Universally Composable Two-Party and Multi-party Secure Computation

(Extended Abstract)

Ran Canetti* Yehuda Lindell† Rafail Ostrovsky‡ Amit Sahai§

ABSTRACT

We show how to securely realize any multi-party functionality in a *universally composable* way, regardless of the number of corrupted participants. That is, we consider a multi-party network with open communication and an adversary that can adaptively corrupt as many parties as it wishes. In this setting, our protocols allow any subset of the parties (with pairs of parties being a special case) to securely realize any desired functionality of their local inputs, and be guaranteed that security is preserved regardless of the activity in the rest of the network. This implies that security is preserved under concurrent composition of an unbounded number of protocol executions, it implies non-malleability with respect to arbitrary protocols, and more. Our constructions are in the common reference string model and make general intractability assumptions.

1. INTRODUCTION

Traditionally, cryptographic protocol problems were considered in a model where the only involved parties are the actual participants in the protocol, and only a single execution of the protocol takes place. This model allowed for relatively concise problem statements, and simplified the design and analysis of protocols. Indeed, this relatively simple model is a natural choice for the initial study of protocols. Some of the many works in this model are [43, 4, 25, 36, 47, 33, 28, 3, 15, 2, 34, 38, 44, 31].

*IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA. email: canetti@watson.ibm.com

†Department of Computer Science, The Weizmann Institute of Science, Rehovot 76100, ISRAEL. email: lindell@wisdom.weizmann.ac.il

‡Telcordia Technologies, MCC-1C357B, 445 South Street, Morristown, New Jersey 07960 6438, USA. email: rafail@research.telcordia.com url: http://www.argreenhouse.com/bios/rafail/index.shtml

§Department of Computer Science, Princeton University. email: sahai@cs.princeton.edu

STOC'02, May 19-21, 2002, Montreal, Quebec, Canada.

However, this model of "stand-alone computation" does not fully capture the security requirements from cryptographic protocols in a modern computer network. In such networks, a protocol execution may run concurrently with an unknown number of other protocols. These arbitrary protocols may be executed by the same parties or other parties, they may have potentially related inputs and the scheduling of message delivery may be adversarially coordinated. Furthermore, the local outputs of a protocol execution may be used by other protocols in an unpredictable way. These concerns, or "attacks" on a protocol are not captured by the stand-alone model.

One way to guarantee that protocols withstand some specific security threats in multi-execution environments is to explicitly incorporate these threats into the security model and analysis. Such an approach was taken, for instance, in the cases of non-malleable commitments and zero-knowledge [23, 20, 46, 21, 19], and concurrent composition of zero-knowledge and oblivious transfer protocols [24, 45, 29]. However, this approach is inherently limited since it needs to explicitly address each new concern, whereas in a realistic network setting, the threats may be unpredictable. Furthermore, it inevitably results in definitions with ever-growing complexity.

In contrast, we take the approach where a protocol is designed and analyzed as "stand alone", and security in a multi-execution environment is guaranteed via a *secure composition theorem*. recently proposed framework of universally composable security [10] which builds and extends on many previous works, e.g. [40, 9]. Here a generic definition is given for what it means for a protocol to "securely realize a given ideal functionality," where an "ideal functionality" is a natural algorithmic way for capturing the desired functionality of the protocol problem at hand. In addition, it is shown that security of protocols is preserved under a general composition operation called universal composition. This essentially means that any protocol that securely realizes an ideal functionality when considered as stand-alone, continues to securely realize the same functionality even when composed with any other set of protocols that may be running concurrently in the same system.

It is known that *any* ideal functionality can be securely realized in a universally composable way using standard constructions, as long as a majority of the participants remain uncorrupted [3, 44, 11, 10]. However, this result does not hold when half or more of the parties may be corrupted. In particular, it does not hold for the important case of *two-party* protocols, where each party wishes to maintain its security even if the other party is corrupted. In fact, it was

shown in [12, 10] that a number of basic two-party functionalities (such as commitment, zero-knowledge, and common coin-tossing) cannot be securely realized in this framework by two-party protocols. Nonetheless, protocols that securely realize the commitment and zero-knowledge functionalities in the common reference string (CRS) model were shown in [12, 19]. (In the CRS model all parties are given a common, public reference string that was ideally chosen from a given distribution. This notion was originally proposed in the context of non-interactive zero-knowledge proofs [6] and since then proved useful in other cases as well.)

Our results. We show that any functionality can be realized in a universally composable way, in the CRS model and under general cryptographic assumptions, regardless of the number of corrupted parties. More specifically, consider a multi-party network where the communication is open and delivery of messages is not guaranteed. The network contains an unbounded (and unspecified) number of parties, and any number of these parties can be adaptively corrupted throughout the computation. In this setting, we show how arbitrary subsets of parties can securely realize any functionality of their inputs in a universally composable way. The functionality may be reactive, namely it may receive inputs and generate outputs multiple times throughout the computation. In addition to a common reference string, our protocols assume that the participants in each protocol execution have an authenticated and synchronous broadcast channel among themselves. No global synchronization is otherwise assumed. (Such synchronization and authentication primitives can be achieved using standard methods, see discussions in [10, 39]. In particular, in the case of two-party protocols these assumptions fall back to a standard asynchronous, pairwise, authenticated network.)

Outline of the construction. Our construction follows the general outline of the construction of Goldreich, Micali and Wigderson [33, 31], where the basic primitives are replaced with universally composable counterparts. On top of guaranteeing universal composability, this results in a modular construction and analysis that highlights the functionality and role of each ingredient in the construction. We first concentrate on the case of two-party functionalities, which contains most of the cryptographic ideas in a simplified form. Here, we first consider semi-honest (or, eavesdropping) adversaries. We begin by defining and realizing an ideal Oblivious Transfer (OT) functionality. Then we show that the [33] construction, given access to the ideal OT functionality, can be used to securely realize any two-party ideal functionality in a universally composable way. (No common reference string is used in the semi-honest case.)

Next we show how to transform any two-party protocol in the semi-honest model into a protocol that guarantees equivalent input-output relations in the presence of general, malicious adversaries. This is done as follows. Our starting point is a new adaptively secure universally composable (UC) commitment protocol in the CRS model, assuming only existence of trapdoor permutations. (UC commitment protocols are protocols that securely realize the ideal commitment functionality [12]. Existing constructions [12, 17] are based on stronger computational assumptions.) Our scheme uses tools from [35, 26, 11, 12, 23, 46].

Next, plugging the new scheme into the UC zero-knowledge protocol of [12] (which assumes access to the ideal commitment functionality), we obtain an adaptively secure UC zero-knowledge protocol in the CRS model, for any NP relation, and based on any trapdoor permutation. Here multiple proof instances, between different parties, can use the same copy of the common string. (Alternatively, we could use the protocol of [19]. However, this protocol provides security only against non-adaptive adversaries.)

Next, we define and realize a new ideal functionality, called commit-and-prove. This functionality allows a a party to commit to values and then prove "in zero knowledge" arbitrary NP-statements about the committed values. (This notion was implicitly present in the work of Goldreich, Micali, and Wigderson [33], and explicitly proposed by Kilian [37]. We formalize it as an ideal functionality in the UC framework.) We realize the commit-and-prove functionality given access to (multiple copies of) the ideal zero-knowledge functionality.

Finally, we cast the *protocol compiler* of [33]) in a model where the parties have access to the ideal commit-and-prove functionality, and use the universal composition theorem to compose all ingredients into a general, UC protocol compiler in the CRS model. Here we also use *universal composition with joint state* [14], which allows several protocol instances to use the same copy of the common string.

