Cryptographic Protocols Provably Secure Against Dynamic Adversaries

Donald Beaver[1] and Stuart Haber[2]

[1] 313 Whitmore Lab, Penn State University, University Park, PA 16802, U.S.A.
beaver@cs.psu.edu
[2] Bellcore, 445 South St., Morristown, NJ 07960-1910, U.S.A.
stuart@bellcore.com

Abstract. We introduce new techniques for generating and reasoning about protocols. These techniques are based on protocol transformations that depend on the nature of the adversaries under consideration. We propose a set of definitions that captures and unifies the intuitive notions of correctness, privacy, and robustness, and enables us to give concise and modular proofs that our protocols possess these desirable properties.
Using these techniques, whose major purpose is to greatly simplify the design and verification of cryptographic protocols, we show how to construct a multiparty cryptographic protocol to compute any given feasible function of the parties' inputs. We prove that our protocol is secure against the malicious actions of any adversary, limited to feasible computation, but with the power to eavesdrop on all messages and to corrupt any *dynamically chosen* minority of the parties. This is the first proof of security against dynamic adversaries in the "cryptographic" model of multiparty protocols. We assume the existence of a one-way function and allow the participants to erase small portions of memory. Our result combines the superior resilience of the cryptographic setting of [GMW87] with the stronger (dynamic) fault pattern of the "non-cryptographic" setting of [BGW88,CCD88].

1 Introduction

A large body of recent work in distributed computing has addressed the problem of constructing protocols whereby n parties can cooperatively compute a function of n arguments, each secretly held by one party. Much of this work can be classified either as the "cryptographic" approach or as the "non-cryptographic" approach. In the non-cryptographic scenario, one posits the existence of a complete network of private communications channels connecting all pairs of users, something that may not exist in practice. In this optimistic situation, [10, 12, 28, 3] have shown that one can tolerate misbehavior by any dynamically chosen colluding minority of faulty players. In the cryptographic scenario, on the other hand, we rely on unproven assumptions about the complexity of certain computational problems instead of on "physical" assumptions about the network. Here again, [23, 24] have shown that one can tolerate faulty behavior by a minority of the players, but the proof techniques in the literature seem to require that the

R.A. Rueppel (Ed.): Advances in Cryptology - EUROCRYPT '92, LNCS 658, pp. 307-323, 1993.

subset of misbehaving parties be chosen "statically," that is at the beginning of the protocol execution.

Our contribution. We introduce a new technique for enhancing the security of multiparty protocols. Beginning with an "ideal protocol" for a computational problem, we proceed by a sequence of reductions in order to generate a maximally secure protocol. One result we can achieve by this technique is to combine the virtues of the best known constructions for both the *cryptographic* and the *non-cryptographic* scenarios, so that we are able to achieve all desired properties of a multiparty protocol for the evaluation of any feasible probabilistic function on private inputs: our protocol does not require private channels, it tolerates a fully dynamic, fully Byzantine adversary, and it is very simple to describe. With novel, concise, and penetrating definitions, our techniques support direct, modular proofs that provide a deep understanding not only of the bounds of cryptographic protocols but of the nature of security itself.

Defining security; desirable properties; proving security. In the past few years many different multiparty protocols have been proposed [23, 24, 17, 10, 12, 2, 28, 3, 7, 16] for various purposes, and authors have given separate arguments for the correctness, privacy, resilience, independence, fairness, robustness, simultaneity, etc. of the computations performed by these protocols. It can be very difficult to compare different attempts to define the same property, let alone to compare the (more or less formal) definitions of these different properties. Continuing the work of [3, 5, 6], our new definitions enable us to write surprisingly simple and clear proofs of the security of many of the protocol transformations in the literature—including, of course, the new ones we introduce in this abstract. Following Chor and Rabin [13], we argue that "in the slippery business of distributed cryptographic protocols, simpler proofs are important."

The introduction of zero-knowledge proofs [21] made two important contributions to the field of cryptography: first, the new idea of proving an assertion to be true without giving away any additional information; and second, the technical insight that feasible simulatability is a good way to talk about and reason about "not giving away information." The idea of zero-knowledge proofs inspired much fruitful research, resulting in many new procedures. In trying to state the properties of these procedures precisely, researchers have had trouble. Distracted, as it were, by the technical difficulties of simulation, they have been drawn away from the main point, which is to achieve, in the absence of trusted parties, various goals that would be easy to achieve in a world where trusted hosts of various sorts were available.

Here, we get back to the point. We describe a way to formally compare two different protocols, and then we compare any proposed protocol to the ideal protocol—usually ridiculously easy to design—that one can write for an ideal world that (by *fiat*) includes trusted hosts. Zero-knowledge uses Yao's notion of indistinguishability of ensembles [31] in reasoning about feasible simulation. Our definitions and proof techniques use indistinguishability to reason not just about the *information* an adversary can glean from a protocol execution but also about the *influence* the adversary can have on the course of an execution.

Static *vs.* dynamic adversaries. The difference between tolerating static and tolerating dynamic adversaries is crucial to distributed system security. It is inconceivable that in exactly the scenario where all the transmitted information is exposed to the adversary (even though in an encrypted form), the adversary is not allowed to take advantage of this in its corruption policy. The major difficulty in the case of cryptographic protocols is that, having access to on-line communication, a dynamic adversary's behavior can depend on this communication in an unpredictable manner. Standard simulation arguments fail, preventing even a proof of *privacy* against a dynamic adversary from having appeared in the literature (despite claims of dynamic security (*cf.* [24, 16])).

It has recently come to the authors' attention that cryptographic techniques similar to ours have been used previously to address the question of simulating private-channels protocols in the cryptographic model [15]. However, this work was not primarily concerned with the question of properly defining the desired security, and the proof sketches are stated in terms of rather incomplete definitions.

With clear and simple definitions and proof techniques, we are able to describe a simple protocol transformation demonstrating that if processors can erase very small portions of memory, then *any result developed for the non-cryptographic model holds also for the cryptographic model, even in the presence of dynamic adversaries.*

Efficiency. Our protocol transformation is remarkably efficient in terms of overhead. The simplifying but unrealistic assumption that private channels exist provides efficient protocols for the non-cryptographic model; our transformation preserves the simplicity of non-cryptographic protocols while deftly handling the difficulties encountered in proof techniques for the cryptographic scenario. At little additional cost we supply the guarantee of *provable security* with the ease of simple protocol design.

Remarks. From a practical standpoint, our assumption that processors may erase memory is far less unreasonable than absolutely private channels, given physical circuits with volatile memory that is destroyed upon tampering. Our transformations require each processor to erase a very small portion of memory, on the order of the size of the keys needed to encrypt or decrypt messages.

Our purpose in this abstract is, first, to demonstrate the power of our new technique to generate and reason about protocols; second, to apply the technique in order to show that it is *possible* to achieve security against a fully dynamic adversary in the cryptographic scenario; and, third, to *prove* that we have achieved this level of security.

Contents. Section 2.2 defines relative resilience, our main new technical tool; §2.3 proves some supporting theorems; §3 describes modular protocol transformations and proves that they produce a secure protocol. Details of our proofs are contained in the appendix.

