Designing Fully Secure Protocols for Secure Two-Party Computation of Constant-Domain Functions

Vanesa Daza[1] and Nikolaos Makriyannis[2(✉)]

[1] Universitat Pompeu Fabra, Barcelona, Spain
vanesa.daza@upf.edu
[2] Tel Aviv University, Tel Aviv, Israel
n.makriyannis@gmail.com

Abstract. In a sense, a two-party protocol achieves fairness if the output from the computation is obtained simultaneously by both parties. A seminal result by Cleve (STOC 1986) states that fairness is impossible, in general. Surprisingly, Gordon et al. (JACM 2011) showed that there exist interesting functions that are computable with fairness. The two results give rise to a distinction between *fair* functions and *unfair* ones. The question of characterizing these functions has been studied in a sequence of works leading to the complete characterization of (symmetric) Boolean functions by Asharov et al. (TCC 2015). In this paper, we design new fully secure protocols for functions that were previously unknown to be fair. To this end, our main technical contribution is a generic construction of a fully secure (fair) protocol, starting with a constant-round protocol satisfying limited security requirements. Our construction introduces new conceptual tools for the analysis of fairness that apply to arbitrary (constant-domain) functions. While the characterization remains open, we believe that our results lay the foundation for a deeper understanding of fairness.

Keywords: Fairness · Secure two-party computation · Malicious adversaries · Cryptographic protocols

1 Introduction

A popular definition of two-party computation is that it enables two mutually distrusting parties to compute a joint function of their inputs while only revealing what the output suggests. However, the popular definition does not capture all the security requirements one may expect from such a computation. Among these requirements is *fairness*, which states that either both parties receive output or none of them do. It is a natural security requirement for many real-world tasks.

V. Daza—Research supported by Project TEC2015-66228-P (MINECO/FEDER, UE).

N. Makriyannis—Research supported by ERC starting grant 638121.

Y. Kalai and L. Reyzin (Eds.): TCC 2017, Part I, LNCS 10677, pp. 581–611, 2017.
https://doi.org/10.1007/978-3-319-70500-2_20

For example, when two parties are signing a contract, the contents of which may be legally binding, it is imperative that one party signs the contract if and only if the second party signs as well.

The study of two-party computation started with the work of Yao [14] in 1982. Secure computation was expanded to the multiparty case by Goldreich, Micali, and Wigderson [10] in 1987. Flagship results from the theory of secure computation state that, when an absolute majority of honest parties can be guaranteed, every task can be realized with full security, i.e. the relevant protocols provide correctness, privacy, independence of inputs, as well as fairness. However, when the honest parties are in the minority, as it happens in the important two-party case, classic protocols satisfy a weaker notion of security known as security-with-abort, which captures all the aforementioned security requirements, except for fairness. This relaxation is often attributed to an inherent limitation that was shown by Cleve [7].

Cleve showed that fairness is impossible to achieve in general when one of the parties behaves dishonestly. Specifically, Cleve proved that the coin-tossing functionality, i.e the inputless functionality that returns the same uniform random bit to the parties, is not computable with fairness. His proof exploits the fact that interactive computation involves exchanging messages back and forth, and thus at some point one party may break fairness by aborting prematurely. It goes without saying that any function that implies coin-tossing is not computable with fairness either, as is the case with the XOR function.

Amazingly, for more than two decades, Cleve's result led to the mistaken conclusion that interesting functions are not computable with fairness in the two-party setting, or the multi-party setting with dishonest majority. Only in 2008 was this interpretation proven false by Gordon, Hazay, Katz and Lindell [11], who showed that Cleve's impossibility does not apply to *all* non-trivial functions, and there are many interesting functions that are inherently *fair*. The remarkable work of Gordon et al. begins by making a distinction between XOR-embedded[1] and non XOR-embedded functions. Functions of the latter type, which includes OR and the greater-than function, are shown to be fair. Yet XOR-embedded functions are not necessarily excluded from fully secure computation. Gordon et al. propose a specific protocol, referred to as GHKL throughout the present paper, that computes many XOR-embedded functions with full security. The authors also show that fair computation of XOR-embedded functions requires super-logarithmic round complexity.

In this paper, we focus on the fundamental question raised by Gordon, Hazay, Katz and Lindell; the characterization of functions with respect to fairness. In particular, we propose a methodology for designing fully secure protocols.

1.1 Previous Works

The problem of characterizing fairness is equivalent to identifying a necessary and sufficient condition for a given two-party function to be fair. There are thus

[1] A function is XOR-embedded if restricting the function to a subset of inputs yields the XOR function.

two complementary ways to engage with the problem. The first one attempts to identify necessary conditions for fairness by means of impossibility results [1,4,7,13]. The second one attempts to identify sufficient conditions by means of feasibility results, i.e. by proving fairness for explicit protocols [2,3,11,13]. We mention that most of these works focus on fair computation of Boolean functions that are symmetric – the function returns the same output to both parties, deterministic – the output is fully determined by the inputs, and constant-domain – the function is independent of the security parameter. By abusing terminology, we refer to such functions simply as Boolean functions.

Necessary conditions can be traced back to Cleve's seminal work [7]. In [1], Agrawal and Prabhakaran generalized the impossibility of coin-tossing to non-trivial sampling functionalities, that is, inputless functionalities that return statistically correlated outputs are not computable with fairness. Asharov, Lindell, and Rabin [4] investigated the problem of characterizing Boolean functions that imply coin-tossing, and are thus inherently unfair. They showed that certain functions, dubbed *balanced*, can be used to toss a uniform random coin. Conversely, they found that coin-tossing is not reducible to any balanced function in the information theoretic-sense. Boolean functions that imply non-trivial sampling where identified by Makriyannis [13], who expanded the class of Boolean functions that are known to be unfair.

Regarding sufficient criteria, Gordon, Hazay, Katz and Lindell laid the foundation with [11], and all subsequent papers [2,3,13] on the topic are based on the GHKL protocol. By digging deep into the security analysis of the GHKL protocol, Asharov [2] deduced sufficient conditions for the protocol to compute functions with full security. Furthermore, the author showed that almost all Boolean functions with unequal-size domains satisfy these conditions, and thus a surprisingly large amount of functions are fair. Sufficient conditions for GHKL were also deduced independently by Makriyannis in [13].

Recently, Asharov, Beimel, Makriyannis and Omri [3] showed that a counter-intuitive modification of GHKL allows for the complete characterization of all Boolean functions. The characterization states that a Boolean function is computable with full security if and only if the all-one vector or the all-zero vector belong to the affine span of either the rows or the columns of the matrix describing the function. Remarkably, the characterization extends to randomized Boolean functions as well as multiparty Boolean functions when exactly half of the parties are corrupted.

Finally, we mention that Gordon and Katz [12] constructed a fully secure three-party protocol for the majority function and a n-party protocol for the AND of n bits.

Limits of the GHKL Approach. While significant progress has been made towards characterizing fairness in secure computation, we argue that the methods that appear in the literature have reached their limits in terms of usefulness. Specifically, regarding the design of fully secure protocols for arbitrary functions, the "standard" approach of generalizing and modifying GHKL to extract

sufficient conditions seems to offer few gains. Even for the limited case of Boolean functions that are not symmetric, straightforward generalizations of GHKL are either function-specific i.e. the resulting protocol is tailored to a specific function, or, the protocol computes a family of functions whose description is rather mysterious and artificial. Arguably, the present state of affairs calls for a systematic analysis of fair protocols.

1.2 Our Contributions

In this paper, we propose a framework for designing fully secure protocols. To this end, we introduce two new conceptual tools, refered to as *locking strategies* and *sampling attacks*, which are inspired by the impossibility results of [1,4,7,13]. Our investigation naturally leads to a new security notion that we call *security against sampling attacks*; a strictly weaker notion than fairness and therefore a necessary requirement for fair protocols. An appealing feature of the proposed security notion is that it bypasses lower-bounds on fairness. Specifically, as was shown by Gordon et al. [11], fair functions may require computation in super-logarithmic round-complexity. In contrast, security against sampling attacks seems to be achievable in a constant number of rounds for the same functions. What's more, security against sampling attacks can be efficiently tested via a collection of linear algebraic properties. The appeal of our approach is further strengthened by our main result, stated next.

We propose a generic construction that transforms any protocol that is – constant-round – passively secure – secure against sampling attacks, into a fully-secure protocol. In the spirit of GHKL, this is achieved by introducing a special threshold round i^*. Our main result may be viewed as a framework for designing fair protocols, and we believe that it demystifies the "standard" approach that appears in the literature. What's more, it applies to any constant-domain two-party function (i.e. randomized, asymmetric and non-Boolean). Our main result is stated informally below.

Theorem 1.1 (informal). *A two-party function is fair if and only if it admits a suitable protocol that is secure against sampling attacks.*

Our techniques show the existence of a fair non-Boolean function where both parties have the same number of inputs. We stress that previous results [2] on non-Boolean functions only applied to functions where one party has at least twice as many inputs as the other.

Theorem 1.2 (informal). *The non-Boolean function described by the matrix below is computable with full security.*

$f(x,y)$	y_1	y_2	y_3	y_4
x_1	1	1	2	2
x_2	1	0	1	2
x_3	1	1	0	2
x_4	2	2	0	2

Next, we propose an algorithm for designing suitable protocols (constant-round, passively secure, secure against sampling attacks). Our algorithm takes an asymmetric Boolean function as input, and it either returns an appropriate protocol, or it returns that it failed to do so. The algorithm is accompanied with a proof of correctness. In Sect. 5.3, we show how our algorithm handles the asymmetric function that was suggested as an open problem in [3], and we prove that it is fair.

Theorem 1.3 (informal). *The function from [3] described by the matrices below is computable with full security.*

$f_1(x,y)$	y_1	y_2	y_3	y_4
x_1	0	1	1	0
x_2	1	0	1	1
x_3	1	0	0	0
x_4	0	1	0	1

$f_2(x,y)$	y_1	y_2	y_3	y_4
x_1	1	1	1	0
x_2	1	0	1	1
x_3	0	1	0	1
x_4	1	1	0	0

Unfortunately, our methods do not settle the characterization of constant-domain two-party functions, even for the asymmetric Boolean case. That being said, we believe that the questions that are left unanswered may be as interesting as the results themselves. Specifically, for a function that lies in the gap, we show that it is computable with fairness as long as privacy is relaxed.

Theorem 1.4 (informal). *The function described by the matrices below admits a protocol that is fair-but-not-private.*

$f_1(x,y)$	y_1	y_2	y_3	y_4	y_5
x_1	1	1	1	1	0
x_2	0	1	0	1	1
x_3	1	1	1	1	1
x_4	0	0	1	0	1
x_5	1	0	0	0	1

$f_2(x,y)$	y_1	y_2	y_3	y_4	y_5
x_1	1	1	0	0	0
x_2	1	0	0	0	1
x_3	1	0	0	1	0
x_4	0	0	1	1	1
x_5	0	1	0	1	0

We emphasize that the function in question may still be computable with full security. However, we believe that our present analysis together with the theorem above strongly indicate that there is an inherent trade-off between fairness and privacy. To the best of our knowledge, the literature does not entertain the idea that fairness and privacy may be attainable only at the expense of one another; the two notions might as well be incomparable.

Organization of the Paper. After recalling some preliminaries in Sect. 2, we introduce locking strategies and sampling attacks in Sect. 3. Section 4 is dedicated to our main result and its proof. In Sect. 5, we show how to obtain suitable protocols by means of the algorithm mentioned above. Finally, open problems and future directions are discussed in Sect. 6.

For clarity, and to alleviate notation, we have decided to restrict our analysis to the family of asymmetric (possibly randomized) Boolean functions.

We emphasize that, with the exception of Sect. 5, our results generalize to arbitrary non-Boolean functions. While the generalization is not straightforward, it is beyound the scope of the present abstract. We refer to the full version [8] for the general case.

2 Preliminaries

Throughout this paper, n denotes the security parameter and $\mathbb{N}$ denotes the set of positive integers. All vectors are column vectors over the real field $\mathbb{R}$. Vectors are denoted using bold letters, e.g. $\mathbf{v}$, $\mathbf{1}$ (the all-one vector). The i-th entry of some vector $\mathbf{v}$ is denoted $\mathbf{v}(i)$. If $\mathbf{v}_1, \ldots, \mathbf{v}_s$ denotes a family of vectors, then $\langle \mathbf{v}_1, \ldots, \mathbf{v}_s \rangle$ denotes the vector space generated by those vectors, and let $\langle \mathbf{v}_i \,|\, \mathbf{v}_j \rangle = \mathbf{v}_i^T \mathbf{v}_j$. Matrices are denoted with capital letters, e.g. M, P. The i-th row and j-th column of some matrix M are denoted $[M]_{i,*}$ and $[M]_{*,j}$, respectively. Furthermore, the element indexed by (i, j) in M is denoted $M(i, j)$.

