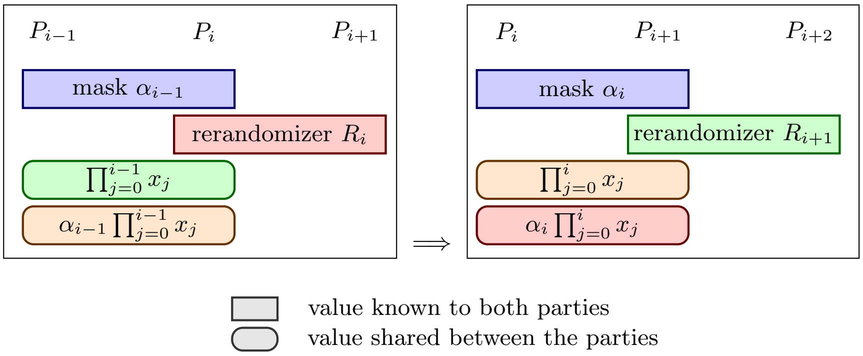


Fig. 5: Pictural representation of the transition during the main phase. The color of a rectangle indicates which random bit has been used to randomize the value. Straight corners indicate values known to both parties, rounded corners indicate values shared between the parties. The transition shows the colors of the bottom three values being cycled upward, while the top color remains the same.

Initialization phase. The initialization phase sets up the invariant, for $i = 1$, in a relatively straightforward way. The source, P_0, samples the four random bits (the blue, green, orange, and red bits). It sets the first mask α_0 to be the blue bit, the first rerandomizer R_1 to be the red bit, and uses the green and orange bits to share its input x_0 as well as the value $\alpha_0 x_0$; that is, it sets the green bit to be $\langle x_0\rangle_0$ and the red bit to be $\langle \alpha_0 x_0\rangle_0$, and defines $\langle x_0\rangle_1 \leftarrow x_0 + \langle x_0\rangle_0$ and $\langle \alpha_0 x_0\rangle_1 \leftarrow \alpha_0 x_0 + \langle \alpha_0 x_0\rangle_0$. It sends $(\langle x_0\rangle_1, \langle \alpha_0 x_0\rangle_1)$ to P_1, together with α_0 and R_1. P_1 sends R_1 to P_2, and the first transition phase can begin (where P_0 will send the remaining shares $(\langle x_0\rangle_0, \langle \alpha_0 x_0\rangle_0)$ to P_2; observe that those are exactly the green and the orange random bits).

Output phase. It remains to describe how the parties obtain the final output. Towards the end of the protocol, we slightly change the last (the $n-1$'th) transition. Let us assume without loss of generality that the rerandomization bit R_{n-1} is the green bit; P_{n-1} does *not* send this rerandomization bit R_{n-1} (which P_{n-1} received from P_{n-3}) to P_n. Otherwise the $n-1$'th transition remains the same and after its execution P_{n-1} and P_n hold random shares of $\prod_{j=0}^{n-1} x_j$.

The players P_{n-1} and P_n will then execute an information theoretic oblivious transfer protocol with the help of P_0 and P_{n-3}, where P_{n-1} will be the sender with inputs $s_0 = 0$ and its share of $\prod_{j=0}^{n-1} x_j$ (denoted s_1), and P_n will be the receiver, with input x_n. This way, P_n receives 0 if $x_n = 0$ and $\prod_{j=0}^{n-1} x_j$ otherwise; that is, P_n learns exactly $\prod_{j=0}^{n} x_j$.

The oblivious transfer proceeds as follows: the source P_0 generates two additional random bits (r, m_0). The bit r will be used to mask the *selection bit* x_n, and the two bits (m_0, R_{n-1}) will be used to mask the sender inputs (this is where we recycle R_{n-1}). P_0 sends $(r, r\cdot m_0)$ to P_n, and m_0 to P_{n-1}. Additionally, it sends r to P_{n-3}, who sends $R_{n-1}\cdot(1+r)$ to P_n. This way, P_n knows r and either R_{n-1} (if $r=0$) or m_0 (if $r=1$); this is exactly a random oblivious transfer correlation.

