Efficient Concurrent Covert Computation of String Equality and Set Intersection*

Chongwon Cho[1] **, Dana Dachman-Soled[2] ***, and Stanisław Jarecki[3] †

[1] Information and Systems Science Laboratory, HRL Laboratories
[2] Department of Computer Science, University of Maryland, College Park
[3] Department of Computer Science, University of California, Irvine

Abstract. The notion of covert computation, an enhanced form of secure multiparty computation, allows parties to jointly compute a function, while ensuring that *participating* parties cannot distinguish their counterparties from a random noise generator, until the end of the protocol, when the output of the function is revealed, if favorable to all parties. Previous works on covert computation achieved super-constant round protocols for general functionalities [16, 5], with efficiency at least linear in the size of the circuit representation of the computed function. Indeed, [9] showed that constant-round covert computation of any non-trivial functionality with black-box simulation is impossible in the plain model.

In this work we construct the first practical constant-round covert protocol for a non-trivial functionality, namely the set-intersection functionality, in the Random Oracle Model. Our construction demonstrates the usefulness of covert subprotocols as building blocks in constructing larger protocols: We show how to compile a *concurrently covert* protocol for a single-input functionality, e.g. string equality, into an efficient secure and covert protocol for a corresponding multi-input functionality, e.g. set intersection.

Our main contributions are summarized as follows:

- We upgrade the notion of covert computation of [5] to *concurrent* covert computation.
- We provide a general compiler that converts *concurrent covert* protocols for single-input functionalities to *concurrent covert* protocols for corresponding multi-input counterparts of these functionalities, at linear cost, in the Random Oracle Model.
- To demonstrate the usefulness of our compiler, we construct a concurrently covert string equality protocol and then apply our compiler to achieve a two-message *concurrent covert* protocol for Set Intersection (SI) with a linear cost in the Random Oracle Model.

* This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF award #CNS-1523467.

** A part of work was performed while visiting University of California, Irvine

*** Work supported in part by NSF CAREER award #CNS-1453045 and by a Ralph E. Powe Junior Faculty Enhancement Award.

† Work supported in part by NSF CAREER award #CNS-0747541.

1 Introduction

Steganography addresses a security question that is not usually considered in cryptography, namely how to make the very fact that a (secure) protocol is being executed, hidden from an eavesdropping adversary. Such hiding of a protocol instance is, in principle, possible if the public channels connecting the communicating parties are *steganographic* in the sense that they have intrinsic entropy. A protocol is steganographic, or *covert*, if its messages can be efficiently injected into such channels in a way that the resulting communication cannot be distinguished from the a priori behavior of these channels. A simple example of a steganographic channel is a *random channel*, which can be implemented e.g. using protocol nonces, random padding bits, lower bits of time stamps, and various other standard communication mechanisms which exhibit inherent entropy. Assuming such random communication channels, if protocol participants encode their protocol messages as binary strings which are indistinguishable from random, they can inject their out-going messages into the random channel, and interpret the information received on those channels as the protocol messages from other parties. The participants must synchronize the timing of using these channels, so they know which bits to interpret as protocol messages, but this can be public information, because the covertness of the protocol implies that the exchanged messages cannot be distinguished from the a priori behavior of these random channels.

Covert computation was formalized for the two-party setting by von Ahn, Hopper and Langford in [16] and in the multi-party setting by Chandran et al. in [5], as a protocol that lets the participants securely compute the desired functionality on their inputs, with the additional property that no *participating* party can distinguish the other participants from "random beacons" that send random binary strings of fixed length instead of proscribed protocol messages, until the end of the protocol, when the output of the function is revealed, if favorable to all parties. Both [16] and [5] show protocols for covert computation of any functionality which tolerates malicious adversaries, resp. in the two-party and the multi-party setting, but the costs of these protocols are linear in the size of the circuit representation of the computed function. Moreover, these protocols are not constant-round, and the subsequent work of [9] showed that this is a fundamental limitation on maliciously-secure covert computation (with black-box simulation) in the standard model, i.e., without access to trusted parameters or public keys. In a recent work, Jarecki [12] showed a constant-round covert *mutual-authentication* protocol, but that protocol satisfied only a game-based definition of an authentication problem. This leaves a natural open question whether useful two-party (or multi-party) tasks can be computed covertly in a more practical way, with constant-round protocols, in stronger but commonly assumed computational models, like the Random Oracle Model (ROM), or equivalently the Ideal Cipher Model [6, 10].

Our contributions: In this work we construct the first practical *two-message covert* protocol for the set-intersection functionality in the Ideal Cipher Model.

That is, two parties, where each party holds a private set of size n, compute the intersection of their private sets. If the two input sets do not have an intersection, then no party can tell apart the following two cases: (1) the other party did not participate in the protocol execution, and (2) the other party did participate but the intersection was empty. Towards this goal, our contribution is three-fold:

(1) We introduce an upgraded version of the covert computation definition of [5], *concurrent covert (C-covert) computation*. We provide a definition of C-covert computation that enjoys advantages over the "single-shot" definition of covert computation in [5] because multiple instances of such protocols can execute concurrently, and the covertness and security properties are assured for each protocol instance.

(2) We show that covert protocols can serve as useful *tools* in constructing secure (and covert) protocols. Namely, we exhibit a general compiler which converts a *covert* protocol (supporting a concurrent composability) for a single-input functionality, e.g., a String Equality Test (SEQ) functionality which takes two strings and outputs 1 if they are equal and 0 (or $\perp$) otherwise, into a *covert* protocol computing a corresponding multi-input functionality, e.g. which in the case of SEQ would be a Set Intersection (SI) functionality. Our compiler is instantiated in the Ideal Cipher Model (equivalently the Random Oracle Model [6, 10]) and it preserves the covertness and the round complexity of the underlying protocol for the single-input functionality, at the increase of the computational and bandwidth costs which is only linear in the number of inputs contributed by each party. (Technically, this compiler is slightly stronger because the covert protocol for the underlying single-input functionality must only satisfy a weaker version of the C-covert computation.) The construction of our compiler is rooted in the idea of the Index-Hiding Message Encoding (IHME) scheme of Manulis, Pinkas, and Poettering [14]. While the security of IHME scheme is defined in terms of a game-based definition, the security of our compiler is generalized and defined in terms of simulation-based security, while the instantiation is provided in the Ideal Cipher Model.

