Universally Composable Symbolic Analysis for Two-Party Protocols based on Homomorphic Encryption

Morten Dahl and Ivan Damgård

Aarhus University, Denmark *

Abstract. We consider a class of two-party function evaluation protocols in which the parties are allowed to use ideal functionalities as well as a set of powerful primitives, namely commitments, homomorphic encryption, and certain zero-knowledge proofs. With these it is possible to capture protocols for oblivious transfer, coin-flipping, and generation of multiplication-triples.
We show how any protocol in our class can be compiled to a symbolic representation expressed as a process in an abstract process calculus, and prove a general computational soundness theorem implying that if the protocol realises a given ideal functionality in the symbolic setting, then the original version also realises the ideal functionality in the standard computational UC setting. In other words, the theorem allows us to transfer a proof in the abstract symbolic setting to a proof in the standard UC model.
Finally, we have verified that the symbolic interpretation is simple enough in a number of cases for the symbolic proof to be partly automated using the ProVerif tool.

Key words: Cryptographic protocols, Security analysis, Symbolic analysis, Automated analysis, Computational soundness, Universal composition, Homomorphic encryption.

1 Introduction

Giving security proofs for cryptographic protocols is often a complicated and error-prone task, and there is a large body of research targeted at this problem using methods from formal analysis [AR02,BPW03,CH06,CC08,CKW11]. This is interesting because the approach could potentially lead to automated or at least computer-aided (formal) proofs of security.

It is well known that the main difficulty with formal analysis is that it is only feasible when enough details about the cryptographic primitives have been

* The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, and also from the CFEM research centre (supported by the Danish Strategic Research Council) within which part of this work was performed.

abstracted away, while on the other hand this abstraction may make us "forget" about issues that make an attack possible. One solution is to show once and for all that a given abstraction is *computational sound*, which loosely speaking means that for any protocol, if we know there are no attacks on its abstract *symbolic* version then this (and some appropriate complexity assumption) implies there are no attacks on the original *computational* version. Such soundness theorems are known in some cases (see related work), in particular for primitives such as public-key encryption, symmetric encryption, signatures, and hash functions.

Another issue with formal analysis is how security properties should be specified. Traditionally this has been done either through trace properties or "strong secrecy" where two instances of the protocol running on different values are compared to each other[1]. This approach can be used to specify security properties such as authenticity and key secrecy. However, it is much less clear how it can capture security of protocols such as oblivious transfer where players take input from the environment. In the cryptographic community it is standard to give simulations-based definitions of security for such protocols, yet this approach have so far only received little attention in formal analysis.

Finally, making protocol (and in particular system) analysis feasible in general requires some way of breaking the task into smaller components which may be analysed independently. While also this has been standard in the cryptographic community for a while (in the form of, e.g., the UC framework [Can01]) it has not yet received much attention in the symbolic community (but see [CH06] for an exception).

1.1 Our Results

In this paper we make progress on expanding the class of protocols for which a formal analysis can be used to show security in the computational setting. We are particularly interested in two-party function evaluation protocols and the primitives used by many of these, namely homomorphic public-key encryption, commitments, and certain zero-knowledge proofs. We aim for proofs of UC security against an active adversary where one party may be (statically) corrupted.

Protocol model. Besides the above primitives protocols are also allowed to use ideal functionalities and communicate over authenticated channels. We put some restrictions on how the primitives may be used. First, whenever a player sends a ciphertext he actually sends a package which also contains a zero-knowledge proof that the sender knows how the ciphertext was constructed: if the ciphertext was made from scratch then he knows the plaintext and randomness used, and if he constructed it from other ciphertexts using the homomorphic property then he knows randomness that "explains" the ciphertext as a function of

[1] For strong secrecy one runs the same protocol on two fixed but different inputs (or with one instance patched to give an independent output) and then ask if it is possible to tell the difference between the two executions. This can for instance be used to argue that a key-exchange protocol is independent of the exchanged key given only the transmitted messages.

that randomness and ciphertexts that were already known. We make a similar assumption on commitments and allow also zero-knowledge proofs that committed values relate to encrypted values in a given way. Second, we assume that honest players use the primitives in a black-box fashion, i.e. an honest player can run the protocol using a (private) "crypto module" that holds all his keys and handles encryption, decryption, commitment etc. This means that all actions taken by an honest player in the protocol may depend on plaintext sent or received but not, for instance, on the binary representation of ciphertexts. We emphasise that we make no such restriction on the adversary.

We believe that the assumptions we make are quite natural: it is well known that if a player provides input to a protocol by committing to it or sending an encryption then we cannot prove UC security of the protocol unless the player proves that he knows the input he provides. Furthermore, active security usually requires players to communicate over authenticated channels and prove that the messages they send are well-formed. We stress, however, that our assumptions do not imply that an adversary must be semi-honest; for instance, our model does not make any assumptions on what type and relationship checks the protocol must perform, nor on the randomness distributions used by a corrupted player.

Security properties. We use *ideal functionalities* and *simulators* to specify and prove security properties. More concretely, we say that a *protocol ϕ is secure* (with respect to the ideal functionality $\mathcal{F}$) if no adversary can tell the difference between interacting with ϕ and interacting with $\mathcal{F}$ and simulator *Sim*, later written $\phi \sim \mathcal{F} \diamond Sim$ for concrete notions of indistinguishability. When this equivalence is satisfied we also say that the protocol *(UC) realises* the ideal functionality. We require that ideal functionalities only operate on plain values and do not use cryptography. Like honest players in protocols, our simulators will only use the primitives and their trapdoors in a black-box fashion which allows us to specify them on an abstract level.

Proof technique. Our main result is quite simple to state on a high level: given a protocol ϕ, ideal functionality $\mathcal{F}$, and simulator *Sim*, we show how these may be compiled to symbolic versions such that if we are given a proof *in the symbolic world* that ϕ realises $\mathcal{F}$ then it follows that ϕ realises $\mathcal{F}$ *in the usual computational world* as well (assuming the crypto-system, commitment scheme and zero-knowledge proofs used are secure). As usual for UC security, we need to make a set-up assumption which in our case amounts to assuming a functionality that initially produces reference strings for the zero-knowledge proofs and keys for the crypto-system.