We also extend our results from the two-party case to the multi-party case. The semi-honest case is treated as in [33]. For the case of general adversaries, we first extend the zero-knowledge and commit-and-prove functionalities to allow a prover to commit and prove statements to a *set* or parties (rather than to a single party). Next, we generalize the protocol compiler, which now has ideal access to the one-to-many version of the commit-and-prove functionality.

Adaptive security. We provide the first general construction that guarantees security against adaptive adversaries, both in the two-party case and in the case of multi-party protocols with honest minority, both for semi-honest and for malicious adversaries. (We stress that no adaptively secure general construction was known in these cases, even in the stand-alone model.) In the semi-honest case, guaranteeing adaptively secure OT requires additional work on top of guaranteeing non-adaptive security. The general protocol, given ideal OT, is the same for the adaptive and non-adaptive cases. In the case of malicious adversaries, guaranteeing adaptively secure zero-knowledge and commit-and-prove requires additional work on top of guaranteeing non-adaptive security. The general protocol compiler given ideal commit-and-prove is the same for the adaptive and non-adaptive cases. (We remark that, in contrast to the case of stand-alone protocols, in our setting adaptive security is a relevant concern even for protocols with a small number of participants, such as two-party protocols. Furthermore, it is important to protect even against adversaries that eventually break into *all* the participants in an interaction. This is so since we consider multiple interactions that take place between different sets of parties in the system.)

Cryptographic assumptions. Overall, our protocols are based on the following cryptographic assumptions. For the non-adaptive case (both semi-honest and malicious) we assume the existence of trapdoor permutations only. For the adaptive, semi-honest case we additionally assume the existence of *obliviously generatable* public-key encryption-schemes as in [22, 16] where public keys can be generated without knowing the corresponding private keys. Alternatively, if we assume existence of *dense cryptosystems* [22]

(where public key is uniformly distributed), a requirement that clearly implies obliviously generatable public-key then we can assume that our reference string is distributed uniformly over $\{0,1\}^k$ for some large enough k. (Otherwise, we need a reference string that is taken from a different distribution.)

Related work. An alternative general construction of two-party protocols, attributed to Yao [47], has the advantage that only a small constant number of rounds is required in order to compute each output value, regardless of the complexity of the evaluated functionality. Potentially, this construction can be used as an alternative to our general construction for semi-honest adversaries given ideal OT (which is based on [33]). We remark, however, that adaptive security seems to be problematic for this specific construction.

In a concurrent and independent work [17], Damgard and Nielsen consider commit-and-prove composable functionality that has great resemblance to our commit-and-prove functionality, and propose protocols that realize this functionality under specific number-theoretic assumptions. Our protocols are based on more general assumptions, though their protocols have considerably better complexity than the ones here.

Organization. Section 2 briefly overviews the model of [10] and the relevant composition theorems. Section 3 presents our new universally composable commitment scheme. Section 5 overviews our construction of general *two-party* protocols. Section 6 elaborates on the commit-and-prove functionality, which is a main ingredient in the construction of Section 5. Section 7 presents the extensions needed for the multi-party case. Throughout, the presentation is kept informal and high-level, leaving out many essential details. A more complete presentation, including the proofs of all claims, can be found in [13].

2. THE MODEL

We overview the framework of [10]. The framework allows defining the security properties of cryptographic tasks so that security of protocols is preserved under a general composition operation with an unbounded number of copies of arbitrary protocols running concurrently in the system.

As in other general definitions (e.g., [34, 40, 1, 42, 9]), the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). Informally, a protocol securely carries out a given task if running the protocol with a realistic adversary amounts to "emulating" an ideal process where the parties hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction. We call the algorithm run by the trusted party an ideal functionality. A protocol that realizes a task as described here is said to securely realize the corresponding ideal functionality.

In order to allow proving the composition theorem, the notion of emulation in this framework is considerably stronger than previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary, $\mathcal{A}$, that controls the communication channels and potentially corrupts parties. "Emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or, simulator) $\mathcal{S}$ that results in similar distribution on the outputs for the parties. Here an additional adversarial entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. (In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$.) Note that the definition requires the "ideal-process adversary" (or, simulator) $\mathcal{S}$ to interact with $\mathcal{Z}$ throughout the computation. Furthermore, $\mathcal{Z}$ cannot be "rewound".

Universal Composition. The following universal composition (UC) operation is considered. Let π be a protocol that operates in a *hybrid* model of computation where parties can communicate as usual, and in addition have ideal access to an unbounded number of copies of some ideal functionality $\mathcal{F}$. (This model is called the $\mathcal{F}$-hybrid model.) Let ρ be a protocol that securely realizes $\mathcal{F}$ as sketched above. Then, let the "composed protocol" π^ρ be identical to π with the exception that each interaction with some copy of $\mathcal{F}$ is replaced with an invocation of an appropriate instance of ρ. Similarly, ρ-outputs are treated as values provided by the appropriate copy of $\mathcal{F}$. It is shown that π and π^ρ have essentially the same input/output behavior. In particular, if π securely realizes some ideal functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then π^ρ securely realizes $\mathcal{I}$ from scratch. (If ρ is a protocol in the $\mathcal{G}$-hybrid model for some ideal functionality $\mathcal{G}$ then π^ρ is also a protocol in the $\mathcal{G}$-hybrid model.)

Universal Composition with joint state. Universal composition requires that all copies of the "subroutine protocol" ρ have separate local states and independent local randomness. This does not allow the subroutine protocols to have any amount of joint state. Indeed, if protocol ρ is designed in the CRS model then the protocol π^ρ obtained using the UC operation described above uses an independent copy of the reference string for each copy of ρ. In contrast, we wish to allow multiple protocol instances to use the same instance of the reference string. We do this using the following variant of the UC operation, proposed in [14] and called universal composition with joint state (JUC). Here the first ingredient, protocol π in the $\mathcal{F}$-hybrid model, remains unchanged. However, instead of replacing each copy of $\mathcal{F}$ with a different instance of some protocol ρ, we replace all copies of $\mathcal{F}$ with a *single instance* of some "joint protocol" $\hat{\rho}$. The protocol obtained from this composition operation is denoted $\pi^{[\hat{\rho}]}$. (The brackets $[\ ]$ in the notation $\pi^{[\hat{\rho}]}$ distinguish this composition operation from standard UC.)

The following condition (on protocol $\hat{\rho}$) is shown to suffice for security to be preserved under JUC. Intuitively, protocol $\hat{\rho}$ should exhibit the same functionality as multiple "independent" instances of ρ. A bit more precisely, given an ideal functionality $\mathcal{F}$, we define $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$, to be the functionality that behaves like multiple independent copies of $\mathcal{F}$. (That is, $\hat{\mathcal{F}}$ runs multiple copies of $\mathcal{F}$. Upon receipt of a query (i,q), $\hat{\mathcal{F}}$ forwards the query q to the ith copy of $\mathcal{F}$.) It is shown in [14] that if $\hat{\rho}$ securely realizes $\hat{\mathcal{F}}$ then protocol $\pi^{[\hat{\rho}]}$ behaves essentially like π does with ideal access to multiple copies of $\mathcal{F}$. In particular, if π

securely realizes some ideal functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then $\pi^{[\hat{\rho}]}$ securely realizes $\mathcal{I}$ from scratch.

The basic model. The underlying communication network is assumed to be asynchronous without guaranteed delivery of messages. In addition, we assume a synchronous and authenticated broadcast channel among each set of interacting parties. (In the two-party case, this falls back to the standard model of pairwise authenticated communication.) It is stressed that our model allows the adversary to disrupt the computation at any time by refraining from delivering messages. Indeed, our protocols do not protect against "early stopping" by the corrupted parties.