Definition 2.1. Let A and B be arbitrary matrices. We write $C = A * B$ if C is equal to the entry-wise (Hadamard) product of the two matrices, i.e. $C(i, j) = A(i, j) \cdot B(i, j)$.

Finally, if $\mathcal{X}$ and $\mathcal{Y}$ denote distribution ensembles, we write $\mathcal{X} = \mathcal{Y}$, $\mathcal{X} \overset{s}{\equiv} \mathcal{Y}$ and $\mathcal{X} \overset{c}{\equiv} \mathcal{Y}$, respectively, if the ensembles are perfectly, statistically or computationally indistinguishable.

2.1 Secure Two-Party Computation

Let P_1 and P_2 denote the parties. A *two-party function* $f = (f_1, f_2)$ is a random process that maps pair of inputs (one for each party), to pairs of random variables called outputs (again, one for each party). The domain of f is denoted $X \times Y$. For our purposes, we assume that $X = \{1, \ldots, \ell\}$, $Y = \{1, \ldots, k\}$ and the parties' outputs are sampled from $\{0, 1\}^2$. To every function f, we associate four matrices $\{M^{(a,b)}\}_{a,b \in \{0,1\}}$ such that

$$M^{(a,b)}(x, y) = \Pr\left[f(x, y) = (a, b)\right].$$

In addition, define $M^{(1,*)}$ and $M^{(*,1)}$, associated with f_1 and f_2 respectively, such that $M^{(1,*)}(x, y) = \Pr\left[f_1(x, y) = 1\right]$ and $M^{(*,1)}(x, y) = \Pr\left[f_2(x, y) = 1\right]$. A two-party protocol Π for computing f is a polynomial-time protocol such that, on global input 1^n (the security parameter) and private inputs $x \in X$, $y \in Y$, the joint distribution of the outputs $\{\Pi(1^n, x, y)\}_n$ is statistically close $(f_1, f_2)(x, y)$, assuming both parties behave honestly. The parties run in polynomial-time in n.

The Adversary. We introduce an adversary $\mathcal{A}$ given auxiliary input $z \in \{0, 1\}^*$ corrupting one of the parties. We assume the adversary is computationally bounded and malicious, i.e. the adversary runs in polynomial-time in n and she may instruct the corrupted party to deviate from the protocol arbitrarily.

The Fully-Secure Model

- **Inputs:** P_1 is holding 1^n and $x \in X$, P_2 is holding 1^n and $y \in Y$. The adversary is given auxiliary input $z \in \{0,1\}^*$. The trusted has no input.
- **Parties send inputs:** The honest party sends his input to the trusted party. The corrupted party sends a value of the adversary's choice. Write (x', y') for the pair of values received by the trusted party.
- **Trusted party performs computation:** If either x' or y' is not in the appropriate domain, the trusted party reassigns the aberrant input to some default value. Write (x', y') for the pair of inputs after (possible) reassignment. The trusted party chooses a random string $r \in \{0,1\}^*$ and computes $f(x, y; r)$.
- **Parties receive outputs:** P_1 receives $f_1(x, y; r)$, P_2 receives $f_2(x, y; r)$.
- **Outputs:** The honest party outputs whatever he received for the trusted party, the corrupted party outputs nothing, the adversary outputs a probabilistic polynomial-time function of its view.

Fig. 1. The ideal model with full-security for computing f.

Write $(\mathsf{out}, \mathsf{view})^{\mathsf{Real}}_{\mathcal{A}(z),\Pi}$ for the pair consisting of the honest party's output and the adversary's view in an execution of protocol Π. Next, we define security in terms of the ideal model.

Let $\mathcal{S}$ denote the ideal-world adversary. Write $(\mathsf{out}, \mathsf{view})^{\mathsf{Ideal}}_{\mathcal{S}(z),f}$ for the pair consisting of the honest party's output and the adversary's view in the ideal model (Fig. 1).

Definition 2.2. Let Π be a protocol for computing f. We say that Π is fully secure if for every non-uniform polynomial time adversary $\mathcal{A}$ in the real model, there exists a non-uniform polynomial time adversary $\mathcal{S}$ in the ideal model such that

$$\left\{(\mathsf{out}, \mathsf{view})^{\mathsf{Real}}_{\mathcal{A}(z),\Pi}(1^n, x, y)\right\}_{n\in\mathbb{N},(x,y)\in X\times Y,z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{(\mathsf{out}, \mathsf{view})^{\mathsf{Ideal}}_{\mathcal{S}(z),f}(1^n, x, y)\right\}_{n\in\mathbb{N},(x,y)\in X\times Y,z\in\{0,1\}^*}.$$

It is important to note that the only way for the ideal-world adversary to affect the honest party's output is through the choice of input. Finally, we remark that the fully-secure model is the standard model for the honest-majority multi-party setting.

The Hybrid Model. The hybrid model with ideal access to $\mathcal{F}$ is a communication model where the parties have access to a trusted computing some functionality $\mathcal{F}$ with full security. In this model, the parties communicate as in the plain model and they are allowed to make a single call to the trusted party for computing $\mathcal{F}$. Protocols and security for this communication model are defined along the same lines as above. By [6], as long as $\mathcal{F}$ admits a secure real-world

protocol, the existence of a secure hybrid protocol for f implies the existence of a secure protocol for f in the real model. By contraposition, if f cannot be realized securely, then the existence of a secure protocol for f in the hybrid model implies the impossibility of realizing $\mathcal{F}$ securely in the real model.

The Dealer Model. Throughout the paper, we define protocols by describing the number of rounds $r(n)$ and the *backup outputs* $\{a_i\}_{i=0}^{r}$ for P_1 and $\{b_i\}_{i=0}^{r}$ for P_2. When executing a protocol, the parties hand their inputs to an entity called the *dealer*. In turn, the dealer performs all the computations and hands the relevant backup outputs to P_1 and then P_2 in a sequence of $r(n)$ iterations. Either party may abort the execution at any time and the protocol terminates at that point. The remaining party is instructed to output the last backup output he received. This approach is known as the *online dealer model*, and it does not incur any loss of generality as there is a standard transformation from the online dealer model to the plain model [2,3,5]. The online dealer model is convenient in that it provides clarity to our presentation and it greatly simplifies the security analysis.

3 Locking Strategies and Sampling Attacks

In this section, we introduce the notions of locking strategies and sampling attacks. To motivate our discussion, we use specific functions from the literature as illustrative examples. Namely, the XOR function encoded by matrices

$$M^{(1,*)} = M^{(*,1)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

the function f^{nm} from [13] encoded by matrices

$$M^{(1,*)} = M^{(*,1)} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix},$$

the function f^{sp} from [3] encoded by matrices

$$M^{(1,*)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \quad M^{(*,1)} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

We remark that since the functions above are deterministic, the corresponding matrices fully describe these functions. In addition, we note that f^{sp} is computable with fairness [3], while XOR and f^{nm} are not [7,13]. Next, we briefly discuss why that is the case.

3.1 Warm-Up

It is not hard to see that a fully-secure realization of XOR yields a fully-secure coin-toss. Indeed, by instructing the parties to choose their inputs uniformly at random, the output from a fully-secure computation of XOR is uniformly distributed, even in the presence of malicious adversaries. A slightly more involved procedure allows the parties to sample correlated bit, using a fully-secure protocol for f^{nm}. Indeed, instruct P_1 to choose his input among $\{x_1, x_3, x_4\}$ with uniform probability, instruct P_2 to choose y_4 with probability $2/5$ or one of his other inputs with probability $1/5$. Let c denote the the output from the computation of f^{nm}. Party P_1 outputs c, party P_2 outputs $1 - c$ if he chose y_2 and c otherwise.

For us, it is important to note that the procedures described above are encoded by certain vectors. For XOR, these vectors are $(1/2, 1/2)$ for P_1 and $(1/2, 1/2)$ for P_2. For f^{nm}, they are $(1/3, 0, 1/3, 1/3)$ for P_1 and $(1/5, -1/5, 1/5, 2/5)$ for P_2. To elaborate further, each vector instructs the relevant party how to choose its input (by taking the absolute value) and whether to flip the output from the computation of the function (negative values indicate that the party must flip the output). Observe that

$$(1/2, 1/2) \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \in \langle \mathbf{1}_2^T \rangle, \qquad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} \in \langle \mathbf{1}_2 \rangle,$$

and

$$(1/3, 0, 1/3, 1/3) \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \in \langle \mathbf{1}_4^T \rangle, \qquad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1/5 \\ -1/5 \\ 1/5 \\ 2/5 \end{pmatrix} \in \langle \mathbf{1}_4 \rangle .$$

The relations above capture the fact that the procedure encoded by the vector yields an output whose distribution is independent of the opponent's input, i.e. P_i's output resulting from the procedure is independent of P_{3-i}'s choice of input, assuming the underlying function is computed with full security. It is straightforward to check that the parties' outputs exhibit statistical correlation, and thus the functions in question are not computable with full-security, by [1,7].

On the other hand, it is interesting to note that similar vectors and procedures can be defined for function f^{sp}. Specifically, observe that

$$(1/2, 1/2, 0, 0) \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \in \langle \mathbf{1}_4^T \rangle , \qquad \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \\ 0 \end{pmatrix} \in \langle \mathbf{1}_4 \rangle .$$

In more detail, by choosing one of their first two inputs uniformly at random, the outputs from a fully-secure computation of f^{sp} are uniformly random, even in the presence of malicious adversaries. However, contrary to the previous cases, the parties' outputs are independent as random variables.

3.2 Locking Strategies

For an arbitrary function f, let $\mathcal{L}_2$ denote a basis of the vector space consisting of all vectors $\mathbf{y}$ such that $M^{(*,1)} \cdot \mathbf{y} \in \langle \mathbf{1}_\ell \rangle$. Similarly, let $\mathcal{L}_1$ denote a basis of the vector space consisting of all vectors $\mathbf{x}$ such that $M^{(1,*)T} \cdot \mathbf{x} \in \langle \mathbf{1}_k \rangle$.

Definition 3.1. Elements of $\langle \mathcal{L}_1 \rangle$ and $\langle \mathcal{L}_2 \rangle$ are referred to as *locking strategies* for P_1 and P_2, respectively.

As discussed above, a locking strategy (after normalization) encodes a distribution over the inputs and a local transformation that depends on the chosen input. Since $M^{(*,1)} \cdot \mathbf{y} \in \langle \mathbf{1}_\ell \rangle$ and $M^{(1,*)T} \cdot \mathbf{x} \in \langle \mathbf{1}_k \rangle$, it follows that the parties' outputs resulting from the locking strategies are independent of each others' inputs, assuming ideal access to f. In loose terms, a party applying some locking strategy "locks" the distribution of its output.

For us, it is important to note that fully-secure protocols "preserve" locking strategies, even in the presence of malicious adversaries. Specifically, the distribution of the honest party's output resulting from some locking strategy is independent of the adversary's course of action (e.g. premature abort). We elaborate on this point next.

3.3 Sampling Attacks

Consider the following single-round protocol for $f^{\mathsf{sp}} = (f_1, f_2)$ defined by means of the backup outputs $\{a_i, b_i\}_{i=0,1}$:

$$\begin{array}{ll} a_0 = f_1(x, \widetilde{y}) \text{ where } \widetilde{y} \in_U Y & b_0 = f_2(\widetilde{x}, y) \text{ where } \widetilde{x} \in_U X \\ a_1 = f_1(x, y) & b_1 = f_2(x, y) \end{array}$$

Suppose that party P_2 applies locking strategy $\mathbf{y} = (1/2, 1/2, 0, 0)^T$. Notice that in an honest execution of Π, party P_2 outputs a uniform random bit. Now, suppose that an adversary corrupting P_1 uses x_3 for the computation, and aborts the computation prematurely if $a_1 = 0$ (In that case P_2 outputs b_0). Deduce that the honest party outputs 1 with probability 3/4 and thus the protocol is not fully-secure.

On the other hand, consider the following two-round protocol Π^{sp} for f^{sp} defined by means of the backup outputs $\{a_i, b_i\}_{i=0\ldots2}$:

$$\begin{array}{lll} a_0 = f_1(x, \widetilde{y}) \text{ where } \widetilde{y} \in_U Y & & b_0 = f_2(\widetilde{x}, y) \text{ where } \widetilde{x} \in_U X \\ a_1 = \begin{cases} f_1(x, y) & \text{if } x \in \{x_1, x_2\} \\ f_1(x, \widetilde{y}') \text{ where } \widetilde{y}' \in_U Y & \text{if } x \in \{x_3, x_4\} \end{cases} & & b_1 = f_2(x, y) \\ a_2 = f_1(x, y) & & b_2 = f_2(x, y) \end{array}$$

Already, we see that the attack described above will not work for this protocol. In fact, a straightforward analysis shows that it is impossible to alter the distribution of the honest party's output resulting from a locking strategy, both for P_1 and P_2. To see that, let $\widehat{b}_j$ (resp. $\widehat{a}_j$) denote the bit obtained from b_j

(resp. a_j) by applying some locking strategy, and observe that the random variables $\widehat{b}_{i-1}$ and $\widehat{b}_2$ (resp. $\widehat{a}_i$ and $\widehat{a}_2$) conditioned on the adversary's view at round i are identically distributed. For similar attacks on arbitrary protocols and functions, security is captured by the definition below.