(3) To make this general compiler result more concrete, we show an example of a two-party single-input functionality for the SEQ functionality (here presented in a one-sided output version), which on a pair of inputs (x, y) outputs $(b, \perp)$ where $b = 1$ if $x = y$ and 0 otherwise. The two-party multi-input functionality corresponding to the SEQ functionality is a Set Intersection (SI) functionality which takes a pair of *vectors* $((x_1, ..., x_n), (y_1, ..., y_n))$ as its inputs, and outputs $((b_1, ..., b_n), \perp)$ where $b_i = 1$ iff there exists j s.t. $x_i = y_j$. We construct a C-covert protocol for SEQ, and by applying the above compiler we obtain a C-covert protocol for the Set Intersection (SI) functionality. Since the C-covert protocol we show for SEQ takes 2 rounds and $O(1)$ group exponentiations, the resulting C-covert Set Intersection protocol takes 2 rounds and performs $O(n)$ group exponentiations. This compares well to existing standard, i.e. *non-covert*, Set Intersection protocols, e.g. [13, 8]. Standard SI protocols have received lots of attention,, and in particular there are multiple solutions which trade off public

key operations for increased communication complexity, e.g. based on garbled circuits [11], Bloom Filters [7] or OT extensions [15] (see also the last reference for comparisons between various SI protocols). Still, we take our results as showing that covertness can be achieved for non-trivial functionalities of general interest, like the SI functionality, at the cost which is comparable to the non-covert protocols for the same functionality.

1.1 Technical Overview

Concurrent Covert Computation. We introduce a new notion of concurrent covert (C-covert) computation. Covert computation was first introduced by von Ahn et al. [16] in the two-party setting. Later, Chandran et al. [5] formulated the notion of covert multiparty computation based on the simulation paradigm. In this work, we initiate a study of composable covert computation by considering the case of of concurrent self-composition. We give a formal definition of concurrent covert computation, which provides a framework for arguing whether a protocol remains covert while many instances of this protocol are executed concurrently in the system. In particular, our notion of C-covert computation follows the framework of universal composability (UC) by Canetti [4] although our notion has a limitation on its composability property compared to the notion of UC. Still, the notion we define upgrades the covert ("one-shot") computation notion of Chandran et al. by enabling concurrent and parallel self-composition of a covert protocol. Such a composability guarantee is at the crux of our application which compiles single-input (weakly) C-covert protocol to a C-covert protocol for the corresponding multi-input functionality. We note that our focus here is on concurrent composability and not full universal composability (UC) because only the former notion is required by the single-input to multi-input compiler: Our compiler executes multiple instances of the covert protocol for the single-input functionality, and hence its security requires that the underlying covert protocol is self-composable.

Intermediate security notions. In the course of achieving concurrent covert security, we introduce a special class of functionalities that we call *indexed single-input* functionalities. Namely, we call a two-party functionality F indexed single-input (ISI) if there exists an *index function* I s.t. for all inputs (x, y) to F we have that $\mathsf{F}(x, y) = \perp$ if $I(x) \neq I(y)$. We also introduce an intermediate security notion for ISI functionalities, called Weakly Concurrent Covert (in short, wC-covert) computation, which is a relaxation of C-covert computation. The high-level insight for this relaxation is that the simulator is allowed to possess additional advice which enables the simulation to go through. This relaxed notion of C-covert is sufficient in our compiler because the compiler construction ensures that the simulator has access to this advice, and hence it suffices that the underlying covert protocol is simulatable given this advice.

From wC-covert single input protocol to C-covert multi-input protocol. We construct, in the Ideal Cipher Model, a compiler that converts any wC-covert protocol for ISI functionality to C-covert protocol for the indexed mult-input (IMI)

version of the same functionality, where the IMI functionality on inputs $\boldsymbol{x} = (x^1, ..., x^n)$ and $\boldsymbol{y} = (y^1, ..., y^n)$ is defined, in short, as an $n \times n$ execution of the underlying ISI functionality on pairs of *matching* inputs, i.e. pairs (x^i, y^j) s.t. $I(x^i) = I(y^j)$. The compiler builds on the compiler idea proposed by Manulis, Pinkas, and Poettering [14]. The compiler of [14] converts a particular protocol for single-input functionality, which in their case was a Secret Handshake protocol (see e.g. [1]), into a secure protocol for multi-input functionality, e.g. a multi-input version of a Secret Handshake, where each party puts a vector of credentials, which are then pair-wise matched by the functionality.

In this work, we give a general-purpose version of this compiler, where we show that *covertness* and *self-composability* are the crucial properties needed of the protocol for the single-input functionality to be compiled. And this shows, very interestingly, that covertness is not just an interesting goal in itself but also can be useful as a tool in building more efficient (e.g. linear time) two-party protocols for multi-input functionalities. We exemplify it with the construction of C-covert Set-Intersection protocol secure under Decisional Diffie-Hellman (DDH) assumption in the random oracle model, which uses $O(n)$ exponentiations and O(n polylog n) multiplications of group elements where n is the number of elements in the set contributed by each party. This compares quite well to the existing *non-covert* SI protocols (see the discussion of various SI protocols in [15], although that discussion concentrates on efficiency inthe honest-but-curious setting).