The actual oblivious transfer follows the standard information-theoretic oblivious transfer using a random oblivious transfer: P_n sends $b = x_n + r$ (its masked selection bit), and P_{n-1} replies with $(s_b + R_{n-1}, s_{1-b} + m_0)$. If $r = 0$, this pair is $(s_{x_n} + R_{n-1}, s_{1+x_n} + m_0)$ and P_n knows the mask R_{n-1}; otherwise, if $r = 1$, this pair is $(s_{1+x_n} + R_{n-1}, s_{x_n} + m_0)$ and P_n knows the mask m_0. Either way, P_n can extract and output $s_{x_n} = \prod_{j=0}^{n} x_j$. In total, two additional random bits were generated by P_0, bringing the total randomness complexity to six bits.

The three-party case. Reading the above, it seems that the protocol requires at least four parties, since P_{n-3} to P_n are involved. However, P_{n-3} is involved solely because P_n should receive $R_{n-1}\cdot(1+r)$, but P_0 might not know the value R_{n-1}. In the three party case, where $P_0 = P_{n-2}$, this is actually not an issue, because $R_1 = R_{n-1}$ is the first rerandomizer, generated by P_0 during the initialization. Therefore, in the three party case, we only need to slightly change the output phase by letting P_0 send $R_{n-1}\cdot(1+r)$ directly to P_n, without involving P_{n-3} (who does not exist).

5.3 Security Analysis

Consider a static adversary corrupting a party P_i. We exhibit a simulator Sim which emulates the view of this corrupted party given i, x_i, and the output of the function, $\prod_{j=0}^{n} x_j$. First, if $\prod_{j=0}^{n} x_j = 1$, then the simulation is trivial, since Sim knows all inputs: Sim can simply honestly emulate the role of all honest parties. Therefore, we assume without loss of generality that $\prod_{j=0}^{n} x_j = 0$. The proof is somewhat different depending on which is the corrupted player, whether it is a “middle" player, one of the “end" players, or one of the two “starting" players.

Case 1: $i \notin \{0, 1, n-1, n\}$. This case corresponds to parties P_i which participate only in the main phase, except for $n-3$, which receives an additional bit from P_0 in the output phase. We first prove the simulation of the views at the end of the main phase.
In the real protocol, each P_i receives exactly four bits by the end of the main phase:

- From player P_{i-1}: a mask α_{i-1} and a rerandomizer R_{i-1}; and
- From player P_{i-2}: P_{i-2}'s random share of $\prod_{j=0}^{i-2} x_j$ and of $\alpha_{i-2}\cdot\prod_{j=0}^{i-2} x_j$.

The simulation is straightforward: Sim samples and sends to P_i two random bits on behalf of P_{i-1} and two other random bits on behalf of P_{i-2}. It remains to show that this simulation is perfectly indistinguishable from a real protocol execution. For α_{i-1}, observe that by construction,

$\alpha_{i-1} = \alpha_0 + \sum_{j=1}^{i-1} x_j$, where α_0 is a uniform random bit sampled by (the honest party) P_0; hence, the simulated α_{i-1} is distributed exactly as in the real protocol.

For the three remaining bits, the proof proceeds by induction on i, $i = 2$ being the base case. For $i = 2$, P_2 receives in the real protocol the red random bit from P_1 and the green and the orange random random bits from P_0; i.e., in the real protocol, P_2 receives three independent random bits, as in the simulation.

Assume now that it has already been established that the simulation is correct for all players P_2 to P_{i-1}. Let (u, u') denote the two values sent by P_{i-3} to P_{i-1} in the protocol, and let (v, v') be the two corresponding shares held by P_{i-2}. Let α_{i-3} be the mask received by P_{i-2}, and let (α_{i-2}, R_{i-2}) be the two bits sent by P_{i-2} to P_{i-1}. By the induction hypothesis, it holds that both the joint distribution of $(\alpha_{i-3}, u, v', R_{i-2})$ and the joint distribution of $(\alpha_{i-3}, v, v', R_{i-2})$ are each exactly the uniform distribution over $\{0,1\}^4$. Then, observe that P_i receives in the real protocol the following:

- From P_{i-2}: $\alpha_{i-3}v + v'$ and $\alpha_{i-3}(v + v') + R_{i-2}$; and
- From P_{i-1}: α_{i-1} and $R_{i-1} = u$.

