Secure two-party computation in applied pi-calculus: models and verification

Sergiu Bursuc

School of Computer Science, University of Bristol, UK
s.bursuc@bristol.ac.uk

Abstract

Secure two-party computation allows two mutually distrusting parties to compute a function together, without revealing their secret inputs to each other. Traditionally, the security properties desired in this context, and the corresponding security proofs, are based on a notion of simulation, which can be symbolic or computational. Either way, the proofs of security are intricate, requiring first to find a simulator, and then to prove a notion of indistinguishability.

Furthermore, even for classic protocols such as Yao's (based on garbled circuits and oblivious transfer), we do not have adequate symbolic models for cryptographic primitives and protocol roles, that can form the basis for automated security proofs. We therefore propose new models in applied pi-calculus in order to address these gaps. Our contributions, formulated in the context of Yao's protocol, include:

- an equational theory for specifying the primitives of garbled computation and oblivious transfer;
- process specifications for the roles of the two parties in Yao's protocol;
- definitions of security that are more clear and direct: result integrity, input agreement (both based on correspondence assertions) and input privacy (based on observational equivalence).

We put these models together and illustrate their use with ProVerif, providing a first automated verification of security for Yao's two-party computation protocol.

1 Introduction

Secure two-party computation is a fundamental problem in cryptography [1]: two parties with inputs a and b wish to compute a function $f(a, b)$ such that each party can both preserve the privacy of its inputs and be sure to receive the correct result of the computation. Even more, each party would like assurance that the other party does not learn more from the protocol, like the evaluation of the function f on other inputs, e.g. $f(a', b')$, or the evaluation of another function on the same inputs, e.g. $g(a, b)$.

A classic, and still most efficient, way of achieving secure two-party computation is Yao's protocol based on garbled circuits [2]. It allows two parties to exchange a garbled circuit and garbled inputs for a function, that can be used to compute the corresponding output, without leaking private information about inputs. In addition, cut-and-choose techniques or zero-knowledge proofs can be incorporated into this protocol to ensure that a malicious party cannot cheat, this way learning more than it should or leading to compromized function outputs [3, 4].

Security proofs in computational models. The active security of Yao's protocol has been defined and proved in the simulation-based model [5, 6, 3], which states that, by executing a two-party computation protocol for a function f, an attacker can obtain essentially nothing more than the output of the function. First, this requires the definition of an ideal model where the desired functionality can be securely computed in a trivial

This work has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO.

manner, for instance relying on a trusted third party and private channels. Secondly, one has to show that the view of every attacker on the real protocol can be matched by a computationally indistinguishable view that comes from the idealized model. This requires a simulator, whose role is to decorate an ideal run with innocous data that makes it look like a real run to any polynomially bounded adversary. This level of generality comes however at a cost, the security proofs being complex and challenging to automate.

Security proofs in symbolic models. On the other hand, significant progress has been made in the field of automated verification of security protocols in formal (or symbolic) models [7, 8]. However, even symbolic definitions of simulation-based security, e.g. [9, 10] or [11, 12] (in applied pi-calculus), are still a challenging task for such methods, which are tailored for basic properties like secrecy, authentication or privacy. Indeed, recent work aiming to automate verification for multi-party computation protocols is either relying on additional manual input [13, 12], or only captures properties of correctness [14]. For Yao's protocol in particular, we also lack symbolic models for the required cryptographic primitives of garbled computation and oblivious transfer. Overall, we do not yet have the models that could be given directly to a verification tool and ask the basic question: is a particular two-party computation protocol secure or not? We propose such models for Yao's protocol.

Our approach and contributions. The main challenge in automating simulation-based security proofs comes from the fact that a simulator first needs to be found, and, for some methods (e.g. [13, 12]), processes need to be rearranged to have the same structure in order to check indistinguishability - this requires some human input in order to be tractable by tools. In this paper, we propose an alternative approach, formulating two-party computation security for Yao's protocol as a conjunction of three basic properties: result integrity, input agreement and input privacy (Section 5). They are based on the standard symbolic notions of correspondence assertions and observational equivalence (of two processes with the same structure), do not require a simulator, and are directly amenable to automation. We also propose formal models in applied pi-calculus for the cryptographic primitives (Section 3) and the processes (Section 4) of Yao's two-party computation protocol. We show that our models can be combined and verified with ProVerif, deriving a first automated proof of security for Yao's protocol.

Relations among notions. Computational soundness results in [9, 10, 14, 13] show that it is sufficient to prove security in the symbolic model, in order to derive security guarantees in the corresponding computational model. The models in [11, 12] have not yet been shown to be computationally sound, to our knowledge. Our models are related to [11, 12, 14, 13], being formulated in the same language of applied pi-calculus. In future work, we aim to show an even stronger relation, deriving conditions under which our properties imply, or not, simulation-based security in these formal models. We discuss this open problem and related work in more detail in Section 6.

2 Preliminaries

2.1 Secure two-party computation with garbled circuits

Assume two parties $\mathcal{A}$ (with secret input x) and $\mathcal{B}$ (with secret input y) want to compute $f(x, y)$, for a function f. The basic tool in Yao's two-party computation protocol [2, 6] is a garbling construction that can be applied to any circuit representing the function f. For a fresh key k, it generates a garbled circuit $GF(f, k)$ and garbled input wires $GW(x, k, a)$, $GW(y, k, b)$, where a and b mark the circuit wires corresponding to the input of $\mathcal{A}$ or $\mathcal{B}$. Then,

- the output of the circuit $GF(f,k)$ on inputs $GW(x,k,a)$, $GW(x,k,b)$ is equal to $f(x,y)$, as depicted in the left part of Figure 1.
- without access to the key k, $f(x,y)$ is the only meaningful information that can be derived from $GF(f,k)$, $GW(x,k,a)$, $GW(y,k,b)$. In particular, the values x and y remain secret and, for any $\{x',y'\} \neq \{x,y\}$, these garbled values do not allow to compute $f(x',y')$.

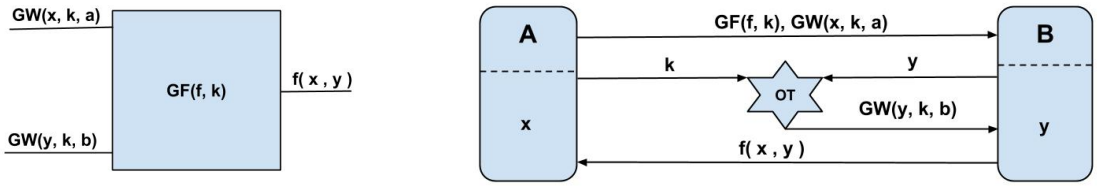

Figure 1 Garbled computation and Yao's protocol for two parties

Relying on garbling, one of the two parties, say $\mathcal{A}$, can play the role of a sender and the other party, say $\mathcal{B}$, can play the role of a receiver. The role of the sender, as depicted in the right part of Figure 1, is to garble the circuit and the inputs of the two parties. The role of the receiver is to execute the garbled computation and send the result back to $\mathcal{A}$. Note, however, that the party $\mathcal{A}$ does not have access to the private input of $\mathcal{B}$, so we need another tool to ensure that $\mathcal{A}$ and $\mathcal{B}$ can agree on a garbled input for $\mathcal{B}$.

This is where $\mathcal{A}$ and $\mathcal{B}$ rely on oblivious transfer [15, 16]. An oblivious transfer protocol allows a receiver to obtain a message from a set computed by the sender such that: (i) only one message can be received and (ii) the sender does not know which message has been chosen by the receiver. In Yao's protocol, the receiver $\mathcal{B}$ can then get one message, which is the garbling of his desired input for the function, and nothing else, whereas the sender $\mathcal{A}$ does not learn what value $\mathcal{B}$ has chosen as input. Having obtained $GF(f,k)$, $GW(x,k,a)$ and $GW(y,k,b)$, $\mathcal{B}$ can evaluate the garbled circuit and obtain $f(x,y)$, which can be sent back to $\mathcal{A}$ as the result of the computation.