2 Definitions and proof techniques

2.1 Background mechanics

Background. Let $\Sigma = \{0,1\}$ with symbols such as delimiters (#) encoded in a natural way. Let $[n] = \{1,\ldots,n\}$ and let $\mathbf{x} = (x_1,\ldots,x_n)$. Let $\mathtt{dist}(X)$ be the set of distributions on a set X (normally finite), and let $uniform(X)$ be the uniform distribution. Let PPTM be the set of probabilistic polynomial-time TM's. Let PFF be the set of functions mapping $(\Sigma^m)^n \to \mathtt{dist}((\Sigma^m)^n)$ that are described by poly-size circuit families $\{C_F(n,m)\}$ where each circuit has a certain number of distinguished, "random" inputs.

The difference $|P-Q|$ between distributions $P,Q \in \mathtt{dist}(X)$ is $\sum_X |\Pr_P[x] - \Pr_Q[x]|$. A **probabilistic function** is a function whose range contains distributions, namely $f : X \to \mathtt{dist}(Y)$. The **composition** of probabilistic functions g and f is given by $\Pr_{g \circ f(x)}[z] = \sum_y \Pr_{f(x)}[y] \Pr_{g(y)}[z]$. An **ensemble** is a probabilistic function $P : \Sigma^* \times \mathbf{N} \to \mathtt{dist}(\Sigma^*)$ such that $\Pr_{P(z,k)}[x] = 0$ for $|x| > k^c$ and some fixed c. Two ensembles P and Q are **computationally indistinguishable** ($P \approx Q$) if

$$(\forall T \in \text{PPTM})(\forall c > 0)(\exists k_0)(\forall k \geq k_0)(\forall z) \quad |T(P(z,k)) - T(Q(z,k))| < k^{-c}$$

where $T(D)$ indicates the probability T outputs 1 on input distribution D. Two *families* of ensembles $\{P_i\}$ and $\{Q_i\}$ are **uniformly**[3] **computationally indistinguishable** if

$$(\forall T \in \text{PPTM})(\forall c > 0)(\exists k_0)(\forall k \geq k_0)(\forall z) \ (\forall i) \ |T(P_i(z,k)) - T(Q_i(z,k))| < k^{-c}.$$

Cryptographic tools. When we take a protocol designed for a network with private channels and try to implement it without such channels, the only method we have available to send private messages is to use cryptographic means. Therefore, in order to prove the security of our protocols, we shall make the complexity-theoretic assumption that **one-way trapdoor permutations** exist. Let **pkc** denote a key generator for a polynomially secure **probabilistic public-key encryption scheme** [20]; this is a probabilistic algorithm that, on input 1^K and $\kappa(K)$ random bits, produces an (encryption key, decryption key) pair that we write (E,D). For an encryption key E, we write $c = E(m,\rho)$ for an encryption of message m using random bits ρ. Let **prg** denote a **cryptographically strong pseudorandom number generator**, an algorithm mapping K bits to $B(K)$ bits [27, 26]. We will also make use of **zero-knowledge proofs of knowledge**; in our application, these will be proofs of knowledge of m performed by the sender of a ciphertext $c = E(m,\rho)$ [14, 30, 19, 9].

Networks and protocols. A player is an interactive PPTM having a random tape, input tape, output tape, and work tape. The I/O tapes may encode several (n) different tapes for communication with several machines. The superstate S_i of player M_i is a string describing its finite control, current state, and contents of

[3] "Uniform" refers to uniform convergence, not TM-computability.

all tape squares read or written so far. With messages $\mu(1,i),\ldots,\mu(n,i)$ written on its input tape, M_i induces a transition $\delta(S_i,\mu(1,i)\cdots\mu(n,i))=(S_i',\boldsymbol{\mu})$, where $\boldsymbol{\mu}$ is a list of outgoing messages to be sent on specific communication channels.

A **channel** is a probabilistic function $C:\Sigma^\star \to \mathtt{dist}(2^{\mathbf{N}\times\mathbf{N}\times\Sigma^\star})$. For example, a **private channel** from i to j satisfies $\Pr_{C(m)}[\{(i,j,m)\}]=1$, while a **broadcast channel** from i is $\Pr_{C(m)}[\{(i,1,m),\ldots,(i,n,m)\}]=1$. Other channels, such as oblivious transfer (noisy) channels, are equally easily described. A **network** is a set of channel functions.

A (synchronous) **protocol** is a collection of sets of players $\Pi=\{(M_1,\ldots,M_n)\}_{n\in\mathbf{N}}$. It is implementable on a network if the messages output by players specify channels in that network. For each n (number of players), m (size of inputs), k (protocol security parameter), $\mathbf{x}\in(\Sigma^m)^n$ (inputs), and $\mathbf{a}\in(\Sigma^\star)^n$ (auxiliary inputs), a protocol Π induces a distribution $\Pi(n,m,k,\mathbf{x}\circ\mathbf{a})$ on outputs by running player M_i on input tape $X_i=1^n\#1^m\#1^k\#x_i\#a_i$. Letting $z=1^n\#1^m\#1^k\#\mathbf{x}\#\mathbf{a}$, this defines an ensemble. The mechanics of protocol execution are straightforward but tedious (see *e.g.* [3, 5]): a sketch with loose notation follows. The notation $\mu(A,B,r)$ denotes the messages sent from players in set A to those in B during round r. Let $R(n,m,k)$ be the number of rounds taken by a protocol (alternatively, the protocol can run until all nonfaulty players decide to halt).

Synchronous Protocol Execution
For $i=1..n$ **do**
  $\mu^{in}(i,i,0)=X_i$
For $r=1..R(n,m,k)$ **do**
  For $i=1..n$ **do in parallel**
    /* `Compute locally (function Local):` */
      $(S_i^r,\mu^{out}(i,[n],r))\leftarrow\delta(S_i^{r-1},\mu^{in}([n],i,r-1))$
    /* `Apply channel functions (function Channel):` */
      $\mu^{del}([n],[n],r)\leftarrow C(\mu^{out}([n],[n],r))$
    /* `Deliver messages output from channels` */
    /* `(function Deliver):` */
      $\mu^{in}([n],[n],r+1)\leftarrow\mu^{del}([n],[n],r)$

The main point is to consider the induced probabilistic functions. Let $S_g=[(S_1,\mu([n],1),\ldots,(S_n,\mu([n],n))]$ represent a global state of the system. A round and an execution of a synchronous protocol are described by

$$\mathbf{Round}(S_g)=\mathbf{Deliver}(\mathbf{Channel}(\mathbf{Local}(S_g)))$$
$$\mathbf{Exec}(S_g)=\mathbf{Round}^{R(n,m,k)}(S_g).$$

Let $Y_i:\Sigma^\star\to\Sigma^\star$ be an **output function**; the **output** of M_i is $y_i=Y_i(S_i^{R(n,m,k)})$. The **view** v_i of M_i is the list of states and messages it has seen. In the **memoryless model**, however, the view consists only of the final state and output; intermediate states and tape contents are *not* recorded, and information can thus be *erased* by a player. The view of a player during an intermediate

round of the protocol includes all its states and messages up to that point; in the memoryless model, an intermediate view contains only the current state, tape contents, and incoming and outgoing messages.