Organization. In Section 2, we introduce the notions of C-covert computation, the indexed single-input and multi-input functionalities (ISI and IMI), and the related notions such as wC-covert computation, which will be utilized by our ISI-to-IMI compiler. In Section 3 we present the construction of compiler which converts a wC-covert protocol for a single-input (ISI) functionality into a C-covert protocol for the multi-input (IMI) version of this functionality, in the Ideal Cipher Model. In Section 4, we present an application of this compiler by exhibiting a wC-covert two-message $O(1)$-exponentiations covert protocol for the SEQ functionality in the Random Oracle Model.

2 Preliminaries

2.1 The Ideal Cipher Model

The ideal cipher model is an idealized model of computation in which entities (i.e., parties) has a public accessible to a ideal (random) block cipher. Such ideal cipher is a block cipher indexed by a key which is a k-bit string (or a field element) s.t. each key k defines a random permutation on l-bit strings. All entities in the ideal cipher model can make encryption and decryption queries to the cipher by specifying its index. In this work, we denote an ideal block cipher by $\Psi_k : \{0,1\}^l \to \{0,1\}^l$ and its inverse by $\Psi_k^{-1} : \{0,1\}^l \to \{0,1\}^l$. Coron et al. [6, 10] showed that the ideal cipher model is equivalent to the Random Oracle Model (ROM), first formalized by Bellare and Rogaway [2]. Therefore,

all results in this work can be translated into the same results in the Random Oracle Model. Throughout this work, we use these two names interchangeably.

2.2 Concurrent Covert Computation

We provide the definition of *concurrent covert* computation (C-covert) for a given functionality. Our definition of C-covert computation follows the framework of Universally composability (UC) by Canetti [4] as well as the definition of stand-alone (i.e. "single-shot") covert computation given by Chandran et al. [5]. Note that we provide the definition of concurrent covert computation for the multi-party case but in the remainder of the paper we will concentrate solely on the two-party functionalities and protocols, leaving general multi-party protocols to future work. Even though our definition builds upon the UC framework, its composability guarantee is restricted to concurrent self-composition. The main reason for this restrictiveness is that the definition guarantees only self-composability of covert computation for *functions*, i.e. not for general reactive functionalities as in the case of standard UC definition of Canetti [4]. We make this definitional choice because it is alrady sufficient in many applications, as exemplified e.g. by the compiler construction we present in this paper. Moreover, composing functionally distinct covert protocols is a challenge. Consider for example a protocol Π formed as a composition of protocols Π_1 and Π_2, where protocol Π_1 runs Π_2 as a subroutine, and note that an adversary might discover the participation of honest parties in the protocol from the outputs of subroutine Π_2 before the completion of protocol Π. In this work we concentrate on concurrent covertness and leave establishment of the framework of fully UC covert computation for future work.

Intuitively, the differences between the concurrent covert notion for functionality F we define below and the standard notion of concurrent computation for F is that (1) F's inputs and outputs are extended to include a special sign $\perp$ designating non-participation; (2) F is restricted to output a non-participation symbol $\perp$ to each party if *any* of these parties contributed $\perp$ as its input; and (3) the real-world protocol of an honest party on the non-participation input $\perp$ is fixed as a "random beacon", i.e. a protocol which sends out random bitstrings of fixed length independently of the messages it receives.

The Ideal Model. The definition of the ideal model is the UC analogue of the ideal model of Chandran et al. [5], except that composability guarantees are restrictted to self-composition. The ideal process involves an ideal functionality $\mathcal{F}$, an ideal process adversary (simulator) Sim, an environment $\mathcal{Z}$ with input z, and a set of dummy parties $P_1, \ldots, P_n$. Parties may input a value $x \in \{0,1\}^k$ to the functionality or a special symbol $\perp$ to indicate that they do not participate in the protocol. Let $\overline{x}$ denote the vector of inputs (including $\perp$) of all parties.

Similarly to the stand-alone covert computation notion of [5], an ideal functionality $\mathcal{F}$ in the C-covert computation is defined by a pair of functions f, g, where $g : \{\{0,1\}^k \cup \{\perp\}\}^n \to \{0,1\}$ is a *favor function* where $g(\overline{x}) = 0$ if and only if $\overline{x}$ is either a non-favorable input (i.e. inputs on which parties want to hide

their participation, e.g. two distinct strings in the case $\mathcal{F}$ is SEQ) or a subset of parties set their inputs to $\perp$ (which indicates that those parties do not participates in the computation). The function $f : \{\{0,1\}^k \cup \{\perp\}\}^n \to \{\{0,1\}^k \cup \{\perp\}\}^n$ is the actual functionality to be jointly computed, and it is restricted so that $f(\overline{x}) = \overline{y} \in \{\{0,1\}^k\}^n$ if $g(\overline{x}) = 1$, and $f(\overline{x}) = \{\perp\}^n$ if $g(\overline{x}) = 0$. In other words, function f outputs non-bot outputs if and only if the output of g on the inputs is favorable. We note that g and f can be randomized functions, in which case functionality $\mathcal{F}$ picks the randomness which is appended to input $\overline{x}$ before g and f execute.

Let $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(k,z,r)$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary S and ideal functionality $\mathcal{F}$, on security parameter k, input z, and random input $r = r_{\mathcal{Z}}, r_{\mathsf{Sim}}, r_{\mathcal{F}}$ as described above. Let $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(k;z)$ denote the random variable describing $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(k,z,r)$ when r is uniformly chosen. We denote the distribution ensemble of $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(k;z)$ by $\{\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(k,z)\}_{k\in N;z\in\{0,1\}^*}$.

The Real Model. The definition of the real model is also the UC analogue of the real model of Chandran et al. [5]. It is as the real model in the standard UC security model, except that each honest party on the non-participation input $\perp$ is assumed to execute a "random beacon" protocol, i.e. to send out random bitstrings of lengths appropriate to a given protocol round. Let $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z,r)$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{A}$ and parties running protocol Π on security parameter k, input z, and random tapes $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, \ldots, r_n$ as described above. Let $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(k;z)$ denote the random variable describing $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z,r)$ when r is uniformly chosen. Similar to the notations in the ideal model, we denote the distribution ensemble of $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z,r)$ by $\{\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{F}}(k,z)\}_{k\in N;z\in\{0,1\}^*}$.