By the induction hypothesis, the joint distribution of $(\alpha_{i-3}v + v', \alpha_{i-3}(v + v') + R_{i-2}, u, \alpha_{i-1})$ is the uniform distribution over $\{0,1\}^3$, which completes the induction step.

As to player number $n-3$, in addition to the messages received during the main phase it receives a single additional independent (from all other messages) random bit from P_0. The simulation is trivial by sending on behalf of P_0 a fresh random bit.

Case 2: $i \in \{n-1, n\}$. We now look at the parties that are involved in the output phase. The view of P_{n-1} in the protocol consists of (m_0, b), where m_0 is an independent random bit, and $b = x_n + r$, where r is an independent random bit. This is again perfectly simulated by sending a random bit m_0 on behalf of P_0 and a random bit b on behalf of P_n. Finally, the view of P_n is the following:

- P_n knows α_{n-1} and the two shares (u, u') of P_{n-2} (u, u' are P_{n-2}'s shares of $\prod_{i=0}^{n-2} x_i$ and $\alpha_{n-2} \cdot \prod_{i=0}^{n-2} x_i$ with P_{n-1});
- P_n receives $(r, r \cdot m_0, R_{n-1} \cdot (1 + r))$ from P_0 and P_{n-3};
- P_0 receives the two OT messages of P_{n-1}.

The two OT messages of P_{n-1} are equal to $(s_b + R_{n-1}, s_{1-b} + m_0)$, where $s_0 = 0$, and s_1 is P_{n-1}'s share of $\prod_{i=0}^{n-1} x_i$, the other share being, by construction, $\alpha_{n-1} \cdot u + u'$. We can open up the terms using $b = r + x_n$ and $s_1 = \prod_{i=0}^{n-1} x_i + \alpha_{n-1} \cdot u + u'$. Then, two cases can happen: (1) $r = 0$ or (2) $r = 1$.

Suppose that $r = 0$; then P_n got $(0, 0, R_{n-1})$ from P_0 and P_{n-3}. By construction, we have

$$s_b + R_{n-1} = bs_1 + R_{n-1} = x_n \cdot (\prod_{i=0}^{n-1} x_i + \alpha_{n-1}u + u') + R_{n-1} = x_n \cdot (\alpha_{n-1}u + u') + R_{n-1},$$

where we use that $\prod_{i=0}^{n} x_i = 0$ by assumption. On the other hand, if $r = 1$, then P_n got $(1, m_0, 0)$ from P_0 and P_{n-3}, and by construction, we have

$$s_{1-b} + m_0 = (1 + b)s_1 + m_0 = x_n \cdot (\prod_{i=0}^{n-1} x_i + \alpha_{n-1}u + u') + m_0 = x_n \cdot (\alpha_{n-1}u + u') + m_0,$$

using again the assumption. Now, to simulate, Sim will sample five random bits $(u, u', r, m_0, R_{n-1}) \overset{\$}{\leftarrow} \{0,1\}^5$ and send (u, u') on behalf of P_2 and $(r, r \cdot m_0, R_{n-1} \cdot (1 + r))$ on behalf of P_0 and P_{n-3}. It remains to simulate the two OT messages. If $r = 0$, then Sim will construct the $s_b + R_{n-1}$ term as $x_n \cdot (\alpha_{n-1}u + u') + R_{n-1}$, which is distributed exactly as in the protocol (recall that Sim knows the input x_n of the corrupted party). The remaining term, $s_{1-b} + m_0$, is simulated by sampling a uniformly random independent bit, which is a perfect simulation since P_n never received any information about m_0 (since it received $(0, 0, R_{n-1})$). If $r = 1$, Sim will construct the $s_b + m_0$ term as $x_n \cdot (\alpha_{n-1}u + u') + m_0$, which is distributed exactly as in the protocol, and simulate the $s_{1-b} + R_{n-1}$ term by a uniform random bit, which is a perfect simulation since P_{n-1} never received any information about the random masking bit R_{n-1}.