Definition 3.2. Let Π be an arbitrary protocol defined by means of its backup outputs $\{a_i, b_i\}_{i\in\{0,\ldots,r\}}$. We say that Π is secure against sampling attacks if

- for every $i \leq r$, for every $x \in X$, for every $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$, it holds that the random sequences $(a_0, \ldots, a_i, \widehat{b}_{i-1})$ and $(a_0, \ldots, a_i, \widehat{b}_r)$ are statistically close.
- for every $i \leq r$, for every $y \in Y$, for every $\mathbf{x} \in \langle \mathcal{L}_1 \rangle$, it holds that the random sequences $(b_1, \ldots, b_i, \widehat{a}_i)$ and $(b_1, \ldots, b_i, \widehat{a}_r)$ are statistically close.

Remark 3.3. Rather awkwardly, we define security against sampling attacks without defining sampling attacks. For the purposes of the present abstract, sampling attacks are simply fail-stop attacks with the intent of altering the distribution of the honest party's output resulting from some locking strategy. Furthermore, we note that Definition 3.2 is information-theoretic. We remark that this is probably too strong. However, since the protocols we will consider are constant-round, it does not affect our analysis.

3.3.1 Sampling Attacks in Linear-Algebraic Terms

In this section, we show how security against sampling attacks can be expressed in linear-algebraic terms. First, we define closeness for vectors. Let $\{\mathbf{v}_n\}_{n\in\mathbb{N}}$ and $\{\mathbf{u}_n\}_{n\in\mathbb{N}}$ denote two families of vectors indexed by $\mathbb{N}$. We say that $\mathbf{v}_n$ is *close* to $\mathbf{u}_n$ if $\|\mathbf{u}_n - \mathbf{v}_n\| \leq \mathsf{negl}(n)$. By abusing notation, we write $\mathbf{u}_n \overset{s}{\equiv} \mathbf{v}_n$ if the vectors are close.

Definition 3.4. For every $i \leq r$, for every $\vec{\alpha}_i = (\alpha_1, \ldots, \alpha_i) \in \{0,1\}^i$, and every $\beta \in \{0,1\}$, define matrices $B_-^{(\vec{\alpha}_i,\beta)}$, $B_+^{(\vec{\alpha}_i,\beta)} \in \mathbb{R}^{\ell\times k}$ such that

$$B_-^{(\vec{\alpha}_i,\beta)}(x,y) = \Pr\left[(\vec{a}_i, b_{i-1})(x,y) = (\vec{\alpha}_i, \beta)\right]$$
$$B_+^{(\vec{\alpha}_i,\beta)}(x,y) = \Pr\left[(\vec{a}_i, b_r)(x,y) = (\vec{\alpha}_i, \beta)\right].$$

Similarly, for every $\vec{\beta}_i = (\beta_1, \ldots, \beta_i) \in \{0,1\}^i$ and every $\alpha \in \{0,1\}$ define matrices $A_-^{(\alpha,\vec{\beta}_i)}$, $A_+^{(\alpha,\vec{\beta}_i)} \in \mathbb{R}^{\ell\times k}$ such that

$$A_-^{(\alpha,\vec{\beta}_i)}(x,y) = \Pr\left[(a_i, \vec{b}_i)(x,y) = (\alpha, \vec{\beta}_i)\right]$$
$$A_+^{(\alpha,\vec{\beta}_i)}(x,y) = \Pr\left[(a_r, \vec{b}_i)(x,y) = (\alpha, \vec{\beta}_i)\right].$$

Proposition 3.5. *Protocol Π is secure against sampling attacks if and only if*

– *for every* $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$*, for every* $i \leq r$*, for every* $\vec{\alpha}_i \in \{0,1\}^i$*, the vector below is close to* $\mathbf{0}_\ell$*.*

$$\left(B_+^{(\vec{\alpha}_i,1)} - B_-^{(\vec{\alpha}_i,1)}\right) \cdot \mathbf{y}. \tag{1}$$

– *for every* $\mathbf{x} \in \langle \mathcal{L}_1 \rangle$*, for every* $i \leq r$*, for every* $\vec{\beta}_i \in \{0,1\}^i$*, the vector below is close to* $\mathbf{0}_k$*.*

$$\left(A_+^{(1,\vec{\beta}_i)T} - A_-^{(1,\vec{\beta}_i)T}\right) \cdot \mathbf{x}. \tag{2}$$

For example, for protocol Π^{sp}, the distributions of (a_1, b_0) and (a_1, b_2) is given by the following matrices.

$$B_-^{(0,1)} = \begin{pmatrix} 0 & 0 & 0 & 1/2 \\ 3/4 & 1/4 & 1/2 & 0 \\ 3/8 & 1/8 & 1/4 & 1/4 \\ 3/8 & 1/8 & 1/4 & 1/4 \end{pmatrix}, \qquad B_+^{(0,1)} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 1/2 \end{pmatrix}$$

$$B_-^{(1,1)} = \begin{pmatrix} 3/4 & 1/4 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 \\ 3/8 & 1/8 & 1/4 & 1/4 \\ 3/8 & 1/8 & 1/4 & 1/4 \end{pmatrix}, \qquad B_+^{(1,1)} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 1/2 \end{pmatrix}.$$

Similarly, the distributions of (a_1, b_1) and (a_2, b_1) is given by the following matrices.

$$A_-^{(0,1)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 1/2 \\ 1/2 & 0 & 0 & 0 \end{pmatrix}, \qquad A_+^{(0,1)} = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,0 \end{pmatrix}$$

$$A_-^{(1,1)} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 1/2 \end{pmatrix}, \qquad A_+^{(1,1)} = \begin{pmatrix} 1\,0\,1\,0 \\ 0\,0\,0\,1 \\ 1\,0\,0\,0 \\ 0\,1\,0\,1 \end{pmatrix}.$$

Notice that the matrices above satisfy Proposition 3.5.

4 Towards Full Security

In this section, we show that constant-round protocols that satisfy passive security *and* security against sampling attacks are easily transformed into fully secure protocols. The present section is dedicated to the construction and its security proof. Let Π be a protocol for computing f. We model the protocol in the usual way. The parties' backup outputs for Π will be denoted $(c_0, \ldots, c_{r'})$ and $(d_0, \ldots, d_{r'})$, respectively, where r' denotes the number of rounds.

Assumption on the round-complexity. We assume that r' is *constant* in the security parameter. This assumption is desirable for for the proof of our main theorem, and it is good enough for our purposes. Nevertheless, the question of determining the optimal round complexity for protocols that are passively secure and secure against sampling attacks may be of independent interest.

We assume that the protocol is passively secure. Therefore, there exist simulators, denoted $\{\mathcal{S}_i^{\mathsf{p}}\}_{i\in\{1,2\}}$, that can recreate the backup sequences in the ideal model. In addition, since the protocol is constant-round, it follows that the ideal sequences are statistically close to the real ones. Formally,

$$(c_0,\ldots,c_{r'},d_{r'})^{\mathsf{Real}} \stackrel{s}{\equiv} (c_0,\ldots,c_{r'},f_2)^{\mathsf{Ideal}}$$
$$(d_0,\ldots,d_{r'},c_{r'})^{\mathsf{Real}} \stackrel{s}{\equiv} (d_0,\ldots,d_{r'},f_1)^{\mathsf{Ideal}}.$$

Finally, we assume that Π is secure against sampling attacks. Theorem 3.5 applies to Π in a very straightforward way. Using the notation from the previous section,

- For every $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$, for every $i = 1,\ldots,r'$, for every $\vec{\alpha}_i \in \{0,1\}^{i+1}$, the vector below is close $\mathbf{0}_\ell$.

$$\left(B_+^{(\vec{\alpha}_i,1)} - B_-^{(\vec{\alpha}_i,1)}\right)\cdot \mathbf{y}. \tag{3}$$

- For every $\mathbf{x} \in \langle \mathcal{L}_1 \rangle$, for every $i = 0,\ldots,r'-1$, for every $\vec{\beta}_i \in \{0,1\}^{i+1}$, the vector below is close to $\mathbf{0}_k$.

$$\left(A_+^{(1,\vec{\beta}_i)^T} - A_-^{(1,\vec{\beta}_i)^T}\right)\cdot \mathbf{x}. \tag{4}$$

4.1 Protocol SecSamp2Fair(Π)

We are going to *combine* the main ingredient of the GHKL protocol – the threshold round i^* – with the protocol above. Specifically, we are going to instruct the parties to run a protocol such that, at some point in the execution, unbeknownst to them, the parties begin running Π.

This is achieved by choosing a random threshold round according to a geometric distribution. Prior to that round, the parties exchange backup outputs that are independent of each other, and, once the threshold round has been reached, the parties exchange backups according to the specifications of Π. Formally, consider protocol SecSamp2Fair(Π) from Fig. 2. For the new protocol, $i^* \geq r'+1$ is chosen according to a geometric distribution with parameter γ. If $i < i^* - r'$, then a_i and b_i are independent of one another. If $i^* - r' \leq i < i^*$, then a_i and b_i are equal to $c_{i-i^*+r'}$ and $d_{i-i^*+r'}$, respectively. Finally, if $i \geq i^*$, then $(a_i,b_i) = (c_{r'},d_{r'}) \stackrel{s}{\equiv} (f_1,f_2)$.

Protocol SecSamp2Fair(Π)

1. The parties P_1 and P_2 hand their inputs, denoted x and y respectively, to the dealer.[a]
2. The dealer executes Π locally on inputs x and y and security parameter n. Write $(c_0, \ldots, c_{r'})$ and $(d_0, \ldots, d_{r'})$ for the sequences of backup outputs computed by the dealer.
3. The dealer chooses $i^* \geq r' + 1$ according to the geometric distribution with parameter γ.
4. The dealer computes $(\mathsf{out}_1, \mathsf{out}_2) = (c_{r'}, d_{r'})$, and, for $0 \leq i \leq r$,

$$a_i = \begin{cases} f_1(x, \widetilde{y}^{(i)}) \text{ where } \widetilde{y}^{(i)} \in_U Y & \text{if } i < i^* - r \\ c_{i-(i^*-r')} & \text{if } i^* - r' \leq i < i^* \\ \mathsf{out}_1 & \text{otherwise} \end{cases}$$

and

$$b_i = \begin{cases} f_2(\widetilde{x}^{(i)}, y) \text{ where } \widetilde{x}^{(i)} \in_U X & \text{if } i < i^* - r \\ d_{i-(i^*-r')} & \text{if } i^* - r' \leq i < i^* \\ \mathsf{out}_2 & \text{otherwise.} \end{cases}$$

5. The dealer gives b_0 to P_2.
6. For $i = 1, \ldots, r$,
   (a) The dealer gives a_i to P_1. If P_1 aborts, then P_2 outputs b_{i-1} and halts.
   (b) The dealer gives b_i to P_2. If P_2 aborts, then P_1 outputs a_i and halts.

[a] If x is not in the appropriate domain or P_1 does not hand an input, then the dealer sends $f(\hat{x}, y)$ (where $\hat{x}$ is a default value) to P_2, which outputs this value and the protocol is terminated. The case of an inappropriate y is dealt analogously.

Fig. 2. Protocol SecSamp2Fair(Π) for computing f.

Theorem 4.1. *Suppose that protocol Π for f is constant-round, passively secure, and secure against sampling attacks. There exists $\gamma_0 \in [0, 1]$ such that protocol* SecSamp2Fair(Π) *is fully secure for f, for every $\gamma < \gamma_0$.*

As a corollary, we show the existence of fair non-Boolean function where both parties have roughly the same number of inputs. We stress that previous results [2] on non-Boolean functions only applied to functions where one party has at least twice as many inputs as the other.

Corollary 4.2. *The non-Boolean function described by the matrix below is computable with full security.*

$f(x,y)$	y_1	y_2	y_3	y_4
x_1	1	1	2	2
x_2	1	0	1	2
x_3	1	1	0	2
x_4	2	2	0	2

Proof. Consider the following 2-round protocol defined by means of the backup outputs $\{a_i, b_i\}_{i=1\ldots 2}$.

$$a_0 = f(x, \widetilde{y}) \text{ where } \widetilde{y} \in_U Y \qquad b_0 = 2$$

$$a_1 = \begin{cases} a \in_U \{0,1\} & \text{if } x = x_2 \text{ and } f(x,y) \neq 2 \\ f(x,y) & \text{otherwise} \end{cases} \qquad b_1 = f(x,y).$$

$$a_2 = f(x,y) \qquad b_2 = f(x,y)$$

The protocol is constant-round, passively secure and secure against sampling attacks. By Theorem 4.1, the function is fair.