Adversaries. An **adversary** $\mathcal{A}$ is an interactive TM with *one* communication line, on which it makes *corruption requests*. In this paper, all adversaries are probabilistic polynomial-time TM's. A **Byzantine** adversary requests either the view of a player i or requests to replace outgoing messages from corrupted players. Note that an adversary may change input and random tapes before the protocol starts (*i.e.* before any messages are generated). A **passive** adversary cannot change messages. If the adversary superstate is S_A, let $T = T(S_A)$ denote the set of players it has corrupted. A **t-adversary** satisfies $|T(S_A)| \leq t$. A **static** adversary satisfies $T(S_A) = T_0$ for some fixed T_0. A **rushing, dynamic** adversary sees $\mu^{del}(\overline{T}, T, r)$ (step 2.2.2) before choosing whom to corrupt and how to corrupt them. We remark that our adversary model is quite robust; we allow players to be interrupted and corrupted even in the middle of a computation. The function **Fault** provides $\mathcal{A}$ with requested information and allows it to compute a new request. The function **Replace** allows $\mathcal{A}$ to change outgoing messages $\mu^{out}(T, [n], r)$, which are then passed through channels, changing some of the messages in $\mu^{del}(T, [n], r)$ in step 2.2.2. An execution of a protocol with adversary is:

$$\mathbf{RoundA}(S_g, S_A) = \mathbf{Deliver}(\mathbf{Replace}(\mathbf{Fault}^t(\mathbf{Channel}(\mathbf{Local}(S_g, S_A)))))$$
$$\mathbf{ExecA}(S_g, S_A) = \mathbf{RoundA}^{R(n,m,k)}(S_g, S_A).$$

Let a_A be an auxiliary input for $\mathcal{A}$ and let y_A denote its output $y_a = Y_A(S_A)$. An execution thus maps $\mathbf{x}\cdot\mathbf{a}\cdot a_A$ to $\mathbf{y}\cdot\mathbf{v}\cdot y_A$, where $\mathbf{v}$ is the list of player views (including final states) and $\mathbf{y} = (Y_1(S_1), \ldots, Y_n(S_n))$ is the list of outputs. We define the induced ensemble that describe adversary and player outputs as

$$[\mathcal{A}, \Pi]' = (Y_1(S_1), \ldots, Y_n(S_n), Y_A(S_A)),$$

and we use a subscript A (resp. $1..n$) to refer to the restriction to $Y_A(S_A)$ (resp. $(Y_1(S_1), \ldots, Y_n(S_n))$).

2.2 Security

We come to the important definitions: *relative resilience* and *absolute resilience*.

The principle behind zero-knowledge [21] is that a simulator that produces accurate verifier (adversary) views, given only information "$x \in L$," shows that a real execution leaks no additional information. This idea covers only half the picture of interaction, however. In addition to *information*, there is the *influence* an adversary has on the outputs of nonfaulty players. (This is overlooked by ZK because a faulty verifier doesn't "influence" the final output of the prover, which is irrelevant in ZK proof systems—only the verifier's decision is considered.) In a protocol, the influence is reflected in the distributions on final outputs of nonfaulty players. Previous approaches to security attempted to deal with

desired properties (*e.g.* correctness, independence) separately; we unify them all by choosing to consider the ensemble $[\mathcal{A}, \Pi]$.

We present a tool called **relative resilience**, introduced by the first author in [3, 5], that allows us to compare the security of one protocol to another. Resilience is a combination of several properties, including privacy and correctness. In order to say that α is as resilient as β, we should like that an adversary who attacks α is also allowed to attack β. If its *information* and *influence* on α are the same as on β, then α is intuitively as secure—or *resilient*—as β. But $\mathcal{A}$ may be incompatible with β for many reasons: the network may differ from protocol α, the communication format may differ, and so on. We give $\mathcal{A}$ an *interface* $\mathcal{I}$ to allow it to attack β. Because the interface should not give $\mathcal{A}$ undeserved extra power, the combination $\mathcal{I}(\mathcal{A})$ (interface together with adversary, regarded as a single machine) should not exceed the power allowed to adversaries attacking β. For example, when adversaries are poly-time bounded, the interface itself must be poly-time as well.

Definition. *An* **interface** *is an interactive TM $\mathcal{I}$ with two tapes, an "environment simulation" tape and an "adversarial" tape. On the former, $\mathcal{I}$ receives and responds to messages from an adversary; on the latter, $\mathcal{I}$ passes on requests and receives responses as an adversary in its own right. An* **interface from α to β** *is such that for all $\mathcal{A} \in \mathtt{A}_\alpha$ we have $\mathcal{I}(\mathcal{A}) \in \mathtt{A}_\beta$, where $\mathtt{A}_\alpha$ (resp., $\mathtt{A}_\beta$) is the class of allowed adversaries for α (resp., β).*

The interface is the proper generalization of a simulator to an interactive setting where all properties (not just privacy) are important. Unless otherwise specified, we shall be concerned with t-bounded polynomial-time message-rushing, dynamic, Byzantine adversaries.

A preliminary definition for resilience captures the essential intuition: protocol α is *weakly as resilient as* protocol β if there exists an interface satisfying $[\mathcal{A}, \alpha]' \approx [\mathcal{I}(\mathcal{A}), \beta]'$. While this would suffice for a single protocol execution, we require an additional property in order to show that concatenating protocols preserves resilience.

Post-Protocol Corruption. After a protocol is finished, the views become auxiliary inputs to later protocols. Therefore, an interface for an early protocol α should be able to generate views of players that had not been corrupted *during* α, so that an interface for a later protocol can include these views in the auxiliary information captured by the later adversary. An interface satisfying our weak definition might not even be well-defined for corruption requests occurring *after* the protocol is finished. We therefore consider an interface $\mathcal{I}$ designed to respond to post-protocol requests. The adversary is allowed to send request ppc, in which case it receives *all* outputs $(y_1, \ldots, y_n)$, but *not* the views. The interface receives nothing. Adversary $\mathcal{A}$ can then request new corruptions j (up to its t-limit) and $\mathcal{I}$ must respond with outputs and newly-synthesized views. $\mathcal{I}$ is allowed to continue requesting corruptions in completed protocol β (up to its t-limit), obtaining accurate output y_j, input x_j, and view v_j^β of β—from which it must synthesize a convincing view v_j^α of α. Thus, $\mathcal{A}$ is able to test—in a strong fashion—the ability

of the interface to answer continual requests, even after the given protocols are finished.

The variable Y_A^{ppc} refers to $\mathcal{A}$'s output when it is allowed to elect post-protocol corruption. The induced protocol ensemble is denoted:

$$[\mathcal{A}, \Pi] = (Y_1(S_1), \ldots, Y_n(S_n), Y_A^{ppc}(S_A)).$$

We can now concisely define relative resilience:

Definition. (Relative Resilience) *Protocol α is* **as resilient as** *protocol β, written $\alpha \succeq \beta$, if there exists an interface $\mathcal{I}$ from α to β such that for all $\mathcal{A} \in \mathtt{A}_\alpha$,*

$$[\mathcal{A}, \alpha] \approx [\mathcal{I}(\mathcal{A}), \beta].$$

Protocol α is **as private as** β if $[\mathcal{A}, \alpha]_A \approx [\mathcal{I}(\mathcal{A}), \beta]_A$ and it is **as correct as** β if $[\mathcal{A}, \alpha]_{1..n} \approx [\mathcal{I}(\mathcal{A}), \beta]_{1..n}$.