Active security. In the case when $\mathcal{B}$ might be malicious, we have to ensure that $\mathcal{A}$ can obtain from $\mathcal{B}$ the correct result. For this, the functionality of the garbled circuit is modified such that its output is a pair of values $f(x,y)$ and $enc(f(x,y),k)$, where k is a fresh secret key chosen by $\mathcal{A}$ for each protocol session. Then, instead of $f(x,y)$, $\mathcal{B}$ returns $enc(f(x,y),k)$ to $\mathcal{A}$: the result $f(x,y)$ is authenticated by the key k. To counter the case of a malicious $\mathcal{A}$, the sender $\mathcal{A}$ can prove that the garbling is correct, relying on cut-and-choose techniques [3, 17] or zero-knowledge proofs [18, 4].

2.2 Applied pi-calculus and ProVerif [19, 20, 21, 22, 23]

Equational theories. We assume given an infinite set of *names*, $a,b,c,k,n\ldots$, an infinite set of *variables*, $x,y,z,\ldots$ and a *signature* $\mathcal{F}$ formed of a set of *constants* and *function symbols*. Names, constants and variables are basic *terms* and new terms can be built by applying a function symbol $f \in \mathcal{F}$ to already defined terms. The signature $\mathcal{F}$ can be split into *public* and *private* symbols: $\mathcal{F} = \mathcal{F}^{pub} \cup \mathcal{F}^{priv}, \mathcal{F}^{pub} \cap \mathcal{F}^{priv} = \emptyset$. A *substitution* σ is a partial function from the set of variables to the set of terms, whose application to a term T is the term $T\sigma$, called an instance of T, obtained by replacing every variable x of T with the corresponding term $x\sigma$. A *term context* is a term $\mathcal{C}[\_1,\ldots,\_n]$ containing special constants $\_1,\ldots,\_n$ (also called holes). For a context $\mathcal{C}[\_1,\ldots,\_n]$ and a sequence

of terms $T_1, \ldots, T_n$, we denote by $\mathcal{C}[T_1, \ldots, T_n]$ the term obtained by replacing each $\_i$ with the corresponding T_i in $\mathcal{C}$. A context with a single hole is denoted by $\mathcal{C}[\_]$.

The semantics of terms is given by a set $\mathcal{R}$ of rewrite rules of the form $U \to V$, where U, V are terms with variables. Then, a term T_1 rewrites to T_2 in one step, denoted by $T_1 \to T_2$, if there is a context $\mathcal{C}[\_]$, a substitution σ and a rewrite rule $U \to V$ such that $T_1 = \mathcal{C}[U\sigma]$ and $T_2 = \mathcal{C}[V\sigma]$. More generally, T_1 rewrites to T_2, denoted by $T_1 \to^* T_2$, if T_2 can be derived from T_1 by applying zero or more rewrite steps [24, 25]. A pair $\mathcal{E} = (\mathcal{F}, \mathcal{R})$ formed of a signature and a set of rewrite rules is called an equational theory. We assume equational theories that are convergent: for any term T, there is a unique term $T{\downarrow}$ such that $T \to^* T{\downarrow}$ and no rewrite steps can be applied to $T{\downarrow}$. We write $U =_{\mathcal{E}} V$ if $U{\downarrow} = V{\downarrow}$. We say that a term T can be deduced from a sequence of terms S, denoted by $S \vdash_{\mathcal{E}} T$ (or simply $S \vdash T$), if and only if there is a context $\mathcal{C}[\__1, \ldots, \__n]$ and terms $T_1, \ldots, T_n$ in S such that $\mathcal{C}[T_1, \ldots, T_n]{\downarrow} = T$ and $\mathcal{C}$ does not contain function symbols in $\mathcal{F}^{priv}$. Such a context, together with the positions of terms $T_1, \ldots, T_n$ in S, is called a proof of $S \vdash_{\mathcal{E}} T$, which may admit several different proofs, denoted by annotations of π.

Processes. Processes of the calculus are built according to the following grammar

$P, Q, R ::=$	processes			
0		null process	$P \mid Q$	parallel composition
$!P$		replication	$new\, n; P$	name restriction
$in(c, x); P$		input on channel	$out(c, T); P$	output on channel
$if\ U = V\ then\ P\ else\ Q$		conditional	$let\ x = T\ in\ P$	term evaluation
$event\ T; P$		event occurence		

where c, n are names, x is a variable, T, U, V are terms. Replication allows the creation of any number of instances of a process. Names introduced by *new* are called *private*, or *fresh*, otherwise they are *public*, or *free*. In input/output actions, the name c represents the communication channel, the variable x stands for the message to be received, and the term T is the message sent over the channel. The construct *event* T is used for specifying security properties, recording that some event has occured. The term T is usually of the form $\mathcal{A}(T_1, \ldots, T_n)$, where $\mathcal{A}$ is a special symbol representing the name of the event (e.g. the start or end of a protocol session), while the terms $T_1, \ldots, T_n$ represent the parameters of the event (e.g. the names or inputs of parties).

A variable x is *free* in a process P if P does not contain x in any of its input actions and any term evaluation of the form $x = T$. A process P with free variables $x_1, \ldots, x_n$ is denoted by $P(x_1, \ldots, x_n)$, i.e. $x_1, \ldots, x_n$ are parameters of P that will be instantiated in the context where P is used. We denote by $sig(P)$ the set of function symbols that appear in P. A *process context* $\mathcal{C}[\_]$ is defined similarly as a term context.

Formally, the operational semantics of processes is defined as a relation on tuples of the form $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P})$, called configurations, whose elements represent the following information in the execution of a process: $\mathcal{N}$ is the set of freshly generated names; $\mathcal{M}$ is the sequence of terms output on public channels (i.e. to the attacker); $\mathcal{L}$ is the set of occured events; $\mathcal{P}$ is the multiset of processes being executed in parallel. The rules that define the operational semantics, presented in the appendix, are quite standard and correspond to the informal meaning previously discussed. We denote by $P \to^* (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P})$ if the configuration $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P})$ can be reached from the initial configuration of P, which is $(\emptyset, \emptyset, \emptyset, \{P\})$.

Security properties. We rely on both *correspondence assertions* [21] and *observational equivalence* [22]. Correspondence assertions allow to specify that some events in the execution of the protocol satisfy certain constraints. They are based on formulas whose syntax

and semantics, for a configuration $\mathcal{C} = (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P})$ and equational theory $\mathcal{E}$, are defined below:

Syntax	Φ, Ψ	$::=$	$ev : T$	$att : T$	$U = V$	$\Phi \wedge \Psi$	$\Phi \vee \Psi$	$\neg\Phi$
Semantics	$\mathcal{C} \models_{\mathcal{E}} ev : T \iff \exists T' \in \mathcal{L}.\, T' =_{\mathcal{E}} T$			$\mathcal{C} \models_{\mathcal{E}} att : T \iff \mathcal{M} \vdash_{\mathcal{E}} T$			
	$\mathcal{C} \models_{\mathcal{E}} U = V \iff U =_{\mathcal{E}} V$						

Therefore, $ev : T$ is true for a configuration if the event T occured in the execution trace leading to it, and $att : T$ is true if the attacker can deduce T from the public messages of the configuration. Formulas $\Phi \wedge \Psi, \Phi \vee \Psi$ and $\neg\Phi$ represent boolean functions that are satisfied as usual. More generally, a *correspondence assertion* is a formula of the form $\Phi \rightsquigarrow \Psi$. Such a formula is satisfied for a process P if and only if, for every process Q, with $sig(Q) \cap \mathcal{F}^{priv} = \emptyset$, and every configuration $\mathcal{C}$ reachable from $P \mid Q$, i.e. $P \mid Q \to^* \mathcal{C}$, and any substition σ, we have that $\mathcal{C} \models \Phi\sigma$ implies $\mathcal{C} \models \Psi\sigma'$, for some substition σ' that extends σ, i.e. if $x\sigma$ is defined, then $x\sigma' = x\sigma$. Intuitively, a correspondence assertion requires that every time the formula Φ is true during the execution of a process, the constraints specified in Ψ must also be true for the same parameters. The process Q stands for any computation that may be performed by the attacker.