4.2 Security Analysis

We only deal with the case where P_1 is corrupted. The other case is virtually analogous. Write $\mathcal{A}$ for the adversary corrupting P_1. We begin with a high-level description of the simulator. The simulator $\mathcal{S}$ chooses i^* according to the specifications of the protocol, and simulates the rounds of the protocol as follows. Prior to iteration/round $i^* - r'$, the simulator generates backup outputs in exactly the same way as the dealer does in the real model. If the adversary decides to abort, $\mathcal{S}$ sends $x_0 \in X$ to the trusted party, where x_0 is sampled according to probability vector $\mathbf{z}_x^{(\vec{\alpha}_{r'})} \in \mathbb{R}^\ell$. As the notation suggests, $\mathbf{z}_x^{(\vec{\alpha}_{r'})}$ depends on x (the input handed by the adversary for the computation) and the last $r'+1$ backup outputs computed by the simulator. At iteration $i^* - r'$, assuming the adversary is still active, the simulator hands x to the trusted party, and receives output $a = f_1(x,y)$. In order to reconstruct the next values of the backup sequence, the simulator invokes $\mathcal{S}_2^{\mathsf{p}}$, and hands one-by-one to $\mathcal{A}$ the values computed by $\mathcal{S}_2^{\mathsf{p}}$. At every iteration following i^*, the simulator hands a to $\mathcal{A}$. At any given point, if the adversary aborts, the simulator outputs the sequence of values he handed to $\mathcal{A}$, and halts.

Intuition. By definition, the simulator's output together with the honest party's output in the ideal model is required to be indistinguishable from the adversary's view and the honest party's output in the real model. In our case, the adversary's view corresponds to the sequence of backup outputs she observes. Notice that the backup up sequences of each world are statistically close, which follows from the way i^* is chosen in both worlds, the passive security of Π, and the fact that prior to $i^* - r'$ the backup outputs in the real and ideal world are identically distributed. The hard part is to argue that there exists $\mathbf{z}_x^{(\vec{\alpha}_{r'})}$ from which the simulator can sample from. As we shall see, the existence of $\mathbf{z}_x^{(\vec{\alpha}_{r'})}$ follows from a corollary of the fundamental theorem of Linear Algebra, which comes into play because of the security against sampling attacks assumption (Fig. 3).

The simulator $\mathcal{S}$ for Protocol SecSamp2Fair(Π)

- The adversary $\mathcal{A}$ gives its input x to the simulator.[a]
- The simulator chooses $i^* \geq r' + 1$ according to the geometric distribution with parameter γ.
- For $i = 1, \ldots, i^* - r' - 1$:
  - The simulator gives $a_i = f(x, \widetilde{y}^{(i)})$ to the adversary $\mathcal{A}$, where $\widetilde{y}^{(i)}$ is chosen according to the uniform distribution.
  - If $\mathcal{A}$ aborts, then the simulator chooses an input x_0 according to a distribution $\mathbf{z}_x^{(a_{i-\rho}, \ldots, a_i)}$ (which depends on the input x and the last sequence $\rho + 1$ values that the simulator generated, where $\rho = \min(r', i)$), gives x_0 to the trusted party, outputs the bits $a_1, \ldots, a_i$, and halts.
- At round $i = i^* - r'$, the simulator gives x to the trusted party and gets the output $a = f_1(x, y)$.
  - The simulator constructs $(a_{i^*-r'}, \ldots, a_{i^*-1})$ such that $a_{i^*-r'+j} = \widehat{a}_j$ by invoking $\mathcal{S}_2^{\mathsf{p}}$ on input x, output $a = f_1(x, y)$ and security parameter n.
- For $i = i^* - r', \ldots, i^* - 1$: The simulator gives a_i to the adversary $\mathcal{A}$, if $\mathcal{A}$ aborts, then the simulator outputs the bits $a_1, \ldots, a_i$ and halts.
- For $i = i^*, \ldots, r$: The simulator gives $a_i = a$ to the adversary $\mathcal{A}$, if $\mathcal{A}$ aborts, then the simulator outputs the bits $a_1, \ldots, a_i$ and halts.
- The simulator outputs the bits $a_1, \ldots, a_r$ and halts.

[a] If the adversary gives an inappropriate x (or no x), then the simulator sends some default $\hat{x} \in X$ to the trusted party, outputs the empty string, and halts.

Fig. 3. The simulator $\mathcal{S}$ for protocol SecSamp2Fair(Π)

Recall that for $i = 1 \ldots r'$ matrices $B_-^{(\alpha_0, \ldots, \alpha_i, \beta)}$ and $B_+^{(\alpha_0, \ldots, \alpha_i, \beta)}$ denote

$$B_-^{(\alpha_0 \ldots \alpha_i, \beta)}(x, y) = \Pr\left[(c_0, \ldots, c_i, d_{i-1})(x, y) = (\alpha_0, \ldots, \alpha_i, \beta)\right]$$
$$B_+^{(\alpha_0 \ldots \alpha_i, \beta)}(x, y) = \Pr\left[(c_0, \ldots, c_i, d_{r'})(x, y) = (\alpha_0, \ldots, \alpha_i, \beta)\right]$$

Now, define $p_x^{(\alpha)} = \Pr\left[f_1(x, \widetilde{y}) = \alpha \,|\, \widetilde{y} \in_U Y\right]$. To alleviate notation, we will omit the security parameter. As mentioned earlier, the corrupted party's backup sequences in the real and ideal world are statistically close. Therefore, if the adversary quits in the real world, then the adversary quits in the ideal world as well, with all but negligible probability – and vice versa. The whole point of the simulation is to show that early aborts do not breach security. In particular, if the adversary quits *after* round i^*, then the relevant distributions in the real and ideal world are statistically close. Our analysis only deals with aborts that take place prior to round i^*.

We only focus on the last $r' + 1$ elements of the corrupted party's backup sequence. Having assumed that i^* has not been surpassed, anything prior to the

last $r'+1$ elements is essentially noise, and it has no bearing on the security analysis. For every sequence of elements $\vec{\alpha}_{r'} \in \{0,1\}^{r'+1}$ and every $\beta \in \{0,1\}$, we compute the probability that the adversary's view and honest party's output in the real world is equal to $(\vec{\alpha}_{r'}, \beta)$, and we express the result in terms of the $B_{-}^{(\,,\,)}$-matrices. Similarly, for the ideal world, we compute the probability that the simulator's output and honest party's output is equal to $(\vec{\alpha}_{r'}, \beta)$, and we express the result in terms of the $B_{+}^{(\,,\,)}$-matrices and vector $\mathbf{z}_x^{(\vec{a}_{r'})}$.

The point of the exercise is to obtain (linear) constraints for vector $\mathbf{z}_x^{(\vec{a}_{r'})}$. Then, we ask if the constraints are satisfiable, and, if so, whether solutions can be found efficiently. The second question can be readily answered. If an appropriate solution exists, the simulator can compute it efficiently. Indeed, the simulator can approximate the probability of all possible sequences of size $r'+1$, and, assuming it exists, the simulator computes $\mathbf{z}_x^{(\vec{a}_{r'})}$ by solving a linear system of size $|X| \times |Y|$. Thus, it suffices to show that $\mathbf{z}_x^{(\vec{a}_{r'})}$ exists. The security features of Π come into play in this regard.

An early abort on the part of the adversary alters the conditional[2] probability distribution of the honest party's output. Security against sampling attacks guarantees that the output remains consistent with the function at hand. Thus, by introducing a threshold round and fine-tuning its parameter, we restrict the distribution of the output until it falls within the range of the function, and the simulator can match it with an appropriate input.

Three Simplifying Assumptions. The case where the adversary aborts before round r' needs special consideration. However, the only difference is that $\mathbf{z}_x^{(\vec{a}_i)}$ depends on fewer elements. The analysis is largely the same and we do not address this case any further. Furthermore, we assume that $p_x^{(\alpha)} \neq 0$, for every $\alpha \in \{0,1\}$ and $x \in X$. This assumption allows for a smoother exposition by disregarding degenerate cases. Finally, regarding Π, we will assume that security against sampling attacks holds perfectly, i.e. (3) and (4) are equal to $\mathbf{0}_\ell$ and $\mathbf{0}_k$ respectively. Again, the latest assumption is not necessary to prove the theorem. We do so in order to avoid introducing notions from Topology to deal with the convergent sequences.

4.3 Real vs Ideal

For every sequence $\vec{\alpha}_{r'} = (\alpha_0, \ldots, \alpha_{r'}) \in \{0,1\}^{r'+1}$ and every $\beta \in \{0,1\}$, we compute the probability that the adversary quitting at round $i \leq i^*$ observes $\vec{\alpha}_{r'}$ and the honest party outputs β. The adversary is assumed to use input $x \in X$ for the computation. To account for every possible input of the honest party, the relevant probabilities are expressed in terms of vectors.

[2] Conditioned on the adversary's view.

Claim 4.3. *In the real model, it holds that*

$$\begin{aligned}
&\Pr\left[(a_{i-r'},\ldots,a_i,b_{i-1})^{\mathsf{Real}} = (\vec{\alpha}_{r'},\beta) \,\middle|\, i \le i^*\right] \\
&\quad = (1-\gamma)^{r'+1}\cdot p_x^{(\alpha_0)}\cdots p_x^{(\alpha_{r'})}\cdot \mathbf{q}^{(\beta)T} + \gamma(1-\gamma)^{r'}\cdot p_x^{(\alpha_0)}\cdots p_x^{(\alpha_{r'-1})}\cdot \left[B_-^{(\alpha_{r'},\beta)}\right]_{x,*} \\
&\quad + \ldots + \gamma(1-\gamma)\cdot p_x^{(\alpha_0)}\cdot \left[B_-^{(\alpha_1\ldots\alpha_{r'},\beta)}\right]_{x,*} + \gamma\cdot\left[B_-^{(\vec{\alpha}_{r'},\beta)}\right]_{x,*},
\end{aligned}$$

where $\mathbf{q}^{(\beta)} = M^{(*,\beta)T}\cdot \mathbf{1}_\ell/\ell$.

Proof. Simple expansion over possible values of i^*.

Define $\mathbf{c}_x^{(\vec{\alpha}_{r'},\beta)} \in \mathbb{R}^k$ *such that* $\mathbf{c}_x^{(\vec{\alpha}_{r'},\beta)}(y) = \Pr\left[f_2(x_0,y)=\beta \,\middle|\, x_0 \leftarrow \mathbf{z}_x^{(\vec{\alpha}_{r'})}\right]$.

Claim 4.4. *In the ideal model, it holds that*

$$\begin{aligned}
&\Pr\left[(a_{i-r'},\ldots,a_i,f_2)^{\mathsf{Ideal}} = (\vec{\alpha}_{r'},\beta) \,\middle|\, i \le i^*\right] \\
&= (1-\gamma)^{r'+1}\cdot p_x^{(\alpha_0)}\cdots p_x^{(\alpha_{r'})}\cdot \mathbf{c}_x^{(\vec{\alpha}_{r'},\beta)T} + \gamma(1-\gamma)^{r'}\cdot p_x^{(\alpha_0)}\cdots p_x^{(\alpha_{r'-1})}\cdot \left[B_+^{(\alpha_{r'},\beta)}\right]_{x,*} \\
&\quad + \ldots + \gamma(1-\gamma)\cdot p_x^{(\alpha_0)}\cdot \left[B_+^{(\alpha_1\ldots\alpha_{r'},\beta)}\right]_{x,*} + \gamma\cdot\left[B_+^{(\vec{\alpha}_{r'},\beta)}\right]_{x,*}.
\end{aligned}$$

Thus, for every $\beta\in\{0,1\}$, we require that $\mathbf{c}_x^{(\vec{\alpha}_{r'},\beta)T}$ is close to

$$\mathbf{q}^{(\beta)T} + \sum_{i=0}^{r'} \lambda_i(\gamma,\vec{\alpha}_{r'})\cdot\left[B_-^{(\alpha_{r'-i}\ldots\alpha_{r'},\beta)} - B_+^{(\alpha_{r'-i}\ldots\alpha_{r'},\beta)}\right]_{x,*},$$

where

$$\lambda_i(\gamma,\vec{\alpha}_{r'}) = \frac{\gamma(1-\gamma)^{r'-i}\cdot p_x^{(\alpha_0)}\cdots p_x^{(\alpha_{r'-i-1})}}{(1-\gamma)^{r'+1}\cdot p_x^{(\alpha_0)}\cdots p_x^{(\alpha_{r'})}} = \frac{\gamma}{(1-\gamma)^{i+1}}\cdot\frac{1}{p_x^{(\alpha_{r'-i})}\cdots p_x^{(\alpha_{r'})}}.$$