Absolute Resilience. The measure of absolute resilience is given by the standard of an *ideal* protocol. A **real** protocol has a t-adversary class and n players; no player is above corruption. An **ideal** protocol contains one or more *trusted hosts* (players $(n+1), (n+2), \ldots$) who cannot be corrupted. The **ideal t-adversary class** includes all t-bounded poly-time adversaries that corrupt players in the range $1..n$ (excluding trusted hosts $n+1$, $n+2$, *etc.*). If $F \in \mathrm{PFF}$ is a probabilistic finite function, the **ideal protocol for** F, $\mathtt{ID}(F)$, has two rounds: all players send their inputs to trusted host $(n+1)$, who then computes F and returns the values.

Definition. (Resilience) *Π is a t-resilient protocol for F if $\Pi \succeq \mathtt{ID}(F)$.*

Computational issues. Although our definitions are presented with resource-bounded adversaries in mind, our approach works equally well when the notions of *statistical* or *perfect* indistinguishability is employed, and adversary classes include unbounded TM's (or even nonrecursive functions).

Comparison to other definitions. Several authors [18, 8, 4] have considered a *fault-oracle* approach, an extension of ZK in which a simulator constructs an adversary's view based on a single request for a computation of F, which thereby induces player outputs as well. Though the fault-oracle approach can be regarded as a step in the right direction, it is limited to comparing a protocol to a function computation, and it is rather inflexible. It does not support modular proofs: it is not clear how the concatenation of two secure protocols can be proven secure, since one must convert two oracle calls to a single one. Other approaches [7, 25] also suffer from the limitation that arbitrary, different protocols cannot easily be compared.

Zero-Knowledge at One Blow. In the **ideal zero-knowledge proof system,** denoted $\mathtt{ID}(L)$, player P sends "$x \in L$" to trusted host TH, who calculates whether $x \in L$ and sends "$x \in L$" to V if so. The host otherwise sends "?" to indicate a failed proof. One of the more attractive uses of *relative resilience* is an equivalent definition of the classical notion of zero-knowledge proof system [21] in one sentence: A two-party protocol $\Pi = (P, V)$ is a **zero-knowledge proof system** for L iff it is as resilient as $\mathtt{ID}(L)$, against static 1-adversaries.

2.3 Proof techniques and now-provable folk theorems

Many intuitively justified approaches become provable using relative resilience. Pitfalls in the naive application of intuition are brought out by our formal definitions. The contribution of this section is a formal statement of provable theorems and the necessary conditions under which they hold. Concise proofs of the non-cryptographic versions of these theorems appear in the dissertation of the first author [5]. Full proofs for the cryptographic setting are short and direct but would require more space than permitted here.

The concatenation $\alpha_2 \circ \alpha_1$ is defined by setting the inputs and auxiliary inputs $\mathbf{x}(2)\cdot\mathbf{a}(2)\cdot a_A(2)$ to protocol α_2 to be the outputs and views $\mathbf{y}(1)\cdot\mathbf{v}(1)\cdot y_A(1)$ generated by running α_1 on the original inputs $\mathbf{x}(1)\cdot\mathbf{a}(1)\cdot a_A(1) = \mathbf{x}\cdot\mathbf{a}\cdot a_A$. We say that protocol family $\{\alpha_i\}$ is **uniformly as t-resilient as** family $\{\beta_i\}$ if there exists a family $\{\mathcal{I}_i\}$ of interfaces so that for all adversaries $\mathcal{A} \in \mathtt{A}_{\alpha_i}$ the ensemble families $\{[\mathcal{A}, \alpha_i]\}$ and $\{[\mathcal{I}_i(\mathcal{A}), \beta_i]\}$ are uniformly indistinguishable. Recall that the term "uniform" denotes not the Turing-machine notion but the mathematical notion of uniform convergence. If the requirement of uniform convergence is relaxed, counterexamples to the results stated below are easy to find [3], demonstrating unforeseen pitfalls in the blind use of folk-theorems.

Define the **composition** of $f(n, m, k)$-many ensembles from $\{P_i\}$ as $P^f(z, k) = P_{f(n,m,k)}(P_{f(n,m,k)-1}(\cdots P_1(z, k)\cdots), k)$. The **concatenation** of $f(n, m, k)$-many protocols from $\{\alpha_i\}$ is the protocol α^f such that $\alpha^f(n, m, k, \mathbf{x}\cdot\mathbf{a}\cdot a_A) = \alpha_{f(n,m,k)} \circ \cdots \circ \alpha_1(n, m, k, \mathbf{x}\cdot\mathbf{a}\cdot a_A)$.

Lemma 1. (Composing poly-many ensembles) *If $\{P_i\}$ is* uniformly *computationally indistinguishable from $\{Q_i\}$ and if $f(n, m, k)$ is polynomially bounded, then $P^f \approx Q^f$.*

Proof. A proof for perfect and statistical indistinguishability appears in [3, 5]; the proof for computational indistinguishability is direct and follows the same lines. □

Theorem 2. (Concatenating poly-many protocols) *If $\{\alpha_i\}$ is* uniformly *as t-resilient as $\{\beta_i\}$ and if $f(n, m, k)$ is polynomially bounded, then $\alpha^f \succeq \beta^f$.*

Proof. The proof uses Lemma 1 and follows a proof for statistical resilience in [3]. □

Theorem 3. (Transitivity of $\succeq$) *$\alpha \succeq \beta$ and $\beta \succeq \gamma$ imply $\alpha \succeq \gamma$. Polynomially many applications work as well: If $\{\alpha_{i+1}\}$ is* uniformly *as t-resilient as $\{\alpha_i\}$,[4] and if $f(n, m, k)$ is polynomially bounded, then $\alpha_f \succeq \alpha_0$.*

Proof. See [3, 5]; the interface $\mathcal{I}$ from α_f to α_0 internally runs $f(n, m, k)$ nested interfaces $\mathcal{I}_{f,f-1} \circ \cdots \circ \mathcal{I}_{2,1} \circ \mathcal{I}_{1,0}$. □

[4] Roughly speaking, $(\forall i)\alpha_{i+1} \succeq \alpha_i$.

The **ideal vacuous protocol**, $\mathtt{ID}(0)$, returns no result. A **t-threshold scheme** is a pair of protocols $(\textsc{Sha}, \textsc{Rec})$ computing probabilistic functions $(\mathbf{sha}, \mathbf{rec})$ such that

1. $\mathbf{rec}$ is t-robust (*i.e.*, insensitive to $\leq t$ changes in inputs);
2. $\mathbf{sha}$ is t-private (*i.e.*, $\mathtt{ID}(\mathbf{sha}) \succeq \mathtt{ID}(0)$);
3. $\mathbf{rec} \circ \mathbf{sha}(x) - x$.