Also, as usual, the adversary is allowed to corrupt parties. In the case of non-adaptive adversaries the set of corrupted parties is fixed at the onset of the computation. In the adaptive case the adversary corrupts parties at will throughout the computation. If the adversary is malicious then corrupted parties follow the instructions of the adversary. In the semi-honest case even corrupted parties follow the prescribed protocol and the adversary only gets read access to the states of corrupted parties.

3. UC COMMITMENT

We describe our new universally-composable non-interactive commitment scheme. Our construction is in the common reference string model, and assumes only the existence of trapdoor permutations. (If the common reference string must come from a uniform distribution, then we require trapdoor permutations with dense public descriptions [22].) Recall that UC commitment schemes are protocols that securely realize the many-time ideal commitment functionality, presented in Figure 1. (The fact that $\mathcal{F}_{\text{MCOM}}$ handles multiple commitments within a single instance will become useful in Section 4.)

Functionality $\mathcal{F}_{\text{MCOM}}$

$\mathcal{F}_{\text{MCOM}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$:

- **Commit Phase:** Upon receiving a message $(\mathsf{commit}, sid, cid, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the tuple (cid, P_i, P_j, b) and send the message $(\mathsf{receipt}, sid, cid, P_i)$ to P_j and $\mathcal{S}$. Ignore any future commit messages with the same cid.
- **Reveal Phase:** Upon receiving a message $(\mathsf{reveal}, sid, cid)$ from P_i: If a tuple (cid, P_i, P_j, b) was previously recorded, then send the message $(\mathsf{reveal}, sid, cid, P_i, b)$ to P_j and $\mathcal{S}$. Otherwise, ignore.

Figure 1: The ideal commitment functionality

Informally speaking, in order to achieve universal composability against adaptive adversaries, our commitment scheme must have the following two properties:

• *Polynomial equivocability:* the simulator (i.e., the adversary in the ideal process) should be able to produce commitments for which it can decommit to both 0 and 1 polynomially many times using the same reference string. (An additional property is needed for adaptive security. See below.) Still, the real committer must be able to decommit to only a single value.

• *Simulation extractability:* the simulator should be able to extract the contents of any valid commitment, even after having supplied an adversary with an arbitrary number of equivocable commitments.

We remark that equivocable commitment protocols of [20, 21] can be used only once on the same reference string to "equivocate", and then a new reference string is needed. In contrast, [12] show how to achieve it (though with a stronger assumptions) polynomially many times.

We describe our construction in phases. First we describe a new non-interactive variant of the Feige-Shamir trapdoor commitment scheme [26], which is at the heart of our construction. Then we show how to transform this scheme into one that is universally composable.

Underlying standard commitment. The basic underlying commitment scheme Com is the standard non-interactive commitment scheme based on a one-way permutation f and a hard-core predicate b of f. That is, in order to commit to a bit σ, one computes $Com(\sigma) = \langle f(U_k), b(U_k) \oplus \sigma \rangle$, where U_k is the uniform distribution over $\{0,1\}^k$. Note that Com is computationally secret, and produces pseudorandom commitments: that is, the distributions $Com(0)$, $Com(1)$, and U_{k+1} are computationally indistinguishable.

Simplified Feige-Shamir Commitment. We briefly describe a simplified version of the Feige-Shamir trapdoor commitment scheme [26], which is based on the zero-knowledge proof for Hamiltonicity of Blum [5]. First, a graph G (with q nodes) is found, so that it is hard to find a Hamiltonian cycle in G within polynomial-time. This is achieved as follows: choose $x \in_R \{0,1\}^k$ and compute $y = f(x)$, where f is a one-way function. Then, use the reduction of the language $\{y \mid \exists x \text{ s.t. } y = f(x)\}$ to that of Hamiltonicity, to obtain a graph G so that finding a Hamiltonian cycle in G is equivalent to finding the preimage x of y. This graph G is sent from the receiver to the sender. [1] Then, in order to commit to 0, the committer commits to a random permutation of G using the underlying commitment scheme Com (and decommits by revealing the entire graph and the permutation). In order to commit to 1, the committer commits to a graph containing a randomly labeled q-cycle only (and decommits by opening this cycle only). Note that this commitment scheme is binding because the ability to decommit to both 0 and 1 implies that the committer knows a Hamiltonian cycle in G. On the other hand, given a Hamiltonian cycle in G, it is possible to generate commitments that are indistinguishable from legal ones, and yet have the property that one can decommit to both a 0 and a 1.

A non-interactive, adaptively secure commitment. We proceed to describe our first step for obtaining adaptive security. In general, the following additional property is required of the simulator: Let c be a commitment produced by the simulator who knows a Hamiltonian cycle in G. Then, for any $b \in \{0,1\}$, the simulator should be able to produce a random string r, so that the honest committer upon input b and random-tape r, outputs the commitment c. Thus, the simulator can provide an "explanation" of its actions, as if it were an honest committer. (We note that the [26] scheme does not have this property.)

The adaptively secure scheme, denoted aHC_G (for *adaptive Hamiltonicity Commitment*), differs from [26] in two

[1] In [26], the graph G is chosen by computing f on *two* randomly chosen inputs, which is important for making use of *witness-indistinguishability*, which underlies the proof of security for [26]. Our construction and proof of security, however, do not require this property; thus, we can use the simpler method described above.

respects. First, instead of letting the receiver choose the string y, we place y (which determines the graph G) in a common reference string. (Note that if we want y to be uniformly distributed, then we need only ask that f is a one-way function.) Next, the sender proceeds as follows.

- To commit to a 0, the sender picks a random permutation π of the nodes of G, and commits to the entries of the adjacency matrix of the permuted graph one by one, using Com. To decommit, the sender sends π and decommits to every entry of the adjacency matrix. The receiver verifies that the graph it received is $\pi(G)$.

- To commit to a 1, the sender chooses a randomly labeled q-cycle, and for all the entries in the adjacency matrix corresponding to edges on the q-cycle, it uses Com to commit to 1 values. However, for all the other entries, it simply produces random values from U_{k+1} (for which it does not know the decommitmenet!) That is, the edges that are not on the q-cycle are not committed to. Instead, random strings are sent.) To decommit, the sender opens only the entries corresponding to the randomly chosen q-cycle in the adjacency matrix.

This commitment scheme immediately has the property of being computationally secret, *i.e.* the distributions $\mathsf{aHC}_G(0)$ and $\mathsf{aHC}_G(1)$ are computationally indistinguishable for any graph G. Also, given the opening of any commitment to both a 0 and 1, one can extract a Hamiltonian cycle in G. Therefore, the committer cannot decommit to both 0 and 1, and the binding property holds. Finally, as with the scheme of [26], given a Hamiltonian cycle in G, a simulator can generate commitments to 0 and then open those commitments to both 0 and 1. (This is because the simulator knows a simple q-cycle in G itself.) However, in contrast to [26], here the simulator can also produce a random tape for the sender explaining a commitment as a commitment to either 0 or 1. Specifically, the simulator generates each commitment string c as a commitment to 0. If, upon corruption of the sender, the simulator has to demonstrate that c is a commitment to 0 then all randomness is revealed. To demonstrate that c was generated as a commitment to 1, the simulator opens the commitments to the edges in the q-cycle and claims that all the unopened commitments are merely uniformly chosen strings (rather than commitments to the rest of G). This can be done since commitments produced by the underlying commitment scheme Com are pseudorandom. The key point here is unlike the schemes of [20, 21], this gives us polynomial equivocability, where the same common reference string can be reused polynomially-many times.

