# Computational Soundness of Uniformity Properties for Multi-party Computation based on LSSS

Hui Zhao[1,2] and Kouichi Sakurai[2]

[1] Shandong University of Technology, Zibo SD 255000, China
[2] Kyushu University, 744 Motooka, Nishi-ku, Fukuoka, 819-0395, Japan

**Abstract.** We provide a symbolic model for multi-party computation based on linear secret-sharing scheme, and prove that this model is computationally sound: if there is an attack in the computational world, then there is an attack in the symbolic (abstract) model. Our original contribution is that we deal with the uniformity properties, which cannot be described using a single execution trace, while considering an unbounded number of sessions of the protocols in the presence of active and adaptive adversaries.

## 1 Introduction

Provable security are now widely considered an essential tool for validating the design of cryptographic schemes. While Dolev-Yao models traditionally comprise only non-interactive cryptographic operations (i.e.,cryptographic operations that produce a single message and do not involve any form of communication, such as encryption and digital signatures), recent cryptographic protocols rely on more sophisticated interactive primitives (i.e., cryptographic operations that involve several message exchanges among parties), with unique features that go far beyond the traditional goals of cryptography to solely offer secrecy and authenticity of communication.

Secret-sharing cryptographic operations constitutes arguably one of the most prominent and most amazing such primitive [1-9]. Traditionally, in an linear secret-sharing scheme with threshold $(t, l)$, a dealer $D$ and a number of players $P_1$, $P_2$, ..., $P_l$ wish to securely generate the secret share $ss_1$,$ss_2$, ..., $ss_l$, where each player $P_i$ holds a private input $d_i$. This secret-sharing scheme is considered sound and complete if any $t$ or more valid secret shares make the secret $s$ computable, and knowledge of any $t-1$ or fewer secret shares leaves the secret completely undetermined.

Indeed, verifiable secret sharing (VSS) and secure multi-party computation (MPC) among a set of $n$ players can efficiently be based on any linear secret-sharing scheme (LSSS) for the players, provided that the access structure of the LSSS allows MPC or VSS at all [9, 18]. Secure multi-party computation (MPC) can be defined as the problem of $n$ players to compute an agreed function of their inputs in a secure way, where security means guaranteeing the correctness of the output as well as the privacy of the players' inputs, even when some players cheat. A key tool for secure MPC, interesting in its own right, is verifiable secret sharing (VSS): a dealer distributes a secret value s among the players, where the dealer and/or some of the players may be cheating. It is guaranteed that if the dealer is honest, then the cheaters obtain no information about $s$, and all honest players are later able to reconstruct $s$. Even if the dealer cheats, a unique such value $s$ will be determined and is reconstructible without the cheaters' help.

Thus, it is important to develop abstraction techniques to reason about LSSS-based MPC and to offer support for the automated verification of their security. Further, computational soundness results (in the sense of uniformity properties) for MPC built upon the abstraction of LSSS should be presented.

### 1.1 Related works

Starting with the seminal work of Abadi and Rogaway [10-12], a lot of efforts has been directed to bridging the gap between the formal analysis system and computational-soundness model. The goal is to obtain the best of both worlds: simple, automated security proofs that entail strong security guarantees. Research over the past decade has shown that many of these Dolev-Yao models are computationally sound, i.e., the absence of attacks against the symbolic abstraction entails the security of suitable cryptographic realizations. Most of these computational soundness results against active attacks, however, have been specific to the class of trace properties, which is only sufficient as long as strong notions of privacy are not considered, e.g., in particular for establishing various authentication properties [13-18]. Only few computational soundness results are known for the class of equivalence properties against active attackers, most of these results focus on abstractions for which it is not clear how to formalize any equivalence property beyond the non-interactive cryptographic operations [19-22], such as multi-party computation that rely on more sophisticated interactive primitives [13, 18].

### 1.2 Challenging issues

Canetti et. proposed framework of universally composable security [18, 21]. In this framework, security properties is defined by what it means for a protocol to realize a given ideal functionality, where an ideal functionality is a natural algorithmic way of capturing the desired functionality of the protocol problem at hand. A protocol that is secure within the universally composable framework is called universally composable (UC).

We are facing a situation where computational soundness results, despite tremendous progress in the last decade, still fall short in comprehensively addressing the class of equivalence properties and protocols that formal verification tools are capable to deal with. Moreover, it is still unknown the composability in UC framework can be extended to achieve more comprehensive computational soundness results for equivalence properties.

### 1.3 Our main contributions

In this paper, we present an abstraction of LSSS within the pi-calculus. In this abstraction, linear secret-sharing scheme is defined by equational theory. Further, abstraction of LSSS-based MPC is provided as a process that receives the inputs from the parties involved in the protocol over private channels, create secret shares of the MPC result, and sends secret shares to the parties again over private channels. This abstraction can be used to model and reason about larger MPC protocols that employ LSSS as a building block.

We establish computational soundness results of preservation of uniformity properties for protocols built upon our abstraction of LSSS-based MPC. This result is obtained in essentially two steps: We first establish an ideal functionality for LSSS-based MPC in the UC framework. Second, we obtain an secure cryptographic realization of our symbolic abstraction of LSSS-based MPC. This computational soundness result holds for LSSS-based MPC that involve arbitrary arithmetic operations; moreover, we will show that it is compositional, in the sense that uniformity properties of bi-processes in pi-calculus implies computational soundness results of preservation of uniformity properties. Such a result allows for soundly modeling and verifying many applications employing LSSS-based MPC as a building block.