Definition 1. *Let $n \in N$. Let $\mathcal{F}$ be an ideal functionality and Π be an n-party protocol. We say that Π concurrently securely realizes $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an ideal-process adversary* Sim *such that for any environment $\mathcal{Z}$,*

$$\{\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(k,z)\}_{k\in N;z\in\{0,1\}^*} \overset{c}{\approx} \{\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in N;z\in\{0,1\}^*}.$$

2.3 Indexed functionalities

Below we define two special classes of functionalities, ISI and IMI, which specify syntactic requirements on the functionalities involved in the compiler described in Section 3. The first notion, of Indexed Single-Input (ISI) two-party functionality, is a syntactic constraint which makes such function subject to a compilation from a "single-input" to a "multi-input" functionality. The second notion, of Indexed Multi-Input (IMI) two-party functionality, describes the functionality that results from such compilation, as it is defined by the underlying ISI functionality and the numbers of inputs contributed by each party. Finally, in definition 4, we define a security requirement on a protocol for computing some ISI functionality F which is a *technical relaxation* of the C-covert notion of definition 1, and

which turns out to suffice for the compilation procedure described in Section 3 to produce a C-covert protocol for the IMI functionality corresponding to F.

Definition 2 (Indexed single-input two-party functionalities.). *F is said to be an indexed single-input (ISI) two-party functionality (over domain D), with an index function I, if on a pair of inputs $(x, y) \in \mathsf{D} \times \mathsf{D}$ it outputs $(\mathsf{out}_A, \mathsf{out}_B)$ where out_A and out_B are outputs to A and B respectively s.t. $\mathsf{out}_A = \mathsf{out}_B = \perp$ whenever $I(x) \neq I(y)$.*

Many natural functionalities are of the ISI type. The notion of an "index agreement" between parties' inputs appears natural especially in the case of functionalities which one would want to compute covertly. Note that the notion of covert computation for F involves an admission function g on inputs s.t. if $g(x, y) = 0$ then F outputs $\perp$ to all parties, in which case neither party can distinguish its counter-party from a random beacon. The notion of an index function I specializes this agreement function by requiring that $g(x, y) = 0$ whenever $I(x) \neq I(y)$. Consider the case of F being a PKI-based authentication policy verification, a Password Authenticated Key Exchange (PAKE), or a String Equality (SEQ) test. In each of these cases the inputs have to "match" for the function to return a positive output. In the case of PAKE and SEQ, the index function can be an identity, as both functionalities might want to return $\perp$ if $x \neq y$, while in the first case function I can output the hash of a public key, either the public key held by the verifier or the public key which issued the certificate held by the prover. Note that an ISI functionality models a computation where each party contributes a single such input, e.g. a string, a password, or a certificate, etc. Hence, a natural extension of any ISI functionality F is a multi-input version of this functionality, which we denote $\tilde{\mathsf{F}}$, where each party can input a vector of n such inputs, and $\tilde{\mathsf{F}}$ computes a pair-wise matching of these inputs (out of n^2 input pairs) and then runs F on input pairs which match successfully. In the following we present a relaxed version of such multi-input functionality where the number of inputs that a malicious party can enter into the functionality might deviate by some δ from the number of honest party's inputs.

Definition 3 (Indexed δ-relaxed multi-input two-party functionalities.). *Let D be the domain of inputs, I be a function defined on D, and F be an indexed single-input two-party functionality over D with an index function I. Let $\tilde{\mathsf{F}}$ be a two party functionality which for some integer $\delta \geq 0$, takes input $\boldsymbol{x} = (x^1, \ldots, x^{n_1})$ from party A, and input $\boldsymbol{y} = (y^1, \ldots, y^{n_2})$ from party B, where $n_1, n_2 \in [n, n + \delta]$ and $x^i, y^j \in \mathsf{D}$ for every $i \in [n_1]$ and $j \in [n_2]$.*

$\tilde{\mathsf{F}}$ is said to admit input $\boldsymbol{x}$ (resp. $\boldsymbol{y}$) if $I(x^i) \neq I(x^j)$ (resp. $I(y^i) \neq I(y^j)$) for all $i, j \in [n_1]$ (resp. $i, j \in [n_2]$). Then, $\tilde{\mathsf{F}}$ is said to be an indexed δ-relaxed multi-input *(IMI) two-party functionality corresponding to F if $\tilde{\mathsf{F}}(\boldsymbol{x}, \boldsymbol{y})$ computes its output as follows:*

1. *If $\tilde{\mathsf{F}}$ does not admit inputs $\boldsymbol{x}$ or $\boldsymbol{y}$, then it outputs $(\perp, \perp)$.*
2. *F computes output sets S_A and S_B as follows: It initializes S_A and S_B as empty sets, and then for each pair of inputs (x^i, y^j) for $(i, j) \in [n_1] \times [n_2]$*

s.t. $z = I(x^i) = I(y^j)$*, computes* $(\mathsf{out_A}, \mathsf{out_B})$ *as an output of* F *on* (x^i, y^j)*, and if* $\mathsf{out_A}$ *or* $\mathsf{out_B} \neq \perp$*, then it adds* $(z, \mathsf{out_A})$ *to* S_A *and* $(z, \mathsf{out_B})$ *to* S_B*. Note that this computation invokes* $O(n)$ *instances of* F *because if* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are admitted by* $\tilde{\mathsf{F}}$ *then there can be at most* $\min(n_1, n_2)$ *pairs* (x^i, y^j) *s.t.* $I(x^i) = I(y^j)$*. Finally, if* S_A *and* S_B *are nonempty, then* $\tilde{\mathsf{F}}$ *outputs* S_A *to party* A *and* S_B *to party* B*. Otherwise it outputs* $\perp$ *to both parties.*

When $\delta = 0$ where the size of inputs from both parties is equal, we simply call $\tilde{\mathsf{F}}$ an indexed multi-input two-party functionality corresponding to F.