We arrive at our result as follows. First we define a simple programming language for specifying, on a rather high and abstract level, the programmes for honest players, ideal functionalities, and simulators that participate in a session of both the *real protocol* containing ϕ and the *ideal protocol* containing $\mathcal{F}$ and the simulator. The language is parameterised by the three corruption scenarios, indicated by which players are honest $\mathcal{H} \in \{AB, A, B\}$, and the class of protocols and properties we consider is implicitly defined as whatever can be described in

it. We call such a set of programmes a *system* and may hence fully describe real and ideal protocols by system triples $(Sys^{AB}, Sys^{A}, Sys^{B})$.

We then define three different ways of interpreting such systems:

- *Real-world interpretation* $\mathcal{RW}(Sys)$: Assuming concrete instantiations of the cryptographic primitives this interpretation produces from system Sys a set of interactive Turing machines that fits in the usual UC model. For instance, if Sys^{AB}_{real} is the system for a real protocol in the scenario where both players are honest then $\mathcal{RW}(Sys^{AB}_{real})$ contains two ITMs M_A, M_B executing the player programmes.
- *Intermediate interpretation* $\mathcal{I}(Sys)$: This interpretation also produces a set of ITMs fitting into the UC model, but does not use concrete cryptographic primitives. Instead we postulate an ideal functionality $\mathcal{F}_{aux}$ that receives all calls from all parties to cryptographic functions and returns handles to objects such as encrypted plaintexts while storing these plaintexts in its memory. Players then send such handles instead of actual ciphertexts and commitments. In this interpretation, the adversary is limited to a certain benign cryptographic behaviour as he too can only access cryptographic objects through $\mathcal{F}_{aux}$.
- *Symbolic interpretation* $\mathcal{S}(Sys)$: This interpretation closely mirrors the intermediate interpretation but instead produces a set of *processes* described in a well-known process calculus.

Having defined these interpretations we define notions of equivalence of systems in each representation: $\mathcal{RW}(Sys_1) \overset{c}{\sim} \mathcal{RW}(Sys_2)$ means that no polynomial time environment can distinguish the two cases given only the public and corrupted keys, and may for instance be used to capture that a protocol UC-securely realises $\mathcal{F}$ in the standard sense; for the intermediate world $\mathcal{I}(Sys_1) \overset{c}{\sim} \mathcal{I}(Sys_2)$ means the same but in the $\mathcal{F}_{aux}$-hybrid model; finally, $\mathcal{S}(Sys_1) \overset{s}{\sim} \mathcal{S}(Sys_2)$ means the two processes are *observationally equivalent* in the standard symbolic sense.

We then prove two soundness theorems stating first, that $\mathcal{I}(Sys_1) \overset{c}{\sim} \mathcal{I}(Sys_2)$ implies $\mathcal{RW}(Sys_1) \overset{c}{\sim} \mathcal{RW}(Sys_2)$ and second, that $\mathcal{S}(Sys_1) \overset{s}{\sim} \mathcal{S}(Sys_2)$ implies $\mathcal{I}(Sys_1) \overset{c}{\sim} \mathcal{I}(Sys_2)$, so that in order to prove UC security of a protocol it is now sufficient to show equivalence in the symbolic model and this is the part we may automate using e.g. ProVerif [BAF05].

Finally, we note that in some cases (in particular when both players are honest) it is possible to use a standard simulator construction and instead check a different symbolic criteria along the lines of previous work [CH06]. This removes the manual effort required in constructing simulators.

Analysis approach. Given the above, a protocol ϕ may be analysed as follows:

1. formulate in our model protocol ϕ and the ideal functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_n$ it uses as a triple $(Sys^{AB}_{real}, Sys^{A}_{real}, Sys^{B}_{real})$
2. likewise formulate the target ideal functionality $\mathcal{G}$ and suitable simulators as a triple $(Sys^{AB}_{ideal}, Sys^{A}_{ideal}, Sys^{B}_{ideal})$
3. show in the symbolic model that $\mathcal{S}(Sys^{\mathcal{H}}_{real}) \overset{s}{\sim} \mathcal{S}(Sys^{\mathcal{H}}_{ideal})$ for all three $\mathcal{H}$

4. the soundness theorem then gives $\mathcal{RW}(Sys^{\mathcal{H}}_{real}) \overset{c}{\sim} \mathcal{RW}(Sys^{\mathcal{H}}_{ideal})$, and in turn that ϕ realises $\mathcal{G}$ under static corruption

Note that as usual in the UC framework we only need to consider one session of the protocol since the compositional theorem guarantees that it remains secure even when composed with itself a polynomial number of times. Note also that we may apply our result to a broader class of protocols through a hybrid-symbolic approach where the protocol in question is broken down into several sub-protocols and ideal functionalities analysed independently either within our framework or outside in an ad-hoc setting (possibly using other primitives).

We have tried to make the symbolic model suitable for automated analysis using current tools such as ProVerif, and although our approach requires the manual construction of a simulator for the symbolic version of the protocol, this is usually a very simple task. As a case study we have carried out a full analysis of the OT protocol from [DNO08] in the full version of this paper[2], where we also illustrate compositional analyses through a coin-flipping protocol, and that the model may express the preprocessing phase of the multi-party computation protocol in [BDOZ11][3].

1.2 Related Work

The main area of related work is *computational soundness* as discussed below (see also [CKW11] for an in-depth survey of this area), but there is also a large body of work on *symbolic modelling of security properties* which at this point has not given much attention to the simulation-based paradigm (see [DKP09,BU13] for two examples without computational soundness), as well as a substantial amount of work on the *direct approach* where the symbolic model is altogether avoided but instead used as inspiration for creating a computational model easier to analyse; this latter line of work includes [Bla08,BGHB11,MRST06,DDMR07] and while it is more expressive than the symbolic approach we have taken here, our focus has been on abstracting and automating as much as possible.

Computational soundness. The line of work started by Backes et al. in [BPW03] and known as "the BPW approach" gives an ideal cryptographic library based on the ideas behind abstract Dolev-Yao models. The library is responsible for all operations that players and the adversary want to perform (such as encryption, decryption, and message sending) with every message being kept in a database by the library and accessed only through handles. Using the framework for reactive simulatability [PW01] (similar to the UC framework) the ideal library is realised using cryptographic primitives. This means that a protocol may be analysed relative to the ideal library yet exhibit the same properties when using the realisation instead. The original model supporting nested nonce generation, public-key

[2] Available at http://eprint.iacr.org/2013/296.