### 1.4 Comparison with existing works

While computational soundness proofs for Dolev-Yao abstractions of multi-party computation use standard techniques[13, 18, 23-25] finding a sound ideal functionality for multi-party computation do not support equivalence properties. Securely realizable ideal functionalities constitute a useful tool for proving computational soundness for equivalence properties of a Dolev-Yao model. In our proof, we establish a connection between our symbolic abstraction of LSSS-based MPC and such an ideal functionality which supports uniformity properties: for an expressive class of equivalence properties, the uniformity implies observational equivalence for bi-processes , which are pairs of processes that differ only in the messages they operate on but not in their structure. We exploit that this ideal functionality is securely realizable.

## 2 The Abstraction of Secret-Sharing in Multi-party Computation

In this section, we present a symbolic abstraction of LSSS-based MPC based on Backes's abstraction for MPC [13]. Since the overall protocol may involve several secure LSSS-based MPC, a session identifier $sid$ is often used to link the private inputs to the intended session. We then represent $SS_{l,t}(m, r)$ as a function that explicitly generates the LSSS-based proofs. The resulting abstraction of LSSS-based MPC is depicted as the pi-calculus process $LMPC$ as follows.

$$\begin{aligned} Input_i = {} & !(inloop_i(z).in_i(d_i, sid').\overline{adv}(sid). \text{ if } sid = sid' \\ & \text{then } \overline{lin_i}(d_i) \text{ else } \overline{inloop_i}(sync())) \mid \overline{inloop_i}(sync()) \end{aligned}$$

$$Deliver_i = \overline{in_i}(y_i, sid).\overline{inloop_i}(sync())$$

$$\begin{aligned} & SSCompu(l, t, F) \\ & \quad = lin_1(d_1).lin_2(d_2)...lin_l(d_l).\nu r \\ & \qquad \mathit{let}\ y = SS_{l,t}(F(d_1, d_2, ..., d_l), r)\ \mathit{in} \\ & \qquad \mathit{let}\ y_1 = COE_{l,t,1}(y)\ \mathit{in} \\ & \qquad \mathit{let}\ y_2 = COE_{l,t,2}(y)\ \mathit{in} \\ & \qquad . \\ & \qquad . \\ & \qquad . \\ & \qquad \mathit{let}\ y_l = COE_{l,t,l}(y)\ \mathit{in} \\ & \qquad (Deliver_1 \mid Deliver_2 \mid ... \mid Deliver_l) \end{aligned}$$

$$\begin{aligned} LMPC(l, t, F, sidc, adv, \widetilde{in}) = {} & sidc(sid).\nu lin.\nu inloop. \\ & (Input_1 \mid Input_2 \mid ... \mid Input_l \mid !SSCompu(l, t, F) \end{aligned}$$

In the abstraction of LSSS-based MPC above, process $LMPC$ is parametrized by threshold $(l, t)$, $l$-function $F$, a session identifier channel $sidc$, a adversary channel $adv$, and $l$ private channels $in_i$ for the $l$ players. We implicitly assume that private channels are authenticated such that only the $i$th player can send messages on channel $in_i$. The computational implementation of $LMPC$ implements this authentication requirement. Furthermore, $LMPC$ contains two restricted channels for every party $i$: an internal loop channel $inloop_i$ and an internal input channel $lin_i$. $LMPC$ receives a session identifier over the channel $sidc$. Then $l + 2$ subprocesses are spawned: a process $input_i$ for each of the $l$ players that is responsible for collecting the $i$th input and for divulging public information, such as the session identifier, to the adversary, and a process that generates the actual secret-sharing proof. Here $input_i$ waits (under a replication) on the loop channel $inloop_i$ for the trigger message $sync()$ of the next round, and expects the private input $d_i$ and a session identifier $sid'$ over $in_i$. It then sends the session identifier $sid'$ to the adversary, checks whether the session identifier $sid'$ equals $sid$, and finally sends the private input $d_i$ on the internal input channel $lin_i$. The actual generation of secret share proof is performed in the last subprocess: after the private inputs of the individual parties are collected from the internal input channels $lin_i$, the function $SS_{l,t}(F(d_1, d_2, ..., d_l), r)$ is executed. After each computation round, the subprocesses $deliver_i$ send the individual secret-sharing proof $COE_{l,t,i}(SS_{l,t}(F(d_1, d_2, ..., d_l), r))$ over the private channels $in_i$ to every player $i$ along with the session identifier $sid$. In order to trigger the next round, $sync()$ is sent over the internal loop channels $inloop_i$.

The abstraction allows for a large class of $l$-function $F$ as described below.

**Definition 1.** *( l-function) We call a term $F$ an l-function if the term contains neither names nor variables, and if for every $\alpha_m$ occurring therein, we have $m \in [1, l]$.*

In our abstraction model for LSSS-based MPC with $(l, t)$ threshold, the values $\alpha_i$ in $F$ constitute placeholders for the private input $d_i$. Based on that, Dealer and players' ability to produce LSSS-based MPC proofs is modeled by introducing symbolic constructor $SS_{l,t}(F(\widetilde{d}), r)$, called *secret share key*. Its arguments are a message $F(\widetilde{d})$ and $d_i$ will serve as substitutes for the variables $\alpha_i$ in $F$. The semantics of these constructors will guarantee two properties: First, a secret $D = F\{\widetilde{d} \;/\; \widetilde{\alpha}\}$ can only be constructed by providing $t$ secret share proofs $COE_{l,t,i_1}(SS_{l,t}(F(\widetilde{d}), r)$, $COE_{l,t,i_2}(SS_{l,t}(F(\widetilde{d}), r))$, ..., $COE_{l,t,i_t}(SS_{l,t}(F(\widetilde{d}), r))$. Second, while the sum of secret shares is less than $t$, the values $D$ are kept secret. These properties imply that the secret share proof indeed guarantees that in the abstract model the soundness and the completeness of the secret-sharing schemes with threshold $(l, t)$. In the symbolic secret share proof above, $d_1$, $d_2$, ..., $d_l$ represent player 1,2,...,$l$'s respectful private input.