Knowing that $\mathbf{c}_x^{(\vec{\alpha}_{r'},\beta)T} = \mathbf{z}_x^{(\vec{\alpha}_{r'})T}\cdot M^{(*,\beta)}$ and that $M^{(*,0)} = \mathbf{1}_{\ell\times k} - M^{(*,1)}$, the simulation is successful if there exists probability vector $\mathbf{z}_x^{(\vec{\alpha}_{r'})}\in\mathbb{R}^k$ such that

$$\begin{aligned}
&\mathbf{z}_x^{(\vec{\alpha}_{r'})T}\cdot M^{(*,1)} \\
&\quad \overset{s}{\equiv} \mathbf{q}^{(1)T} + \lambda_0\cdot\left[B_+^{(\alpha_{r'},1)} - B_-^{(\alpha_{r'},1)}\right]_{x,*} + \ldots + \lambda_{r'}\cdot\left[B_+^{(\vec{\alpha}_{r'},1)} - B_-^{(\vec{\alpha}_{r'},1)}\right]_{x,*}.
\end{aligned} \tag{5}$$

Define $\mathbf{u}_x^{(\vec{\alpha}_{r'})} = \mathbf{z}_x^{(\vec{\alpha}_{r'})} - \mathbf{1}_\ell/\ell$ and notice that (5) is equivalent to

$$\begin{aligned}&\mathbf{u}_x^{(\vec{\alpha}_{r'})T} \cdot M^{(*,1)} \\ &\quad \stackrel{s}{\equiv} \lambda_0 \cdot \left[B_+^{(\alpha_{r'},1)} - B_-^{(\alpha_{r'},1)}\right]_{x,*} + \cdots + \lambda_{r'} \cdot \left[B_+^{(\vec{\alpha}_{r'},1)} - B_-^{(\vec{\alpha}_{r'},1)}\right]_{x,*}, \end{aligned} \tag{6}$$

and

$$\begin{cases} \sum_{x_0} \mathbf{u}_x^{(\vec{\alpha}_{r'})}(x_0) = 0 \\ \forall x_0 \in X,\ \mathbf{u}_x^{(\vec{\alpha}_{r'})}(x_0) \in [-1/\ell, 1 - 1/\ell] \end{cases}.$$

Lemma 4.5. *Let* $\mathbf{c}$ *be an arbitrary vector and let* M *be an arbitrary matrix. There exists* $\mathbf{u}$ *such that* $\sum_z \mathbf{u}(z) = 0$ *and* $\mathbf{u}^T \cdot M = \mathbf{c}^T$ *if and only if* $\mathbf{c}^T\mathbf{v} = 0$*, for every* $\mathbf{v}$ *such that* $M\mathbf{v} \in \langle \mathbf{1} \rangle$*.*

Proof. Define

$$M' = \begin{pmatrix} -1\ 1 \ldots 0 \\ \vdots\ \vdots\ \ddots\ \vdots \\ -1\ 0 \ldots 1 \end{pmatrix} \cdot M.$$

Observe that the row-space of M' is equal to the image of the hyperplane $\{\mathbf{u} \mid \sum_z \mathbf{u}(z) = 0\}$ by M^T and that $\ker(M') = \{\mathbf{v} \mid M\mathbf{v} \in \langle \mathbf{1} \rangle\}$. Conclude by applying the fundamental theorem of linear algebra. □

Proof of Theorem 4.1. We show that there exist suitable vectors $\mathbf{u}_x^{(\vec{\alpha}_{r'})}$ satisfying (6), for every $x \in X$ and $\vec{\alpha}_{r'} \in \{0,1\}^{r'+1}$. By assumption, security against sampling attacks holds perfectly for Π. It follows that

$$\left(B_+^{(\vec{\alpha}_i,1)} - B_-^{(\vec{\alpha}_i,1)}\right) \cdot \mathbf{y} = \mathbf{0}_\ell,$$

for every $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$. By Lemma 4.5, there exists $\mathbf{u}_{x,i}^{(\vec{\alpha}_{r'})}$ such that $\sum_{x_0} \mathbf{u}_{x,i}^{(\vec{\alpha}_{r'})T}(x_0) = 0$ and

$$\mathbf{u}_{x,i}^{(\vec{\alpha}_{r'})T} \cdot M^{(*,1)} = \left[B_+^{(\vec{\alpha}_i,1)} - B_-^{(\vec{\alpha}_i,1)}\right]_{x,*}.$$

Thus, $\mathbf{u}_x^{(\vec{\alpha}_{r'})} \stackrel{\text{def}}{=} \sum_i \lambda_i \mathbf{u}_{x,i}^{(\vec{\alpha}_{r'})}$ satisfies (6). To conclude, we argue that there exists γ_0 such that $\mathbf{u}_x^{(\vec{\alpha}_{r'})}(x_0) \in [-1/\ell, 1 - 1/\ell]$, for every $\gamma < \gamma_0$. Recall that

$$\lambda_i(\gamma, \vec{\alpha}_{r'}) = \frac{\gamma}{(1-\gamma)^{i+1}} \cdot \frac{1}{p_x^{(\alpha_{r'-i})} \cdots p_x^{(\alpha_{r'})}}.$$

Observe that λ_i tends to 0 as γ tends to 0. □

5 The Asymmetric Case

Our analysis of locking strategies and sampling attacks culminates in Theorem 4.1 from the previous section. The theorem states that, in order to demonstrate that a given function is computable with full security, it suffices to design a constant-round, passively-secure protocol that is secure against sampling attacks. In this section, we look for relevant protocols for asymmetric Boolean functions. We propose an algorithm that takes a description of the function as input, and, depending on the termination step, either returns the description of an appropriate protocol, or it returns that it failed to do so.

We begin by visiting some mathematical tools and a few useful lemmas. Next, we define a game involving the parties computing f and the dealer. The game simulates the last interaction in a correct protocol computing f, and whose purpose is for the dealer to hand a backup[3] output to the disadvantaged party *without* compromising any of the security requirements. Finally, largely as an extension of the game, we obtain an algorithm for designing constant-round protocols that are passively secure and secure against sampling attacks. Using the tools and the lemmas from Sect. 5.1, we demonstrate that our algorithm satisfies correctness.

Speculative Remark. For what it is worth, numerical results on small cases indicate that our algorithm accounts for the overwhelmingly majority of non semi-balanced functions. We also encountered a handful of non semi-balanced functions for which our algorithm fails to come up with a suitable protocol. These functions are noteworthy because we suspect that their unknown status cannot be attributed to potential shortcomings of our algorithm. We believe that our algorithm is as good at finding suitable protocols as can be expected.

5.1 Irreducible Locking Strategies

Let $f : X \times Y \to \{0,1\}^2$ denote some Boolean asymmetric (possibly randomized) finite function. Since f is asymmetric, it has four associated matrices $M^{(0,0)}$, $M^{(0,1)}$, $M^{(1,0)}$, $M^{(1,1)} \in [0,1]^{\ell \times k}$. Recall that locking strategies for P_1 and P_2 correspond to elements of the vector spaces $\langle \mathcal{L}_1 \rangle = \{\mathbf{x} \in \mathbb{R}^\ell \,|\, \mathbf{x}^T M^{(1,*)} \in \langle \mathbf{1}_k^T \rangle\}$ and $\langle \mathcal{L}_2 \rangle = \{\mathbf{y} \in \mathbb{R}^k \,|\, M^{(*,1)}\mathbf{y} \in \langle \mathbf{1}_\ell \rangle\}$, where $\mathcal{L}_1$ and $\mathcal{L}_2$ denote arbitrary bases of each space. Without loss of generality, assume $|\mathcal{L}_1| = s_1$ and $|\mathcal{L}_2| = s_2$. Locking strategies endow a matrix with a matroid structure, in the same way that linear dependence does. We define the matroid by means of its *minimally dependent sets*, i.e. circuits.

Definition 5.1. We say that the columns of $M^{(*,1)}$ indexed by $Y' \subseteq Y$ are *minimally dependent* if

– $\left\{M^{(*,1)}\mathbf{e}_y\right\}_{y \in Y'} \cup \{\mathbf{1}_\ell\}$ are linearly dependent,

[3] Other than the actual output.

– for every $y_0 \in Y'$, it holds that $\left\{M^{(*,1)}\mathbf{e}_y\right\}_{y\in Y'\setminus\{y_0\}} \cup \{\mathbf{1}_\ell\}$ are linearly independent.

Similarly, we say that the rows of $M^{(1,*)}$ indexed by $X' \subseteq X$ are *minimally dependent* if

– $\left\{\mathbf{e}_x^T M^{(1,*)}\right\}_{x\in X'} \cup \left\{\mathbf{1}_k^T\right\}$ are linearly dependent,
– for every $x_0 \in X'$, it holds that $\left\{\mathbf{e}_x^T M^{(1,*)}\right\}_{x\in X'\setminus\{x_0\}} \cup \left\{\mathbf{1}_k^T\right\}$ are linearly independent.

Proposition 5.2. *Suppose that the columns of $M^{(*,1)}$ indexed by $Y' \subseteq Y$ are minimally dependent. Up to a multiplicative factor, there exists a unique $\mathbf{q} \in \mathbb{R}^k \setminus \{\mathbf{0}_k\}$ such that $M^{(*,1)}\mathbf{q} \in \langle \mathbf{1}_\ell \rangle$ and $\mathsf{supp}(\mathbf{q}) = Y'$.*

Proof. By definition, there exists $\mathbf{q} \in \mathbb{R}^k$ such that $M^{(*,1)}\mathbf{q} \in \langle \mathbf{1}_\ell \rangle$ and $\mathsf{supp}(\mathbf{q}) = Y'$. The non-trivial task is to show that this vector is unique, *up* to a multiplicative factor. Suppose there exists $\mathbf{q}'$ such that $\mathsf{supp}(\mathbf{q}') \subseteq Y'$ and $M^{(*,1)}\mathbf{q}' \in \langle \mathbf{1}_\ell \rangle$. In pursuit of a contradiction, assume that $\mathbf{q}' \neq \lambda\mathbf{q}$, for every $\lambda \in \mathbb{R}$. Equivalently, there exists $i, j \in Y'$ such that $\mathbf{q}(i) = \lambda_i \mathbf{q}'(i)$ and $\mathbf{q}(j) = \lambda_j \mathbf{q}'(j)$, with $\lambda_i \neq \lambda_j$. Without loss of generality, say that $\lambda_i \neq 0$ and define $\mathbf{q}'' = \lambda_i \cdot \mathbf{q}' - \mathbf{q}$. Deduce that $M^{(*,1)}\mathbf{q}'' \in \langle \mathbf{1}_\ell \rangle$ and $\mathsf{supp}(\mathbf{q}'') \subsetneq Y'$, in contradiction with the fact that the columns indexed by Y' are minimally dependent. □

Definition 5.3. If $\mathbf{q} \in \mathbb{R}^k$ is as in Proposition 5.2, we say that $\mathbf{q}$ is *irreducible*.