[3] Due to limitations on expressibility of probabilistic choice in our model we analyse a slight variant of the protocol where the verification of the generated triples is pushed into the online phase.

encryption, and MACs has later been extended to support symmetric encryption [BP04] and a simple form of homomorphic threshold-encryption [LN08] allowing a single homomorphic evaluation. The approach has been used to analyse protocols for trace-based security properties such as authentication and key secrecy [BP03,BP06].

Comparing our work to the BPW approach we see that the functionality $\mathcal{F}_{aux}$ in our intermediate model corresponds to the ideal cryptographic library, and the real-world operation modules to the realisation. The difference lies in the supported operations, namely our more powerful homomorphic encryption and simulation operations – the former allows us to implement several two-party functionalities while the latter allows us to express simulators for ideal functionalities within the model. This not only allows us to capture an entirely different class of indistinguishability-based security properties[4] (such as the standard assumptions on OT with static corruption) but also to do modular and hybrid-symbolic analysis. The importance of this was elaborated on in [Can08].

The next line of closely related work is that started by Canetti et al. in [CH06] and building on [MW04,BPW03] but adding support for modular analysis. They first formulate a programming language for protocols using public-key encryption and give both a computational and symbolic interpretation. They then give a mapping lemma showing that the traces of the two interpretations coincide, i.e. the computational adversary can do nothing that the symbolic adversary cannot also do (except with negligible probability). This is used to give symbolic criteria for realising authentication and key-exchange functionalities, and show that ProVerif may be used to automate the analysis of the original Needham-Schroeder-Lowe protocol (relative to authenticity) and two of its variants (relative to key-exchange). Later work [CG10] again targets key-exchange protocols but adds support for digital signatures, Diffie-Hellman key-exchanges, and forward security under adaptive corruption.

Most importantly, our approach has been that of not fixing the target ideal functionalities but instead letting it be expressible in the model (along with the realising protocol and simulator). Hence it is relatively straight-forward to analyse protocols implementing other functionalities than what we have done here, whereas adapting [CH06] to other classes of protocols requires manually finding and showing soundness of a symbolic criteria. It is furthermore not clear which functionalities may be captured by symbolic criteria expressed as trace properties and strong secrecy. In particular, the target functionalities of [CH06] and [CG10] do not take any input from the players nor provide any security guarantees when a player is corrupt, and hence the criteria do not need to account for these case. Again we also show soundness for a different set of primitives.

[4] In principle the BPW model could be used as a stepping-stone to analyse cases where the simulator may simply run the protocol on constants. However, the simulator is sometimes required to use trapdoors in order to extract information needed to simulate an ideal functionality in the simulation-based paradigm. These cases cannot be analysed with the operations of the BPW model.

The final line of related work is showing soundness of indistinguishability-based (instead of trace-based) properties. This was started by Comon-Lundh et al. in [CC08] and, unlike the two previous lines of work, aims at showing that if the symbolic adversary cannot distinguish between two systems in the symbolic interpretation then the computational adversary cannot do so either for the computational interpretation. [CC08] showed this for symmetric encryption and was continued in [CHKS12] for public-key encryption and hash functions.

Our work obviously relates in that we are also concerned about soundness of indistinguishability. Again the biggest difference is the choice of primitives, but also that our framework seems more suitable for expressing ideal functionalities and simulators: although mentioned as an application, their model does not appear to be easily adapted to capturing the typical structure of a composable analysis framework such as the UC framework (private channels are not allowed for instance). To this end the result is closer to what might be achieved through the BPW approach. Note that the work in [CHKS12] does not require computable parsing (as we do through the NIZK proofs). However, for secure function evaluation in the simulation-based paradigm some form of computational extraction is typically required in general.

The work in [BMM10] is also somewhat related in that they also aim at analysing secure function evaluation, namely secure multi-party computations (MPC). However, they instead analyse protocols using MPC as a primitive whereas we are interested in analysing the (lower-level) protocols realising MPC. Moreover, they are again limited to trace properties.

Organisation. The rest of the paper is organised in a "top-down" approach of progressively removing cryptography and bitstrings, and ending up with an highly idealised model. Section 2 specifies our protocol class including the interface of the operation modules. Section 3 gives the preliminaries for the real-world interpretation in Section 4. The intermediate and symbolic worlds are given in Section 5 and 6 respectively together with their soundness statements. Further details including definitions and proofs are given in the full version of this paper.

2 Protocol Model

The specific form of protocols introduced here is an essential part of our soundness result in that it characterises the class of protocols for which the result holds. The model is parameterised by a finite domain of *values* $\{V_n\}$, two finite sets of *types* $\{T_i\}, \{U_j\}$, and two finite sets of arithmetic *expressions* $\{e_k\} \subseteq \{f_\ell\}$ which for simplicity we often assume to be over four variables.

Programmes are given in a simple programming language allowing input, output, conditionals, and invocation of operations. We consider three kinds of programmes, *plain*, *player*, and *simulator*, differing in what operations they may use and whether or not they accept cryptographic packages.

Plain programmes, such as ideal functionalities, may only use operations

$$\mathsf{isValue}(x) \to b,\ \mathsf{eqValue}(v,w) \to b,\ \mathsf{inType}_U(v) \to b,\ \mathsf{inType}_T(v) \to b,$$
$$\mathsf{peval}_f(v_1,v_2,w_1,w_2) \to v,\ \mathsf{isConst}(x) \to b,\ \mathsf{eqConst}_c(x) \to b,$$
$$\mathsf{isPair}(x) \to b,\ \mathsf{pair}(x_1,x_2) \to x,\ \mathsf{first}(x) \to x_1,\ \mathsf{second}(x) \to x_2$$

where for instance $\mathsf{isValue}$ determines if a message is a value, inType_U if a value belongs to type U, peval_f evaluates expression f on the four values, pair forms a pairing, and first projects the first component of a pairing. Their input command aborts if any cryptographic package is received.