In the following we define the symbolic model in which the execution of a symbolic protocol involving secret share proofs takes place.

First, we fix several countably infinite sets. By $Nonce$ we denote the set of nonces. We use elements from *Garbage* to represent ill formed messages (corresponding to unparseable bitstrings in the computational model). Finally, elements of *Rand* denote symbolic randomness used in the construction of secret share proofs. We assume that $Nonce$ is partitioned into in infinite sets $Nonce_{ag}$ and $Nonce_{adv}$, representing the nonces of honest agents and the nonces of the adversary. Similarly, *Rand* is partitioned into in infinite sets $Rand_{ag}$ and $Rand_{adv}$.

We proceed by defining the syntax of messages that can be sent in a protocol execution. Since such messages can contain secret share proofs, and these are parametrized over constructors that are to be computed, we first have to define the syntax of these constructors. Let the message type $T$ be defined by the following grammar:

$$\begin{aligned} T &::= COE_{l,t}(SS_{l,t}(T, N)) \mid SS_{l,t}(T, N) \mid \\ &\quad\; pair(T, T) \mid S \mid N \mid garbage(N) \\ S &::= empty \mid string_0(S) \mid string_1(S) \end{aligned}$$

The intuitive interpretation of a $l$-function is that it is a term with free variables $\alpha_i$. The $\alpha_i$ will be substituted with messages.

We define destructors as follows:

$D := \{Combin_{l,t}/t, fst/1, snd/1, unstring_0/1, unstring_1/1, equals/2\}$. The destructor $Combin_{l,t}$ extracts the secret from a secret share sequence. The destructors $fst$ and $snd$ are used to destruct pairs, and the destructors $unstring_0$ and $unstring_1$ allow to parse payload-strings.

We further define the equational rule as follows:

$$Combin_{l,t}(COE_{l,t,i_1}(SS_{l,t}(m,r)), COE_{l,t,i_2}(SS_{l,t}(m,r)), ..., COE_{l,t,i_t}(SS_{l,t}(m,r)) = m.$$

Based on the symbolic model above, we then give the definition of symbolic pi-calculus execution of LSSS-based MPC.

**Definition 2.** *(Symbolic pi-calculus execution). Let $\Pi$ be a closed process, and let Adv be an interactive machine called the attacker. We define the symbolic pi-execution as an interactive machine $SExec_{\Pi}$ that interacts with Adv:*

1. *Start. Let $P = \Pi$, where we rename all bound variables and names (including nonces and randomness) such that they are pairwise distinct and distinct from all unbound ones. Let $\eta$ and $\mu$ be a totally undefined partial functions from variables and names, respectively, to terms. Let $a_1, a_2, ..., a_n$ denote the free names in $P_0$. For each $i$, pick $r_i \in Nonces_{ag}$ at random. Set $\mu := \mu \bigcup \{a_1 := r_1, a_2 = r_2, ..., a_n := r_n\}$. Send $(r_1, r_2, ..., r_n)$ to Adv.*
2. *Main loop. Send $P$ to the adversary and expect an evaluation context $E$ from the adversary. Distinguish the following cases:*
   - *(a) $P = E[M(x) : P_1]$: Request two terms $c$, $m$ from the adversary. If $c = eval_{\eta,\mu}(M)$, set $\eta := \eta \bigcup \{x := m\}$ and $P := E[P_1]$.*
   - *(b) $P = E[va.P_1]$: Pick $r \in Nonce_{ag} \setminus range(\mu)$, set $P := E[P_1]$ and $\mu := \mu(a := r)$.*
   - *(c) $P = E[\widetilde{M}(N).P_1][M_2(x).P_2]$: If $eval_{\eta,\mu}(M_1) = eval_{\eta,\mu}(M_2)$, then set $P := E[P_1][P_2]$ and $\eta := \eta \bigcup \{x := eval_{\eta,\mu}(N)\}$.*
   - *(d) $P = E[let\ x\ =\ D\ in\ P_1\ else\ P_2]$: If $m := eval_{\eta,\mu}(D)\bot$, set $\eta := \eta \bigcup \{x := m\}$ and $P := E[P_1]$; Otherwise set $P := E[P_2]$.*
   - *(e) $P = E[!P_1]$: Rename all bound variables of $P_1$ such that they are pairwise distinct and distinct from all variables and names in $P$ and in the domains of $\eta$ and $\mu$, yielding a process $\widetilde{P_1}$. Set $P := E[\widetilde{P_1}|!P_1]$.*
   - *(f) $P = E[\widetilde{M}(N).P_1]$: Request a term $c$ from the adversary. If $c = eval_{\eta,\mu}(M)$, set $P := E[P_1]$ and send $eval_{\eta,\mu}(N)$ to the adversary.*
   - *(g) In all other cases, do nothing.*

We are now ready to define what uniformity properties of a bi-process in the applied pi-calculus is.

**Definition 3.** *(Uniformity properties for bi-process) We say that the bi-process $P$ is uniform when $fst(P) \to Q_1$ implies that $P \to Q$ for some bi-process $Q$ with $fst(Q) = Q_1$, and symmetrically for $snd(P) \to Q_2$.*

## 3 Computational Soundness of Secret-Sharing in Multi-party Computation

In this section, we present a computational soundness result of preservation of uniformity properties for our abstraction of LSSS-based MPC. Our result builds on the universal composablity (UC) framework [18, 21], where the security of a protocol is defined by comparison with an ideal functionality $I$. The proof proceeds in three steps, as depicted in the following.

In the first step, we prove that the uniformity properties of an applied pi-calculus process carries over to the computational setting, where the protocol is executed by interactive Turing machines operating on bitstrings instead of symbolic terms and using cryptographic algorithms instead of constructors and destructors.The first part of the proof entails the computational soundness of a process executing the abstraction $LMPC(l, t, F, sidc, adv, \widetilde{in})$. A computational implementation of the protocol, instead, should execute an actual MPC protocol.