Achieving simulation extractability. As discussed above, the commitment scheme aHC_G has the equivocability property, as required. However, our commitment scheme must also have the *simulation extractability* property. We must modify our scheme in such a way that we add extractability without sacrificing equivocability. Simulation-extractability alone could be achieved by including a public-key for an encryption scheme secure against chosen ciphertext attack (CCA-2) [23, 46, 19] into the common reference string, and having the committer send an encryption of the decommitment information along with the commitment itself. The CCA security of the encryption can be used to guarantee that an adversary cannot use an equivocable commitment prepared by the simulator to produce a non-extractable commitment. Thus, a simulator knowing the associated decryption key can decrypt and obtain the decommitment information, thereby extracting the committed value from any adversarially prepared commitment. However, the operation above – of encrypting decommitment information with a general CCA-secure encryption scheme – may indeed destroy the equivocability of the overall scheme. In order to obtain equivocability, we require the encryptions to be pseudorandom, which is not necessarily the case for known CCA-secure schemes based on general assumptions [23, 46, 19]. We achieve the security we desire by using double-encryption: by first encrypting using a CCA2-secure scheme, which may result in a ciphertext which is not pseudorandom, and then re-encrypting the encryption using a scheme for which all encryptions look pseudorandom. (The second scheme needs to be secure only against chosen plaintext attacks.) The key is to show that this combination gives the right composition, and this is indeed what we show in the full version [13].

For the CCA-secure scheme, denoted E_{cca}, we can use any known scheme based on trapdoor permutation, e.g., [23, 46, 19]. For the second encryption scheme, denoted E, we use the standard encryption scheme based on trapdoor-permutations and hard-core predicates [32], where the public key is a trapdoor permutation f, and the private key is f^{-1}. Here encryption of a bit b is $f(x)$ where x is a randomly chosen element such that the hard-core predicate of x is b. Note that encryptions of both 0 and 1 are pseudorandom. The commitment scheme, called Universal Adaptive Hamiltonicity Commitment UAHC, is presented in Figure 2.

Protocol UAHC

- **Common Reference String:** The string consists of a random string y (which determines the graph G), and public-keys for the encryption schemes E and E_{cca}. (The security parameter k is implicit.)
- **Commit Phase:**
  1. On input $(\texttt{commit}, sid, cid, P_j, b \in \{0,1\})$, P_i computes $z \leftarrow \mathsf{aHC}_G(b)$. Next, P_i computes $C_b = E(E_{cca}(P_i, cid, r))$ and sets C_{1-b} to a random string of length $|C_b|$. Finally, P_i records (cid, r, b), where r is the randomness used in aHC_G and the encryptions, and sends $c = (sid, cid, z, C_0, C_1)$ to P_j.
  2. P_j receives and records c, and outputs $(\mathsf{receipt}, sid, cid, P_i,)$.
- **Reveal Phase:**
  1. On input $(\texttt{Open}, sid, cid)$, P_i retrieves (cid, r, b) and sends to P_j.
  2. P_j verifies the decommitment information and outputs $(\mathsf{open}, sid, cid, P_i, b)$ if it is correct.

Figure 2: The commitment protocol UAHC

Let $\mathcal{F}_{\text{CRS}}$ denote the common random string functionality (that is, $\mathcal{F}_{\text{CRS}}$ provides all parties with a common, public string drawn from the above distribution). Then, we have:

PROPOSITION 1. *Protocol* UAHC *securely realizes* $\mathcal{F}_{\text{MCOM}}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model, assuming exostence of trapdoor permutations.*

4. UC ZERO-KNOWLEDGE

Universally composable zero-knowledge serves as a starting point for our general construction. Specifically, in the two-party case we present our constructions in the $\mathcal{F}_{\text{ZK}}$-hybrid model, where $\mathcal{F}_{\text{ZK}}$ is the ideal functionality described in Figure 3. (In the multi-party case our constructions are based on an extension of $\mathcal{F}_{\text{ZK}}$ to the case of single prover and multiple verifiers. See Section 7.)

Functionality $\mathcal{F}_{\text{ZK}}$

$\mathcal{F}_{\text{ZK}}$ proceeds as follows, running with a prover P, a verifier V and an adversary $\mathcal{S}$, and parameterized with relation R:

- Upon receiving (ZK-prover, sid, x, w) from P: if $R(x, w) = 1$, then send (ZK-proof, sid, x) to V and $\mathcal{S}$ and halt. Otherwise, ignore.

Figure 3: The $\mathcal{F}_{\text{ZK}}$ functionality

Note that, in contrast to $\mathcal{F}_{\text{MCOM}}$ where a single copy of the functionality handles multiple commitments, $\mathcal{F}_{\text{ZK}}$ handles only a single proof (i.e., it sends only a single message before halting). Indeed, our protocols in the $\mathcal{F}_{\text{ZK}}$-hybrid model use multiple copies of $\mathcal{F}_{\text{ZK}}$. We then use universal composition with joint state (JUC) in order to realize all the copies of $\mathcal{F}_{\text{ZK}}$ using a single copy of the reference string. This is done as follows. The JUC theorem states that any protocol in the $\mathcal{F}_{\text{ZK}}$-hybrid model can be composed with any protocol ρ that securely realizes $\hat{\mathcal{F}_{\text{ZK}}}$, so that all calls to $\mathcal{F}_{\text{ZK}}$ are replaced with a single instance of ρ. (Here $\hat{\mathcal{F}_{\text{ZK}}}$ is the multi-session extension of $\mathcal{F}_{\text{ZK}}$, as described in Section 2.) It thus suffices to demonstrate protocols that securely realized $\hat{\mathcal{F}_{\text{ZK}}}$. We point to two such protocols that work for any NP relation. First is the adaptively secure, three-round zero-knowledge protocol for graph Hamiltonicity described in [12]. This protocol operates in the $\mathcal{F}_{\text{MCOM}}$-hybrid model, and use only a single copy of $\mathcal{F}_{\text{MCOM}}$. Composing with the protocol from Section 3, we obtain a protocol that uses a single copy of the reference string. Second is the non-interactive zero-knowledge protocol of [19]. This protocol operates in the $\mathcal{F}_{\text{CRS}}$-hybrid model, assumes existence of trapdoor permutations, and uses only a single copy of the reference string. It is secure only against non-adaptive adversaries.

5. UC TWO-PARTY COMPUTATION

This section presents an outline of our protocols for securely realizing any *two-party* ideal functionality in a universally composable way. In high level, our construction is similar to that of Goldreich, Micali and Wigderson [33, 31]. Recall that the GMW construction proceeds in two stages. First, they present a protocol for securely realizing any functionality in the semi-honest adversarial model (where the adversary follows the protocol specification exactly, yet attempts to learn more than is intended from the execution transcript). Next, they construct a *protocol compiler* that transforms any semi-honest protocol into a protocol that has the same functionality in the malicious adversarial model.

5.1 The case of semi-honest adversaries

Let us first briefly recall the [33, 31] construction for securely evaluating a function of the inputs of two parties. The parties have access to an arithmetic circuit over $GF(2)$ for evaluating the given functionality. Furthermore they hold shares of the input lines of the circuit. That is, for each input line l, party A holds a value a_l and party B holds a value b_l, such that both a_l and b_l are random under the restriction that $a_l + b_l$ equals the value of this line. Next, the parties evaluate the circuit gate by gate, where for each gate they compute shares of the output value of the gate, given their shares of the input values. Addition gates are evaluated by having each party locally add its shares of the input values. Multiplication gates are evaluated using 1-out-of-4 Oblivious Transfer. Once the parties obtain shares of the output gates of the circuit, they reveal their shares to the prescribed party.

Our general construction, given idealized Oblivious Transfer, is that of GMW. That is, we define an ideal Oblivious Transfer functionality, $\mathcal{F}_{\text{OT}}$, and show that in the $\mathcal{F}_{\text{OT}}$-hybrid model the GMW protocol securely realizes any two-party functionality in the presence of semi-honest, *adaptive* adversaries. This holds unconditionally and even if the adversary and environment are computationally unbounded. (In fact, our construction is somewhat more general in that it deals with reactive functionalities. This is done by allowing the parties to hold shares of the state of the ideal functionality between activations. See details in [**?**].)