Player programmes, in addition to those of plain programmes, may also use operations

$$\mathsf{isComPack}(x) \to b,\ \mathsf{isEncPack}(x) \to b,\ \mathsf{isEvalPack}(x) \to b,$$
$$\mathsf{commit}_{U,ck,crs}(v,r) \to d,\ \mathsf{encrypt}_{T,ek,crs}(v,r) \to c,$$
$$\mathsf{eval}_{e,ek,ck,crs}(c_1,c_2,v_1,r_1,v_2,r_2) \to c,\ \mathsf{decrypt}_{dk}(c) \to v,$$
$$\mathsf{verComPack}_{U,ck,crs}(d) \to b,\ \mathsf{verEncPack}_{T,ek,crs}(c) \to b,$$
$$\mathsf{verEvalPack}_{e,ek,ck,crs}(c,c_1,c_2,[d_1,d_2]) \to b$$

to respectively determine: whether a message is a cryptographic package and its kind; form a new commitment package under their own commitment key and CRS using the value and randomness supplied, and with a proof of plaintext membership in type U[5]; form a new encryption package under either encryption key and their own CRS using the value and randomness supplied, and with a proof of plaintext membership in type T; form a new evaluation package under the encryption key of the inputs and their own commitment key and CRS, with a fresh ciphertext, a proof that it was created through homomorphic evaluation of expression e on inputs c_1, c_2, v_1, v_2, and commitments to v_1, v_2 under the randomness supplied[6]; decrypt a ciphertext under their own encryption key; and finally verify cryptographic packages under the specified keys and, in case of evaluation packages, that the correct ciphertexts and (optional) commitments were used. Their input command aborts on cryptographic packages not created under the commitment key and CRS of the other player.

Finally, simulator programmes instead use simulation versions of the player operations

$$\mathsf{simcommit}_{U,ck,simtd}(v,r) \to d,\ \mathsf{simencrypt}_{T,ek,simtd}(v,r) \to c,$$
$$\mathsf{simeval}_{e,ek,ck,simtd}(c_1,c_2,v_1,r_1,v_2,r_2) \to c,$$
$$\mathsf{simeval}_{e,ek,ck,simtd}(v,c_1,c_2,d_1,d_2) \to c$$

[5] Note that the NIZK proofs allow us to realise an ideal commitment functionality with opening despite no explicit opening operation for commitments.

[6] Note that as an artefact from wanting a symbolic model easier to analysis with available tools, operation eval_e (unlike commit_U and $\mathsf{encrypt}_T$) only takes r_1, r_2 for commitments d_1, d_2 as input, and not an r for re-randomisation of the resulting ciphertext; instead, the implementations will choose fresh randomness internally.

for an honest player (and no decryption operation), and operations

$$\mathsf{extractCom}_{extd}(d) \to v,\ \mathsf{extractEnc}_{extd}(c) \to v,$$
$$\mathsf{extractEval}_{1,extd}(c) \to v,\ \mathsf{extractEval}_{2,extd}(c) \to v$$

for a corrupt player. The operations for an honest player are similar to those of a player programme except that less checks are performed and proofs are simulated. The operations for a corrupt player allows the programme to extract the plaintext value of commitment and encryption packages, and the two plaintext values of commitments in evaluation packages, as long as they were created under the CRS of the corrupt player. Their input command behaves as for player programmes.

As an example, consider the OT protocol from [DNO08]. Intuitively, the receiver gets a bit b from the environment, encrypts it as c_b under his own encryption key, and sends c_b to the sender along with a proof that it really contains either 0 or 1. After checking the proof, the sender uses the homomorphic property to evaluate expression $sel(b, v_0, v_1) = (1 - b) \cdot v_0 + b \cdot v_1$ on the received ciphertext and the values v_0, v_1 given by the environment. He then sends the resulting ciphertext c_v back to the receiver along with a proof that it was constructed correctly. Finally, the receiver checks the proof to ensure that c_v was created using c_b, and outputs the decrypted value.

In our protocol model we may express the two players as the programmes in Figure 1 with sender P^S_{OT} on the left and receiver P^R_{OT} on the right. Under the three scenarios of static corruption the real protocol may then be described by system triple

$$\left(\quad P^S_{OT} \diamond Auth_{RS} \diamond Auth_{SR} \diamond P^R_{OT} \quad , \quad P^S_{OT} \quad , \quad P^R_{OT} \quad \right)$$

where the first system for when both players are honest also have one authenticated channel in each direction (and no ideal functionalities), and the next two systems for when one player is corrupted contain just the honest player programme. Likewise we may describe the ideal protocol with an ideal OT functionality and simulators in the protocol model, obtaining system triple

$$\left(\quad \mathcal{F}^{SR}_{OT} \diamond Sim^{SR,R}_{OT} \diamond Auth_{RS} \diamond Auth_{SR} \diamond Sim^{SR,S}_{OT} \quad , \right.$$
$$\left. \mathcal{F}^S_{OT} \diamond Sim^S_{OT} \quad , \quad \mathcal{F}^R_{OT} \diamond Sim^R_{OT} \quad \right)$$

with simulators that respectively run the protocol on dummy values, use extraction to obtain b, and use extraction to obtain v_0, v_1. In our case analysis we use ProVerif to conclude for each of the three cases that the two corresponding systems are indistinguishable.

Note that the input command $\mathrm{input}_{\mathcal{P}}[p : x]$ is specified with a set of ports $\mathcal{P}$ on which the programme is also listening but which will result in the programme aborting. The motivation for having these is that the symbolic soundness result requires that systems are *non-losing*, in the sense that whenever a programme

$\text{input}_{\emptyset}[\mathit{receive}_{RS} : c_b]$;
  if $\mathsf{verEncPack}_{bit,ek_R,crs_R}(c_b)$ then
  $\text{output}[\mathit{out}^S_{OT} : \texttt{getInput}]$;
$\text{input}_{\emptyset}[\mathit{in}^S_{OT} : (v_0, v_1)]$;
  if $\mathsf{isValue}(v_0)$ and $\mathsf{isValue}(v_1)$ then
  let $c_v \leftarrow \mathsf{eval}_{sel,R,S,S}(c_v, v_0, r_0, v_1, r_1)$;
  $\text{output}[\mathit{send}_{SR} : c_v]$;
stop

$\text{input}_{\emptyset}[\mathit{in}^R_{OT} : b]$;
  if $\mathsf{inType}_{bit}(b)$ then
  let $c_b \leftarrow \mathsf{encrypt}_{bit,ek_R,crs_R}(b, r)$;
  $\text{output}[\mathit{send}_{RS} : c_b]$;
$\text{input}_{\emptyset}[\mathit{receive}_{SR} : c_v]$;
  if $\mathsf{verEvalPack}_{sel,R,S,S}(c_v, c_b)$ then
  let $v_b \leftarrow \mathsf{decrypt}_{dk_R}(c_v)$;
  $\text{output}[\mathit{out}^R_{OT} : v_b]$;
stop

$$\text{where } \mathsf{eval}_{sel,R,S,S}(\dots) = \mathsf{eval}_{sel,ek_R,ck_S,crs_S}(\dots)$$
$$\text{and } \mathsf{verEvalPack}_{sel,R,S,S}(\dots) = \mathsf{verEvalPack}_{sel,ek_R,ck_S,crs_S}(\dots)$$

Fig. 1. Player programmes for OT sender (left) and receiver (right)

sends a message on a closed port p the receiving programme must also be listening on p. This also accounts for the atypical specification of the sender programme above; making it explicit ask the environment for its input by sending `getInput` means we may use $\mathcal{P} = \emptyset$ for all input commands, thereby simplifying the symbolic analysis.