In the second step of the proof, we show that for each $l$-function $F$, the computational execution of our abstraction $LMPC(l, t, F,\ sidc, adv, \widetilde{in})$ is indistinguishable from the execution of a ideal $LMPC$ protocol $I$ that solely comprises a single incorruptible machine.

The third step of the proof ensures that for a $l$-function $F$ there is a protocol that securely realizes $I$ in the UC framework, which ensures in particular that the uniformity properties of $I$ carry over to the actual implementation.

These three steps allow us to conclude that for each abstraction $LMPC(l, t, F, sidc, adv, \widetilde{in})$ there exists an implementation such that the uniformity properties of any process $P$, carry over from the execution that merely executes a process $P$ and executes $LMPC(l, t, F, sidc, adv, \widetilde{in})$ as a regular subprocess to the execution that communicates with upon each call of a subprocess $LMPC(l, t, F, sidc, adv, \widetilde{in})$. By leveraging the composablity of the UC framework and the realization result for LSSS-based MPC in the UC framework, we finally conclude that if a protocol based on our LSSS-based MPC abstraction is robustly safe then there exists an implementation of that protocol that is computationally safe.

### 3.1 Computational execution of a process

We firstly give computational soundness definition of LSSS. Two properties are expected from a secret share proof in LSSS: knowledge of any $t$ or more valid secret shares makes the secret easily computable (completeness), it is computationally infeasible to produce the secret with knowledge of any $t-1$ or fewer valid secret shares(soundness):

**Definition 4.** *(Computational sound LSSS, $\Upsilon_{LSSS}$). A symbolically-sound LSSS is a tuple of polynomial-time algorithms ($K$, $SCon$, $SCom$, $S$) with the following properties (all probabilities are taken over the coin tosses of all algorithms and adversaries):*

1. *Completeness. Let a nonuniform polynomial-time adversary $A$ be given. Let ($crs$, $simtd$, $extd$) $\leftarrow K(1^\eta)$. Let ($l$, $t$, $m$) $\leftarrow A(1^\eta, crs)$. Let ($ss_1$, $ss_2$, ..., $ss_l$) $\leftarrow SCon(l, t, m, crs)$, then with overwhelming probability in $\eta$, $SCom(l, t, ss_{i_1}, ss_{i_2}, ..., ss_{i_t}, crs) = m$.*
2. *Soundness. Let a nonuniform polynomial-time adversary $A$ be given. Consider the following experiment parameterized by a bit $c$: Let ($crs$, $simtd$,*

*extd) $\leftarrow K(1^\eta)$. Let (l, t, m) $\leftarrow A^{SCon(.^*),SCom(.^*)}(1^\eta$, crs, simtd). Then let ($ss_1$, $ss_2$, ..., $ss_l$) $\leftarrow$ SCon(l, t, m, crs) if c=0 and ($ss_1$, $ss_2$, ..., $ss_l$) $\leftarrow$ S(l, t, simtd) if c=1. Let guess $= A^{SCon(.^*),SCom(.^*)}$(crs, simtd, $ss_{i_1}$, $ss_{i_2}$, ..., $ss_{i_r}$) with $r < t$. Let $P_c(\eta)$ denote the following probability: $P_c(\eta)$ $:= Pr[guess = m]$, then $|P_0(\eta) - P_1(\eta)|$ is negligible.*

3. *Length-regularity. Let polynomial-size circuit sequence $\widetilde{C}(.^*)$, and secret share parameters $m_1$, $m_2$ be given such that $|m_1| = |m_2|$. Let (crs, simtd) $\leftarrow$ $K(1^\eta)$. Let ($ss_1$) $\leftarrow$ SCon(l, t, $m_1$, crs) and ($ss_2$) $\leftarrow$ SCon(l, t, $m_2$, crs). Then $|ss_1| = |ss_2|$ holds with probability 1.*

Since the applied pi-calculus only has semantics in the symbolic model (without probabilities and without the notion of a computational adversary), we need to introduce a notion of computational execution for symbolic protocol.

Our computational implementation of a symbolic protocol is a probabilistic polynomial-time algorithm that expects as input the symbolic protocol $\Pi$, a set of deterministic polynomial-time algorithms $A$ for the constructors and destructors in $\Pi$, and a security parameter $k$. This algorithm executes the protocol by interacting with a computational adversary. In the operational semantics of the applied calculus, the reduction order is non-deterministic. This non-determinism is resolved by letting the adversary determine the order of the reduction steps. The computational execution sends the process to the adversary and expects a selection for the next reduction step.

**Definition 5.** *(Computational implementation of the symbolic model) We require that the computational implementation A of the symbolic model M has the following properties:*

1. *A is an implementation of M (in particular, all functions $A_f$ ($f \in C \bigcup D$) are polynomial-time computable). For bitstring m, Type(m) denotes the type of m.*
2. *There are disjoint and efficiently recognizable sets of bitstrings representing the types nonces, and payload-strings. The set of all bitstrings of type nonce we denote $Nonces_k$.(Here and in the following, k denotes the security parameter.)*
3. *The functions $A_{SS_{l,t}}$ are length-regular. We call an function f length regular if $|m_i| = |m_i'|$ for i = 1, 2, ..., n implies $|f(m_i)| = |f(m_i')|$. All $m \in Nonces_k$ have the same length.*
4. *$A_N$ for $N \in Nonce_{ag} \bigcup Nonce_{adv}$ returns a uniformly random $r \in Nonces_k$.*
5. *For all m, the image of $A_{SS_{l,t}}(m, r)$ is the sequence of the type $<$ secret share, $Type(m) >$.*
6. *For all m, the image of $A_{COE_{l,t,i}}(A_{SS_{l,t}}(m, r))$ is of the type $<$ secret share, $Type(m) >$.*
7. *For all $m_1, ..., m_t \in \{0,1\}^*$, if $m_1 = A_{COE_{l,t,i_1}}(A_{SS_{l,t}}(m, r))$, ..., $m_t = A_{COE_{l,t,i_t}}(A_{SS_{l,t}}(m, r))$, we have $A_{Combin_{l,t}}(m_1, ..., m_t) = m$. Else, $A_{Combin_{l,t}}(m_1, ..., m_t) = \perp$.*