Case 3: $i \in \{0, 1\}$. It remains to deal with the first two parties, which is straightforward: in the real protocol P_1 receives four bits from P_0, two of which are truly random, and two of which are $(x_0, \alpha_0 x_0)$ masked by two more independent random bits. Sim can perfectly simulate this message by sending four random bits on behalf of P_0. P_0 never receives any message throughout the protocol (except the value of the function), hence privacy against P_0 holds trivially. This concludes the proof.

References

AL17. G. Asharov and Y. Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, January 2017.

BDSPV99. C. Blundo, A. De Santis, G. Persiano, and U. Vaccaro. Randomness complexity of private computation. *computational complexity*, 8(2):145–168, 1999.

Bea92. D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO'91*, *LNCS* 576, pages 420–432. Springer, Heidelberg, August 1992.

Bea95. D. Beaver. Precomputing oblivious transfer. In *CRYPTO'95*, *LNCS* 963, pages 97–109. Springer, Heidelberg, August 1995.

BGW88. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

CCD88. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

Cha90. D. Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO'89*, *LNCS* 435, pages 591–602. Springer, Heidelberg, August 1990.

CK91. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discret. Math.*, 4(1):36–47, 1991.

DPP16. D. Data, V. M. Prabhakaran, and M. M. Prabhakaran. Communication and randomness lower bounds for secure computation. *IEEE Transactions on Information Theory*, 62(7):3901–3929, 2016.

GMW87a. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

GMW87b. O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO'86*, *LNCS* 263, pages 171–185. Springer, Heidelberg, August 1987.

Gol09. O. Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

GR03. A. Gál and A. Rosén. Lower bounds on the amount of randomness in private computation. In *35th ACM STOC*, pages 659–666. ACM Press, June 2003.

JLR03. A. Jakoby, M. Liśkiewicz, and R. Reischuk. Private computations in networks: Topology versus randomness. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 121–132. Springer, 2003.

KLR06. E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. In *38th ACM STOC*, pages 109–118. ACM Press, May 2006.

KM96. E. Kushilevitz and Y. Mansour. Randomness in private computations. In *15th ACM PODC*, pages 181–190. ACM, August 1996.

KOP$^+$19. E. Kushilevitz, R. Ostrovsky, E. Prouff, A. Rosén, A. Thillard, and D. Vergnaud. Lower and upper bounds on the randomness complexity of private computations of AND. In *TCC 2019, Part II*, *LNCS* 11892, pages 386–406. Springer, Heidelberg, December 2019.

KOP$^+$21. E. Kushilevitz, R. Ostrovsky, E. Prouff, A. Rosén, A. Thillard, and D. Vergnaud. Lower and upper bounds on the randomness complexity of private computations of AND. *SIAM J. Discret. Math.*, 35(1):465–484, 2021.

KOR98. E. Kushilevitz, R. Ostrovsky, and A. Rosén. Amortizing randomness in private multiparty computations. In *17th ACM PODC*, pages 81–90. ACM, June / July 1998.

KOR99. E. Kushilevitz, R. Ostrovsky, and A. Rosén. Characterizing linear size circuits in terms of privacy. *Journal of Computer and System Sciences*, 58(1):129–136, 1999.

KR94. E. Kushilevitz and A. Rosén. A randomnesss-rounds tradeoff in private computation. In *CRYPTO'94*, *LNCS* 839, pages 397–410. Springer, Heidelberg, August 1994.

LHA$^+$12. A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, whit is right. *IACR Cryptology ePrint Archive*, 2012:64, 2012.

NN93. J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM journal on computing*, 22(4):838–856, 1993.

NSS$^+$. M. Nemec, M. Sýs, P. Svenda, D. Klinec, and V. Matyas. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1631–1648.

NTS99. N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1):148–173, 1999.

RU19. A. Rosén and F. Urrutia. A new approach to multi-party peer-to-peer communication complexity. In *ITCS 2019*, pages 64:1–64:19. LIPIcs, January 2019.

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.