3 Computational Model and Cryptographic Primitives

Our computational model is that of the UC framework as described in [Can01]. In this model ITMs in a network communicate by writing to each others tapes, thereby passing on the right to execute. In other words, the scheduling is token-based so that any ITM may only execute when it is holding the token. Initially the special *environment* ITM $\mathcal{Z}$ holds the token. When it writes on a tape of an ITM M in the network it passes on the token and M is now allowed to execute. If the token ever gets stuck it goes back to the environment.

For environment $\mathcal{Z}$, adversary $\mathcal{A}$, and network N, we write $\texttt{Exec}_{\mathcal{Z},\mathcal{A},N}(\kappa, z)$ for the random variable denoting the output bit (guess) of $\mathcal{Z}$ after interacting with $\mathcal{A}$ and N, and denote ensemble $\{\texttt{Exec}_{\mathcal{Z},\mathcal{A},N}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^\star}$ by $\texttt{Exec}_{\mathcal{Z},\mathcal{A},N}$. We may then compare networks as follows:

Definition 1 (Computational Indistinguishability). *Two networks of ITMs N_1 and N_2 are* computational indistinguishability *when no polynomial time adversary $\mathcal{A}$ may allow a polynomial time environment $\mathcal{Z}$ to distinguish between them with more than negligible probability, i.e. for all PPT $\mathcal{Z}$ and $\mathcal{A}$ we have $\texttt{Exec}_{\mathcal{Z},\mathcal{A},N_1} \overset{c}{\approx} \texttt{Exec}_{\mathcal{Z},\mathcal{A},N_2}$ which we write $N_1 \overset{c}{\sim} N_2$.*

By allowing different adversaries in the two networks we also obtain a notion of one network realising another, namely network N_1 *realises* network N_2 when,

for any PPT $\mathcal{A}$, there exists a PPT *simulator Sim* such that for all PPT $\mathcal{Z}$ we have $N_1 \overset{c}{\sim} N_2$.

We require the following primitives and security properties:

Commitment scheme. We assume two PPT algorithms $\mathbf{ComKeyGen}(1^\kappa) \to ck$ and $\mathbf{Com}_{ck}(V, R) \to D$ for key-generation and commitment, respectively. We require that the scheme is *well-spread*, *computationally binding* and *computationally hiding*. Intuitively, well-spread means that it is hard to predict the outcome of honestly generating a commitment.

Homomorphic encryption scheme. An *encryption scheme* is given by three PPT algorithms $\mathbf{EncKeyGen}(1^\kappa) \to (ek, dk)$, $\mathbf{Enc}_{ek}(V, R) \to C$, and $\mathbf{Dec}_{dk}(C) \to V$. A *homomorphic* encryption scheme furthermore contains a PPT algorithm $\mathbf{Eval}_{e,ek}(C_1, C_2, V_1, V_2, R) \to C$ for arithmetic expression $e(x_1, x_2, y_1, y_2)$ and randomness R for re-randomisation. We require that the scheme is *well-spread*, *correct*, *history hiding* (or *formula private*), and IND-CPA secure for the entire domain. Here, correct means that decryption almost always succeeds for well-formed ciphertexts, and history hiding that a ciphertext produced using $\mathbf{Eval}_{e,ek}$ is distributed as $\mathbf{Enc}_{ek}$ on the same inputs.

Non-Interactive Zero-Knowledge Proof-of-Knowledge scheme. For binary relation $\mathcal{R}$ we assume PPT algorithms $\mathbf{CrsGen}_{\mathcal{R}}(1^\kappa) \to crs$, $\mathbf{SimCrsGen}_{\mathcal{R}}(1^\kappa) \to (crs, simtd)$, and $\mathbf{ExCrsGen}_{\mathcal{R}}(1^\kappa) \to (crs, extd)$ for CRS generation, PPT algorithms $\mathbf{Prove}_{\mathcal{R},crs}(x, w) \to \pi$, $\mathbf{SimProve}_{\mathcal{R},simtd}(x) \to \pi$, and $\mathbf{Ver}_{\mathcal{R},crs}(x, \pi) \to \{0, 1\}$ for respectively generating, simulating, and verifying proofs π, and finally deterministic polynomial time algorithm $\mathbf{Extract}_{\mathcal{R},extd}(x, \pi) \to w$ for extracting witnesses. We require that such schemes are *complete*, *computational zero-knowledge*, and *extractable*, and assume instantiations for:

- $\mathcal{R}_U = \left\{ (x, w) \,\middle|\, D = \mathbf{Com}_{ck}(V, R) \wedge V \in U \right\}$ with $x = (D, ck), w = (V, R)$
- $\mathcal{R}_T = \left\{ (x, w) \,\middle|\, C = \mathbf{Enc}_{ek}(V, R) \wedge V \in T \right\}$ with $x = (C, ek), w = (V, R)$
- $\mathcal{R}_e = \left\{ (x, w) \,\middle|\, C = \mathbf{Eval}_{e,ek}(C_1, C_2, V_1, V_2, R) \wedge D_i = \mathbf{Com}_{ck}(V_i, R_i) \right\}$ with $x = (C, C_1, C_2, ek, D_1, D_2, ck)$ and $w = (V_1, R_1, V_2, R_2, R)$.