**Definition 6.** *(Computational pi-calculus execution). Let $\Pi$ be a closed process, A be a computational implementation of the symbolic model. Let Adv be an* *interactive machine called the adversary. We define the computational pi-calculus execution as an interactive machine $Exec_{\Pi,A}(1^k)$ that takes a security parameter $k$ as argument and interacts with Adv:*

1. *Start. Let $P$ be obtained from $\Pi$ by deterministic $\alpha$-renaming so that all bound variables and names in $P$ are distinct. Let $\eta$ and $\mu$ be a totally undefined partial functions from variables and names, respectively, to bitstrings. Let $a_1, a_2, ..., a_n$ denote the free names in $P$. For each $i$, pick $r_i \in Nonces_k$ at random. Set $\mu := \mu \bigcup \{a_1 := r_1, a_2 = r_2, ..., a_n := r_n\}$. Send $(r_1, r_2, ..., r_n)$ to Adv.*
2. *Main loop. Send $P$ to the adversary and expect an evaluation context $E$ from the adversary. Distinguish the following cases:*
   *(a) $P = E[M(x) : P_1]$: Request two bitstrings $c$, $m$ from the adversary. If $c = ceval_{\eta,\mu}(M)$, set $\eta := \eta \bigcup \{x := m\}$ and $P := E[P_1]$.*
   *(b) $P = E[va.P_1]$: Pick $r \in Nonces_k$ at random, set $P := E[P_1]$ and $\mu := \mu(a := r)$.*
   *(c) $P = E[\widetilde{M}(N).P_1][M_2(x).P_2]$: If $ceval_{\eta,\mu}(M_1) = ceval_{\eta,\mu}(M_2)$, then set $P := E[P_1][P_2]$ and $\eta := \eta \bigcup \{x := ceval_{\eta,\mu}(N)\}$.*
   *(d) $P = E[let\ x\ =\ D\ in\ P_1\ else\ P_2]$: If $m := ceval_{\eta,\mu}(D)\bot$, set $\eta := \eta \bigcup \{x := m\}$ and $P := E[P_1]$; Otherwise set $P := E[P_2]$.*
   *(e) $P = E[!Q]$: Let $Q'$ be obtained from $Q$ by deterministic $\alpha$-renaming so that all bound variables and names in $Q'$ are fresh. Set $P := E[Q'|!Q]$. $P = E[\widetilde{M}(N).P_1]$: Request a bitstring $c$ from the adversary. If $c = ceval_{\eta,\mu}(M)$, set $P := E[P_1]$ and send $ceval_{\eta,\mu}(N)$ to the adversary.*
   *(f) In all other cases, do nothing.*

*For any interactive machine Adv, we define $Exec_{\Pi,A,Adv}(1^k)$ as the interaction between $Exec_{\Pi,A}(1^k)$ and Adv; the output of $Exec_{\Pi,A,Adv}(1^k)$ is the output of Adv.*

In the preceding section, we have described the trace properties and the uniformity properties involving LSSS proofs. We firstly formulate our soundness result for trace properties, Namely, with overwhelming probability, a computational trace of computational pi-calculus execution is a computational instantiation of some symbolic Dolev-Yao trace.

We construct a interactive machine called simulator, which simulates against an adversary *Adv* the execution *Exec* while actually interacting with *SExec*. The definition of such a simulator based on computational implementation $A$ will be used for the definition for computational soundness for trace properties in the following.

**Definition 7.** *(Hybrid pi-calculus execution) The simulator $Sim_A$ based on computational implementation $A$ is constructed as follows: whenever it gets a term from the protocol, it constructs a corresponding bitstring and sends it to the adversary, and when receiving a bitstring from the adversary it parses it and sends the resulting term to the protocol.*

1. *Constructing bitstrings is done using a function $\beta$, parsing bitstrings to terms using a function $\tau$. The simulator picks all random values and keys himself: For each protocol nonce $N$, he initially picks a bitstring $r_N$. He then translates, e.g., $\beta(N) := r_N$ and $\beta(SS_{l,t}(M,N)) := A_{SS_{l,t}}(r_M, r_N)$.*
2. *Translating back is also natural: Given $\widetilde{m} = \widetilde{r_N}$, we let $\tau(m_i) := COE_{l,t,i_1}($ $SS_{l,t}(M,N))$, and if $c$ is a LMPC result that can be decrypted as $m$ using $A_{Com}(\widetilde{m})$, we set $\tau(c) := M$.*

*Let $\Pi$ be a closed process, $Sim_A$ be a simulator based on computational implementation of the symbolic model A. Let Adv be an interactive machine called the adversary, we define the hybrid pi-calculus execution as an interactive machine $Exec_{\Pi,Sim_A}(1^k)$ that takes a security parameter $k$ as argument and interacts with Adv. We also define $Exec_{\Pi,Sim_A,Adv}(1^k)$ as the interaction between $Exec_{\Pi,Sim_A}(1^k)$ and Adv; the output of $Exec_{\Pi,Sim_A,Adv}(1^k)$ is the output of Adv. .*

We stress that the simulator $Sim$ does not have additional capabilities compared to a usual adversary against $Exec$. We then give the definition of the computational soundness for trace properties.