Proposition 5.4. *There exists a basis of $\langle \mathcal{L}_2 \rangle$ consisting of irreducible strategies.*

Proof. It is a well known that any generating set contains a basis. Thus, it suffices to show that irreducible locking strategies form a generating set. Let $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$ and consider $\mathsf{supp}(\mathbf{y})$. Let $\mu_1, \ldots, \mu_{t_\mathbf{y}}$ denote all the subsets of $\mathsf{supp}(\mathbf{y})$ that index minimally dependent columns, and write $\mathbf{q}_1, \ldots, \mathbf{q}_{t_\mathbf{y}}$ for the associated unique *irreducible* locking strategies. We show that $\mathbf{y} \in \langle \mathbf{q}_1, \ldots, \mathbf{q}_{t_\mathbf{y}} \rangle$ by constructing a sequence of locking strategies $\mathbf{y}_0, \ldots, \mathbf{y}_{s_\mathbf{y}}$ such that

$$\begin{cases} \mathbf{y}_0 = \mathbf{y} \\ \mathbf{y}_{j+1} = \mathbf{y}_j - \alpha_j \cdot \mathbf{q}^{(j)} \\ \mathbf{y}_{s_\mathbf{y}} = \mathbf{0}_\ell \end{cases},$$

where $\alpha_j \in \mathbb{R}$ and $\mathbf{q}^{(j)} \in \{\mathbf{q}_1, \ldots, \mathbf{q}_{t_\mathbf{y}}\}$. Let $\mathbf{q}^{(0)}$ be an arbitrary element of $\{\mathbf{q}_1, \ldots, \mathbf{q}_{t_\mathbf{y}}\}$ and fix j_0 such that $\mathbf{q}^{(0)}(j_0) \neq 0$. Define $\mathbf{y}_1 = \mathbf{y} - \frac{\mathbf{y}(j_0)}{\mathbf{q}^{(0)}(j_0)} \cdot \mathbf{q}^{(0)}$. Notice that $\mathbf{y}_1$ is a locking strategy and that $\mathsf{supp}(\mathbf{y}_1) \subsetneq \mathsf{supp}(\mathbf{y})$. Since $\mathbf{y}_1$ is a locking strategy, it follows that $\mu^{(1)} \subset \mathsf{supp}(y_1)$, for some $\mu^{(1)} \in \{\mu_1, \ldots, \mu_{t_\mathbf{y}}\}$. Write $\mathbf{q}^{(1)}$ for the associated locking strategy. Similarly to what we just did, fix j_1 such that $\mathbf{q}^{(1)}(j_1) \neq 0$, define $\mathbf{y}_2 = \mathbf{y}_1 - \frac{\mathbf{y}_1(j_1)}{\mathbf{q}^{(1)}(j_1)} \cdot \mathbf{q}^{(1)}$, and notice that $\mathbf{y}_2$ is a locking strategy and that $\mathsf{supp}(\mathbf{y}_2) \subsetneq \mathsf{supp}(\mathbf{y}_1)$. Repeat the procedure and conclude that it terminates in at most $|\mathsf{supp}(\mathbf{y})|$ steps. □

Define $Y_0, \ldots, Y_{k'}$ to be a partitioning of the input domain Y that we construct as follows. First, $y \in Y_0$ if $\mathbf{e}_y$ is orthogonal to $\langle \mathcal{L}_2 \rangle$. Next, for $i \geq 1$, let $\mathbf{q}^{(i)}$ be an irreducible locking strategy such that $\mathsf{supp}(\mathbf{q}^{(i)}) \cap (Y_{i-1} \cup \ldots, \cup Y_0) = \emptyset$. Finally, $y \in Y_i$ if there exist irreducibles $\mathbf{q}_1^{(i)}, \ldots, \mathbf{q}_{t_y}^{(i)}$ such that

$$\begin{cases} \mathbf{q}^{(i)} = \mathbf{q}_1^{(i)} \\ \mathsf{supp}(\mathbf{q}_j^{(i)}) \cap \mathsf{supp}(\mathbf{q}_{j+1}^{(i)}) \neq \emptyset \\ y \in \mathsf{supp}(\mathbf{q}_{t_y}^{(i)}) \end{cases} .$$

5.2 The Dealer Game

In this section, we present a game involving the parties computing f and the dealer. The purpose of the game is to define a simplified variant of the security against sampling attacks requirement. Assume that the honest party, say P_2, applies some locking strategy $\mathbf{y}$ while executing a protocol for computing f. If the protocol is secure against sampling attacks, then the adversary cannot distinguish between the correct output and the backup output of the honest party. In the worst case, the adversary is handed the output of the corrupted party before the honest party's receives his. In such an event, we ask what the honest party's backup output ought to be, other than the correct output.

Write a_i (resp. b_i) for P_1's (resp. P_2's) backup output at round i. Let $\widehat{b}_*$ denote the bit obtained from b_* by applying[4] $\mathbf{y}$, and r denotes the number of rounds. From an honest P_2's perspective, we require that the pairs $(a_i, \widehat{b}_{i-1})$ and $(a_i, \widehat{b}_r)$ are statistically close, for every $x \in X$, $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$ and $i \in \{1 \ldots r\}$. Consider the following process involving a dealer. The dealer receives inputs x and y from P_1 and P_2, respectively, and computes $f(x, y) = (f_1(x, y), f_2(x, y))$. Then, the dealer hands $f_1(x, y)$ to P_1 and a bit b to P_2, where b is a probabilistic function of P_2's input and $f_2(x, y)$. We investigate how to construct b with the following goals in mind.

1. *minimize* the information b contains about $f_2(x, y)$
2. $(f_1, \widehat{f_2})$ is statistically close to $(f_1, \widehat{b})$, for every $x \in X$ and $\mathbf{q} \in \langle \mathcal{L}_2 \rangle$.

Let us introduce vectors $\mathbf{b}^{(0)}$, $\mathbf{b}^{(1)} \in \mathbb{R}^k$ such that

$$\mathbf{b}^{(\beta)}(y_0) = \Pr\left[b = 1 \,\middle|\, f_2(x, y) = \beta \wedge y = y_0 \right].$$

Fix $y \in Y$, and notice that $b \equiv f_2$ on input y if $\mathbf{b}^{(0)}(y) = 0$ and $\mathbf{b}^{(1)}(y) = 1$. On the other hand, b contains no information about $f_2(x, y)$ if and only if $\mathbf{b}^{(0)}(y) = \mathbf{b}^{(1)}(y)$. Consequently, our aim is for $\mathbf{b}^{(0)}$ and $\mathbf{b}^{(1)}$ to be equal on as many indices as possible.

[4] Recall that $\mathbf{y}$ encodes an input distribution but also a certain transformation.

Claim 5.5. *Using the notation above, It holds that* $(f_1, \widehat{f_2})$ *is statistically close to* $(f_1, \widehat{b})$ *if and only if, for every* $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$,

$$\begin{cases} M^{(0,0)}\left(\mathbf{b}^{(0)} * \mathbf{y}\right) + M^{(0,1)}\left(\mathbf{b}^{(1)} * \mathbf{y}\right) = M^{(0,1)}\mathbf{y} \\ M^{(1,0)}\left(\mathbf{b}^{(0)} * \mathbf{y}\right) + M^{(1,1)}\left(\mathbf{b}^{(1)} * \mathbf{y}\right) = M^{(1,1)}\mathbf{y} \end{cases}. \tag{7}$$

Proof. Fix $x \in X$, $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$, $\alpha \in \{0,1\}$, and note that

$$\begin{aligned} \Pr\left[(f_1, \widehat{f_2}) = (\alpha, 1)\right] &= \mathbf{e}_x^T \left(\sum_{\mathbf{y}(y) \geq 0} \left[M^{(\alpha,1)}\right]_{*,y} \mathbf{y}(y) + \sum_{\mathbf{y}(y) < 0} \left[M^{(\alpha,0)}\right]_{*,y} |\mathbf{y}(y)| \right) \\ &= \mathbf{e}_x^T \left(M^{(\alpha,1)}\mathbf{y} + \sum_{\mathbf{y}(y) < 0} \left[M^{(\alpha,*)}\right]_{*,y} |\mathbf{y}(y)| \right) \end{aligned}$$

On the other hand, $\Pr\left[(f_1, \widehat{b}) = (\alpha, 1)\right]$

$$\begin{aligned} &= \sum_{\mathbf{y}(y) \geq 0} \mathbf{e}_x^T \left(\left[M^{(\alpha,1)}\right]_{*,y} \cdot \mathbf{b}^{(1)}(y) + \left[M^{(\alpha,0)}\right]_{*,y} \cdot \mathbf{b}^{(0)}(y) \right) \mathbf{y}(y) \\ + \sum_{\mathbf{y}(y) < 0} &\left(\left[M^{(\alpha,1)}\right]_{*,y} \cdot (1 - \mathbf{b}^{(1)}(y)) + \left[M^{(\alpha,0)}\right]_{*,y} \cdot (1 - \mathbf{b}^{(0)}(y)) \right) |\mathbf{y}(y)|, \end{aligned}$$

and thus $\Pr\left[(f_1, \widehat{b}) = (\alpha, 1)\right]$

$$= \mathbf{e}_x^T \left(M^{(\alpha,0)}\left(\mathbf{b}^{(0)} * \mathbf{y}\right) + M^{(\alpha,1)}\left(\mathbf{b}^{(1)} * \mathbf{y}\right) + \sum_{\mathbf{y}(y) < 0} \left[M^{(\alpha,*)}\right]_{*,y} |\mathbf{y}(y)| \right).$$

To conclude, note that since $\mathbf{b}^{(0)}, \mathbf{b}^{(1)}$ are fixed vectors, it holds that $(f_1, \widehat{f_2})$ and $(f_1, \widehat{b})$ are statistically close if and only if they are *identically distributed*. □

Moving on, fix $Y_i \in \{Y_0, \ldots, Y_{k'}\}$ and suppose there exist $\mathbf{b}^{(0)}, \mathbf{b}^{(1)}$ satisfying Eq. (7) such that $\mathbf{b}^{(0)}(y_0) \neq 0$ or $\mathbf{b}^{(1)}(y_0) \neq 1$, for some $y_0 \in Y_i$. We show that there exist $\mathbf{b}'^{(0)}, \mathbf{b}'^{(1)}$ satisfying Eq. (7) such that $\mathbf{b}'^{(0)}(y) = \mathbf{b}'^{(1)}(y)$, for every $y \in Y_i$. This is where the underlying matroid structure will come in handy.

Proposition 5.6. *It holds that* $\mathbf{b}^{(1)}(y) - \mathbf{b}^{(0)}(y) = \mathbf{b}^{(1)}(y_0) - \mathbf{b}^{(0)}(y_0)$, *for every* $y \in Y_i$. *In addition, vectors* $\mathbf{b}'^{(1)}$, $\mathbf{b}'^{(0)}$ *satisfy Eq. (7), where*

$$\mathbf{b}'^{(b)}(y) = \begin{cases} \mathbf{b}^{(b)}(y) & \text{if } y \notin Y_j \\ \dfrac{\mathbf{b}^{(0)}(y)}{\mathbf{b}^{(0)}(y_0) - \mathbf{b}^{(1)}(y_0) + 1} & \text{if } y \in Y_j \end{cases}.$$

Proof. For the first part of the claim, we apply Proposition 5.2. The case $i = 0$ is left to the reader. Let $i \geq 1$ and fix irreducible $\mathbf{q}$ such that $y_0 \in \mathsf{supp}(\mathbf{q})$. We know that, for any $\mathbf{y} \in \langle \mathcal{L}_2 \rangle$,

$$M^{(0,0)} \left(\mathbf{b}^{(0)} * \mathbf{y} \right) + M^{(0,1)} \left(\mathbf{b}^{(1)} * \mathbf{y} \right) = M^{(0,1)} \mathbf{y}, \tag{8}$$

$$M^{(1,0)} \left(\mathbf{b}^{(0)} * \mathbf{y} \right) + M^{(1,1)} \left(\mathbf{b}^{(1)} * \mathbf{y} \right) = M^{(1,1)} \mathbf{y}. \tag{9}$$

Let $\mathbf{y} = \mathbf{q}$ and add the two expressions.

$$\left(\mathbf{1}_{\ell \times k} - M^{(*,1)} \right) \left(\mathbf{b}^{(0)} * \mathbf{q} \right) + M^{(*,1)} \left(\mathbf{b}^{(1)} * \mathbf{q} \right) = M^{(*,1)} \mathbf{q}.$$

By moving a few terms around, deduce that $M^{(*,1)} \left((\mathbf{b}^{(1)} - \mathbf{b}^{(0)}) * \mathbf{q} \right) \in \langle \mathbf{1}_\ell \rangle$. Consequently, by Proposition 5.2, $\mathbf{b}^{(1)}(y) - \mathbf{b}^{(0)}(y) = \mathbf{b}^{(1)}(y_0) - \mathbf{b}^{(0)}(y_0)$, for every $y \in \mathsf{supp}(\mathbf{q})$. Moving on, fix an arbitrary $y \in Y_i$. We know there exists a sequence of irreducibles $\mathbf{q}_1^{(i)} \dots \mathbf{q}_{t'_y}^{(i)}$ such that

$$\begin{cases} \mathbf{q} = \mathbf{q}_1^{(i)} \\ \mathsf{supp}(\mathbf{q}_j^{(i)}) \cap \mathsf{supp}(\mathbf{q}_{j+1}^{(i)}) \neq \emptyset \\ y \in \mathsf{supp}(\mathbf{q}_{t'_y}^{(i)}) \end{cases},$$

Apply the same argument as above and, by induction, deduce that $\mathbf{b}^{(1)}(y) - \mathbf{b}^{(0)}(y) = \mathbf{b}^{(1)}(y_0) - \mathbf{b}^{(0)}(y_0)$. For the second part of the claim, we rely on the following observations.

- Vectors $\mathbf{b}_0^{(0)}$ and $\mathbf{b}_0^{(1)}$ satisfy Eqs. (8) and (9), where

$$\mathbf{b}_0^{(0)} = \begin{cases} \mathbf{b}^{(0)}(y) & \text{if } y \notin Y_i \\ 0 & \text{if } y \in Y_i \end{cases}, \qquad \mathbf{b}_0^{(1)} = \begin{cases} \mathbf{b}^{(1)}(y) & \text{if } y \notin Y_i \\ 1 & \text{if } y \in Y_i \end{cases}.$$

- Solutions to Eqs. (8) and (9) can be combined linearly.