4 Real-world Interpretation

In the real-world model all messages sent between entities are annotated bit-strings BS of the following kinds: $\langle \texttt{value} : V \rangle$ and $\langle \texttt{const} : Cn \rangle$ for values and constants, $\langle \texttt{pair} : BS_1, BS_2 \rangle$ for pairings, and $[\texttt{comPack} : D, ck, \pi_U, crs]$, $[\texttt{encPack} : C, ek, \pi_T, crs]$, $[\texttt{evalPack} : C, C_1, C_2, ek, D_1, D_2, ck, \pi_e, crs]$ for commitment, encryption, and evaluation packages. In interpretation $\mathcal{RW}(Sys)$ of a system *Sys* each programme P is executed by ITM M_P with access to its own operation module $\mathcal{O}_P$ enforcing sanity checks on received messages and implementing the operations available to P as described in Section 2. These implementations follow straight-forwardly from the primitives.

The interpretation also contains a setup functionality $\mathcal{F}_{setup}$ connected to the operation modules of the cryptographic programmes. It is set to support either a real or an ideal protocol, is assumed to know the corruption scenario, and is responsible for generating and distributing the cryptographic keys and trapdoors, including leaking the public and corrupted keys to the adversary.

When a message is received by an M_P it is immediately passed to $\mathcal{O}_P$ which checks that every cryptographic package in it comes with a correct proof generated under the other player's CRS. The operation module also keeps a list σ of the ciphertexts received and generated by the player, so that it may enforce a policy of only accepting an evaluation package if it has first seen the ciphertexts it is supposedly constructed from, and rejecting certain ciphertexts that an honest player would never have produced and which cannot occur in the intermediate interpretation[7].

If the message was accepted by the operation module the machine gets back a reference through which it may access the message in the future. It then executes the operations as dictated by the programme and finally either halts or sends a message to another machine.

5 Intermediate Interpretation

The intermediate interpretation uses the same machines M_P for executing programmes as the real-world interpretation, however all operation modules and the setup functionality are now replaced with a single functionality $\mathcal{F}_{aux}$ offering operation implementations to the honest entities as well as a certain set of methods to the adversary. In effect, the cryptographic primitives and setup functionality has been replaced by a global memory with logical restrictions on how the adversary is allowed to access it.

All cryptographic messages passed around among the entities are uniformly random *handles* H of length κ associated to data objects in the global memory: commitment objects take form $(\texttt{com} : V, R, ck)$, encryption objects $(\texttt{enc} : V, R, ek)$, and proof objects[8] $(\texttt{proof}_U : H_D, ck, crs)$, $(\texttt{proof}_T : H_C, ek, crs)$, and $(\texttt{proof}_e : H_C, H_{C_1}, H_{C_2}, ek, H_{D_1}, H_{D_2}, ck, crs)$. Note that the ck, ek, crs here are simply constants chosen by $\mathcal{F}_{aux}$ and indicating the creator and owner of the objects. For packages we have objects $(\texttt{comPack} : H_D, ck, H_\pi, crs)$, $(\texttt{encPack} : H_C, ek, H_\pi, crs)$, and $(\texttt{evalPack} : H_C, H_{C_1}, H_{C_2}, ek, H_{D_1}, H_{D_2}, ck, H_\pi, crs)$.

The intermediate implementation of operations for honest entities follows the real-world implementation closely, yet of course using data objects instead of cryptographic bitstrings. One difference is that some guarantees are now provided by the model itself as a consequence of the adversary being limited in

[7] One example is if it receives two evaluation packages with the same C but with, say, different D_1; an honest player would have re-randomised the result thereby with overwhelming probability not produce the same C twice. As mentioned earlier, rejecting certain ciphertexts gives an easier-to-analyse symbolic interpretation.

[8] Note that proof objects do not have a randomness (or counter) component; we have gone with this option to simplify the symbolic model but it may easily be removed.

what he may do; for instance, it is not possible for him to construct packages with an invalid proof, and even adversarily evaluated ciphertexts are correctly re-randomised. This means that less checks are enforced through the σ list.

The methods offered to the adversary by $\mathcal{F}_{aux}$ essentially allows him to inspect and construct cryptographic packages, including decrypting ciphertexts for corrupted players, and compare arbitrary handles through a method $\mathsf{eq}(H, H') \to \{0, 1\}$. These methods are determined by what is needed by translator[9] $\mathcal{T}_\star$ in the soundness proof (see below and full paper).

5.1 Soundness of Intermediate Interpretation

Through a series of hybrid interpretations $\mathcal{T}[\mathcal{I}(Sys)]$, where leakage and influence ports of the authenticated channels are rewired to run through translator $\mathcal{T}$, we show that a real-world adversary cannot distinguish between $\mathcal{RW}(Sys)$ and $\mathcal{I}(Sys)$ for a well-formed system Sys.

Theorem 1 (Soundness of Intermediate Model). *Let Sys_1 and Sys_2 be two well-formed systems. If $\mathcal{I}(Sys_1) \overset{c}{\sim} \mathcal{I}(Sys_2)$ then $\mathcal{RW}(Sys_1) \overset{c}{\sim} \mathcal{RW}(Sys_2)$.*

Proof (overview). By a series of hybrid interpretations we first use the properties of the primitives to show that for any well-formed real or ideal protocol Sys we have $\mathcal{RW}(Sys) \overset{c}{\sim} \mathcal{T}_\star\big[\mathcal{I}(Sys)\big]$ for a constructed PPT translator $\mathcal{T}_\star$ using only the methods offered to the adversary by $\mathcal{F}_{aux}$. An important property here is that the identity of commitments and ciphertexts are preserved by the translation performed by each (hybrid-)translator. Next, by assumption no polynomially bounded ITM $\mathcal{Z}'$ can tell the difference between $\mathcal{I}(Sys_1)$ and $\mathcal{I}(Sys_2)$ using only the adversarial methods, and hence no $\mathcal{Z}' = \mathcal{Z} \diamond \mathcal{T}_\star$ for a polynomially bounded ITM $\mathcal{Z}$ can tell the difference either. The result then follows.

Corollary 1. *Let $(Sys^{\mathcal{H}}_{real})_{\mathcal{H}}$ specify a real protocol for ϕ and let $(Sys^{\mathcal{H}}_{ideal})_{\mathcal{H}}$ specify an ideal protocol with target functionality $\mathcal{F}$. If $\mathcal{I}(Sys^{\mathcal{H}}_{real}) \overset{c}{\sim} \mathcal{I}(Sys^{\mathcal{H}}_{ideal})$ for all three corruption cases $\mathcal{H}$ then ϕ is a realisation of $M_{\mathcal{F}}$ (with inlined operation module) under static corruption.*

[9] In UC-terms the translator is simply a simulator for $\mathcal{F}_{aux}$ used to show that the real-world interpretation is a realisation of the intermediate interpretation. However, we use this wording to avoid too much overload.