Observational equivalence, denoted by $P_1 \sim P_2$, specifies the inability of the attacker to distinguish between two processes P_1 and P_2. Formally, $P_1 \sim P_2$ is true if and only if, for every process Q, with $sig(Q) \cap \mathcal{F}^{priv} = \emptyset$, and every configuration $(\mathcal{N}_1, \mathcal{M}_1, \mathcal{L}_1, \mathcal{P}_1)$ reachable from $P_1 \mid Q$, there is a configuration $(\mathcal{N}_2, \mathcal{M}_2, \mathcal{L}_2, \mathcal{P}_2)$ reachable from $P_2 \mid Q$, such that for any term T_1 and any two different proofs π_1, π_2 of $\mathcal{M}_1 \vdash_{\mathcal{E}} T_1$, there is a term T_2 such that π_1, π_2 are also proofs of $\mathcal{M}_2 \vdash_{\mathcal{E}} T_2$. There are different flavours of this definition, which are essentially equivalent for the class of processes that we consider [19, 22, 26, 23].

3 Equational theory for garbled computation

In this section we present an equational theory to model the cryptographic primitives used in garbled computation protocols like [2, 6, 3]. We will refer to a party $\mathcal{A}$ as *the sender* (who garbles and transmits data), and to a party $\mathcal{B}$ as *the receiver* (who receives and ungarbles data). The equational theory should enable: $\mathcal{B}$ to evaluate a garbled circuit on garbled inputs; $\mathcal{A}$ to prove that the circuits and its inputs are correctly garbled; $\mathcal{B}$ to obtain by oblivious transfer $\mathcal{B}$'s garbled input. We propose for this purpose the equational theory $\mathcal{E}_{GC}$ from Figure 2, and we discuss next how it captures the desired cryptographic properties.

Garbled circuit evaluation.

- the term $eval(T_{\mathcal{F}}, T_{\mathcal{A}}, T_{\mathcal{B}})$ represents the result of evaluating a circuit, represented by the term $T_{\mathcal{F}}$, on inputs of $\mathcal{A}$ and $\mathcal{B}$, represented by terms $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$ respectively.
- the term $gf(T_{\mathcal{F}}, T_{\mathcal{K}})$ represents the garbling of a circuit $T_{\mathcal{F}}$, given a garbling key $T_{\mathcal{K}}$.
- the term $gw(T, T_{\mathcal{K}}, i)$, with $i \in \{a, b\}$, represents a garbling of the input T with a key $T_{\mathcal{K}}$, where T corresponds to the input wires of party $\mathcal{A}$, when i is a, or of party $\mathcal{B}$, when i is b.
- the term $geval(gf(T_{\mathcal{F}}, T_{\mathcal{K}}), gw(T_{\mathcal{A}}, T_{\mathcal{K}}, a), gw(T_{\mathcal{B}}, T_{\mathcal{K}}, b))$ represents the computation performed on the garbled function and garbled inputs given as arguments to $geval$, the result of which is $eval(T_{\mathcal{F}}, T_{\mathcal{A}}, T_{\mathcal{B}})$, as specified by the rewrite rule $\mathcal{R}_1$.

Moreover, we have an additional function $geval'$, whose rewrite rule $\mathcal{R}_2$ provides an encryption of the evaluated function. As explained in section 2.1, this ciphertext is to be sent as response to $\mathcal{A}$, and it allows $\mathcal{A}$ to have confidence that the final result correctly

Figure 2 Equational theory $\mathcal{E}_{GC}$ for garbled computation

$$\mathcal{F} = \begin{array}{|cccccccc|} \hline eval/3 & gf/2 & gw/3 & a/0 & b/0 & geval/3 & geval'/3 & ok/0 \\ ungarb/2 & checkf/2 & gwot/3 & com/2 & get/3 & enc/2 & dec/2 & \\ \hline \end{array}$$

$$\mathcal{R} = \begin{array}{|lrcl|} \hline 1. & geval(gf(x,y), gw(x_1,y,a), gw(x_2,y,b)) & \to & eval(x,x_1,x_2) \\ 2. & geval'(gf(x,y), gw(x_1,y,a), gw(x_2,y,b)) & \to & enc(eval(x,x_1,x_2),y) \\ 3. & ungarb(gw(x,y,z),y) & \to & x \\ 4. & ungarb(gf(x,y),y) & \to & x \\ 5. & checkf(gf(x,y),x) & \to & ok \\ 6. & get(gwot(com(x,z),y_1,y_2),x,z) & \to & gw(x,y_1,y_2) \\ 7. & dec(enc(x,y),y) & \to & x \\ \hline \end{array}$$

reflects his inputs in the protocol, even while interacting with a malicious $\mathcal{B}$. For brevity, we have considered that the key in this encryption is the same as the one used for garbling, but the model can be easily adapted for more complex scenarios.

Overall, $\mathcal{R}_1$ and $\mathcal{R}_2$ are the only operations that can be performed on garbled values without the garbling key, this way enforcing several security properties at the cryptographic level, which are crucial for deriving the security properties at the process level that we specify in Section 5. First, the function and the inputs of the garbled circuit cannot be modified. Second, the computation in rules $\mathcal{R}_1, \mathcal{R}_2$ succeeds only for circuits and inputs that are garbled with the same key y (otherwise, a malicious party may combine garbled values from different sessions of the protocol in order to derive more information than it should). Third, the inputs must be used consistently, e.g. the garbled input of $\mathcal{A}$ cannot be substituted with a garbled input for $\mathcal{B}$ (ensured by the constants a and b). Garbled data can only be ungarbled by the key holder, as specified by the rule $\mathcal{R}_3$ for garbled functions and the rule $\mathcal{R}_4$ for garbled inputs.

We have seen that several protections are in place in order to ensure that a malicious receiver cannot cheat. In addition, we need to ensure that a malicious sender cannot cheat when garbling a circuit. This is the role of $\mathcal{R}_5$, which allows a party to check that a function is correctly garbled, without access to the garbling key. Cryptographically, there are various ways in which this abstraction can be instantiated, e.g. by zero-knowledge proofs [4] or cut-and-choose techniques [3, 27]. The model of oblivious transfer that we explain next will also allow the receiver to be convinced that his input is correctly garbled.

Garbled oblivious transfer is modeled relying on functions $gwot, get, com$, and the rewrite rule $\mathcal{R}_6$, as follows:

- the term $com(T_{\mathcal{B}}, V)$ represents a commitment to a term $T_{\mathcal{B}}$, which cannot be modified, and is hidden by a nonce V; such a term will be used by $\mathcal{B}$ to request a garbled version of $T_{\mathcal{B}}$ without disclosing it.
- the term $gwot(com(T_{\mathcal{B}}, V), T_{\mathcal{K}}, T)$ is an oblivious transfer term, obtained from a commited input and a garbling key $T_{\mathcal{K}}$; such a term will be constructed by $\mathcal{A}$ and sent in response to $\mathcal{B}$'s commitment.
- the term $get(gwot(com(T_{\mathcal{B}}, V), T_{\mathcal{K}}, T), T_{\mathcal{B}}, V)$ allows to obtain $gw(T_{\mathcal{B}}, T_{\mathcal{K}}, T)$ from an oblivious transfer term, if a party has the secret input $T_{\mathcal{B}}$ and nonce V that have been used to construct the corresponding commitment, relying on the rule $\mathcal{R}_6$.