The second item is trivial. For the first item, we show that vectors $\mathbf{b}_0^{(0)}$ and $\mathbf{b}_0^{(1)}$ are solutions to the equations for a particular basis of $\langle \mathcal{L}_2 \rangle$. By Proposition 5.4, consider a basis of $\langle \mathcal{L}_2 \rangle$ that consists of irreducible strategies. Conclude by observing that $Y_i \cap \mathsf{supp}(\mathbf{q}') = \emptyset$, for every irreducible $\mathbf{q}'$ such that $\mathsf{supp}(\mathbf{q}) \nsubseteq Y_i$. Next, define

$$\mathbf{b}'^{(b)} = \frac{1}{\mathbf{b}^{(0)}(y_0) - \mathbf{b}^{(1)}(y_0) + 1} \cdot \mathbf{b}^{(b)} + \left(1 - \frac{1}{\mathbf{b}^{(0)}(y_0) - \mathbf{b}^{(1)}(y_0) + 1} \right) \cdot \mathbf{b}_0^{(b)}.$$

We note that $\mathbf{b}'^{(0)}$, $\mathbf{b}'^{(1)}$ admit the right expression. It remains to show that $\mathbf{b}'^{(b)}(y) \in [0, 1]$, for every y. Since $\mathbf{b}'^{(b)}(y) = \mathbf{b}^{(b)}(y)$ if $y \notin Y_j$, it suffices to show that

$$\frac{\mathbf{b}^{(0)}(y)}{\mathbf{b}^{(0)}(y) - \mathbf{b}^{(1)}(y) + 1} \in [0, 1], \tag{10}$$

for $y \in Y_i$. We conclude by observing that (10) is equivalent to $0 \leq \mathbf{b}^{(0)}(y)$ and $\mathbf{b}^{(1)}(y) \leq 1$. □

5.3 The Algorithm

Next, we show how to construct passively-secure protocols that are also secure against sampling attacks. The idea is to build the backup outputs from the bottom-up, i.e. start with $a_r \equiv f_1$ and $b_r \equiv f_2$, and construct a_{r-1} and b_{r-1} such that a_{r-1} (resp. b_{r-1}) only depends on x and $f_1(x,y)$ (resp. y and $f_2(x,y)$) without compromising security against sampling attacks.

To this end, we employ a minimization algorithm in combination with Proposition 5.6. Without loss of generality, we begin by assuming that P_1 is corrupted, and that he observes $a_r \equiv f_1(x,y)$. To define b_{r-1}, we run an optimization algorithm that constructs vectors $\{\mathbf{b}^{(\beta)}\}_{\beta\in\{0,1\}}$, and we delete any input $y \in Y$ for which $\mathbf{b}^{(1)}(y) - \mathbf{b}^{(0)}(y) \neq 1$. Then, in order to define a_{r-1}, we run an optimization algorithm that constructs vectors $\{\mathbf{a}^{(\alpha)}\}_{\alpha\in\{0,1\}}$, assuming P_2 is corrupted, and the party is privy to the output *only if the input he used was not deleted in the previous step*. We proceed by deleting any input $x \in X$ for which $\mathbf{a}^{(1)}(y) - \mathbf{a}^{(0)}(y) \neq 1$. We carry on in this fashion until one party runs out of inputs, *or* the process does not allow for any further deletions.

Getting ahead of ourselves, we note that deleted inputs cannot be used by the adversary to mount a successful sampling attack. In light of Proposition 5.6, if an input was deleted at iteration i, then every backup output until round $r-i$ contains no information about the output.

Additional Notation. Before we describe the algorithm, let us introduce some notation. For every $\mathbf{q} \in \mathcal{L}_2$ and $X' \subseteq X$, define

$$A_{\mathbf{q}}(X') = \begin{pmatrix} \left[M^{(0,0)} * Q\right]_{X'} & \left[M^{(0,1)}_{X'} * Q\right]_{X'} \\ \left[M^{(1,0)} * Q\right]_{X'} & \left[M^{(1,1)} * Q\right]_{X'} \\ M^{(*,0)} * Q & M^{(*,1)} * Q \end{pmatrix}, \qquad \vec{b}_{\mathbf{q}} = \begin{pmatrix} \left[M^{(0,1)}\right]_{X'} \mathbf{q} \\ \left[M^{(1,1)}\right]_{X'} \mathbf{q} \\ M^{(*,1)}\mathbf{q} \end{pmatrix}$$

where $Q = \mathbf{1}_\ell \cdot \mathbf{q}^T$ and the notation $[\ \cdot\ ]_{X'}$ indicates that only the rows indexed by $X' \subseteq X$ appear. Write $\mathcal{L}_2 = \{\mathbf{q}_1, \ldots, \mathbf{q}_{s_2}\}$ and consider the following linear system for unknown $(\mathbf{b}^{(0)T}, \mathbf{b}^{(1)T})$.

$$\begin{pmatrix} A_{\mathbf{q}_1}(X') \\ A_{\mathbf{q}_2}(X') \\ \vdots \\ A_{\mathbf{q}_{s_2}}(X') \end{pmatrix} \cdot \begin{pmatrix} \mathbf{b}^{(0)} \\ \mathbf{b}^{(1)} \end{pmatrix} = \begin{pmatrix} \vec{b}_{\mathbf{q}_1}(X') \\ \vec{b}_{\mathbf{q}_2}(X') \\ \vdots \\ \vec{b}_{\mathbf{q}_{s_2}}(X') \end{pmatrix} \tag{11}$$

$$\begin{pmatrix} \mathbf{0}_k \\ \mathbf{0}_k \end{pmatrix} \leq \begin{pmatrix} \mathbf{b}^{(0)} \\ \mathbf{b}^{(1)} \end{pmatrix} \leq \begin{pmatrix} \mathbf{1}_k \\ \mathbf{1}_k \end{pmatrix}$$

Similarly, for every $\mathbf{p} \in \mathcal{L}_1$ and $Y \subseteq Y'$, define

$$B_{\mathbf{p}}(Y') = \begin{pmatrix} \left[M^{(0,0)^T} * P\right]_{Y'} & \left[M^{(1,0)^T} * P\right]_{Y'} \\ \left[M^{(0,1)^T} * P\right]_{Y'} & \left[M^{(1,1)^T} * P\right]_{Y'} \\ M^{(0,*)^T} * P & M^{(1,*)^T} * P \end{pmatrix}, \qquad \vec{a}_{\mathbf{p}} = \begin{pmatrix} \left[M^{(1,0)^T}\right]_{Y'} \mathbf{p} \\ \left[M^{(1,1)^T}\right]_{Y'} \mathbf{p} \\ M^{(1,*)^T}\mathbf{p} \end{pmatrix}$$

where $P = \mathbf{1}_k \cdot \mathbf{p}^T$ and the notation $[\ \cdot\ ]_{Y'}$ indicates that only the rows indexed by $Y' \subseteq Y$ appear. Write $\mathcal{L}_1 = \{\mathbf{p}_1, \ldots, \mathbf{p}_{s_1}\}$ and consider the following linear system for unknown $(\mathbf{a}^{(0)T}, \mathbf{a}^{(1)T})$.

$$\begin{pmatrix} B_{\mathbf{p}_1}(Y') \\ B_{\mathbf{p}_2}(Y') \\ \vdots \\ B_{\mathbf{p}_{s_1}}(Y') \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}^{(0)} \\ \mathbf{a}^{(1)} \end{pmatrix} = \begin{pmatrix} \vec{a}_{\mathbf{p}_1}(Y') \\ \vec{a}_{\mathbf{p}_2}(Y') \\ \vdots \\ \vec{a}_{\mathbf{p}_{s_1}}(Y') \end{pmatrix} \tag{12}$$
$$\begin{pmatrix} \mathbf{0}_\ell \\ \mathbf{0}_\ell \end{pmatrix} \leq \begin{pmatrix} \mathbf{a}^{(0)} \\ \mathbf{a}^{(1)} \end{pmatrix} \leq \begin{pmatrix} \mathbf{1}_\ell \\ \mathbf{1}_\ell \end{pmatrix}$$

As noted earlier, the idea is to delete inputs from the parties in a sequence of iterations. Namely, we begin by running a linear program that minimizes $-\mathbf{1}_k^T \mathbf{b}^{(0)} + \mathbf{1}_k^T \mathbf{b}^{(1)}$ under the constraints of Eq. (11), with $X' = X$. At this point, we delete any input $y \in Y$ for which $\mathbf{b}^{(1)}(y) - \mathbf{b}^{(0)}(y) < 1$. Write $Y^- \subseteq Y$ for the remaining inputs. We proceed by running a linear program that minimizes $-\mathbf{1}_\ell^T \mathbf{a}^{(0)} + \mathbf{1}_\ell^T \mathbf{a}^{(1)}$ under the constraints of Eq. (12), with $Y' = Y^-$. Again, we delete any input $x \in X$ for which $\mathbf{a}^{(1)}(x) - \mathbf{a}^{(0)}(x) < 1$. We repeat the procedure until either one of the parties runs out of inputs *or* no further deletions can be made, for either party. See Fig. 4 for a full description of the algorithm. Before we discuss the general ramifications of the terminating step, we illustrate the usefulness of our algorithm with an example.

Example. Consider the deterministic asymmetric Boolean function from [3] described by the following matrices.

$$M^{(1,*)} = \begin{pmatrix} 0\,1\,1\,0 \\ 1\,0\,1\,1 \\ 1\,0\,0\,0 \\ 0\,1\,0\,1 \end{pmatrix}, \qquad M^{(*,1)} = \begin{pmatrix} 1\,1\,1\,0 \\ 1\,0\,1\,1 \\ 0\,1\,0\,1 \\ 1\,1\,0\,0 \end{pmatrix}.$$

For this function, each party has a unique locking strategy. Namely, $\mathbf{p}^T = (1,1,1,1)$ and $\mathbf{q}^T = (1,1,0,1)$ respectively. Let us walk through each iteration of the algorithm. The first optimization returns $\mathbf{b}^{(0)T} = (0,0,1,0)$ and $\mathbf{b}^{(1)T} = (1,1,0,1)$. Notice that $Y^+ = \{y_1, y_2, y_4\}$. The algorithm assigns $Y^- = Y^+$ and moves on to the next step. The second optimization returns $\mathbf{a}^{(0)T} = (1/2, 0, 1, 1/2)$ and $\mathbf{a}^{(1)T} = (1/2, 0, 1, 1/2)$. Notice that $X^+ = \emptyset$, and the algorithm terminates. Now, we will use these vectors to define backup outputs for the parties. Consider the following two-round protocol described by means of the backup outputs $\{(a_i, b_i)\}_{i=0\ldots2}$. Assuming the parties use $x \in X$ and $y \in Y$ for the computation,

Building the Backup Outputs

1. **Inputs**: f, $\mathcal{L}_1$, $\mathcal{L}_2$.
2. Define $X^- = X$ and $Y^- = Y \cup \{k+1\}$.
3. Minimize $-\mathbf{1}_k^T \mathbf{b}^{(0)} + \mathbf{1}_k^T \mathbf{b}^{(1)}$ subject to

$$\begin{pmatrix} A_{\mathbf{q}_1}(X^-) \\ A_{\mathbf{q}_2}(X^-) \\ \vdots \\ A_{\mathbf{q}_{s_2}}(X^-) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{b}^{(0)} \\ \mathbf{b}^{(1)} \end{pmatrix} = \begin{pmatrix} \vec{b}_{\mathbf{q}_1}(X^-) \\ \vec{b}_{\mathbf{q}_2}(X^-) \\ \vdots \\ \vec{b}_{\mathbf{q}_{s_2}}(X^-) \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{0}_k \\ \mathbf{0}_k \end{pmatrix} \leq \begin{pmatrix} \mathbf{b}^{(0)} \\ \mathbf{b}^{(1)} \end{pmatrix} \leq \begin{pmatrix} \mathbf{1}_k \\ \mathbf{1}_k \end{pmatrix}$$

   Define Y^+ consisting of all the y's such that $\mathbf{b}^{(1)}(y) - \mathbf{b}^{(0)}(y) = 1$.
4. **If** $Y^+ = Y^-$ *or* $Y^+ = \emptyset$ stop, otherwise set $Y^- \stackrel{\text{def}}{=} Y^+$ and go to Step 5.
5. Minimize $-\mathbf{1}_\ell^T \mathbf{a}^{(0)} + \mathbf{1}_\ell^T \mathbf{a}^{(1)}$ subject to

$$\begin{pmatrix} B_{\mathbf{p}_1}(Y^-) \\ B_{\mathbf{p}_2}(Y^-) \\ \vdots \\ B_{\mathbf{p}_{s_1}}(Y^-) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}^{(0)} \\ \mathbf{a}^{(1)} \end{pmatrix} = \begin{pmatrix} \vec{a}_{\mathbf{p}_1}(Y^-) \\ \vec{a}_{\mathbf{p}_2}(Y^-) \\ \vdots \\ \vec{a}_{\mathbf{p}_{s_1}}(Y^-) \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{0}_\ell \\ \mathbf{0}_\ell \end{pmatrix} \leq \begin{pmatrix} \mathbf{a}^{(0)} \\ \mathbf{a}^{(1)} \end{pmatrix} \leq \begin{pmatrix} \mathbf{1}_\ell \\ \mathbf{1}_\ell \end{pmatrix}$$

   Define X^+ consisting of all the x's such that $\mathbf{a}^{(1)}(x) - \mathbf{a}^{(0)}(x) = 1$.
6. **If** $X^+ = X^-$ *or* $X^+ = \emptyset$ stop, otherwise set $X^- \stackrel{\text{def}}{=} X^+$ and go to Step 3.
7. **Output:** The transcript of the execution.