Next we present protocols that securely realizes $\mathcal{F}_{\text{OT}}$ in the semi-honest case. In the non-adaptive case the protocol of [25, 33, 31] suffices. In the adaptive case our protocol uses non-committing encryption (as in [11]), with the additional property that there is an alternative key generation algorithm that generates only public encryption keys without the corresponding decryption key. (Following [16], we call this the *oblivious generation* property. All known non-committing encryption schemes have this property. Such schemes exist under either the RSA assumption or the DDH assumption.) In all, we show:

PROPOSITION 2. *Assume that trapdoor permutations and two-party non-committing encryption protocols with the oblivious generation property exist. Then, for any two-party ideal functionality $\mathcal{F}$, there exists a protocol* Π *that securely realizes $\mathcal{F}$ in the presence of* semi-honest, adaptive *adversaries.*

5.2 Security against malicious adversaries

Having constructed a protocol that is universally composable when the adversary is limited to semi-honest behavior, we construct a protocol compiler to transform this protocol into one that is secure even against malicious adversaries. From here on, we refer to the protocol that is secure against semi-honest adversaries as the "basic protocol". In order to obtain a protocol secure against malicious adversaries, we need to enforce potentially malicious corrupted parties to behave in a semi-honest manner. First and foremost, this involves forcing the parties to follow the prescribed protocol. However, this only makes sense relative to a *given* input and random tape. Furthermore, a malicious party must be forced into using a *uniformly chosen* random tape. This is because the security of the basic protocol depends on the fact that the party has no freedom in setting its own randomness. We begin with a description of the GMW compiler.

An informal description of the GMW compiler. In light of the above discussion regarding enforcing semi-honest

behavior, the GMW compiler begins by having each party commit to its input. Next, the parties run a coin-tossing protocol in order to fix their random tapes. A simple coin-tossing protocol in which both parties receive the same uniformly distributed string is not sufficient here. This is because the parties' random tapes must remain secret. Instead, an *augmented* protocol is used, where one party receives a uniformly distributed string (to be used as its random tape) and the other party receives a commitment to that string. Now, each party holds its own input and uniformly distributed random-tape, and a commitment to the other party's input and random-tape. Therefore, each party can be "forced" into behaving consistently with the committed input and random-tape.

We now describe how this behavior is enforced. A protocol specification is a deterministic function of a party's view consisting of its input, random tape and messages received so far. As we have seen, each party holds a commitment to the input and random tape of the other party. Furthermore, the messages sent so far are public. Therefore, the assertion that a new message is computed according to the protocol is an NP-statement (and the party sending the message knows an adequate NP-witness to it). Thus, the parties can use zero-knowledge proofs to show that their steps are indeed according to the protocol specification. Therefore, in the protocol emulation phase, the parties send messages according to the instructions of the basic protocol, while proving at each step that the message they sent is correct. The key point is that, due to the soundness of the proofs, even a malicious adversary cannot deviate from the protocol specification without being detected. Therefore, the adversary is limited to semi-honest behavior. Furthermore, since the proofs are zero-knowledge, nothing "more" is revealed in the compiled protocol than in the basic protocol. We conclude that the security of the compiled protocol (against malicious adversaries) is derived from the security of the basic protocol (against semi-honest adversaries).

In summary, the GMW compiler has three components: input commitment, coin-tossing and protocol emulation (in which the parties prove that their steps are according to the protocol specification).

Universally Composable Protocol Compilation. A natural way of adapting the GMW compiler to the setting of universally composable secure computation would be to take the same compiler, but use *universally composable* commitments, coin-tossing and zero-knowledge as sub-protocols. However, such a strategy fails because, in contrast to GMW where NP assertions can be stated and proven with respect to the publicly known commitment string, in the $\mathcal{F}_{\text{MCOM}}$-hybrid model no such string is available.

We thus follow a different strategy for constructing a universally composable compiler. Observe that in GMW the use of a commitment scheme is not standard. Specifically, although both parties commit to their inputs etc., they never decommit. Rather, they *prove* NP-statements relative to their committed values. Thus, a natural primitive to use would be a "commit-and-prove" functionality, which is comprised of two phases. In the first phase, a party "commits" (or is bound) to a specific value. In the second phase, this party proves (in zero-knowledge) NP-statements relative to the committed value. We formulate this notion in a universally composable commit-and-prove functionality, denoted $\mathcal{F}_{\text{CP}}$, and then use this functionality to implement all three phases of the compiler. More specifically, our protocol compiler uses the "commit" phase of the $\mathcal{F}_{\text{CP}}$ functionality in order to execute the input and coin-tossing phases of the compiler. The "prove" phase of the $\mathcal{F}_{\text{CP}}$ functionality is then used to force the adversary to send messages according to the protocol specification and consistent with the input and the random-tape resulting from the coin-tossing (as fixed in the commit phase of $\mathcal{F}_{\text{CP}}$). The result is a universally composable analog to the GMW compiler. We remark that in the $\mathcal{F}_{\text{CP}}$-hybrid model the compiler is unconditionally secure against *adaptive* adversaries, even if the adversary and the environment are computationally unbounded.

In Section 6, we show how to securely realize $\mathcal{F}_{\text{CP}}$ in the $\mathcal{F}_{\text{ZK}}$-hybrid model. Furthermore, as noted in Section 4, $\mathcal{F}_{\text{ZK}}$ itself can be securely realized in the $\mathcal{F}_{\text{CRS}}$-hybrid model, assuming the existence of trapdoor permutations. We thus have:

THEOREM 3. *Assume that trapdoor permutations and two-party non-committing encryption protocols with the oblivious sampling property exist. Then, for any two-party ideal functionality $\mathcal{F}$, there exists a protocol Π that securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model in the presence of* malicious, adaptive *adversaries.*

We note that in the case of *non-adaptive* adversaries, non-committing encryption is not needed, and trapdoor permutations suffice.

6. TWO-PARTY COMMIT-AND-PROVE

We define the two-party "commit-and-prove" functionality, $\mathcal{F}_{\text{CP}}$, and present a protocol for securely realizing it. As discussed in Section 5, this functionality, which is a generalization of the commitment functionality, is central for constructing the general protocol compiler. As in the case of $\mathcal{F}_{\text{ZK}}$, the $\mathcal{F}_{\text{CP}}$ functionality is parameterized by a relation R. The first stage is a commit phase in which the receiver obtains a commitment to some value w. However, the second phase is more general than plain decommitment. Rather than revealing the committed value, the functionality receives some value x from the committer, sends x to the receiver, and asserts whether $R(x, w) = 1$. To see that this is indeed a generalization of a commitment scheme, take R to be the identity relation and $x = w$. Then, following the prove phase, the receiver obtains w and is assured that this is the value that was indeed committed to in the commit phase.

In fact, $\mathcal{F}_{\text{CP}}$ is slightly more general, in the following two ways. First it allows the committer to commit to multiple secret values w_ℓ, and then have the relation R depend on all these values in a single proof. (This extension is needed to deal with reactive protocols, where inputs may be received over time.) Second, the committer may ask to prove multiple statements with respect to the same set of secret values. Thus, when receiving a new $(\mathsf{commit}, sid, w)$ request from the committer, $\mathcal{F}_{\text{CP}}$ will add the current w to the already existing list $\overline{w}$ of committed values. When receiving a $(\mathsf{CP\text{-}prover}, sid, x)$ request, $\mathcal{F}_{\text{CP}}$ evaluates R on x and the list $\overline{w}$. Functionality $\mathcal{F}_{\text{CP}}$ is presented in Figure 4.