**Definition 8.** *(Computational soundness for trace properties) Let A be a computational implementation of the symbolic model and $Sim_A$ be a simulator. If for every closed process $\Pi$, A has to satisfy the following two properties:*

1. *Indistinguishability: $Exec_{\Pi,A}(1^k) \approx Exec_{\Pi,Sim_A}(1^k)$, which means the hybrid execution is computationally indistinguishable from the computational execution with any adversary.*
2. *Dolev-Yaoness: The simulator $Sim_A$ never (except for negligible probability) sends terms $t$ to the protocol with $S \mapsto t$ where $S$ is the list of terms $Sim_A$ received from the protocol so far.*

*then A is a computationally sound model for trace properties.*

Further, we give the definition of the computational soundness for uniformity properties. We rely on the notion of termination-insensitive computational indistinguishability (tic-indistinguishability) to capture that two protocols are indistinguishable in the computational world [22].

**Definition 9.** *(Tic-indistinguishability) Given two machines $M$, $M'$ and a polynomial $p$, we write $Pr[(M \mid M') \Downarrow_{p(k)} \; x]$ for the probability that the interaction between $M$ and $M'$ terminates within $p(k)$ steps and $M'$ outputs $x$. We call two machines A and B termination-insensitively computationally indistinguishable for a machine Adv ($A \approx_{tic}^{Adv} B$) if for for all polynomials $p$, there is a negligible function $\eta$ such that for all $z$, $a$, $b \in [0,1]^*$ with $a \neq b$,*

$Pr[(A(k) \mid Adv(k)) \Downarrow_{p(k)} \; a] + Pr[(B(k) \mid Adv(k)) \Downarrow_{p(k)} \; b] \preceq 1 + \eta(k)$

*Here, $z$ represents an auxiliary string. Additionally, we call A and B termination-insensitively computationally indistinguishable $A \approx_{tic} B$ if we have $A \approx_{tic}^{Adv} B$ for all polynomial-time machines Adv.*

Based on tic-indistinguishability, the definition of the computational soundness for uniformity properties is given as follows.

**Definition 10.** *(Computational soundness for uniformity properties) Let $A$ be a computational implementation of the symbolic model and $Sim_A$ be a simulator based on $A$. If $A$ is a computational soundness model for trace properties, and for every uniform bi-process $\varPi$, $Exec_{left(\varPi),Sim_A}(1^k) \approx_{tic} Exec_{right(\varPi),Sim_A}(1^k)$, then $A$ is a computational soundness model for uniformity properties.*

**Lemma 1.** *Assume that the encryption scheme $AE$, the signature scheme $SIG$, and the ISSS proof system $\varUpsilon_{LSSS}$ satisfy the requirements in definition 4. ($AE$, $SIG$, $\varUpsilon_{LSSS}$) is a computationally sound model for trace and uniformity properties.*

Proof. See Appendix

We now proceed to construct a generic ideal functionality $I_{sid,l,t,F}$ that serves as an abstraction of LSSS-based MPC. This construction is parametric over the session identifier $sid$, and the function $F$ to be computed. The ideal functionality receives the (secret) input message of party $i$ from port $in^e_{i,sid}$ along with a session identifier. The input message is stored in the variable $x_i$ and the state $state_i$ of $i$ is set to input. Both the session identifier and the length of the received message are leaked to the adversary on port $out^a_{i,sid}$.

**Definition 11.** *(Ideal model for LMPC) We construct an interactive polynomial-time machine $I_{sid,l,t,F}$, called the LMPC ideal functionality, which is parametric over a session identifier $sid$, LSSS threshold $(l,t)$, and a poly-time algorithm $F$. Initially, the variables of $I_{sid,l,t,F}$ are instantiated as follows: $\forall i \in [1,n]$ : $state_i := input$, $r_i := 1$. Upon an activation with message $m$ on port $p$, $I_{sid,l,t,F}$, behaves as follows.*

1. *Upon $(in^e_{i,sid}(m, sid'))$ If $sid = sid'$ and $state_i = input$, then set $state_i := compute$ and $x_i := m$. If $state_i = input$, then send $(sid', |m|)$ on port $in^a_{i,sid}$.*
2. *Upon $(in^a_{i,sid}(deliver, sid'))$, if $\forall j \in [1;n]$ : $state_j = compute$ and $r_i = r_j$ , then compute $(y_1, y_2, ..., y_n) \leftarrow SCon(F(x_1, x_2, ..., x_n))$ and $\forall j \in [1, n]$ : $state_j := deliver$. If $state_i = deliver$, set $r_i := r_i + 1$, $state_i := input$ and send $y_i$ on port $out^e_{i,sid}$.*

For defining computational computational soundness for uniformity properties with ideal model $I_{sid,l,t,F}$, we modify the scheduling simulator $Sim$ and hybrid computational execution as follow. Simulator $Sim_{A,I}$ simulates against an adversary $Adv$ the execution $Exec$ while interacting with $SExec$, in which $A$ is a computational implementation of the symbolic model and $I$ is a family of LMPC ideal functionalities.

We also construct context, called $state_\gamma$. In our abstraction, every party of a LMPC can be in the following states: *init*, *input*, *compute*, and *deliver*. In the state *init*, the entire session is not initialized yet; in the state *input*, the party expects an input; in the state *compute*, the party is ready to start the main computation; and, in the state *deliver*, the party is ready to deliver secret share. These states are stored in a mapping state (which is maintained by $Sim_{A,I}$) such that $state_\gamma(i)$ is the state of party $i$ in the session.