This way we express an oblivious transfer of garbled values at an abstract level, capturing

formally the security properties that are ensured by a concrete instantiation of the oblivious transfer protocol (e.g. [15, 16, 27, 28]): for a sender $\mathcal{A}$ and a receiver $\mathcal{B}$:

- $\mathcal{B}$ should only learn one garbled value among many possible ones
- $\mathcal{A}$ should not learn which value $\mathcal{B}$ has chosen

The first property is ensured in our model by the fact that a dishonest $\mathcal{B}$ cannot change the commitment $com(T_{\mathcal{B}}, V)$ in an oblivious transfer term $gwot(com(T_{\mathcal{B}}, V), T_{\mathcal{K}}, T)$. The only way to obtain a garbling of a second message would be to run a second instance of the protocol with $\mathcal{A}$, involving another commitment and corresponding oblivious transfer term - this is a legitimate behaviour that is also allowed by our model. The second property is ensured by the fact that a commitment $com(T_{\mathcal{B}}, V)$ does not reveal $T_{\mathcal{B}}$ or V. Furthermore, only the holder of $T_{\mathcal{B}}$ and V can extract the respective garbled value from an oblivious transfer term, ensuring that $\mathcal{B}$ is in fact the only party that can obtain $gw(T_{\mathcal{B}}, T_{\mathcal{K}}, T)$.

4 Formal protocol specification

In this section, we show how the equational theory from section 3 is integrated into higher level protocols modeled by processes communicating over a public network. Figure 3 contains the process specifications of the two roles in Yao's protocol for secure two-party computation: the sender process $\mathcal{A}$ and the receiver process $\mathcal{B}$. Text within **(*** and ***)** represents comments.

The public parameter of $\mathcal{A}$ and $\mathcal{B}$ is the function to be evaluated, represented by the free variable $x_{\mathcal{F}}$. The private parameters of $\mathcal{A}$ and $\mathcal{B}$ are their respective inputs, represented by the free variables $x_{\mathcal{A}}$ and respectively $x_{\mathcal{B}}$. The goal of $\mathcal{A}$ and $\mathcal{B}$ is therefore to obtain $eval(x_{\mathcal{F}}, x_{\mathcal{A}}, x_{\mathcal{B}})$, without disclosing $x_{\mathcal{A}}$ to $\mathcal{B}$ and $x_{\mathcal{B}}$ to $\mathcal{A}$. A public name c represents the communication channel between the two parties, possibly controlled by an attacker.

Sender $\mathcal{A}(x_{\mathcal{F}}, x_{\mathcal{A}})$	**Receiver** $\mathcal{B}(x_{\mathcal{F}}, x_{\mathcal{B}})$
(* Receive $\mathcal{B}$'s commited input *) $in(c, x_c)$; **(* Characterize a started session *)** $event\ \mathcal{A}_{in}(x_{\mathcal{F}}, x_{\mathcal{A}}, x_c)$; **(* Generate a new garbling key *)** $new\, k_{\mathcal{A}}$; **(* Garble and send the function *)** $let\ x_{gf} = gf(x_{\mathcal{F}}, k_{\mathcal{A}})\ in\ out(c, x_{gf})$; **(* Garble and send $\mathcal{A}$'s input *)** $let\ x_{ga} = gw(x_{\mathcal{A}}, k_{\mathcal{A}}, a)\ in\ out(c, x_{ga})$; **(* Oblivious transfer of $\mathcal{B}$'s garbled input *)** $let\ x_{ot} = gwot(x_c, k_{\mathcal{A}}, b)\ in$ $out(c, x_{ot})$; **(* Receive the result *)** $in(c, y_{\mathcal{A}})$; $let\ x_{res} = dec(y_{\mathcal{A}}, k_{\mathcal{A}})\ in$ **(* Characterize an ended session *)** $event\ \mathcal{A}_{res}(x_{\mathcal{F}}, x_{\mathcal{A}}, x_c, x_{res})$	**(* Send $\mathcal{B}$'s commited input *)** $new\, n_{\mathcal{B}}$; $let\ x_c = com(x_{\mathcal{B}}, n_{\mathcal{B}})\ in\ out(c, x_c)$; **(* Receive garbled $x_{\mathcal{F}}$ and $x_{\mathcal{A}}$ *)** $in(c, x_{gf})$; $in(c, x_{ga})$; **(* Characterize a started session *)** $event\ \mathcal{B}_{in}(x_{\mathcal{F}}, x_{ga}, x_{\mathcal{B}})$; **(* Oblivious receival of $\mathcal{B}$'s garbled input *)** $in(c, x_{ot})$; $let\ x_{gb} = get(x_{ot}, x_{\mathcal{B}}, n_{\mathcal{B}})\ in$ **(* Verify the garbled function *)** $if\ checkf(x_{gf}, x_{\mathcal{F}}) = ok\ then$ **(* Perform garbled computation *)** $let\ x_{res} = geval(x_{gf}, x_{ga}, x_{gb})\ in$ $let\ y_{\mathcal{A}} = geval'(x_{gf}, x_{ga}, x_{gb})\ in$ **(* Send the result to $\mathcal{A}$ *)** $out(c, y_{\mathcal{A}})$; **(* Characterize an ended session *)** $event\ \mathcal{B}_{res}(x_{\mathcal{F}}, x_{ga}, x_{\mathcal{B}}, x_{res})$

Figure 3 Processes for two-party computation

Sender. The sender process $\mathcal{A}$ creates a new garbling key $k_{\mathcal{A}}$, which it uses to garble the circuit $x_{\mathcal{F}}$, its input $x_{\mathcal{A}}$ and, obliviously, the input of $\mathcal{B}$. As part of oblivious transfer, $\mathcal{A}$ first receives the commited input of $\mathcal{B}$. The garbled values, as well as the corresponding oblivious transfer term, are sent to $\mathcal{B}$ over the public channel c. As response from $\mathcal{B}$, $\mathcal{A}$ receives the result of the computation encrypted with $k_{\mathcal{A}}$.

Receiver. The receiver process $\mathcal{B}$ obtains garbled data from $\mathcal{A}$ and, to get a garbled version of its own input $x_{\mathcal{B}}$, engages in the oblivious transfer protocol: it makes a commitment to $x_{\mathcal{B}}$, sends the commitment to $\mathcal{A}$ and receives in response the oblivious transfer term that corresponds to the garbled version of the committed input, which $\mathcal{B}$ retreives relying on the function *get*. Next, $\mathcal{B}$ verifies that the function is correctly garbled and, if the verification is successful, it performs the garbled computation. The vaule x_{res} is the result obtained by $\mathcal{B}$, while $y_{\mathcal{A}}$ is the encrypted result that is sent back to $\mathcal{A}$.

Events. The events $\mathcal{A}_{in}$, $\mathcal{A}_{res}$, $\mathcal{B}_{in}$ and $\mathcal{B}_{res}$ are used as part of the formal specification of security properties that we present in section 5.