The $\mathcal{F}_{\text{CP}}$ functionality is defined so that only correct statements (i.e., values x such that $R(x, w) = 1$) are received by P_j in the prove phase. Incorrect statements are ignored by the functionality, and the receiver P_j receives no notification that an attempt at cheating in a proof took place. This

Functionality $\mathcal{F}_{\text{CP}}$

$\mathcal{F}_{\text{CP}}$ proceeds as follows, running with a committer A, a receiver B and an adversary $\mathcal{S}$, and with security parameter k:

- **Commit Phase:** Upon receiving a message $(\mathsf{commit}, sid, B, w)$ from A where $w \in \{0,1\}^k$, append the value w to the list $\overline{w}$, and send the message $(\mathsf{receipt}, sid, A)$ to B and $\mathcal{S}$. (Initially, the list $\overline{w}$ is empty.)
- **Prove Phase:** Upon receiving a message $(\mathsf{CP\text{-}prover}, sid, x)$ from A, where $x \in \{0,1\}^{\text{poly}(k)}$, compute $R(x, \overline{w})$: If $R(x, \overline{w}) = 1$, then send B and $\mathcal{S}$ the message $(\mathsf{CP\text{-}proof}, sid, x)$. Otherwise, ignore.

Figure 4: The commit-and-prove functionality

convention simplifies the description and analysis of our protocols. We note, however, that this is not essential. Error messages can be added to the functionality (and realized) in a straightforward way.

6.1 Securely Realizing $\mathcal{F}_{\text{CP}}$

We present a protocol for securely realizing the $\mathcal{F}_{\text{CP}}$ functionality in the $\mathcal{F}_{\text{ZK}}$-hybrid model. The commit phase involves an interactive version of the aHC_G commitment scheme of Section 3, and the prove phase of the protocol involves a single invocation of $\mathcal{F}_{\text{ZK}}$.

We begin by describing the commit phase of the protocol. Notice that similarly to universally composable commitments, the commitment used here must have the *extractability* property. That is, a simulator must be able to extract the committed value from the committer. The usual way of obtaining such a property is by having the committer prove in zero-knowledge that it knows the decommitment value. Here, we do the same, utilizing the ideal $\mathcal{F}_{\text{ZK}}$ functionality. We note that the use of $\mathcal{F}_{\text{ZK}}$ guarantees extractability regardless of the commitment scheme in use. In particular, in the case of non-adaptive adversaries the simple commitment based on any one-way function and hard-core predicates suffices. However, in order to achieve security against adaptive adversaries, we require that the simulator be able to generate a simulated commitment that can be "explained" as a commitment to any value (recall the discussion in Section 3). We thus base ourselves on scheme aHC_G presented there. However, as presented there, aHC_G relies on the existence of a common reference string, whereas here we only have access to $\mathcal{F}_{\text{ZK}}$. We thus use an interactive version of aHC_G where the receiver B choose the random string y and sends y to the sender using $\mathcal{F}_{\text{ZK}}$, while proving that it knows a preimage of y w.r.t. the one permutation f. (By extracting this preimage, the simulator obtains the "trapdoor information" it needs; see Section 3). In summary, B chooses a random value x in the domain of the one way permutation f, and sends $(\mathtt{ZK\text{-}prover}, sid, y = f(x), x))$ to $\mathcal{F}_{\text{ZK}}$ with the appropriate relation. Then, in order to commit to w, the committer A computes $c = \mathsf{aHC}_G(w)$ using randomness r, and sends $(\mathsf{ZK\text{-}prover}, sid, (c, y), (w, r))$ (the relation used here is that c is a commitment to w by aHC_G, using randomness r). When B receives c, it adds c to its list $\overline{c}$ of accepted commitments. In addition, the committer A keeps the list $\overline{w}$ of all the values w committed to, and the lists $\overline{r}$ and $\overline{c}$ of the corresponding random values and commitment values.

We now proceed to describe the prove phase of the protocol. As is to be expected, this phase involves a single invocation of $\mathcal{F}_{\text{ZK}}$. Let R be the relation parameterizing $\mathcal{F}_{\text{CP}}$. Then, the relation R_P parameterizing the $\mathcal{F}_{\text{ZK}}$ functionality that is used in the prove phase is defined by:

$$R_P \stackrel{\text{def}}{=} \{((x, \overline{c}, y), (\overline{w}, \overline{r})) \mid \forall \ell,\ c_\ell = \mathsf{aHC}_G(w_\ell; r_\ell)\ \&\ R(x, \overline{w}) = 1\}$$

where $\mathsf{aHC}_G(w; r)$ denotes a commitment to w using random-coins r and based on the string y. Thus, the prove phase consists of the committer proving some NP-statement regarding the values committed to previously. (The value x is the NP-statement and the values committed to, plus the randomness used, comprise the "witness" for x). Upon receiving the proof message: $(\mathsf{ZK\text{-}proof}, sid, P_i, P_j, (x, \overline{c}, y))$ from $\mathcal{F}_{\text{ZK}}$, the receiver accepts if y is the string that it initially sent and $\overline{c}$ agrees with the list of commitments it has previously received in this session. Note that if $R \in \text{NP}$, then so too is R_P.

The above-described protocol invokes $\mathcal{F}_{\text{ZK}}$ three times, each time for a different relation. For simplicity, we assume that there are three different $\mathcal{F}_{\text{ZK}}$ functionalities, each one parameterized by the appropriate relation. That is, let $\mathcal{F}_{\text{ZK}}^1$ be an $\mathcal{F}_{\text{ZK}}$ functionality parameterized by the one-way function f. Let $\mathcal{F}_{\text{ZK}}^2$ be an $\mathcal{F}_{\text{ZK}}$ functionality parameterized by $R_2 = \{((c, y), (w, r)) \mid c = \mathsf{aHC}_G(w; r)\}$. Finally, let $\mathcal{F}_{\text{ZK}}^3$ be an $\mathcal{F}_{\text{ZK}}$ functionality parameterized by R_P as described above. The protocol is presented in Figure 5. We have:

PROPOSITION 4. *Protocol* CP *securely realizes $\mathcal{F}_{\text{CP}}$ in the $\mathcal{F}_{\text{ZK}}$-hybrid model.*

Proposition 4 is proven by utilizing the power of the ideal zero-knowledge functionality $\mathcal{F}_{\text{ZK}}$. The key point is that the ideal-model simulator receives all values sent by corrupted parties to the $\mathcal{F}_{\text{ZK}}$ functionalities. In particular, this means that the simulator easily obtains the committed value from a corrupted A, because A sends w and r to $\mathcal{F}_{\text{ZK}}^2$ in order to commit. On the other hand, the security against adaptive adversaries is derived immediately from the fact that aHC_G is adaptively secure, *if* the simulator knows a preimage of y. However, once again, the simulator can obtain such a preimage from a corrupted B because B must send this preimage to $\mathcal{F}_{\text{ZK}}^1$. Finally, the prove messages are also easily simulated. That is, upon receiving a message of the form $(\mathsf{CP\text{-}proof}, sid, x)$ from $\mathcal{F}_{\text{CP}}$, the simulator generates the message that B would have seen in Protocol CP, which is $(\mathsf{ZK\text{-}proof}, sid, (x, \overline{c}, y))$ (where y is the string sent earlier by B and the vector $\overline{c}$ contains simulated commitments that were sent earlier in the simulation). See details in [13].

Finally, we note that the commitment scheme aHC_G can be implemented using one-way functions only. In order for this to be the case, the underlying Com commitment used in aHC_G is replaced by the commitment scheme of [41], that can be based on any one-way function. Indeed, the [41] scheme produces random-looking commitments, as required by aHC_G. In addition, we modify the protocol so that B also sends the receiver message from the [41] commitment in Step 1.