Fig. 4. An algorithm for designing fully-secure protocols.

$$a_0 = f_1(x, \widetilde{y}) \text{ where } \widetilde{y} \in_U Y \qquad b_0 = f_2(\widetilde{x}, y) \text{ where } \widetilde{x} \in_U X$$

$$a_1 = \begin{cases} a \in_U \{0,1\} & \text{if } x \in \{x_1, x_4\} \\ 1 & \text{if } x = x_2 \\ 0 & \text{if } x = x_3 \end{cases} \qquad b_1 = \begin{cases} b \in_U \{0,1\} & \text{if } y = y_3 \\ f_2(x,y) & \text{if } y \neq y_3 \end{cases}.$$

$$a_2 = f_1(x, y) \qquad b_2 = f_2(x, y)$$

Observe that a_1 and b_1 are constructed in accordance with $\mathbf{a}^{(0)}$, $\mathbf{a}^{(1)}$ and $\mathbf{b}^{(0)}$, $\mathbf{b}^{(1)}$, respectively. It is not hard to see that the resulting protocol is passively secure and secure against sampling attacks. In light of Theorem 4.1, function f is computable with full security. Next, we discuss the general case.

General Case. Assume that the algorithm terminates because one of the parties ran out of inputs. Without loss of generality, say that $Y^+ = \emptyset$ and write

$$\begin{pmatrix} \mathbf{b}_0^{(0)} \\ \mathbf{b}_0^{(1)} \end{pmatrix} \cdots \begin{pmatrix} \mathbf{b}_t^{(0)} \\ \mathbf{b}_t^{(1)} \end{pmatrix}, \qquad \begin{pmatrix} \mathbf{a}_1^{(0)} \\ \mathbf{a}_1^{(1)} \end{pmatrix} \cdots \begin{pmatrix} \mathbf{a}_t^{(0)} \\ \mathbf{a}_t^{(1)} \end{pmatrix}$$

for the vectors computed in the execution of the algorithm – starting from the bottom-up – i.e. $\mathbf{b}_0^{(0)}, \mathbf{b}_0^{(1)}$ denote the *last* vectors computed for P_2 and $\mathbf{b}_t^{(0)}, \mathbf{b}_t^{(1)}$ denote the *first* vectors computed for P_2. Similarly, $\mathbf{a}_1^{(0)}, \mathbf{a}_1^{(1)}$ denote the *last* vectors computed for P_1 and $\mathbf{a}_t^{(0)}, \mathbf{a}_t^{(1)}$ denote the *first* vectors computed for P_1. Now, assume[5] that for every $i \in \{1, \ldots, t\}$, and every $j \in \{1, \ldots, \ell\}$, either $\mathbf{a}_i^{(1)}(j) - \mathbf{a}_i^{(0)}(j) = 1$ or $\mathbf{a}_i^{(1)}(j) = \mathbf{a}_i^{(0)}(j)$. Similarly, for every $i \in \{0, \ldots, t\}$, and every $j \in \{1, \ldots, k\}$, either $\mathbf{b}_i^{(1)}(j) - \mathbf{b}_i^{(0)}(j) = 1$ or $\mathbf{b}_i^{(1)}(j) = \mathbf{b}_i^{(0)}(j)$. Write $\mathcal{T}_f$ for the transcript of the algorithm and consider the protocol from Fig. 5.

Theorem 5.7. *Using the notation above, Protocol* SecSamp$(\mathcal{T}_f)$ *is passively secure and secure against sampling attacks.*

Protocol SecSamp$(\mathcal{T}_f)$

1. The parties P_1 and P_2 hand their inputs, denoted x and y respectively, to the dealer.[a]
2. The dealer computes $f(x, y) = (\alpha, \beta)$ and constructs $(a_0, \ldots, a_{t+1})$ and $(b_0, \ldots, b_{t+1})$ such that
   - $a_0 = f_1(x, \widetilde{y})$, where $\widetilde{y} \in_U Y$.
   - b_0 is sampled independently such that $\Pr[b_0 = 1] = \mathbf{b}_0^{(0)}(y) = \mathbf{b}_0^{(1)}(y)$.
   - For every $i \in \{1, \ldots, t\}$, a_i and b_i are bits satisfying

$$\begin{cases} \Pr[a_i = 1] = \mathbf{a}_i^{(\alpha)}(x) \\ \Pr[b_i = 1] = \mathbf{b}_i^{(\beta)}(y) \end{cases}$$

   - $a_{t+1} = f_1(x, y)$ and $b_{t+1} = f_2(x, y)$.
3. The dealer hands a_0 to P_1 and b_0 to P_2.
4. For $i = 1, \ldots, t+1$,
   (a) The dealer gives a_i to P_1. If P_1 aborts, then P_2 outputs b_{i-1} and halts.
   (b) The dealer gives b_i to P_2. If P_2 aborts, then P_1 outputs a_i and halts.

[a] If x is not in the appropriate domain or P_1 does not hand an input, then the dealer sends $f(\hat{x}, y)$ (where $\hat{x}$ is a default value) to P_2, which outputs this value and the protocol is terminated. The case of an inappropriate y is dealt analogously.

Fig. 5. Protocol SecSamp$(\mathcal{T}_f)$ for computing f.

[5] In light of Proposition 5.6, we can construct vectors admitting the required expression.

Proof (Sketch). The fact that the protocol is passively secure is trivial. Regarding security against sampling attacks, notice that, at any given round, the adversary either knows the output or knows nothing about it (other than what the corrupted party's input suggests). The adversary will not be able to mount a successful sampling attack in neither case. If the output has not been revealed to her, then her view is independent of the honest party's output resulting from some locking strategy (regardless of whether she quits at that round or at the end). If the output has been revealed to the adversary, then sampling attacks are foiled by design thanks to the algorithm. □

When the algorithm fails. We turn our attention to functions for which the algorithm returns $Y^+ \neq \emptyset$ and $X^+ \neq \emptyset$. Semi-balanced functions fall under this category. By Cleve [7], protocols that satisfy both correctness *and* security against sampling attacks do not exist in the plain model. However, there are functions other than semi-balanced for which the algorithm fails. Unfortunately, we do not fully understand why that is the case and there appears to be a trade off between fairness and privacy. To illustrate, we show that a certain function that lies in the gap can be computed with fairness but not privacy.

We emphasize that the function in question may still be computable with full security. However, our previous analysis together with the theorem below strongly indicate that the trade-off may be inherent.

Theorem 5.8. *The function described by the matrices below admits a protocol that is fair-but-not-private.*

$$M^{(1,*)} = \begin{pmatrix} 1&1&1&1&0 \\ 0&1&0&1&1 \\ 1&1&1&1&1 \\ 0&0&1&0&1 \\ 1&0&0&0&1 \end{pmatrix}, \qquad M^{(*,1)} = \begin{pmatrix} 1&1&0&0&0 \\ 1&0&0&0&1 \\ 1&0&0&1&0 \\ 0&0&1&1&1 \\ 0&1&0&1&0 \end{pmatrix}.$$

Proof. Consider the following 2-round protocol defined by means of the backup outputs $\{a_i, b_i\}_{i=1\ldots2}$.

$a_0 = f_1(x, \widetilde{y})$ where $\widetilde{y} \in_U Y$ $\qquad$ $b_0 = b \in_U \{0,1\}$

If $y \in \{y_1, y_3, y_5\}$ $\qquad$ $a_1 = \begin{cases} a \in_U \{0,1\} & \text{if } x = x_1 \\ 0 & \text{if } x = x_5 \\ 1 & \text{otherwise} \end{cases}$ $\qquad$ $b_1 = f(x, y)$.

If $y \in \{y_2, y_4\}$ $\qquad$ $a_1 = \begin{cases} 0 & \text{if } x \in \{x_4, x_5\} \\ 1 & \text{otherwise} \end{cases}$

$a_2 = f(x, y)$ $\qquad$ $b_2 = f(x, y)$

A straightforward computation shows that the protocol is secure against sampling attacks. However, the protocol is obviously not passively secure. Notice that the backup output a_1 leaks information about P_2's input. Nevertheless, by plugging the protocol into our compiler, the resulting protocol satisfies fairness.

Formally, by having the trusted party leak the honest party's input to the simulator in the ideal model, one can show that the resulting protocol is secure with respect to the new model. □

6 Conclusions and Open Problems

In this paper, we introduced a notion of security referred to as *security against sampling attacks*. The notion of security is useful because it is necessary for fairness and it appears easier to achieve compared to fairness. What is more, we showed how certain protocols satisfying security against sampling attacks can be transformed into fully-secure protocols. We emphasize that the route towards full-security we propose is not arbitrary; every known protocol based on GHKL can be viewed as a special case of our approach. Finally, for asymmetric functions, we showed how to design suitable protocols by means of an algorithm. Given an asymmetric (possibly randomized) Boolean function, our algorithm either returns an appropriate protocol or it returns that it failed to do so. Unfortunately, our algorithm fails for functions other than semi-balanced, and the status of these functions is still unknown. We provide a few conjectures as to why that may be the case.

First, we believe that a failure on the part of the algorithm is essentially a proof of impossibility. In other words, we believe that if our algorithm fails to come up with an suitable protocol for some function, then any realization of the function is susceptible to some attack. At the same time, we believe that the attack in question cannot rely solely on sampling attacks, but on some combination of passive and sampling attacks. The motivation behind this belief is that we suspect certain functions to be computable with fairness-but-privacy[6] but not with full security. A candidate for such a function is given at the end of the last section.

The Multi-party Case. We note that, like [3], our analysis extends to the multi-party case where the total number of parties is constant and exactly half of the parties are corrupted. Specifically, if $f = (f_1, \ldots, f_t) : Z_1 \times \ldots \times Z_t \to [m]^t$ denotes a (possibly randomized) t-party function, there are $\binom{t}{t/2}$ two-party functions that result from partitioning the set into two equal-sized subsets. These functions can be viewed as non-Boolean asymmetric functions in $X \times Y \to [m^{t/2}]^2$. Using the techniques from [3,5], functionality f is fair if and only if all of the underlying two-party functions are fair as well. Thus, our framework is also useful in this regard.

Finally, our work says little about the multi-party case with absolute dishonest majorities as well as two-party and multi-party functionalities that depend on the security parameter $\mathcal{F} = \{f_n\}_{n \in \mathbb{N}}$. Of course, locking strategies and sampling attacks are still meaningful in these settings, and it would be interesting to see how they can be put to use.

[6] Of course, this notion needs to be formalized.

References

1. Agrawal, S., Prabhakaran, M.: On fair exchange, fair coins and fair sampling. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 259–276. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_15
2. Asharov, G.: Towards characterizing complete fairness in secure two-party computation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 291–316. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_13
3. Asharov, G., Beimel, A., Makriyannis, N., Omri, E.: Complete characterization of fairness in secure two-party computation of boolean functions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 199–228. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_10
4. Asharov, G., Lindell, Y., Rabin, T.: A full characterization of functions that imply fair coin tossing and ramifications to fairness. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 243–262. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_14
5. Beimel, A., Omri, E., Orlov, I.: Protocols for multiparty coin toss with a dishonest majority. J. Cryptol. **28**, 551–600 (2015)
6. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
7. Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: STOC, pp. 364–369 (1986)
8. Daza, V., Makriyannis, N.: Designing Fully Secure Protocols for Secure Two-Party Computation of Constant-Domain Functions. Cryptology ePrint Archive, Report 2017/098 (2017)
9. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. II. Cambridge University Press, Cambridge (2004)
10. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229 (1987)
11. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete fairness in secure two-party computation. J. ACM **58**, 24:1–24:37 (2011)
12. Gordon, S.D., Katz, J.: Complete fairness in multi-party computation without an honest majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 19–35. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_2
13. Makriyannis, N.: On the classification of finite boolean functions up to fairness. In: Proceeding of the Security and Cryptography for Networks Conference, pp. 135–154 (2014)
14. Yao, A.C.: Protocols for secure computations. In: FOCS, pp. 160–164 (1982)