Define $\mathbf{hide}(H) = \mathbf{sha} \circ H \circ \mathbf{rec}$, the probabilistic function that reconstructs a shared value, computes H, and shares it again. The standard share/compute/reveal paradigm for multiparty protocols is to express F as $F^f = \circ_1^f F_i$ and compute $\mathbf{rec} \circ [\circ_1^f \mathbf{hide}(F_i)] \circ \mathbf{sha}$. That is, inputs are secretly shared; intermediate values are secretly computed but not revealed; then the final output is reconstructed. The "folk theorem" claiming that this method is secure, but whose intuitive statement is sometimes false, is formalized and proven as:

Theorem 4. ("Completeness" paradigm) *Let* $(\textsc{Sha}, \textsc{Rec})$ *be a t-threshold scheme, and let* $F = F^f = \circ_1^f F_i$ *for some polynomially bounded* $f(n, m, k)$. *If* $\{\alpha_i\}$ *is* uniformly *as t-resilient as* $\{\mathtt{ID}(\mathbf{hide}(F_i))\}$, *then*

$$\textsc{Rec} \circ [\circ_1^f \alpha_i] \circ \textsc{Sha} \succeq \mathtt{ID}(F^f).$$

Proof. *Uniformity* is essential. The proof uses Theorems 2 and 3 and the robustness and privacy of $(\mathbf{rec}, \mathbf{sha})$ but is omitted (*cf.* [3, 5], however). □

3 Achieving security against dynamic adversaries

In this section we describe our construction of a cryptographic multiparty protocol that is t-resilient against *dynamic* adversaries for $t < n/2$, for any $F \in \mathrm{PFF}$. Recall that n is the number of players and m is the size of the inputs. Let $K = k + n + m$. A **feasible** protocol has polynomial-size messages: assume that each message from i to j has fixed length $L(K)$, for a total of $B(K) = O(K^c)$ bits over $B(K)/L(K)$ rounds, for some c. If p_{ij} is a string of $B(K)$ bits, let $p_{ij}(r)$ denote the r^{th} block of $L(K)$ bits. Let $\mathbf{pkc}$ be a public-key generator requiring $\kappa(K)$ random bits, and let $\mathbf{prg}$ be a cryptographically strong pseudorandom generator from K to $B(K)$ bits. Figure 1 describes the transformation and subprotocols used in the transformation and proof of security for our main results.

Theorem 5. (Main Result I) *If one-way trapdoor functions exist, then in the memoryless model, private channels can be replaced by public (broadcast) channels at a cost of 2 extra rounds: for all feasible* Π, $\textsc{Transform}(\Pi) \succeq \Pi$.

TRANSFORM(Π) = PAD(Π) ∘ SENDSEED.

PAD(Π) /* Use pads, not channels */
- Input for player i is $X_i = (x_i, p_{i1}, \ldots, p_{in}, p_{1i}, \ldots, p_{ni})$
- Run Π on x_i, except replace steps "(r) $i \to j : m_{ij}[\to m'_{ij}]$" by:
  - (r) $c_{ij} \leftarrow p_{ij}(r) \oplus m_{ij}$
  - (r) **i broadcasts:** $c_{ij}[\to c'_{ij}]$
  - $(r+1)$ **receive** c'_{ji}; $m'_{ji} \leftarrow p_{ji}(r) \oplus c'_{ji}$

SENDPAD /* Send random pad over private channel */
1 $p_{ij} \leftarrow$ *uniform*$(\{0,1\}^{B(K)})$
  $i \to j$: $p_{ij}[\to p'_{ij}]$
2 **receive** p'_{ji}; **output** $(x_i, p_{i1}, \ldots, p_{in}, p'_{1i}, \ldots, p'_{ni})$

SENDPSPAD /* Send pseudorandom pad over private channel */
1 $s_{ij} \leftarrow$ *uniform*$(\{0,1\}^K)$; $p_{ij} \leftarrow \mathrm{prg}(s_{ij})$
  $i \to j$: $s_{ij}[\to s'_{ij}]$
2 receive s'_{ji}; **if not** K **bits then** $s'_{ji} \leftarrow 0^K$
  $p'_{ji} \leftarrow \mathrm{prg}(s'_{ji})$
  erase all except output $(x_i, p_{i1}, \ldots, p_{in}, p'_{1i}, \ldots, p'_{ni})$

SENDSEED /* Send pseudorandom pad using encryption */
1 $r_i \leftarrow$ *uniform*$(\{0,1\}^{\kappa(K)})$; $(E_i, D_i) \leftarrow \mathrm{pkc}(1^K, r_i)$
  i broadcasts: $E_i[\to E'_i]$
2 receive E'_j; **if bad then** $E'_j \leftarrow$ identity function
  $s_{ij} \leftarrow$ *uniform*$(\{0,1\}^K)$; $p_{ij} \leftarrow \mathrm{prg}(s_{ij})$
  $r_{ij} \leftarrow$ *uniform*$(\{0,1\}^{\epsilon(K)})$; $\sigma_{ij} \leftarrow E'_j(s_{ij}, r_{ij})$
  i broadcasts: $\sigma_{ij}[\to \sigma'_{ij}]$;
  i and j execute a zero-knowledge proof of i's knowledge of s_{ij}
3 receive σ'_{ji}; **if** proof fails or $s'_{ji} \leftarrow D_i(\sigma'_{ji})$ fails **then** $s'_{ji} \leftarrow 0^K$
  $p'_{ji} \leftarrow \mathrm{prg}(s'_{ji})$
  erase all except output $(x_i, p_{i1}, \ldots, p_{in}, p'_{1i}, \ldots, p'_{ni})$

Fig. 1. Subprotocols for our transformation. Code for player i; take $1 \le i, j \le n, j \ne i$. "$(r)$ $i \to j$: $m[\to m']$" means i should send m to j at round r over a private channel, and m' denotes the message actually sent. See text for pkc, prg, $\kappa(K), \epsilon(K), B(K), p_{ij}(r)$.

Proof. TRANSFORM(Π) assumes only broadcast channels whereas Π may use private channels. The proof is modular: first, replace private channels by uniform one-time pads sent in a preprocessing stage over private channels; then replace uniform one-time pads by pseudorandom pads also sent initially over private channels; finally, replace initial private channels by public-key encryptions of the seeds for the pads. Keys and seeds are erased before the body of the protocol starts. The proof follows from Lemmas 7, 8, 9 (listed below), using

transitivity (Theorem 3):

$$\begin{aligned}\textsc{Transform}(\Pi) &= \textsc{Pad}(\Pi) \circ \textsc{SendSeed} \succeq \textsc{Pad}(\Pi) \circ \textsc{SendPsPad} \\ &\succeq \textsc{Pad}(\Pi) \circ \textsc{SendPad} \succeq \Pi.\end{aligned}$$ □

Theorem 6. (Main Result II) *If one-way trapdoor functions exist, then in the memoryless model, for $t(n) < n/2$ and any probabilistic finite function $F \in$ PFF, there is a computationally t-resilient protocol for F secure against dynamic, message-rushing, Byzantine adversaries.*

Proof. Rabin, Ben-Or [28] and Beaver [3] demonstrate the existence of a protocol $\Pi(F)$ using private and broadcast channels that is statistically (hence computationally) t-resilient against dynamic, rushing, Byzantine t-adversaries, for any $t(n) < n/2$ and $F \in$ PFF. (Beaver's proof uses "resilience" as defined here.) Theorem 5 suffices.[5] □

Proofs of the following lemmas are given in the Appendix.