2.4 Relaxed Covertness Notion for ISI Protocols

To utilize the full power of our ISI-to-IMI compiler construction we introduce a relaxed notion of C-covert security applicable to ISI functionalities, which we call "Weakly Concurrent Covert" (in short, wC-covert) security. The main difference from the definition of C-covert is that the simulator receives additional advice to simulate the view of environment. Namely, the simulator learns the index $I(x)$ (resp. $I(y)$) for an x (resp. y) input by an *honest* party A (resp. B). Intuitively, simulation of a protocol can only be easier if the simulator gets such advice on the honest party's input, and therefore a protocol that satisfies this relaxation is easier to achieve. (We will indeed see such construction in the Random Oracle Model (ROM) in Section 4.) The reason that we consider such a relaxed notion of C-covert security for a protocol computing an ISI functionality is that our compiler, shown in Section 3, compiling C-covert protocol for an ISI functionality F to a C-covert protocol for an IMI functionality $\tilde{\mathsf{F}}$ corresponding to F, is constructed in the ideal cipher model where each party encrypts its messages of an instance of the protocol for F such that if x is a party's input to an instance, then the party uses the ideal cipher $\Psi_a(\cdot)$ with key $a = I(x)$ to encode all messages belonging to the instance. The idea is that the simulator can embed a random output r for the honest party's ideal cipher queries. Only if the adversary then queries $\Psi_a^{-1}(r)$ will the simulator need to simulate the underlying message m of the protocol for F, using the *programmability* of the ideal cipher and the underlying simulator for F. Therefore, whenever the underlying simulator for F is instantiated, $a = I(x)$ is already known.

Definition 4 (wC-covert protocols for an ISI two-party functionality.). *Let* F *be an ISI two-party functionality and let* $\Pi = (A, B)$ *be a two-party protocol that realizes* F*. Let* x *be the input of honest party and let* x^* *be the input of corrupted party. Protocol* Π *is a* ρ*-round* wC-covert *implementation of* F *if* Π *is* C-covert *computation of* F *with the following additional conditions:*

1. (Additional Advice) *For all efficient environments* $\mathcal{Z}$ *and adversaries* $\mathcal{A}$*, there exists an PPT simulator* Sim *s.t. for all inputs* x *in the domain of inputs of* F*, the environment's output in the real world,* $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(k, z, r)$ *is indistinguishable from the environment's output in the ideal world,* $\mathsf{Ideal}_{\mathsf{F},\mathsf{Sim},\mathcal{Z}}(k, z, r)$*, where the way messages are passed between Z and honest ideal players is identical to the one of definition of concurrent covert computation in the Section*

2.2 except for the following change in the ideal world: When $\mathcal{Z}$ sends to an ideal honest party its input x to F, the security game gives an "advice" a to Sim where $a = I(x)$ if $x \neq \perp$ and $a = I(x^*)$ if $x = \perp$.

2. *Consider a malicious strategy for party A (resp. B), where A chooses $s \leftarrow \{0,1\}^t$ such that t is the bit length of the message sent by A in protocol Π and sends s to (honest) B (resp. A) as its message in the j-th round of protocol Π for some $j \in [\rho]$. Then, with probability $1 - \mathsf{neg}(k)$ over choice of s, Sim, given the additional advice a as in Condition 1, queries the ideal functionality F with $\perp$ (and so the ideal party outputs $\perp$). Furthermore, with probability $1 - \mathsf{neg}(k)$, over choice of s for A's (resp. B's) j-th round message as above, Sim's subsequent messages, conditioned on s, are uniformly distributed.*

3 Compiling single-input TPCs to multi-input TPCs

In this section, we present a compiler $\mathsf{Comp}(\Pi, n)$ which takes any $\mathsf{wC\text{-}covert}$ protocol Π for indexed single input two-party functionality F and converts it to a $\mathsf{C\text{-}covert}$ protocol $\tilde{\Pi}$ which securely implements the corresponding indexed multi-input two-party functionality $\tilde{\mathsf{F}}$. We first describe the compiler which results in a multi-input $\tilde{\mathsf{F}}$ which takes exactly n inputs from each party, and then we show how its efficiency can be improved if the resulting functionality $\tilde{\mathsf{F}}$ is relaxed to allow the dihonest parties to input $n + \delta$ inputs instead of n.

We first give some intuition for our compiler and the proof of security. For simple exposition in the following high-level intuition, we restrict ourselves to the case of the two-message protocols where each party sends a single message to each other (i.e., a single-round protocol). The formal construction of multi-round compiler is provided in Fig. 1. The very high-level intuition is that each party encodes n parallel messages (where n is the size of the party's input set) from n instantiations of the underlying protocol Π using an ideal cipher Ψ and a polynomial encoding. Specifically, A constructs a polynomial P_1^A such that for each input x^i of party A, $P_1^A(I(x^i)) = \Psi_{I(x^i)}(m_1^i)$ and sends P_1^A (i.e., its coefficients) to B as its message, where m_1^i is the corresponding first message of protocol Π. Due to the covertness of the underlying protocol Π, the party receiving the encoded message cannot tell which points of the polynomial were programmed. For each of its inputs y^i, party B recovers the value $m_1^i = \Psi^{-1}_{I(y^i)}(P_1^A(I(y^i))$ and uses it to compute the corresponding second message m_2^i of protocol Π. Then, B encodes these messages in a similar fashion using polynomial P_2^B. B sends P_2^B to A who similarly recovers its output values.