7. MULTI-PARTY UC-COMPUTATION

We discuss how the two-party construction of Theorem 3 is extended to the setting of multi-party computation, where any number of parties may be corrupted. (Again, see [13]

Protocol CP

- **Auxiliary Input:** A security parameter k, and a session identifier sid.
- **Commit phase:**
  1. On input $(\texttt{Commit}, sid, B, w)$, A sends sid to B.
  2. . Upon receiving sid from A, B chooses $x \in_R \{0,1\}^k$, computes $y = f(x)$ (where f is a one-way function), and sends $(\mathsf{ZK\text{-}prover}, sid, y, x)$ to $\mathcal{F}_{\mathrm{ZK}}^1$.
  3. Upon receiving $(\mathsf{ZK\text{-}proof}, sid, y)$ from $\mathcal{F}_{\mathrm{ZK}}^1$, A computes $c = \mathsf{aHC}_G(w; r)$ for a random r, and using the y it just received. A then sends $(\mathsf{ZK\text{-}proof}, sid, (c, y), (w, r))$ to $\mathcal{F}_{\mathrm{ZK}}^2$. In addition, A stores in a vector $\overline{w}$ the list of all the values w that were sent, and in vectors $\overline{r}$ and $\overline{c}$ the corresponding lists of random strings and commitment values.
  4. Upon receiving $(\mathsf{ZK\text{-}proof}, sid, (c, y'))$ from $\mathcal{F}_{\mathrm{ZK}}^2$, B verifies that y' equals the string y that it sent in Step 1, outputs $(\mathsf{receipt}, sid)$ and adds the value c to the list $\overline{c}$. (This list is initially empty.) If verification fails then B ignores the message.
- **Prove phase:**
  1. On input $(\texttt{CP-prover}, sid, x)$, A first checks that $R(x, \overline{w}) = 1$. If not, then it sends nothing. If yes, then it sends $(\mathsf{ZK\text{-}prover}, sid, (x, \overline{c}, y), (\overline{w}, \overline{r}))$ to $\mathcal{F}_{\mathrm{ZK}}^3$, where $\overline{w}, \overline{r}, \overline{c}$ are the values described above.
  2. Upon receiving $(\texttt{ZK-proof}, sid, (x, \overline{c}, y'))$ from $\mathcal{F}_{\mathrm{ZK}}^3$, B verifies that y' is the string that it sent above, and that its list of commitments equals $\overline{c}$. If so, then it outputs $(\texttt{CP-proof}, sid, x)$. Otherwise, it ignores the message.

Figure 5: A protocol for realizing $\mathcal{F}_{\mathrm{CP}}$

for details.) We consider a multi-party network where arbitrary subsets of the parties wish to realize arbitrary (possibly reactive) functionalities of their local inputs. For this purpose, any set of parties that engage in a protocol execution are assumed to have access to an *authenticated, synchronous* broadcast channel among them. That is, each message carries a round number and all messages of round i are delivered before any message of round $j > i$. (We stress that these guarantees are local to each protocol execution. No global synchronization or broadcast is assumed.) The broadcast channel is modeled by the following ideal broadcast functionality, $\mathcal{F}_{\mathrm{BC}}$. All communication among the parties is done via $\mathcal{F}_{\mathrm{BC}}$.

Functionality $\mathcal{F}_{\mathrm{BC}}$

Upon receiving a message $(\mathsf{broadcast}, sid, \mathcal{P}, x)$ from P_i, where $\mathcal{P}$ is a set of parties, send $(\mathsf{broadcast}, sid, P_i, x)$ to all parties in $\mathcal{P}$ and to $\mathcal{S}$, and halt.

Figure 6: The ideal broadcast functionality

The outline of our construction is as follows. Similarly to the two-party case, we first construct a multi-party protocol that is secure against semi-honest adversaries. Then, we construct a protocol compiler (like that of GMW), that transforms semi-honest protocols into ones that are secure even against malicious adversaries. This protocol compiler is constructed using a one-to-many extension of the commit-and-prove functionality, denoted $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$. In this brief outline, we focus exclusively on how the extension to $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$ is achieved. See Figure 7 for a formal definition of $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$.

Multi-party Functionality $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$

$\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$, and with security parameter k:

- **Commit Phase:** Upon receiving a message $(\mathsf{commit}, sid, \mathcal{P}, w)$ from P_i where $\mathcal{P}$ is a set of parties and $w \in \{0,1\}^k$, append the value w to the list $\overline{w}$, record $\mathcal{P}$, and send the message $(\mathsf{receipt}, sid, P_i)$ to the parties in $\mathcal{P}$ and $\mathcal{S}$. (Initially, the list $\overline{w}$ is empty.)
- **Prove Phase:** Upon receiving a message $(\mathsf{CP\text{-}prover}, sid, x)$ from P_i, where $x \in \{0,1\}^{\mathrm{poly}(k)}$, compute $R(x, \overline{w})$: If $R(x, \overline{w}) = 1$, then send the message $(\mathsf{CP\text{-}proof}, sid, P_i, x)$ to the parties in $\mathcal{P}$ and to $\mathcal{S}$. Otherwise, ignore.

Figure 7: Multi-party commit-and-prove

To realize $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$, we first realize the multi-party extensions of $\mathcal{F}_{\mathrm{MCOM}}$ and $\mathcal{F}_{\mathrm{ZK}}$. These functionalities, denoted $\mathcal{F}_{\mathrm{MCOM}}^{1:\mathrm{M}}$ and $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$, are defined analogously to $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$. Realizing $\mathcal{F}_{\mathrm{MCOM}}^{1:\mathrm{M}}$ is easy, since the commitment protocol of Section 3 is *non-interactive*. Therefore, a committer P_i can commit to *all* parties by simply broadcasting its commitment string, and $\mathcal{F}_{\mathrm{MCOM}}^{1:\mathrm{M}}$, the multi-party extension of $\mathcal{F}_{\mathrm{MCOM}}$, is immediately obtained. Realizing $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$ is a bit more involved, since we do not know how to securely realize zero-knowledge for adaptive adversaries without interaction. Nevertheless, a multi-party zero-knowledge protocol can be achieved as follows, using the methodology of [31]. The prover runs a copy of the three-round protocol of [12] with each receiver over the broadcast channel, using $\mathcal{F}_{\mathrm{MCOM}}^{1:\mathrm{M}}$ for all the commitments. (Using $\mathcal{F}_{\mathrm{MCOM}}^{1:\mathrm{M}}$ guarantees that all parties receive all the commitments). Furthermore, each verifying party checks that the proofs of all the other parties are also accepting (this is possible because the proof of Hamiltonicity is publicly verifiable and because all parties receive all the commitments). Thus, at the end of the protocol, all honest parties agree (without any additional communication) on whether the proof was successful or not. (Note also that the adversary cannot cause an honest prover's proof to be rejected.) We note that this protocol (denoted MZK) can in fact be used to realize $\hat{\mathcal{F}}_{\mathrm{ZK}}^{1:\mathrm{M}}$, the multi-session extension of $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$, using a single copy of $\mathcal{F}_{\mathrm{MCOM}}^{1:\mathrm{M}}$, and thus a single copy of the reference string. Consequently, we can now use universal composition with joint state to compose any protocol in the $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$-hybrid model (that potentially uses multiple copies of $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$) with a *single instance* of protocol MZK.