Lemma 7. *One-time pads sent initially over private channels are as secure as private channels: for all feasible Π, $\textsc{Pad}(\Pi) \circ \textsc{SendPad} \succeq \Pi$.*

Lemma 8. *A pseudorandomly generated pad (with erasing) is as good as a uniformly random pad: for all feasible Π, $\textsc{Pad}(\Pi) \circ \textsc{SendPsPad} \succeq \textsc{Pad}(\Pi) \circ \textsc{SendPad}$.*

Lemma 9. *Generating a pad by encrypting and sending a seed (with erasing) is as good as sending the pad over a private channel: for all feasible Π, $\textsc{Pad}(\Pi) \circ \textsc{SendSeed} \succeq \textsc{Pad}(\Pi) \circ \textsc{SendPsPad}$.*

Remark. The overhead of the transformation is small enough to be practical. Rather than store a large pad at the outset, a provably secure and practical modification of the protocol permits every player to extend its pad each round (or every few rounds) using the last, unused K bits of the previous pad as a seed which it then erases. No additional messages need be sent. Furthermore, the use of an underlying noncryptographic protocol with a simple algorithm appears more efficient and implementable than the use of, say, n^2 two-party zero-knowledge proofs of behavior after each round.

4 Open Questions

These definitions and proofs suggest a number of stimulating lines of research:

- Zero knowledge and interactive proofs are defined by [21, 1] against *static* adversaries; *i.e.*, the cases of faulty P and faulty V are considered separately. Do there exist "dynamic" zero-knowledge proof systems, which are secure against dynamic adversaries, who choose P or V based on some preliminary, and thus committed, portion of the interaction?

[5] We remark that if the protocols of [10, 12, 8] are proven secure using our definitions, then a *constant-round* protocol for $t < n/3$ exists.

- Our protocol employs the *statistical* zero-knowledge proofs used by Rabin, Ben-Or and Beaver [28, 3] in an underlying private-channel layer to protect against Byzantine adversaries. The standard, widely-adopted approach introduced in [23], using *computational* zero-knowledge proofs on the top broadcast-level to guarantee behavior, seems to fail in a dynamic setting. Are computational two-party zero-knowledge proofs useful for interactive computation with dynamic security?
- Is it possible to prove that erasing is *necessary* in order to achieve security against dynamic adversaries in cryptographic protocols? If this is true, we would have a nice characterization of the difference between "cryptographic" and "non-cryptographic" multi-party computing: either private channels or tape-erasing is necessary in order to withstand a dynamic adversary.
- The use of public-key encryptions to replace private channels seems intuitive. Does the apparent failure of this approach without erasing keys suggest a subtle way that public-key encryptions fail when used in combination, or does it suggest that zero-knowledge based definitions of protocol security (virtually the only sort ever considered in computational models) are not suitable for a general approach to security?

References

1. L. Babai, S. Moran. "Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes." *J. Comput. System Sci.* **36** (1988), 254–276.
2. J. Bar-Ilan, D. Beaver. "Non-Cryptographic Fault-Tolerant Computing in a Constant Expected Number of Rounds of Interaction." *Proceedings of PODC*, ACM, 1989, 201–209.
3. D. Beaver. "Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority." *J. Cryptology*, 4:2, 1991, 75–122. An earlier version appeared as "Secure Multiparty Protocols Tolerating Half Faulty Processors" in *CRYPTO '89*, G. Brassard, ed., Springer–Verlag LNCS **435**, 1990.
4. D. Beaver. "Formal Definitions for Secure Distributed Protocols." *Proceedings of the DIMACS Workshop on Distributed Computing and Cryptography*, Princeton, NJ, October, 1989, J. Feigenbaum, M. Merritt (eds.).
5. D. Beaver. *Security, Fault Tolerance, and Communication Complexity in Distributed Systems.* Ph.D. Thesis, Harvard University, Cambridge, 1990.
6. D. Beaver. "Foundations of Secure Interactive Computation." *Proceedings of Crypto '91* (to appear).
7. D. Beaver, S. Goldwasser. "Multiparty Computation with Faulty Majority." *Proceedings of the* 30^{th} *FOCS*, IEEE, 1989, 468–473.
8. D. Beaver, S. Micali, P. Rogaway. "The Round Complexity of Secure Protocols." *Proceedings of the* 22^{nd} *STOC*, ACM, 1990, 503–513.
9. M. Bellare and O. Goldreich. "On Defining Proofs of Knowledge." *Proceedings of Crypto '92* (to appear).
10. M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." *Proceedings of the* 20^{th} *STOC*, ACM, 1988, 1–10.

11. G. Brassard, D. Chaum, C. Crépeau. "Minimum Disclosure Proofs of Knowledge." *J. Comput. System Sci.* **37** (1988), 156–189.
12. D. Chaum, C. Crépeau, I. Damgård. "Multiparty Unconditionally Secure Protocols." *Proceedings of the* 20^{th} *STOC*, ACM, 1988, 11–19.
13. B. Chor, M. Rabin. "Achieving Independence in a Logarithmic Number of Rounds." *Proceedings of the* 6^{th} *PODC*, ACM, 1987.
14. U. Feige, A. Fiat, and A. Shamir. "Zero knowledge proofs of identity." *J. of Cryptology*, **1**:2, 1988, 77–94.
15. P. Feldman. "One Can Always Assume Private Channels." Unpublished manuscript, 1988.
16. P. Feldman, S. Micali. "Optimal Algorithms for Byzantine Agreement." *Proceedings of the* 20^{th} *STOC*, ACM, 1988, 148–161. (The reader of this paper is referred for the relevant result to Feldman's Ph.D. Thesis, *Optimal Algorithms for Byzantine Agreement* (MIT, 1988), where it apparently does not appear; but see [15].)
17. Z. Galil, S. Haber, M. Yung. "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model." *Proceedings of Crypto 1987*, Springer–Verlag, 1988, 135–155.
18. Z. Galil, S. Haber, and M. Yung. "Minimum-Knowledge Interactive Proofs for Decision Problems." *SIAM J. Comput.* **18**:4 (1989), 711–739.
19. Z. Galil, S. Haber, and M. Yung. "Interactive public-key cryptosystems." Submitted for publication, 1991.
20. S. Goldwasser, S. Micali. "Probabilistic Encryption." *J. Comput. System Sci.* **28** (1984), 270–299.
21. S. Goldwasser, S. Micali, C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems." *SIAM J. Comput.* **18**:1 (1989), 186–208.
22. S. Goldwasser, M. Sipser. "Private Coins vs. Public Coins in Interactive Proof Systems." *Proceedings of the* 18^{th} *STOC*, ACM, 1986, 59–68.
23. O. Goldreich, S. Micali, A. Wigderson. "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design." *Proceedings of the* 27^{th} *FOCS*, IEEE, 1986, 174–187.
24. O. Goldreich, S. Micali, A. Wigderson. "How to Play Any Mental Game, or A Completeness Theorem for Protocols with Honest Majority." *Proceedings of the* 19^{th} *STOC*, ACM, 1987, 218–229.
25. S. Goldwasser, L. Levin. "Fair Computation of General Functions in Presence of Immoral Majority." *Proceedings of Crypto 1990.*
26. J. Håstad. "Pseudo-Random Generators under Uniform Assumptions." *Proceedings of the* 22^{nd} *STOC*, ACM, 1990, 395–404.
27. R. Impagliazzo, L. Levin, and M. Luby. "Pseudorandom Generation from One-Way Functions." *Proceedings of the* 21^{st} *STOC*, ACM, 1989, 12–24.
28. T. Rabin, M. Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." *Proceedings of the* 21^{st} *STOC*, ACM, 1989, 73–85.
29. A. Shamir. "How to Share a Secret." *Communications of the ACM*, **22** (1979), 612–613.
30. M. Tompa and H. Woll. "Random self-reducibility and zero knowledge interactive proofs of possession of information." *Proceedings of the* 28^{th} *FOCS*, IEEE, 1987, 472–482.
31. A. Yao, "Theory and Applications of Trapdoor Functions." *Proceedings of the* 23^{rd} *FOCS*, IEEE, 1982, 80–91.