There are several important points about the proof of security:

- **Using the simulator for Π.** We note that the underlying simulator for Π, denoted by Sim_Π will be used to generate m_1^i when B is corrupt and m_2^i when A is corrupt. However, note that since the simulator Sim for the compiled protocol simulates the ideal cipher, Sim has the advantage that it can construct the polynomials P_1^A, P_2^B at random and then run the underlying simulator Sim_Π to generate m_1^i or m_2^i only when an inversion query is made

to the ideal cipher. The advantage of this is that now Sim_{Π} can be given some auxiliary information about the ideal input $I(x^i)$ or $I(y^i)$. Specifically, when an adversary queries Ψ_a^{-1}, the simulator for the compiled program knows that the underyling message should be either a random message (corresponding to the element not being in the party's set) or it should be a protocol message for Π, computed using input x^i or y^i such that $I(x^i) = a$ or $I(y^i) = a$. Note that obtaining this auxiliary information is exactly the relaxation on Sim_{Π} is formalized in item (1) of Definition 4.

– **Ensuring correctness.** We must account for the fact that a party may not query the ideal cipher but may simply embed a random message m in P_1^A or P_2^B with the hopes that it will be "valid". Specifically, in the case that a random message is embedded in P_2^B, we must ensure correctness. In other words, we must rule out the possibility that B embeds a random message, from which Sim_{Π} cannot extract a corresponding input y^i but which yields a valid output for the real party A. To address this issue, we assume that Π has the property that random messages will cause the other party to output $\perp$ with all but negligible probability. We note that this property of Π is formalized in item (2) of Definition 4.

See Figure 1 for the formal description of the compiler $\mathsf{Comp}(\Pi, n)$.

Theorem 1. *Let k be a security parameter. If Π is a wC-covert protocol for indexed single input two-party functionality F and $\Psi : \{0,1\}^* \to \{0,1\}^l$ is an ideal cipher with $l = n \cdot \omega(k)$, then $\mathsf{Comp}(\Pi)$ is a secure, C-covert protocol for indexed multi-input two-party functionality $\tilde{\mathsf{F}}$ corresponding to F, taking n inputs from each party.*

Remark 1. We note that our compiler does not require the parties to enter the same number of the inputs. An adversarial party's number of inputs to $Comp(\Pi, n)$ might be indeed differ from n even if the honest party's number of inputs is n. First consider the case that the number of inputs of an adversary is smaller than n. For this case, observe that the honest party given a degree-$n+1$ polynomial as a message from its adversary will extract only the messages m according to its own n indexes such that $I(x^i) = I(y^i)$ while automatically treating all the other messages as non-participating messages of adversary (even though the honest party does not notice it). For the case where adversary wants to enter inputs more than n encoded in a $n+1$-degree polynomial, we prove that no PPT adversary can do that if we choose appropriate parameters for random oracle. See the following Lemma 1.

Lemma 1. *Let k be a security parameter, let $q(k)$ be an arbitrary polynomial in k, and let $\delta \geq 1$ be a constant. Let $\mathcal{A}$ be any PPT adversary which runs in time $q(k)$, an arbitrary polynomial in k. Consider the following game $\mathsf{Game}_{q,m,n,k,\delta}$ between adversary $\mathcal{A}$ and challenger $\mathcal{C}$:*

1. *Repeat the following procedure q times:*
   (a) $\mathcal{A}$ chooses a field element a from a field $F = GF(2^m)$. $\mathcal{A}$ sends it to $\mathcal{C}$.

Compiler(Π, n)

Setup: Given a security parameter k, two parties A and B can utilize the following:

- a ρ-round protocol $\Pi = (A_\Pi, B_\Pi)$ for computing an indexed single-input two-party functionality F over domain D, with an index function $I : \mathsf{D} \to F$ where F is a field of order 2^k where A_Π and B_Π are message generators.
- an ideal cipher Ψ, and its inverse Ψ^{-1}, keyed by elements of F, where for all $v \in F$, $\Psi_v(\cdot) : \{0,1\}^l \to \{0,1\}^l$ and $\Psi_v^{-1}(\Psi_v(x)) = x$ for all $x \in \{0,1\}^l$.

Inputs: The inputs of parties A and B are respectively vectors $\boldsymbol{x} = \{x^1, x^2, \cdots, x^n\}$ and $\boldsymbol{y} = \{y^1, y^2, \cdots, y^n\}$ in D^n which are admitted by F, i.e. $I(x^i) \neq I(x^j)$ and $I(y^i) \neq I(y^j)$ for all $i \neq j$.

Protocol Execution: WLOG, party A first sends a message to party B which then responds back to A in each round.

To compute the messages, party A does the following:

For each j-th round such that $j \in [\rho]$,

- For each $i \in [n]$, do the following:
  - Choose random coins $r^i_{A,j}$.
  - Compute the message α^i_j of protocol Π on input x^i, protocol Π transcripts $\alpha^i_1, \beta^i_1, \cdots, \alpha^i_{i-1}, \beta^i_{i-1}$ and randomness $r^i_{A,1}, \cdots, r^i_{A,j}$ by using A_Π.
  - Query the ideal cipher $\Psi_{I(x^i)}(\cdot)$ to obtain $s^i_{A,j} = \Psi_{I(x^i)}(\alpha^i_j)$.
- Interpolate an n-degree polynomial P^A_j over F s.t. $P^A_j(0) = 0$ and $P^A_j(I(x^i)) = s^i_{A,j}$ for $i \in [n]$.
- Send P^A_j to B (i.e. a vector of coefficients of P^A_j or a vector of values of P^A_j on n fixed points).

To compute the messages (responses to A), party B does the following:

For each j-th round such that $j \in [\rho]$,

- For each $i \in [n]$, do the following:
  - Compute $\alpha^i_j = \Psi^{-1}_{I(y^i)}(P^A_j(I(y^i)))$.
  - Choose random coins $r^i_{B,j}$.
  - Compute the message β^i_j of protocol Π on input y^i, protocol Π transcripts $\alpha^i_1, \beta^i_1, \cdots, \alpha^i_{i-1}, \beta^i_{i-1}, \alpha^i_i$, randomness $r^i_{B,1}, \cdots, r^i_{B,j}$ by using B_Π.
  - Query the ideal cipher $\Psi_{I(y^i)}(\cdot)$ to obtain $s^i_{B,j} = \Psi_{I(y^i)}(\beta^i_j)$.
- Interpolate an n-degree polynomial P^B_j over F s.t. $P^B_j(0) = 0$ and $P^B_j(I(y^i)) = s^i_{B,j}$ for $i \in [n]$.
- Send P^B_j to A.