**Definition 12.** *(Hybrid pi-calculus execution with Ideal model for LMPC) Let a mapping $state_\gamma(i)$ from internal session identifiers and party identifiers to states given. We assign a process to each state $state_\gamma$ .*

1. *Upon receiving the initial process $P$, Enumerate every occurrence $LMPC(l, t, F, sidc, adv, \widetilde{in})$ with an internal session identifier $\gamma$, and tag this occurrence $LMPC(l, t, F, sidc, adv, \widetilde{in})$ in $P$ with $\gamma$. Let initially delivery($\gamma$, i) := $false, andletstate_\gamma(i) := init$ for all $i \in [1, n]$. For any $\gamma$, let $corrupt(\gamma, i) :=$ true, if the corresponding $in_i$ in $LMPC(l, t, F, sidc, adv, \widetilde{in})$ is free; otherwise let $corrupt(\gamma, i) := false$. In addition store all channel names in the partial mapping $\mu$.*
2. *Main loop: Send $P$ to the adversary $Adv$. Then, expect an evaluation context $E$ . We distinguish the following cases for $E$.*
   - *(a) $E$ schedules the initialization and evaluation context is $P = E[LMPC(l, t, F, sidc, adv, \widetilde{in})]$: Set $state_\gamma(i) := init$ for all $i \in [1, n]$. The internal state of $I_{session_{id}(\gamma),l,t,F}$ is set to input.*
   - *(b) $E$ schedules an input to a corrupted party $i$ and evaluation context is $P = E[LMPC(l, t, F, sidc, adv, \widetilde{in})]$: Request a bitstring $m$ from the adversary. Check whether $m = (c, s, input, m_0)$, $session_{id}(\gamma) := s$ and $state_\gamma(i) := input$, If the execution accepts the channel name $c$, we proceed and check wether $\mu(in_i)$ is defined and $c = \mu(in_i)$; if $\mu(in_i)$ is not defined set $\mu(in_i) := c$. If the check fails, abort the entire simulation. Send $(m_0, s)$ to $I_{session_{id}(\gamma),l,t,F}$ upon port $in^e_{i,s}$, Set $state_\gamma(i) := compute$.*
   - *(c) $E$ schedules an input to an honest party $i$ and evaluation context is $P = E[\widetilde{c}(x, s).Q][LMPC(l, t, F, sidc, adv, \widetilde{in})]$: Check whether there is an $i \in [1, n]$ such that $\mu(c) = \mu(in_i)$, $state_\gamma(i) := input$, and $session_{id}(\gamma) := \mu(s)$. Send $(\mu(x), \mu(s))$ to $I_{session_{id}(\gamma),l,t,F}$ upon port $in^e_{i,s}$, Set $state_\gamma(i) := compute$.*
   - *(d) Start the main computation upon the first delivery command for a party $i$ and evaluation context: $P = E[LMPC(l, t, F, sidc, adv, \widetilde{in})]$: Check whether $state_\gamma(i) := compute$ for all $i \in [1, n]$. Set $state_\gamma(i) := deliver$ for all $i \in [1, n]$.*
   - *(e) The delivery command for a party $i$ is sent and evaluation context: $P = E[LMPC(l, t, F, sidc, adv, \widetilde{in})]$: Request a bitstring $m = (c, s, deliver)$ from the adversary. Check whether $s = session_{id}(\gamma)$, and $state_\gamma(i) = deliver$ and there is an $i \in [1, n]$ such that $\mu(in_i) = c$. Set $delivery(\gamma, i) := true$, $state_\gamma(i) = input$, receive $m', sid$ from $I_{session_{id}(\gamma),l,t,F}$ on port $out^e_{i,s}$ and forward $m'$ to adversary.*
   - *(f) The output of party $i$ is delivered to an honest party and evaluation context: $P = E[c(x).Q][LMPC(l, t, F, sidc, adv, \widetilde{in})]$: Check whether there is an $i$ such that $\mu(c) = \mu(in_i)$, and $state_\gamma(i) = deliver$. Set $delivery(\gamma, i) := true$, $state_\gamma(i) = input$, receive $m', sid$ from $I_{session_{id}(\gamma),l,t,F}$ on port $out^e_{i,session_{id}(\gamma)}$ and set $\mu(x) = m'$.*

*Let $\Pi$ be a closed process, $Sim_{A,I}$ be a simulator based on computational implementation of the symbolic model $A$ and $I$ be a family of Ideal model for LMPC. Let Adv be an interactive machine called the adversary, we define the hybrid pi-calculus execution as an interactive machine $Exec_{\Pi,Sim_{A,I}}(1^k)$ that takes a security parameter $k$ as argument and interacts with Adv. We also define $Exec_{\Pi,Sim_{A,I},Adv}(1^k)$ as the interaction between $Exec_{\Pi,Sim_{A,I}}(1^k)$ and Adv; the output of $Exec_{\Pi,Sim_{A,I},Adv}(1^k)$ is the output of Adv.*

Then we can give the definition of Computational soundness for trace and uniformity properties with Ideal model for LMPC.

**Definition 13.** *(Computational sound ideal model for LMPC) Let $I$ be a family of ideal model for LMPC. If for every closed process $\Pi$, and computational implementation of the symbolic model $A$, $Exec_{\Pi,Sim_{A,I}}(1^k) \approx Exec_{\Pi,Sim_A}$, then $I$ is a computationally sound ideal model for LMPC.*

We can get the computation soundness result for ideal model for LMPC.

**Lemma 2.** *Assume the family of ideal model for LMPC $I$ satisfy the requirement in definition 13, $I$ is a computationally soundness ideal model for LMPC.*

Proof. See appendix.

## 3.2 Computational soundness results

We now state the main computational soundness result of this work: the robust safety of a process (specifically uniformity properties) using non-interactive primitives and our LMPC abstraction carries over to the computational setting, as long as the non-interactive primitives are computationally sound. This result ensures that the verification technique from Section 3 provides computational safety guarantees. We stress that the non-interactive primitives can be used both within the LMPC abstractions and within the surrounding protocol.

In order to realize the definition of computation soundness for uniformity properties in UC framework, we firstly give a novel definition for the security requirement of UC realization which is stronger that original definition in UC framework.