6 Symbolic Model and Interpretation

The symbolic model and interpretation is tailored to be a conservative approximation of the intermediate model and is based on the well-known dialect in [BAF05] of the applied-pi calculus [AF01], for which automated verification tools exist in the form of ProVerif.

We assume a modelling of the values v in the domain and a modelling of all constants plus $\mathbf{true}, \mathbf{false}, \mathbf{garbage}$. Let *names* $\mathcal{N}$ be a countable set of atomic symbols used to model randomness r, secret key material $dk, extd$, and ports p. A term t is then build from names, a countable set of variables $x, y, z, \ldots$, and constructor symbols

$$\mathbf{pair},\ \mathbf{ek},\ \mathbf{crs},\ \mathbf{com},\ \mathbf{enc},\ \mathbf{proof}_U,\ \mathbf{proof}_T,\ \mathbf{proof}_e,$$
$$\mathbf{comPack},\ \mathbf{encPack},\ \mathbf{evalPack}$$

where the three $\mathbf{proof}_{(\cdot)}$ constructors are unavailable to the adversary. The destructor symbols are

$$\mathbf{isValue},\ \mathbf{eqValue},\ \mathbf{inType}_U,\ \mathbf{inType}_T,\ \mathbf{isConst},\ \mathbf{eqConst}_c,\ \mathbf{equals},$$
$$\mathbf{isPair},\ \mathbf{first},\ \mathbf{second},\ \mathbf{isComPack},\ \mathbf{isEncPack},\ \mathbf{isEvalPack},$$
$$\mathbf{verComPack}_U,\ \mathbf{verEncPack}_T,\ \mathbf{verEvalPack}_e,\ \mathbf{eval}_e,\ \mathbf{peval}_f,$$
$$\mathbf{dec},\ \mathbf{extractCom},\ \mathbf{extractEnc},\ \mathbf{extractEval}_1,\ \mathbf{extractEval}_2,$$
$$\mathbf{ckOf}, \mathbf{ekOf}, \mathbf{crsOf}, \mathbf{comOf}, \mathbf{encOf}, \mathbf{encOf}_1, \mathbf{encOf}_2, \mathbf{comOf}_1, \mathbf{comOf}_2$$

where only $\mathbf{eval}_e$ is unavailable to the adversary. The reason for this is that in order to keep the symbolic model suitable for automated analysis, we do not wish to symbolically model the composition of randomness from encryptions when performing homomorphic evaluations; instead the private $\mathbf{eval}_e$ destructor takes a name r as input and we give the adversary access to it only via an honest process that accepts inputs $c_1, c_2, v_1, r_1, v_2, r_2$, picks a fresh name for r, and applies the destructor before sending back the result. We also use t to range over terms with destructors.

Processes Q are built from grammar

$$\begin{array}{cccc} \text{nil} & \text{in}[p, x]; Q & \text{let } x = t \text{ in } Q \text{ else } Q' & Q \mid\mid Q' \\ \text{new } n; Q & \text{out}[p, t]; Q & \text{if } t = t' \text{ then } Q \text{ else } Q' & !\, Q \end{array}$$

where n is a name, p is a port, and x a variable. The nil process does nothing and represents a halted state. The new $n; Q$ process is used for name and port restriction. Intuitively, the let $x = t$ in Q else Q' process tries to evaluate t to t' by reducing it using our rewrite rules and (trivial) equational theory; if it is successful it binds it to x in Q and proceeds as this process, and if it fails then it proceeds as Q' instead. The if $t = t'$ then Q else Q' process is just syntactic sugar but intuitively proceeds as Q if t and t' can be rewritten to equivalent terms, and as Q' if not. Finally, $Q \mid\mid Q'$ denotes parallel composition, and $!\, Q$ unbounded replication.

An *evaluation context* $\mathcal{E}$ is essentially a process with a hole, built from $[\_]$, $\mathcal{E} \parallel Q$, and new $n; \mathcal{E}$. We obtain process $\mathcal{E}[Q]$ as the result of filling the hole in $\mathcal{E}$ with Q. The formal semantics of a process can then be given by a reduction relation $\rightarrow$ defined as the smallest relation closed under application of evaluation contexts and rules:

$$\mathrm{out}[p,t]; Q_1 \parallel \mathrm{in}[p,x]; Q_2 \rightarrow Q_1 \parallel Q_2\{t/x\}$$

$$\text{let } x = t \text{ in } Q \text{ else } Q' \rightarrow \begin{cases} Q\{t'/x\} & \text{when } t \Downarrow t' \\ Q' & \text{otherwise} \end{cases}$$

where $t \Downarrow t'$ indicates that t may be rewritten to some t' containing no destructors. We write $\rightarrow^*$ for the reflexive and transitive closure of reduction.

Our equivalence notion for formalising *symbolic indistinguishability* is observational equivalence [AF01]. Here we write $Q{\downarrow_p}$ when Q can send an observable message on port p; that is, when $Q \rightarrow^* \mathcal{E}[\mathrm{out}[p,t]; Q']$ for some term t, process Q', and evaluation context $\mathcal{E}$ that does not bind p.

Definition 2 (Symbolic indistinguishability). Symbolic indistinguishability*, denoted* $\overset{s}{\sim}$*, is the largest symmetric relation* $\mathcal{R}$ *on closed processes* Q_1 *and* Q_2 *such that* $Q_1 \mathcal{R} Q_2$ *implies:*

1. *if* $Q_1{\downarrow_p}$ *then* $Q_2{\downarrow_p}$
2. *if* $Q_1 \rightarrow Q_1'$ *then there exists* Q_2' *such that* $Q_2 \rightarrow^* Q_2'$ *and* $Q_1' \mathcal{R} Q_2'$
3. $\mathcal{E}[Q_1] \mathcal{R} \mathcal{E}[Q_2]$ *for all evaluation contexts* $\mathcal{E}$

Intuitively, a context may represent an attacker, and two processes are symbolic indistinguishable if they cannot be distinguished by any attacker at any step: every output step in an execution of process Q_1 must have an indistinguishable equivalent output step in the execution of process Q_2, and vice versa; if not then there exists an evaluation context that "breaks" the equivalence. Note that the definition uses an existential quantification: if $Q_1 \overset{s}{\sim} Q_2$ then we only know that a reduction of Q_1 can be matched by *some* reduction of Q_2.