Output:

- For each $i \in [n]$, A does the following:
  - Compute $\alpha^i_\rho = \Psi^{-1}_{I(x^i)}(P^B_\rho(I(x^i)))$.
  - Choose random coins $r^i_{\rho+1}$.
  - Compute the i-th output $out^{I(x^i)}$ of protocol Π on input x^i, randomness $r^i_1, \cdots, r^i_{\rho+1}$, and protocol Π transcript $\alpha^i_1, \beta^i_1, \cdots, \alpha^i_\rho, \beta^i_\rho$ by using A_Π.
- Let S be the set of i's in $[n]$ such that $out^{I(x^i)} \neq \perp$.
- Output $S_{out} = \{(I(x^i), out^{I(x^i)}) \mid \text{ for } i \in S\}$ if $S \neq \emptyset$ and $\perp$ otherwise.
- B does the same as A by using B_Π on its own input and random coins.

Fig. 1. The compiler $\mathsf{Comp}(\Pi, n)$.

(b) $\mathcal{C}$ responses with s uniformly sampled from field F.

2. *Without loss of generality, assume that $\mathcal{A}$ chooses q distinct a's. Let T be the set of q pairs of (a, s) generated in the above procedure. $\mathcal{A}$ wins the game if there exists a degree n polynomial $f(x)$ such that there exist a subset $S \subset T$ where $|S| = n + \delta$ and $f(a) = s$ for all (a, s) in S.*

Then, for sufficiently large k, for all n and δ, if $m = n\omega(\log k)$, then for any PPT adversary $\mathcal{A}$ running in time q, $\mathcal{A}$ wins the above game except with negligible probability ϵ.

Proof. Towards contradiction, assume that the lemma is false. That is, there exists a PPT adversary $\mathcal{A}$ that runs in time $q = k^d$ for some constant d and wins the game with non-negligible probability: there exists some c such that

$$P := \mathsf{Pr}[\mathcal{A} \text{ wins } \mathsf{Game}_{q,m,n,k,\delta}] \geq \tfrac{1}{k^c}$$

where the probability is taken over the coin toss of adversary $\mathcal{A}$.

Then, we have

$$P = \binom{q}{n+\delta} \frac{1}{|F|^\delta} = \binom{q}{n+\delta} \frac{1}{2^{m\delta}} \leq \frac{q^{n+\delta}}{2^{m\delta}}.$$

This means that

$$\frac{1}{k^c} \leq \frac{q^{n+\delta}}{2^{m\delta}} \Rightarrow \frac{2^{m\delta}}{q^{n+\delta}} \leq k^c \Rightarrow m\delta - (n+\delta)\log q^{n+\delta} \leq c \log k$$
$$\Rightarrow \frac{m\delta}{\log k} - dn - d\delta \leq c \Rightarrow \frac{\omega(\log k) n\delta}{\log k} - dn - d\delta \leq c \Rightarrow n(\omega(1)) \leq c.$$

Therefore, this completes the proof as we have a contradiction. □

We provide the formal proof of Theorem 1 in the full version of this work due to the restriction of the space. One immediate consequence of Theorem 1 is that it compiles a wC-covert protocols for any ISI two-party functionality into a C-covert protocol for the same functionality in the ideal cipher model. That is, if we encode messages with a degree one polynomial (a linear function) vanishing at 0 where messages correspond to underlying weakly secure protocol Π for F, then the resulting compiled protocol $\mathsf{Comp}(\Pi)$ is a C-covert protocol for F.

Corollary 1. *If there is a wC-covert protocol for an ISI two-party functionality F then there is a C-covert protocol for F in the random oracle model.*

Improving efficiency by relaxing the functionality. We note that Theorem 1 and its security proof rely on the fact that ideal cipher Ψ maps protocol messages into a string of length $m = n \cdot \omega(k)$, so the efficiency of protocol degrades linearly in n. To improve the compilation efficiency we can break this dependency between m and n by allowing (corrupted) parties to encode more than n messages into a polynomial. In particular, we relaxed the requirement of compiler that each party must put n inputs by allowing the parties to put $n + \delta$ inputs for $\delta \geq 0$.

That is, if we allow $\delta = O(n)$ in the proof of Lemma 1 above, then it is easy to see $m = O(k)$, which is independent of the number of expected inputs. The simulator's strategy (provided in the full version) remains the same except that the simulator's time complexity increases by $O(n)$.

Theorem 2. *Let k be a security parameter and let $\delta = O(n)$. If Π is a wC-covert protocol for indexed single input two-party functionality F and $\Psi : \{0,1\}^* \to \{0,1\}^l$ is an ideal cipher with $l = O(k)$, then $\mathsf{Comp}(\Pi)$ is a secure, C-covert protocol for δ-relaxed indexed multi-input two-party functionality $\tilde{\mathsf{F}}$ corresponding to F, taking at most $n + \delta$ inputs from each party.*

4 Instantiation of wC-covert string equality protocol

In the following, we construct an efficient one-round C-covert set-intersection protocol in the random oracle model. Given the compiler presented in Section 3, the construction of wC-covert protocol for ISI two-party string equality protocol is sufficient for a C-covert set-intersection protocol.