- The event $\mathcal{A}_{in}(x_{\mathcal{F}}, x_{\mathcal{A}}, x_c)$ records that $\mathcal{A}$ has engaged in a protocol session for the computation of $x_{\mathcal{F}}$, having $\mathcal{A}$'s input equal to $x_{\mathcal{A}}$, and $\mathcal{B}$'s input being committed to x_c.
- The event $\mathcal{A}_{res}(x_{\mathcal{F}}, x_{\mathcal{A}}, x_c, x_{res})$ records in addition that $\mathcal{A}$ has obtained the result x_{res} as outcome of the protocol session.
- The event $\mathcal{B}_{in}(x_{\mathcal{F}}, x_{ga}, x_{\mathcal{B}})$ records that $\mathcal{B}$ has engaged in a procotol session for the computation of $x_{\mathcal{F}}$, having $\mathcal{B}$'s input equal to $x_{\mathcal{B}}$, and $\mathcal{A}$'s input being garbled as x_{ga}.
- The event $\mathcal{B}_{res}(x_{\mathcal{F}}, x_{ga}, x_{\mathcal{B}}, x_{res})$ records in addition that $\mathcal{B}$ has obtained the result x_{res} as outcome of the protocol session.

Attacker. As usual, the attacker can execute any of the operations that we have described, as well as any other operations allowed by the equational theory, and (pretend to) play the role of any party, while interacting with an honest party $\mathcal{A}$ or $\mathcal{B}$ on the public channel c. This is captured formally by the semantics of the applied pi-calculus and the definition of the security properties that we present in the next section.

5 Formal models of security for two-party computation

Informally, we require the following properties from a secure two-party computation protocol:

1. The dishonest parties should not learn too much:
   - **(a)** The only leakage about the input of an honest party should come from the result of the evaluated function (**Input privacy**).
   - **(b)** A dishonest party should be able to evaluate a function on honest inputs only as agreed by the corresponding honest party (**Input agreement**).
2. The honest parties learn the correct result (**Result integrity**).

The distinction between input privacy and input agreement separates the task of input protection for honest parties into (a) protecting the honest input during the protocol flow (without bothering about the output of the function); and (b) ensuring that a dishonest party can learn function outputs on private inputs only as agreed by the corresponding honest party. This distinction helps to address automated verification problems when the public output of the protocol depends on the private input of parties. For example, automating privacy proofs for electronic voting protocols is known to be problematic, because care should be taken to separate the legitimate (e.g. the result of the election) from the illegitimate information flow [29, 30]. This is also a problem for automating simulation-based proofs,

where an ideal functionality models exactly what can be leaked by the protocol, and a simulator needs to be found that shows the protocol not to leak more [11, 12, 13]. Our separation of this property into (a) and (b) is a new way of addressing this problem, and is making more explicit the properties that are achieved, without requiring a simulator as in [11, 12, 13] or additional honest parties as in [29, 30].

These security properties can be specified in a general setting, but for brevity we present them in relation to the models of Sections 3 and 4, and leave their generalization as future work. In this setting, a specification of a two-party computation protocol is given by a triple $(\mathcal{A}, \mathcal{B}, \mathcal{E})$, where $\mathcal{E}$ is an equational theory containing $\mathcal{E}_{GC}$ from Section 3, $\mathcal{A}$ is a sender process with free variables $x_{\mathcal{F}}, x_{\mathcal{A}}$, $\mathcal{B}$ is a receiver process with free variables $x_{\mathcal{F}}, x_{\mathcal{B}}$, and these processes are enriched with events $\mathcal{A}_{in}, \mathcal{B}_{in}, \mathcal{A}_{res}, \mathcal{B}_{res}$ presented in Section 4.

5.1 Result integrity

Result integrity should ensure that the final result obtained by an honest party $\mathcal{P} \in \{\mathcal{A}, \mathcal{B}\}$ after a session of the protocol is consistent with the function that $\mathcal{P}$ expects to be evaluated, with the input of $\mathcal{P}$ in this session, and with the input of the other party, that has responded to this session, or has initiated it. Formally, the events $\mathcal{A}_{res}(x_{\mathcal{F}}, x_{\mathcal{A}}, x_c, x_{res})$ and $\mathcal{B}_{res}(x_{\mathcal{F}}, x_{ga}, x_{\mathcal{B}}, x_{res})$ capture the views of $\mathcal{A}$ and $\mathcal{B}$ after a session of the protocol has ended, recording all the relevant data, in particular the result obtained by the respective party, and the committed (resp. garbled) input of the other party. Therefore, we can specify the requirement of result integrity by the correspondence assertions $\Phi^{\mathcal{A}}_{int}$ and $\Phi^{\mathcal{B}}_{int}$ presented in Definition 1.

▶ **Definition 1** (Result integrity)**.** Let $(\mathcal{A}, \mathcal{B}, \mathcal{E})$ be a specification of a two-party computation protocol. We define the correspondence assertions $\Phi^{\mathcal{A}}_{int}$ and $\Phi^{\mathcal{B}}_{int}$ as follows:

$$\begin{array}{lcll} \Phi^{\mathcal{A}}_{int} & \doteq & ev : \mathcal{A}_{res}(x, y, z, w) & \rightsquigarrow \quad z = com(z_1, z_2) \ \wedge \ w = eval(x, y, z_1) \\ \Phi^{\mathcal{B}}_{int} & \doteq & ev : \mathcal{B}_{res}(x, y, z, w) & \rightsquigarrow \quad y = gw(y_1, y_2, a) \ \wedge \ w = eval(x, y_1, z) \end{array}$$

We say that $(\mathcal{A}, \mathcal{B}, \mathcal{E})$ satisfies result integrity if

$$\begin{array}{lll} !\,(\ in(c, x_{\mathcal{F}}); in(c, x_{\mathcal{A}}); \mathcal{A}(x_{\mathcal{F}}, x_{\mathcal{A}})\ ) & \models_{\mathcal{E}} & \Phi^{\mathcal{A}}_{int} \qquad \text{and} \\ !\,(\ in(c, x_{\mathcal{F}}); in(c, x_{\mathcal{B}}); \mathcal{B}(x_{\mathcal{F}}, x_{\mathcal{B}})\ ) & \models_{\mathcal{E}} & \Phi^{\mathcal{B}}_{int} \end{array}$$

The specification lets the attacker execute any number of sessions of an honest party $\mathcal{A}$ or $\mathcal{B}$, with any function $x_{\mathcal{F}}$ and any values $x_{\mathcal{A}}, x_{\mathcal{B}}$ as inputs, and requires the correspondence assertions $\Phi^{\mathcal{A}}_{int}$ and $\Phi^{\mathcal{B}}_{int}$ to be satisfied by this process. In turn, $\Phi^{\mathcal{A}}_{int}$ and $\Phi^{\mathcal{B}}_{int}$ require that for any occurence of the event $\mathcal{A}_{res}$ or $\mathcal{B}_{res}$, the result obtained by the respective honest party, recorded in the variable w (say T_w), correctly reflects the function and relevant messages of the corresponding session, recorded in variables x, y, z (say T_x, T_y, T_z).

The variables z_1, z_2 in $\Phi^{\mathcal{A}}_{int}$ are existentially quantified by definition. Therefore, $\Phi^{\mathcal{A}}_{int}$ requires that the term T_z, which represents the message that $\mathcal{A}$ received from $\mathcal{B}$, should be of the form $com(T_1, T_2)$, for some terms T_1, T_2 such that the final result T_w is equal to $eval(T_x, T_y, T_1)$. Similarly, $\Phi^{\mathcal{B}}_{int}$ requires T_y, which represents the message that $\mathcal{B}$ received from $\mathcal{A}$, to be of the form $gw(T_1, T_2, a)$, for some terms T_1, T_2 such that T_w is equal to $eval(T_x, T_1, T_z)$.