It remains to describe how to realize $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$ in the $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$-hybrid model. We present the following protocol, MCP. The prove phase of MCP is the same as in protocol CP, except that $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$ is used instead of $\mathcal{F}_{\mathrm{ZK}}$. The commit phase of MCP proceeds as follows. Let P_i be the committer. Then, for every $j \neq i$, party P_j first chooses a value x_j in the domain of the one-way function f, and computes $y_j = f(x_j)$. Next, P_j broadcasts y_j and proves that it knows a preimage x_j for y_j, using $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$ with the appropriate relation. Finally, the committer P_i *separately* commits to w using every y_j and

proves (using a single instance of $\mathcal{F}_{\mathrm{ZK}}^{1:\mathrm{M}}$ with the appropriate relation) that: (1) it knows all the decommitments, and (2) all the commitments are to the same w.

Finally, we note that, as in the two-party case, a multi-party protocol compiler can be constructed in the $\mathcal{F}_{\mathrm{CP}}^{1:\mathrm{M}}$-hybrid model, with no further assumptions. We conclude with the following theorem:

THEOREM 5. *Assume that trapdoor permutations and multi-party non-committing encryption protocols with the oblivious sampling property exist. Then, for any multi-party ideal functionality $\mathcal{F}$, there exists a protocol Π that securely realizes $\mathcal{F}$ in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{CRS}})$-hybrid model in the presence of* malicious, adaptive *adversaries, and for any number of corruptions.*

As in the two-party setting, for the case of non-adaptive adversaries, non-committing encryption protocols are not needed, and secure protocols are obtained assuming only the existence of trapdoor permutations.

8. REFERENCES

[1] D. BEAVER Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority, *Journal of Cryptology*, Vol. 4, pp. 75–122, 1991.

[2] D. BEAVER, AND S. GOLDWASSER, Multiparty Computation with Faulty Majority, *FOCS 89*.

[3] M. BEN-OR, S. GOLDWASSER AND A. WIGDERSON, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *STOC 1998*.

[4] M. BLUM, "Coin flipping by telephone", *IEEE Spring COMPCOM*, pp. 133-137, Feb. 1982.

[5] M. BLUM How to Prove a Theorem So No One Else Can Claim It. *Proceedings of the International Congress of Mathematicians*, Berkeley, California, USA, 1986, pp. 1444-1451.

[6] M. BLUM, P. FELDMAN AND S. MICALI, Non-interactive zero-knowledge and its applications. *STOC 88*.

[7] M. BLUM, A. DE SANTIS, S. MICALI AND G. PERSIANO, Non-Interactive Zero-Knowledge Proofs. *SIAM Journal on Computing*, vol. 6, December 1991, pp. 1084–1118.

[8] G. BRASSARD, D. CHAUM AND C. CRÉPEAU, Minimum Disclosure Proofs of Knowledge, JCSS, v. 37, pp 156-189.

[9] R. Canetti Security and composition of multi-party cryptographic protocols", *Journal of Cryptology*, Vol. 13, No. 1, winter 2000.

[10] R. Canetti, "Universally Composable Security: A New paradigm for Cryptographic Protocols", *42nd FOCS*, 2001. Full version at http://eprint.iacr.org/2000/067.

[11] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-Party Computation. *STOC 96*.

[12] R. Canetti and M. Fischlin. Universally Composable Commitments. *CRYPTO 01*.

[13] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multiparty Secure Computation. In the *ePrint archive:* http://eprint.iacr.org.

[14] R. Canetti and T. Rabin. Universal Composition with Joint State. IACR's Eprint archive, http://eprint.iacr.org/2002/.

[15] D. Chaum, C. Crepeau, and I. Damgard, Multiparty Unconditionally Secure Protocols, *STOC 88*.

[16] I. Damgard and J. Nielsen. Improved non-committing encryption schemes based on general complexity assumption. *CRYPTO 2000*.

[17] I. Damgard and J. Nielsen Perfect Hiding or Perfect Binding Universally Composable Commitment Schemes with Constanst Expansion Factor. Manuscrtipt on Damgard homepage, 2001.

[18] Y. Dodis and S. Micali Secure Computation, *CRYPTO 2000*.

[19] A. DeSantis, G. DiCrescenzo, R. Ostrovsky, G. Persiano, A. Sahai. Robust Non-interactive Zero-Knowledge. *CRYPTO 2001*.

[20] G. DiCrescenzo, Y. Ishai, and R. Ostrovsky. Non-Interactive and Non-Malleable Commitment. *STOC 98*.

[21] G. DiCrescenzo, J. Katz, R. Ostrovsky and A. Smith Efficient and Non-interactive Non-malleable Commitment, *Eurocrypt 2001*.

[22] A. DeSantis and G. Persiano. Zero-Knowledge Proofs of Knowledge Without Interaction. *FOCS 1992*

[23] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *SIAM. J. Computing*, Vol. 30, No. 2, 2000, pp. 391-437.

[24] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. *STOC 1998*.

[25] S. Even, O. Goldreich and A. Lempel, A randomized protocol for signing contracts, *CACM*, vol. 28, No. 6, 1985, pp. 637-647.

[26] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. *CRYPTO 1989*.

[27] U. Feige, D. Lapidot, and A. Shamir, Multiple non-interactive zero knowledge proofs based on a single random string. *FOCS 1990*.

[28] Z. Galil, S, Haber, and M. Yung Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model, *Crypto 1987*.

[29] J. Garay and P. MacKenzie Concurrent Oblivious Transfer, *FOCS 2000*.

[30] O. Goldreich, S. Micali and A. Wigderson, Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38, No. 1, pages 691–729, 1991.

[31] O. Goldreich. Secure Multi-Party Computation. Manuscript. Preliminary version, 1998. www.wisdom.weizmann.ac.il/∼oded/pp.html.

[32] O. Goldreich and L. Levin, A Hard Predicate for All One-way Functions. *STOC 1989*.

[33] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. *STOC 1987*. For details see [31].

[34] S. Goldwasser, and L. Levin, Fair Computation of General Functions in Presence of Immoral Majority, *CRYPTO 1990*.

[35] S. Goldwasser and S. Micali. Probabilistic Encryption. In *JCSS* 28(2), pages 270-299, 1984.

[36] S. Goldwasser, S. Micali and C. Rackoff, The Knowledge Complexity of Interactive Proof Systems, *SIAM Journal on Comput.*, Vol. 18, No. 1, 1989, pp. 186-208.

[37] J. Kilian Uses of Randomness in Algorithms and Protocols, Chapter 3, The ACM Distinghished Dissertation 1989, MIT press.

[38] E. Kushilevitz, S. Micali and R. Ostrovsky, Reducibility and Completeness In Multi-Party Private Computations, *FOCS 94*.

[39] Y. Lindell, A. Lysyanskaya and T. Rabin, On the composition of authenticated Byzantine agreement, *STOC 2002*.

[40] S. Micali and P. Rogaway, Secure Computation, unpublished manuscript, 1992. Preliminary version in *CRYPTO '91, LNCS 576*, Springer-Verlag, 1991.

[41] M. Naor, Bit Commitment using Pseudorandom Generators. *Journal of Cryptology*, Vol. 4, pages 151–158, 1991.

[42] B. Pfitzmann and M. Waidner, Composition and integrity preservation of secure reactive systems, *7th ACM Conf. on Computer and Communication Security*, 2000, pp. 245-254.

[43] M. Rabin, How to exchange secrets by oblivious transfer, Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[44] T. Rabin and M. Ben-Or Verifiable Secret Sharing and Multi-party Protocols with Honest Majority, *STOC 1989*.

[45] R. Richardson and J. Kilian On the Concurrent Composition of Zero-Knowledge Proofs. *Eurocrypt 1999*.

[46] A. Sahai Non-Malleable Non-Interactive Zero-Knowledge and Adaptive Chosen-Ciphertext Security. *FOCS 1999*.

[47] A. Yao How to generate and exchange secrets, *FOCS 1986*.