Appendix

Conventions for conciseness. An interface $\mathcal{I}$ from α to β is **canonical** if it runs internal copies of nonfaulty α-players i on inputs $x_i = 0$ (or some other default value); when $\mathcal{A}$ requests to corrupt i in β, $\mathcal{I}$ corrupts i to obtain the real (x_i, a_i) and overwrites the internal player's state/tape to be "consistent" with (x_i, a_i) and the α-messages already sent between $\mathcal{I}$ and $\mathcal{A}$. The set of faulty players chosen by $\mathcal{A}$ through round r is written $T(r)$. The symbol i^b indicates a faulty player ($i \in T$); i^g indicates a good player ($i \notin T$). Bars over letters indicate internally-simulated values; lowercase letters refer to protocol α whereas capital letters (except E, D) refer to protocol β; apostrophes indicate messages from an adversary.

Even though the lemmas in this appendix hold for any feasible protocol Π, for concreteness the reader may wish to imagine Π as a verifiable secret-sharing protocol (*e.g.* [10]).

Lemma 7. *For all feasible* Π, $\textsc{Pad}(\Pi) \circ \textsc{SendPad} \succeq \Pi$.

Proof sketch. We remark that this lemma holds for computationally-unbounded adversary classes as well as polynomial-time adversary classes (even when erasing is not allowed). Let $\alpha = \textsc{Pad}(\Pi) \circ \textsc{SendPad}$, $\beta = \Pi$. Run a canonical $\mathcal{I}$ that creates "fake" pads $\overline{p}_{ij} \leftarrow \mathit{uniform}(\{0,1\}^{B(K)})$ for all initially nonfaulty $i \notin T(0)$. Supply faulty j^b with $\overline{p}_{ij}$ from internally simulated, nonfaulty i^g; supply nonfaulty, simulated j^g with whatever pads p'_{ij} faulty i^b sends. If $\mathcal{A}$ corrupts i, get (x_i, a_i) from β and use fake $\overline{p}_{ij}$ values to supply view. Now $\mathcal{A}$ enters $\textsc{Pad}(\Pi)$; consider round r. If $i^g \rightarrow j^g : M_{ij}(r)$ in β (a private message not seen by $\mathcal{I}$), $\mathcal{I}$ uses the internal message $\overline{m}_{ij}(r)$ (from fake player i on input $x_i = 0$) with fake pad $\overline{p}_{ij}$ to create broadcast message $c_{ij}(r) \leftarrow \overline{m}_{ij}(r) \oplus \overline{p}_{ij}(r)$ for $\mathcal{A}$ to see. If $i^g \rightarrow j^b : M_{ij}(r)$ in β, $\mathcal{I}$ sends $M_{ij}(r) \ominus \overline{p}_{ij}(r)$ in α. If $i^b \rightarrow j^g : c_{ij}(r)$ in α, $\mathcal{I}$ sends $m_{ij}(r) \leftarrow c_{ij}(r) \oplus p'_{ij}(r)$ in β. If $\mathcal{A}$ newly corrupts i, $\mathcal{I}$ gets (x_i, a_i) and private messages $\{m_{ij}(\rho) \mid \forall \rho \leq r, \forall j\}$ from β; $\mathcal{I}$ constructs consistent pads p_{ij} by, first, overwriting used portions $[\forall \rho \leq r, \forall j]\ p_{ij}(\rho) \leftarrow c_{ij}(\rho) \oplus m_{ij}(\rho)$, which effects no change on pads already given to $\mathcal{A}$ ($p_{ij} = \overline{p}_{ij}$ for $j \in T$), and, second, keeping unused portions $[\forall \rho > r, \forall j]\ p_{ij}(\rho) \leftarrow \overline{p}_{ij}(\rho)$. Because the pads that $\mathcal{A}$ captures are uniformly random whether obtained from α or from $\mathcal{I}$, and because $\mathcal{I}$ constructs consistent sets of messages, it is not difficult to show the collection of all outputs is *identical*: $[\mathcal{A}, \alpha] = [\mathcal{I}(\mathcal{A}), \beta]$. □

Lemma 8. *For all feasible* Π, $\textsc{Pad}(\Pi) \circ \textsc{SendPsPad} \succeq \textsc{Pad}(\Pi) \circ \textsc{SendPad}$.

Proof sketch. Let $\alpha = \textsc{Pad}(\Pi) \circ \textsc{SendPsPad}$, $\beta = \textsc{Pad}(\Pi) \circ \textsc{SendPad}$. Run a canonical $\mathcal{I}$ that creates "fake" seeds $\overline{s}_{ij} \leftarrow \mathit{uniform}(\{0,1\}^K)$ and pads $\overline{p}_{ij} \leftarrow \mathbf{prg}(\overline{s}_{ij})$ for all initially nonfaulty $i \notin T(0)$. In Step 1, do $i^g \rightarrow j^b : \overline{s}_{ij}$ in $\textsc{SendPsPad}$; record uniform pads $i^g \rightarrow j^b : P_{ij}$ sent in $\textsc{SendPad}$. Record $i^b \rightarrow$

$j^g : s'_{ij}$ in SENDPSPAD, compute $P'_{ij} \leftarrow p'_{ij} \leftarrow \mathbf{prg}(s'_{ij})$, and send $i^b \rightarrow j^g : P'_{ij}$ in SENDPAD. If $\mathcal{A}$ corrupts i, then get (x_i, a_i) from β and supply $\overline{s}_{ij}, \overline{p}_{ij}$ in the view. In Step 2, all seeds are erased; from here on, only $\overline{p}_{ij}$ need be supplied in new corruptions. In fact, $\mathcal{I}$ may later overwrite portions of $\overline{p}_{ij}$ to create consistent views; if $\overline{p}_{ij}$ were uniformly random this would make no difference; because $\overline{p}_{ij}$ is pseudorandom the resulting distribution is false, but indistinguishable (without the erased seed).

Now $\mathcal{A}$ enters PAD(Π); consider round r. If $i^g \rightarrow j^g : C_{ij}(r)$ in β, $\mathcal{I}$ sends $c_{ij}(r) \leftarrow \mathit{uniform}(\{0,1\}^L)$ in α; this is false but indistinguishable. If $i^g \rightarrow j^b : C_{ij}(r)$ in β, $\mathcal{I}$ knows the pads P_{ij} in β and $\overline{p}_{ij}$ in α and calculates an α-message $c_{ij}(r) \leftarrow C_{ij}(r) \oplus P_{ij}(r) \oplus \overline{p}_{ij}(r)$. If $i^b \rightarrow j^g : c_{ij}(r)$ in α, $\mathcal{I}$ sends $C_{ij}(r) \leftarrow c_{ij}(r)$ in β; player j^g in β uses the same pad $P'_{ij}(r)$ to decrypt this message as the one $(p'_{ij}(r))$ that the simulated player j^g would use in α. If $\mathcal{A}$ corrupts player i, $\mathcal{I}$ recreates pads p_{ij} as in Lemma 7. For $j \in T$ this changes nothing, but for $j \notin T$, this replaces a portion of a pseudorandom pad by a uniform pad (because c_{ij} was artificially and uniformly created).