5.2 Input agreement

Input agreement should ensure that the function outputs obtained by a dishonest party after executing a session of the protocol are consistent with the expectation of an honest party

when it releases its private inputs. Specifically, consider the case where an honest party $\mathcal{A}$ supplied an input $T_{\mathcal{A}}$ in order to compute a function $T_{\mathcal{F}}$. Then, the other party (possibly dishonest) should only be able to obtain $eval(T_{\mathcal{F}}, T_{\mathcal{A}}, T_{\mathcal{B}})$, where $T_{\mathcal{B}}$ is its own input when playing the role of $\mathcal{B}$ in the corresponding protocol session. In particular, the other party should not be able to obtain $eval(T_{\mathcal{F}}, T_{\mathcal{A}}, T'_{\mathcal{B}})$, for a different input $T'_{\mathcal{B}}$, or $eval(T'_{\mathcal{F}}, T_{\mathcal{A}}, T_{\mathcal{B}})$, for different function $T'_{\mathcal{F}}$. Similar guarantees should hold for an honest party $\mathcal{B}$.

We formally define these requirements as correspondence assertions restricting the function outputs that can be obtained by the attacker. The fact that the attacker knows a particular function output can be expressed by the basic formula $att : eval(x, y, z)$. To express the constraints associated with this formula (restricting the values x, y, z), we rely on events $\mathcal{A}_{in}(x_{\mathcal{F}}, x_{\mathcal{A}}, x_c)$ and $\mathcal{B}_{in}(x_{\mathcal{F}}, x_{ga}, x_{\mathcal{B}})$, that record the parameters of each honest party in a started protocol session. In particular, the event $\mathcal{A}_{in}$ records the commited input of $\mathcal{B}$, received by $\mathcal{A}$, and $\mathcal{B}_{in}$ records the garbled input of $\mathcal{A}$, received by $\mathcal{B}$. Therefore, these events fully determine the result that each party (and in particular a dishonest party) should obtain from the respective protocol session. Then, in Definition 2 we require that to any function output $eval(x, y, z)$ obtained by the attacker, there corresponds an initial event recording the agreement of the respective honest party $\mathcal{A}$ or $\mathcal{B}$.

▶ **Definition 2** (Input agreement)**.** Let $(\mathcal{A}, \mathcal{B}, \mathcal{E})$ be a specification of a two-party computation protocol. We define the correspondence assertions $\Phi^{\mathcal{A}}_{agr}$ and $\Phi^{\mathcal{B}}_{agr}$ as follows:

$$\begin{array}{lcll} \Phi^{\mathcal{A}}_{agr} & \doteq & att : eval(x,y,z) & \rightsquigarrow \quad (\; ev : \mathcal{A}_{in}(x,y,z_1) \; \wedge \; z_1 = com(z,z_2) \;) \; \vee \; att : y \\ \Phi^{\mathcal{B}}_{agr} & \doteq & att : eval(x,y,z) & \rightsquigarrow \quad (\; ev : \mathcal{B}_{in}(x,y_1,z) \; \wedge \; y_1 = gw(y,y_2,a) \;) \vee att : z \end{array}$$

We say that a specification $(\mathcal{A}, \mathcal{B}, \mathcal{E})$ of a two-party computation protocol satisfies *input agreement* if:

$$\begin{array}{lcll} ! \; (\; in(c, x_{\mathcal{F}}); new\, i_{\mathcal{A}}; \mathcal{A}(x_{\mathcal{F}}, i_{\mathcal{A}}) \;) & \models_{\mathcal{E}} & \Phi^{\mathcal{A}}_{agr} & \text{and} \\ ! \; (\; in(c, x_{\mathcal{F}}); new\, i_{\mathcal{B}}; \mathcal{B}(x_{\mathcal{F}}, i_{\mathcal{B}}) \;) & \models_{\mathcal{E}} & \Phi^{\mathcal{B}}_{agr} & \end{array}$$

Note, however, that this property cannot be achieved if the input of the honest party is known to the attacker, who can obtain $eval(x, y, z)$ from x, y, z, by simply evaluating the function. Therefore, input agreement as defined here makes sense only for honest input values that are not available to the attacker. This is captured by the disjunction in the correspondence assertions $\Phi^{\mathcal{A}}_{agr}$ and $\Phi^{\mathcal{B}}_{agr}$ of Definition 2, and by the fact that inputs $i_{\mathcal{A}}, i_{\mathcal{B}}$ of honest parties in the test processes $\mathcal{A}(x_{\mathcal{F}}, i_{\mathcal{A}}), \mathcal{B}(x_{\mathcal{F}}, i_{\mathcal{B}})$ are locally generated for each session.

5.3 Input privacy

Input agreement prevents the attacker from learning unauthorized outputs of the function, but it does not ensure the privacy of an honest input - an equally important and complementary property (indeed, without privacy, the attacker may first derive the honest input and then simply evaluate the function). Traditionally, the privacy of an input x in a process $\mathcal{P}(x)$ is defined as a property of indistinguishability, e.g. observational equivalence, between two of its instances, say $\mathcal{P}(a_1)$ and $\mathcal{P}(a_2)$. For example, this is the case when verifying strong secrecy [31] or vote privacy in electronic voting [29, 30]. Modeling strong secrecy in this way is straightforward, because no information should flow from the secret input to the public output. In the case of e-voting protocols, and in our case of secure two-party computation, the definition should be made robust in order to take into account information flow that is unavoidable from the output of the protocol.

Typically, such information flow results in trivial attacks against formal indistinguishability, yet they are out of the scope of cryptographic protocols, and should not be considered as attacks (they are often called false attacks). In [29, 30], the specification avoids such false attacks by requiring the presence of at least two honest voters, which swap their votes such that the result of the election is the same in two different experiments. More generally, we can require that the only leakage about the input of an honest party should come from the result of the evaluated function. In other words, if the output of the function is withheld from the attacker, *no* leakage should occur about the honest inputs. In this case, we have a standard requirement of strong secrecy, which can be specified as an observational equivalence.

On the other hand, it is not straightforward to formalize the first part of this requirement, withholding the output of the function from the attacker. The attacker might be able to compute the output by combining data gathered throughout the protocol (for example, an attacker playing the role of $\mathcal{B}$ in Yao's protocol can evaluate the function output from the received garbled data). In such cases, it is not clear what data can be legitimately withheld from the attacker when defining input privacy. Instead, we will enrich the equational theory such that, for the honest inputs, all corresponding function outputs are equivalent, i.e. the attacker cannot observe the difference between them. Therefore, rather than suppressing the function output in the protocol specification, we suppress the attacker's ability to gain information from this output. The enriched equational theory relies on special function symbols α and β that will decorate the private inputs of an honest party $\mathcal{A}$, respectively $\mathcal{B}$.

▶ **Definition 3.** Let $\mathcal{E}$ be an equational theory. Consider the function symbols α, β and the constants α_0, β_0. We define the equational theories

$$\begin{array}{lcl} \mathcal{E}_\alpha & = & \mathcal{E} \cup \{\ eval(x, \alpha(y), z) \to eval(x, \alpha_0, z)\ \} \quad \text{and} \\ \mathcal{E}_\beta & = & \mathcal{E} \cup \{\ eval(x, y, \beta(z)) \to eval(x, y, \beta_0)\ \} \end{array}$$

The rewrite rules for *eval* declare function evaluations of inputs decorated with α, β to be equivalent, relying on the constants α_0, β_0, which are used for obtaining a convergent equational theory. Then, the specification in Definition 4 considers two versions of a process: for any number of sessions, and any choice of terms x^0, x^1 for each session, in the first version an honest party $\mathcal{A}$, respectively $\mathcal{B}$, inputs $\alpha(x^0)$, respectively $\beta(x^0)$; in the second version the party inputs $\alpha(x^1)$, respectively $\beta(x^1)$. We say that the protocol satisfies input privacy if these two versions are in observational equivalence, i.e. indistinguishable for the attacker.