At the very high-level, the main idea behind our construction is to utilize the Smooth Projective Hash Function (SPHF) for (Cramer-Shoup like, see [3] for more details) CCA-secure encryption defined in the Random Oracle Model. More specifically, let G be a cyclic group of prime q and H be a random oracle. Given a public key (g, h) where $h = g^\alpha$ for some $\alpha \in Z_q$, a party (called A) can encrypt its message p as $c = (g^r, h^r \cdot g^{H(p)})$ for some random $r \in Z_q$. Given the ciphertext c, another party (called B), if B possesses p, can extract a DDH tuple from the ciphertext c and create a hash value h and a projection key pk which is independent of message p. If A is given the projection key pk, then it may compute the same hash value h using its own witness r and pk. If B does not possess p, then B cannot extract a DDH tuple from the encryption and the hash value h becomes uniformly random in its range in the view of A even given projection key pk. For our wC-covert string equality protocol, we use this SPHF in both ways: from A to B and from B to A, where each direction checks if a party possess an identical string. The formal description of wC-covert string equality protocol is provided in Fig. 2.

Theorem 3. *Assume the DDH problem is hard in group G of order prime q and let H be a random oracle. Then, protocol Π is a one-round wC-covert protocol for the string-equality functionality.*

We prove Theorem 3 by proving two lemmas, Lemma 2 (correctness) and Lemma 3 (security). Due to the space constrain, we provide their proofs in the full version.

Lemma 2. *The protocol Π described in Fig. 2 is correct with overwhelming probability. That is, Π on input (p_A, p_B) outputs $(1, \perp)$ if and only if $p_A = p_B$ except with probability $1/2^q$.*

Lemma 3. *The protocol Π described in Fig. 2 is one-round wC-covert protocol as defined in Definition 4.*

Covert Protocol $\Pi = \langle A, B \rangle$

Setup: A cyclic group G of prime order q, generator g, $h = g^{\alpha}$ for random α in Z_q. $\mathsf{CRS} = (G, q, g, h)$ is published. Let H be a Random Oracle with outputs in Z_q. Random elements of G must be efficiently encodeable as random bitstrings of fixed length. (This is easy e.g. if G is a subgroup of residues mod p for $p-1 = q \cdot c$ s.t. $gcd(q, c) = 1$.)

Inputs: A has input p_A, B has input p_B.

Outputs: A outputs 1 if inputs p_A and p_B are equal. Otherwise, A outputs 0. B always outputs $\perp$. Inputs are favorable if $p_A = p_B$.

First Round:

- On input p_A, A does the following:
  1. Choose $e_A, d_A, r_A \leftarrow Z_q$.
  2. Set $\mathsf{pk}_A := g^{e_A} h^{d_A}$, $c_A^1 := g^{r_A}$, and $c_A^2 := h^{r_A} \cdot g^{H(p_A)}$.
  3. Send $(c_A^1, c_A^2, \mathsf{pk}_A)$ to B.

Second Round:

- B does the following:
  1. Choose $e_B, d_B, r_B \leftarrow Z_q$.
  2. Set $\mathsf{pk}_B := g^{e_B} h^{d_B}$, $c_B^1 := g^{r_B}$, and $c_B^2 := h^{r_B} \cdot g^{H(p_B)}$).
  3. Compute $k_{B1} := H((c_A^1)^{e_B} \cdot (c_A^2 / g^{H(p_B)})^{d_B})$, $k_{B2} := H((\mathsf{pk}_A)^{r_B})$. Send $(c_B^1, c_B^2, \mathsf{pk}_B, k_B = k_{B1} \oplus k_{B2})$ to A.

Output:

- A computes $k_{A1} := H((\mathsf{pk}_B)^{r_A})$ and $k_{A_2} := H((c_B^1)^{e_A} \cdot (c_B^2 / g^{H(p_A)})^{d_A})$.
- A outputs 1 if $k_{A1} \oplus k_{A2} = k_B$, and 0 otherwise.

Fig. 2. A simple covert protocol Π for string equality functionality

Combining Theorem 1 (resp. Theorem 2) with Theorem 3 immediately yields two-pass C-covert set-intersection protocol (resp. with δ-relaxation). For the completeness, we provide the formal corollary as follows.

Corollary 2. *Let k be a security parameter and let $\delta = 0$ (resp. $\delta = O(n)$). If Π is a wC-covert protocol for indexed single input two-party string equality functionality F and $\Psi : \{0,1\}^* \to \{0,1\}^l$ is an ideal cipher with $l = O(n\omega(\log k))$ (resp. $l = O(k)$), then $\mathsf{Comp}(\Pi, n)$ is a one-round C-covert protocol for (resp. δ-) relaxed set-intersection two-party functionality, taking a set of n (resp. at most $n + \delta$) elements from each party.*

References

1. D. Balfanz, G. Durfee, N. Shankar, D.K. Smetters, J. Staddon, and H.C. Wong. Secret handshakes from pairing-based key agreements. In *IEEE Symposium on Security and Privacy*, 2003.
2. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

3. Fabrice Benhamouda, Olivier Blazy, Cline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for sphfs and efficient one-round pake protocols. In *In CRYPTO 13*, pages 449–475. Springer-Verlag, 2013.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA, 2001. IEEE Computer Society.
5. Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multiparty computation. In *FOCS*, pages 238–248, 2007.
6. Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 1–20, 2008.
7. C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *Computer and Communications Security (CCS)*, page 789800, 2013.
8. Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, pages 1–41, 2014.
9. Vipul Goyal and Abhishek Jain. On the round complexity of covert computation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 191–200, New York, NY, USA, 2010. ACM.
10. Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 89–98, 2011.
11. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security (NDSS)*, 2012.
12. Stanislaw Jarecki. Practical covert authentication. In Hugo Krawczyk, editor, *Public-Key Cryptography PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 611–629. Springer Berlin Heidelberg, 2014.
13. Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, pages 418–435, 2010.
14. Mark Manulis, Benny Pinkas, and Bertram Poettering. Privacy-preserving group discovery with linear complexity. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, pages 420–437, Berlin, Heidelberg, 2010. Springer-Verlag.
15. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 797–812. USENIX Association, 2014.
16. Luis von Ahn, Nicholas Hopper, and John Langford. Covert two-party computation. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 513–522, New York, NY, USA, 2005. ACM.