**Definition 14.** *(UC realization) Let $I$ be an ideal functionality and let $\Pi$ be an n-party protocol, $A$ be a computational implementation of the symbolic model. We say that real protocol $\rho$ securely realizes $I$ if for any adversary Adv, $Exec_{\Pi,Sim_{A,I}}($ $1^k) \approx Exec_{\Pi,Sim_{A,\rho}}$.*

**Lemma 3.** *Assume that enhanced trapdoor permutations exists. Then for all ideal function for LMPC $I$ which security requirements in definition 4, there exists a family of non-trivial protocols $\rho$in the CRS-model that UC realizes $I$ in the presence of malicious, static adversaries.*

Proof. See Appendix.

Finally, we can get the computation soundness result for our abstraction for LMPC.

**Theorem 1.** *Assume that the encryption scheme AE, the signature scheme SIG, and the ISSS proof system $\Upsilon_{LSSS}$ satisfy the security requirements in definition 4, also the family of ideal model for LMPC I satisfy the requirement in definition 13, there exists a family of non-trivial protocol ρin the CRS-model that UC realizes I.*

Proof. See Appendix.

## 4 Conclusions

We have presented the first computational soundness theorem for multi-party computation based on linear secret-sharing scheme (LMPC). This allows to analyze protocols in a simple symbolic model supporting encryptions, signatures, and secret-sharing schemes; the computational soundness theorem then guarantees that the uniformity properties shown in the symbolic model carry over to the computational implementation.

## References

[1] Shamir, A.: How to share a secret. Communications of the ACM. 22(11), 612–613 (1979).

[2] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, 427–437 (1987).

[3] Shoup, V.: Practical threshold signatures. In Proceedings of the EUROCRYPT 2000. Springer-Verlag, 207–220 (2000).

[4] He, A. J., Dawson, E.: Multistage secret sharing based on one-way function. Electronics Letters. 30(9), 1591–1592 (1994).

[5] Chien, H.-Y., Tseng, J. K.: A practical (t, n) multi-secret sharing scheme. IEICE Transactions on Fundamentals of Electronics. Communications and Computer 83-A, 12(2000), 2762–2765 (2000).

[6] Shao, J., Cao, Z. F.: A new efficient (t,n) verifiable multi-secret sharing (VMSS) based on YCH scheme. Applied Mathematics and Computation. 168(1), 135–140 (2005).

[7] Zhao, J., Zhang, J., Zhao, R.: A practical verifiable multi-secret sharing scheme. Computer Standards and Interfaces. 29(1), 138–141 (2007).

[8] Yang, C. C., Chang, T. Y., Hwang, M. S.: A (t, n) multi-secret sharing scheme. Applied Mathematics and Computation. 151, 483–490 (2004).

[9] Cramer, R., Damgard, I., and Maurer, U.: General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. In Proceedings of EUROCRYPT '00, Springer LNCS. May 2000.

[10] Abadi, M., Baudet, M., and Warinschi, M.: Guessing attacks and the computational soundness of static equivalence. In Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS). Springer, Volume 3921 of Lecture Notes in Computer Science, 398-412 (2006).

[11] Abadi, M., and Fournet, C.: Mobile values, new names, and secure communication. In Proceedings of the 28th Symposium on Principles of Programming Languages (POPL), ACM Press. 104–115 (2001).

[12] Abadi, M. and Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology. 15(2), 103–127 (2002).

[13] Backes, M., Maffei, M., Mohammadi, E.: Computationally Sound Abstraction and Verification of Secure Multi-Party Computations. In Proceedings of ARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), 2010.

[14] Backes, M. Hofheinz, D. and Unruh, D.: A general framework for computational soundness proofs or the computational soundness of the applied pi-calculus. IACR ePrint Archive 2009/080, 2009.

[15] Backes, M., Bendun, F., Unruh, D.: Computational Soundness of Symbolic Zero-knowledge Proofs: Weaker Assumptions and Mechanized Verification. In Proceedings of Principles of Security and Trust: Second International Conference (POST 2013). Springer, 206–225 (2013).

[16] Backes, M., Malik, A., Unruh, D.: Computational Soundness without Protocol Restrictions. CCS. ACM Press. 699–711 (2012).

[17] Kusters, R., Tuengerthal, M.: Computational Soundness for Key Exchange Protocols with Symmetric Encryption. In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS). ACM Press. 91–100 (2009).

[18] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A.: Universally composable two-party and multiparty secure computation. In Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), ACM Press. 494–503 (2002).

[19] Comon-Lundh, H., Cortier, V.: Computational soundness of observational equivalence. In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS), ACM Press. 109–118 (2008).

[20] Comon-Lundh, H., Cortier, V., Scerri, G.: Security Proof with Dishonest Keys. In Proceedings of Principles of Security and Trust: Second International Conference (POST 2012), Springer Press. 149–168 (2012).

[21] Canetti, R., Herzog: Universally Composable Symbolic Security Analysis. Journal of Cryptology. 24(1), 83–147 (2011).

[22] Backes, M., Mohammadi, E., Rung, T.: Bridging the Gap from Trace Properties to Uniformity. In Proceedings of Principles of Security and Trust: Second International Conference (POST 2014), Springer Press (2014).

[23] Canetti, R., Feige, U., Goldreich, O. and Naor, m.: Adaptively Secure Multi-Party Computation, In Proceedings of 28th STOC. 639–648 (1996).

[24] Canetti, R., Fischlin, M.: Universally Composable Commitments. In Proceedings of CRYPTO'01, Springer-Verlag. LNCS 2139, 19–40 (2001).

[25] Canetti, R., Rabin, T.: Universal Composition with Joint State. Cryptology ePrint Archive. Report 2002/047, http://eprint.iacr.org/ (2002).