▶ **Definition 4** (Input privacy)**.** Let $(\mathcal{A}, \mathcal{B}, \mathcal{E})$ be a specification of a two-party computation protocol and $\mathcal{E}_\alpha, \mathcal{E}_\beta$ be the equational theories from Definition 3. Let $\mathcal{C}_{in}[\_]$ be the process context $in(c, x_\mathcal{F})$; $in(c, x^0)$; $in(c, x^1)$; $[\_]$. We say that $(\mathcal{A}, \mathcal{B}, \mathcal{E})$ satisfies input privacy if

$$\begin{array}{lcl} !\,\mathcal{C}_{in}[\,\mathcal{A}(x_\mathcal{F}, \alpha(x^0))\,] & \sim_{\mathcal{E}_\alpha} & !\,\mathcal{C}_{in}[\,\mathcal{A}(x_\mathcal{F}, \alpha(x^1))\,] \quad \text{and} \\ !\,\mathcal{C}_{in}[\,\mathcal{B}(x_\mathcal{F}, \beta(x^0))\,] & \sim_{\mathcal{E}_\beta} & !\,\mathcal{C}_{in}[\,\mathcal{B}(x_\mathcal{F}, \beta(x^1))\,] \end{array}$$

Note that $\alpha(x^0)$ and $\alpha(x^1)$ remain distinct terms with respect to $\mathcal{E}_\alpha$ when considered in any context other than in terms of the form $eval(y, \alpha(x^0), z), eval(y, \alpha(x^1), z)$; and similarly for $\mathcal{E}_\beta$. That is why, if there is a privacy weakness in the protocol, the attacker will be able to spot the difference between the two experiments in Definition 4, for either $\mathcal{A}$ or $\mathcal{B}$.

6 Conclusion and related work

The ProVerif code for the models introduced in this paper is in appendix. ProVerif returns within seconds positive results for all queries, and we also perform reachability tests to

ensure that all parties can execute the protocol correctly. Our contributions differ from related work in several respects and also open some new research questions:

The model of Backes et al [14] considers multi-party computation functionalities abstractly, allowing to reason about their use in larger protocols, without necessarily representing the cryptographic primitives that realize the functionality. Their framework comes equipped with a computational soundness result and is applied to the case study of an auction protocol [32]. The security properties that are specified and verified automatically (relying on type-checking) are limited to robust safety, which can be related to our property of result integrity.

Dahl and Damgård [13] also propose a formal framework for specifying two-party computation protocols and present the automated verification with ProVerif of an oblivious transfer protocol based on homomorphic encryption [28]. They formulate a definition of simulation-based security in applied pi-calculus and show it to be computationally sound. In order to obtain an input for ProVerif, they have to find a simulator and additionally massage the resulting processes manually. On the other hand, our methodology does not require the explicit construction of a simulator and our models can be readily given as input to automated tools. Our case study is also different, relying on garbled circuits rather than homomorphic encryption, and on oblivious transfer as a sub-protocol to compute any function the two parties may agree on. However, we do not provide a soundness result, and the relation of our models to simulation-based security remains an open question. In that direction, we can also explore extensions of our models into a general framework allowing the verification of other protocols, for two or multiple parties, and relying on various cryptographic primitives.

Delaune et al [11] and Böhl and Unruh [12] study definitions of simulation-based security in applied pi-calculus, showing their application to the analysis of several protocols. Although quite general, their frameworks are not easily amenable to automation. Similarly to [13], the authors of [12] have to perform a significant amount of manual proof before applying ProVerif, and they have left automated verification as an open question. Earlier symbolic models for simulation-based security, which are computationally sound, yet more complex, are presented in [9, 10, 33]. Our paper is an argument for a different approach: rather than directly expressing simulation-based security in formal models, we propose several security notions whose conjunction should be sufficient for secure two-party computation, while it remains to be seen under what conditions they imply simulation-based security. This methodology promises not only better automation, but also a better understanding of what security properties are achieved. In turn, this may aid the design of new protocols, where some of the properties can be relaxed.

A formal model for oblivious transfer in applied pi-calculus is presented by Dahl and Damgård [13]. Their specification is a process modeling a particular protocol, whereas we propose a more abstract equational theory. This equational theory is however specific, because it only models the oblivious transfer of garbled values. It would be possible to propose a generic equational theory for oblivious transfer, but it may cause problems for automated verification - it remains a problem for future work. Conversely, the model of Goubault et al [34] aims to capture formally the probabilistic aspect of some oblivious transfer protocols.

Acknowledgement. We thank the anonymous reviewers for their valuable comments.

References

1 Andrew Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

2 Andrew Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.

3 Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Naor [35], pages 52–78.

4 Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Naor [35], pages 97–114.

5 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

6 Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

7 Martín Abadi, Bruno Blanchet, and Hubert Comon-Lundh. Models and proofs of protocol security: A progress report. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 2009.

8 Véronique Cortier and Steve Kremer, editors. *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*. IOS Press, 2011.

9 Ran Canetti and Jonathan Herzog. Universally composable symbolic security analysis. *J. Cryptology*, 24(1):83–147, 2011.

10 Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, 2003.

11 Stéphanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. In Ravi Kannan and K. Narayan Kumar, editors, *Proceedings of the 29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'09)*, volume 4, pages 169–180, Kanpur, India, December 2009.

12 Florian Böhl and Dominique Unruh. Symbolic universal composability. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26-28, 2013*, pages 257–271. IEEE, 2013.

13 Morten Dahl and Ivan Damgård. Universally composable symbolic analysis for two-party protocols based on homomorphic encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 695–712. Springer, 2014.

14 Michael Backes, Matteo Maffei, and Esfandiar Mohammadi. Computationally sound abstraction and verification of secure multi-party computations. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 352–363, 2010.

15 Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

16 Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

17 Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2013.

18 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

19 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

20 Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW'01)*, 2001.

21 Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

22 Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008.

23 Mark Ryan and Ben Smyth. Applied pi calculus. In V. Cortier and S. Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, Cryptology and Information Security Series. IOS Press, 2011.

24 Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 243–320. MIT Press, 1990.

25 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

26 Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *Computer Security Foundations Symposium (CSF), Port Jefferson, New York, USA, July 8-10, 2009*, pages 266–276. IEEE Computer Society, 2009.

27 Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.

28 Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2008.

29 Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

30 Michael Backes, Cătălin Hriţcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Computer Security Foundations Symposium (CSF)*, pages 195–209. IEEE Computer Society, 2008.

31 Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, page 86. IEEE Computer Society, 2004.

32 Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security, FC 2009*, pages 325–343, 2009.

33 Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.

34 Jean Goubault-Larrecq, Catuscia Palamidessi, and Angelo Troina. A probabilistic applied pi-calculus. In Zhong Shao, editor, *Asian Symposium on Programming Languages and Systems (APLAS'07)*, volume 4807 of *Lecture Notes in Computer Science*, pages 175–290, Singapore, 2007. Springer.

35 Moni Naor, editor. *Advances in Cryptology - EUROCRYPT 2007, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*. Springer, 2007.

7 Operational semantics and verification with ProVerif

The rules that define the operational semantics of applied pi-calculus and ProVerif, adapted from [21] and [22], are presented in Figure 4. We have a transition system on configurations of the form $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P})$, where $\mathcal{N}$ is a set of names, $\mathcal{M}$ is a sequence of messages, $\mathcal{L}$ is a set of events and $\mathcal{P}$ is a multiset of processes. We denote by $P[x \mapsto T]$ the process obtained from P by replacing every occurence of x with T, and by $\mathcal{M} \circ T$ the sequence of terms obtained by appending T to the sequence of terms $\mathcal{M}$.