6.1 Symbolic Interpretation

Using the model from above it is somewhat straight-forward to give a symbolic interpretation of a system $\mathcal{S}(Sys)$ by giving an interpretation of a programme P in the form of a process Q_P, as well as a symbolic implementation of its operation module. Doing this we obtain a process Q_h for the honest entities, and for the adversary's operations we get a process Q_{adv}, both of which depend on the corruption scenario $\mathcal{H}$. The symbolic interpretation of a protocol is hence given by the three processes

$$\mathcal{E}^{AB}_{setup}[Q^{AB}_h \parallel Q^{AB}_{adv}] \qquad \mathcal{E}^{A}_{setup}[Q^{A}_h \parallel Q^{A}_{adv}] \qquad \mathcal{E}^{B}_{setup}[Q^{B}_h \parallel Q^{B}_{adv}]$$

where $Q^{\mathcal{H}}_h$ and $Q^{\mathcal{H}}_{adv}$ are put together inside an evaluation context responsible for generating keys.

6.2 Soundness of Symbolic Interpretation

Since the symbolic model already matches the intermediate model quite closely, the main issue for the soundness theorem is to ensure that the two notions of equivalence coincide. This in turn boils down to ensuring that the scheduling that leads to symbolic equivalence coincides with the scheduling policy used in the computational interpretations. Our solution is to restrict systems such that they allow only one choice of symbolic scheduling, namely that of the computational model. It is enough to require that no message is lost, i.e. for any strategy of the adversary, if a programme sends a message on a port then the receiving programme is listening on that port. The motivation behind this is that the two models disagree on what happens when the receiver is not ready: in the computational model the message is lost (read but ignored by the receiver) while in the symbolic model the message hangs around (possibly blocking) until the receiver is ready; this may then lead to non-determinism and several scheduling choices.

Theorem 2. *Let* Sys_1 *and* Sys_2 *be two systems that do not allow messages to be lost. If* $\mathcal{S}(Sys_1) \overset{s}{\sim} \mathcal{S}(Sys_2)$ *then* $\mathcal{I}(Sys_1) \overset{c}{\sim} \mathcal{I}(Sys_2)$.

Proof (overview). By fixing the random bitstrings seen by $\mathcal{Z}$ when interacting with $\mathcal{I}(Sys_i)$ we obtain a deterministic execution that with overwhelming probability will be matched by the symbolic execution on some evaluation context; the only situation where this is not possible is if $\mathcal{Z}$ manages to guess a bitstring drawn uniformly at random from $\{0,1\}^\kappa$. Symbolic indistinguishability between the two systems then implies that with overwhelming probability $\mathcal{Z}$ sees the same when interacting with $\mathcal{I}(Sys_1)$ and $\mathcal{I}(Sys_2)$.

Acknowledgements

We would like to thank Ran Canetti for valuable discussion and insights, and for hosting Morten at BU in the beginning of this work. We would also like to thank Hubert Comon-Lundh for discussion and clarification of his work, and Bogdan Warinschi for comments and valuable suggestions. Finally, we are thankful for the feedback provided by the anonymous reviewers, including mentioning of several places that needed clarification.

References

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, New York, USA, 2001. ACM Press.

[AR02] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15:103–127, 2002.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Symposium on Logic in Computer Science (LICS'05)*, pages 331–340. IEEE, 2005.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, 2011.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, 2011.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.

[BMM10] Michael Backes, Matteo Maffei, and Esfandiar Mohammadi. Computationally sound abstraction and verification of secure multi-party computations. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *LIPIcs*, pages 352–363. Schloss Dagstuhl, 2010.

[BP03] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the needham-schroeder-lowe public-key protocol. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *LNCS*, pages 1–12. Springer, 2003.

[BP04] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Computer Security Foundations Workshop (CSFW'04)*, pages 204–218. IEEE, 2004.

[BP06] Michael Backes and Birgit Pfitzmann. On the cryptographic key secrecy of the strengthened yahalom protocol. In *Security and Privacy in Dynamic Environments*, volume 201 of *IFIP*, pages 233–245. Springer, 2006.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *Computer and Communications Security (CCS'03)*, pages 220–230, New York, USA, 2003. ACM.

[BU13] Florian Böhl and Dominique Unruh. Symbolic universal composability. In *Computer Security Foundations (CSF'13)*, pages 257–271. IEEE, 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science (FOCS'01)*, pages 136–145. IEEE Computer Society, 2001.

[Can08] Ran Canetti. Composable formal security analysis: Juggling soundness, simplicity and efficiency. In *ICALP*, volume 5126 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2008.

[CC08] Hubert Lundh Comon and Véronique Cortier. Computational soundness of observational equivalence. In *Computer and Communications Security (CCS'08)*, pages 109–118. ACM, 2008.

[CG10] Ran Canetti and Sebastian Gajek. Universally composable symbolic analysis of diffie-hellman based key exchange. *IACR Cryptology ePrint Archive*, 2010:303, 2010.

[CH06] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *Theory of Cryptography (TCC'06)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer Berlin Heidelberg, 2006.

[CHKS12] Hubert Lundh Comon, Masami Hagiya, Yusuke Kawamoto, and Hideki Sakurada. Computational soundness of indistinguishability properties without computable parsing. In *Information Security Practice and Experience (ISPEC'12)*, volume 7232 of *LNCS*, pages 63–79. Springer, 2012.

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46:225–259, 2011.

[DDMR07] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172(0):311–358, 2007.

[DKP09] Stéphanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4, pages 169–180, 2009.

[DNO08] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In *ICISC*, volume 5461 of *LNCS*, pages 318–335. Springer, 2008.

[LN08] Peeter Laud and Long Ngo. Threshold homomorphic encryption in the universally composable cryptographic library. In *Provable Security*, volume 5324 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2008.

[MRST06] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1-3):118–164, 2006.

[MW04] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography (TCC'04)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. of IEEE Symposium on Security and Privacy*, pages 184–200, 2001.