(NIL) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{0\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P})$

(BANG) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{!P\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{P, !P\})$

(PAR) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{P \mid Q\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{P, Q\})$

(NEW) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{new\, n; P\}) \to (\mathcal{N} \cup \{n'\}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{P\})$
where $n' \notin \mathcal{N}$

(COMM) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{out(c, T); P\ ,\ in(c, x); Q\}) \to (\mathcal{N}, \mathcal{M}', \mathcal{L}, \mathcal{P} \cup \{P, Q[x \mapsto T\downarrow]\}\})$
where $\mathcal{M}' = \mathcal{M} \circ T$, if $\mathcal{M} \vdash_{\mathcal{E}} c$, and $\mathcal{M}' = \mathcal{M}$, otherwise

(IF$_\mathrm{T}$) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{if\ U = V\ then\ P\ else\ Q\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{P\})$
if $U =_{\mathcal{E}} V$

(IF$_\mathrm{F}$) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{if\ U = V\ then\ P\ else\ Q\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{Q\})$
if $U \neq_{\mathcal{E}} V$

(LET) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{let\ x = T\ in\ P\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{P[x \mapsto T\downarrow]\})$

(EV) $(\mathcal{N}, \mathcal{M}, \mathcal{L}, \mathcal{P} \cup \{event\ T; P\}) \to (\mathcal{N}, \mathcal{M}, \mathcal{L} \cup \{T\}, \mathcal{P} \cup \{P\})$

Figure 4 Operational semantics

On the following pages, we present the ProVerif input for the equational theory of Section 3, the processes of Section 4, and the properties of Section 5. The code is also available online.

```
(* EQUATIONAL THEORY *)

(* CONSTANTS *)
fun zero/0. fun one/0.

(* FUNCTION EVALUATION *)
fun eval/3.

(* ENCRYPTION *)
fun enc/2.
reduc dec(enc(x,y),y) = x.

(* GARBLED COMPUTATION *)
fun ia/0. fun ib/0.
fun gf/2. fun gw/3.
reduc geval(gf(x,k),gw(y,k,ia),gw(z,k,ib)) = eval(x,y,z).
reduc geval'(gf(x,k),gw(y,k,ia),gw(z,k,ib)) = enc(eval(x,y,z),k).

reduc ungarb(gf(x,k),k) = x;
      ungarb(gw(x,k,y),k) = x;
      ungarb(gw(x,k,y),k) = x.

(* GARBLED OBLIVIOUS TRANSFER *)
fun gwot/3. fun com/2.
reduc get(gwot(com(x,z),y1,y2), x, z) = gw(x, y1, y2).

(* CORRECTNESS PROOF *)
fun ok/0.
reduc checkf(gf(xf,xk), xf) = ok.

(* PUBLIC CHANNEL MODELLING THE COMMUNICATION NETWORK *)
free c.

(* THE TWO PARTIES THAT EXECUTE THE PROTOCOL ARE MODELED BY
partyA - THE PARTY THAT GARBLES THE CIRCUIT
partyB - THE PARTY THAT EVALUATES THE GARBLED CIRCUIT
*)
```

```
let partyA =
    (* RECEIVE A COMMITMENT TO THE INPUT OF B (PART OF OBLIVIOUS TRANSFER)   *)
    in(c, xc);

    (* CHARACTERIZING A STARTED SESSION OF A *)
    event A_in(xf, xa, xc);

    (* CREATE A NEW GARBLING KEY *)
    new key;

    (* GARBLE AND SEND THE FUNCTION *)
    let xgf = gf(xf,key) in out(c, xgf);

    (* GARBLE AND SEND THE INPUT OF A *)
    let xgwa = gw(xa,key,ia) in out(c,xgwa);

    (* COMPLETE THE OBLIVIOUS TRANSFER OF B'S GARBLED INPUT *)
    let otb = gwot(xc, key, ib) in out(c,otb);

           (* RECEIVE THE AND DECRYPT THE RESULT *)
    in(c,enc_result); let result = dec(enc_result,key) in

    (* CHARACTERIZING A FINISHED SESSION OF A *)
    event A_res(xf, xa, xc, result).

let partyB =
    (* SEND THE COMMITMENT OF B'S INPUT (PART OF OBLIVIOUS TRANSFER) *)
    new nonce;
    let input_commit = com(xb,nonce) in
    out(c, input_commit);

    (* RECEIVE GARBLED DATA *)
    in(c, xgf); in(c, xgwa);

    (* COMPLETE THE OBLIVIOUS TRANSFER OF B'S GARBLED INPUT *)
    in(c, otb); let xgwb = get(otb, xb, nonce) in

    (* CHARACTERIZING A STARTED SESSION OF B *)
    event B_in(xf, xgwa, xb);

    (* VERIFY THE GARBLED FUNCTION *)
    if checkf(xgf,xf) = ok then
    (

    (* PERFORM THE GARBLED COMPUTATION *)
    let resultB = geval(xgf, xgwa, xgwb) in
    let resultA = geval'(xgf, xgwa, xgwb) in

    (* SEND THE ENCRYPTED RESULT BACK TO partyA *)
    out(c,resultA);

    (* CHARACTERIZING A FINISHED SESSION OF B *)
    event B_res(xf, xgwa, xb, resultB)
    ).
```

```
(* QUERIES (SHOULD BE COPIED AND RUN SEPARATELY IN 3 DISTINCT FILES) *)

(* RESULT INTEGRITY - HONEST PARTIES GET THE CORRECT RESULT *)
query ev:A_res(x,y,z,w) ==> z=com(z',z'') &  w = eval(x,y,z').
query ev:B_res(x,y,z,w) ==> y=gw(y',y'',ia) & w = eval(x,y',z).

(* MAIN PROCESS FOR RESULT INTEGRITY *)
process ! ( in(c, xf); in(c,xa); partyA ) |
! ( in(c, xf); in(c,xb); partyB )

(* INPUT AGREEMENT - THE ATTACKER ONLY LEARNS AGREED FUNCTION OUTPUTS *)
query attacker:eval(x,y,z) ==> (ev:A_in(x,y,z') & z'=com(z,z'')) | attacker:y.
query attacker:eval(x,y,z) ==> (ev:B_in(x,y',z) & y' = gw(y,y'',ia)) | attacker:z.

(* MAIN PROCESS FOR INPUT AGREEMENT *)
process ! ( in(c, xf); new xa; partyA ) |
! ( in(c, xf); new xb; partyB)

(* INPUT PRIVACY - EQUATIONAL THEORY AND MAIN PROCESS *)
fun alpha/1. fun alpha0/0.
fun beta/1. fun beta0/0.
equation eval(x,alpha(y),z) = eval(x, alpha0, z).
equation eval(x,y,beta(z)) = eval(x, y, beta0).

process ! ( in(c,(xf,xa0,xa1));
      let xa = choice[alpha(xa0),alpha(xa1)] in partyA
   ) |
! ( in(c,(xf,xb0,xb1));
      let xb = choice[beta(xb0),beta(xb1)] in partyB
   )

(* REACHABILITY TESTS FOR HONEST PARTIES *)
free test_circ.
query ev:B_res(x, y, z, eval(test_circ, zero, one)).
query ev:A_res(x, y, z, eval(test_circ, zero, one)).

(* REACHABILITY TESTS FOR ATTACKER - HONEST PARTIES EXECUTED WITH SECRET INPUT *)
private free test_val.
query attacker:eval(test_circ,test_val,one).
query attacker:eval(test_circ,one, test_val).
process ! ( in(c, xf); let xa = test_val in partyA ) |
! ( in(c, xf); let xb = test_val in partyB)
```