To prove that $[\mathcal{A}, \alpha] \approx [\mathcal{I}(\mathcal{A}), \beta]$, we assume there is a machine that K^{-c}-distinguishes $[\mathcal{A}, \alpha]$ from $[\mathcal{I}(\mathcal{A}), \beta]$, in particular for some $z = \mathbf{x}\cdot\mathbf{a}\cdot a_A$. Say the protocols run for $R = R(n, m, k)$ rounds. Define $(Rn^2 + 1)$ hybrid distributions ξ_{IJr}, with $(I, J, r) \in \{1..n\} \times \{1..n\} \times \{1..R\} \cup \{(0,0,0)\}$, in which certain pseudorandom pads are gradually replaced by uniform pads. We say $(i, j, \rho) \leq (I, J, r)$ if $i < I$, or if $i = I$ and $j < J$, or if $i = I$ and $j = J$ and $\rho \leq r$. In ensemble ξ_{IJr} the replacement is performed for those pads $p_{ij}(\rho)$ for which $(i, j \notin T(\rho))$ and $(i, j, \rho) \leq (I, J, r)$. Choose d so that $K^{-d} < 1/K^c Rn^2$. Since $\xi_{000} = [\mathcal{A}, \alpha](\mathbf{x}\cdot\mathbf{a}\cdot a_A)$ and $\xi_{nnR} = [\mathcal{I}(\mathcal{A}), \beta](\mathbf{x}\cdot\mathbf{a}\cdot a_A)$, there is a machine that K^{-d}-distinguishes $\xi_{ij\rho}$ from $\xi_{i,j,\rho+1}$ (or from $\xi_{i,j+1,1}$, if $\rho = R$; or from $\xi_{i+1,1,1}$, if $\rho = R$ and $j = n$) for some i, j, and ρ, and hence one that K^{-d}-distinguishes $\mathbf{prg}(\mathit{uniform}(\{0,1\}^K))$ and $\mathit{uniform}(\{0,1\}^{B(K)})$, implying $\mathbf{prg}$ is not a cryptographically strong pseudorandom generator. □

Lemma 9. *For all feasible* Π, PAD(Π) ∘ SENDSEED $\succeq$ PAD(Π) ∘ SENDPSPAD.

Proof sketch. Let α = PAD(Π) ∘ SENDSEED, β = PAD(Π) ∘ SENDPSPAD. Run a canonical $\mathcal{I}$ that, through the end of SENDSEED, creates "fake" keys (E_i, D_i), seeds $\overline{s}_{ij}$, pads $\overline{p}_{ij}$, and that when given a request to corrupt i, uses the "fake" state with only x_i and a_i overwritten with the values obtained from corrupting i in SENDPSPAD. $\mathcal{I}$ simulates the proofs of knowledge of SENDSEED in a straightforward manner. $\mathcal{I}$ records all seeds S_{ij} and pads P_{ij} obtained from SENDPSPAD in corruptions of players i or j. When $i^b \rightarrow j^g : \sigma'_{ij}$ in step 2 of SENDSEED, then $\mathcal{I}$ sends $S'_{ij} \leftarrow s'_{ij} \leftarrow D_j(\sigma'_{ij})$ in step 1 of SENDPSPAD and records $P'_{ij} \leftarrow p'_{ij} \leftarrow \mathbf{prg}(s'_{ij})$. As in Lemma 8, $\mathcal{I}$ does not supply $\mathcal{A}$ with these recorded values. Rather, as before, $\mathcal{I}$ uses P_{ij} to translate messages in β to messages in α as follows. When $i^g \rightarrow j^b : C_{ij}(r)$ in β, $\mathcal{I}$ converts this to $c_{ij}(r) \leftarrow C_{ij}(r) \oplus P_{ij}(r) \oplus \overline{p}_{ij}(r)$. When $i^b \rightarrow j^g : c_{ij}(r)$ in α, $\mathcal{I}$ sends $C_{ij}(r) \leftarrow c_{ij}(r)$ in β; player j^g in β uses the same pad $P'_{ij}(r)$ to decrypt this message as the

one $(p'_{ij}(r))$ that the simulated player j^g would use in α. When messages are passed between i^g and j^g, $\mathcal{I}$ uses random strings. When $\mathcal{A}$ newly corrupts i, $\mathcal{I}$ constructs pads $p_{ij}(\rho)$ for j^g and $[\forall \rho \leq r]$ by obtaining $m_{ij}(\rho)$ from β and calculating $p_{ij}(\rho) \leftarrow c_{ij}(\rho) \oplus m_{ij}(\rho)$. As in Lemma 8, all distributions are exact except for the calculation of p_{ij} pads (with $j \notin T$) upon corruption, in which case the resulting distribution is uniform (because c_{ij} has uniform distribution) rather than pseudorandom.

Consider $(Rn^2 + 1)$ distributions ξ_{IJr}, as in the proof of Lemma 8, in which pseudorandom pads are progressively replaced by random pads. As before, a K^{-c}-distinguisher between $[\mathcal{A}, \alpha]$ and $[\mathcal{I}(\mathcal{A}), \beta]$ gives a K^{-d}-distinguisher between, not $\mathbf{prg}(\mathit{uniform}(\{0,1\}^K))$ and $\mathit{uniform}(\{0,1\}^{B(K)})$ as in Lemma 8, but distributions $\mathcal{P}_1$ and $\mathcal{P}_4$, where $\mathcal{P}_1 = (E_j(\overline{s}_{ij}, r_{ij}), \mathbf{prg}(\overline{s}_{ij}))$ with uniformly random $E_j, \overline{s}_{ij}$, and r_{ij}, and where $\mathcal{P}_4 = (E_j(\overline{s}_{ij}, r_{ij}), p_{ij})$ with uniformly random $E_j, \overline{s}_{ij}, r_{ij}$, and p_{ij}. This does not directly contradict the strength of **prg** because an encryption of the seed is present. The seed itself, the bits used to encrypt it, and the decryption key, however, are not present; they were erased. Define $\mathcal{P}_2 = (E_j(0, r_{ij}), \mathbf{prg}(\overline{s}_{ij}))$ and $\mathcal{P}_3 = (E_j(0, r_{ij}), p_{ij})$, with E_j, r_{ij}, $\overline{s}_{ij}$, and p_{ij} uniformly random. Let $\delta = \frac{1}{3}K^{-d}$. A δ-distinguisher between $\mathcal{P}_1$ and $\mathcal{P}_2$ provides a means to break the cryptosystem **pkc**: a δ-distinguisher between $\mathcal{P}_2$ and $\mathcal{P}_3$ provides a means to break the pseudorandom generator **prg**; and a δ-distinguisher between $\mathcal{P}_3$ and $\mathcal{P}_4$ provides a means to break the cryptosystem **pkc**. (The reduction from a $\mathcal{P}_1$-$\mathcal{P}_2$ distinguisher to an algorithm that successfully attacks **pkc** makes crucial use of the knowledge extractor corresponding to the proof of knowledge of step 2 of SENDSEED; similarly for the case of a distinguisher between the distributions $\mathcal{P}_3$ and $\mathcal{P}_4$.) Therefore there is no K^{-d}-distinguisher between $\mathcal{P}_1$ and $\mathcal{P}_4$, and hence no K^{-c}-distinguisher between $[\mathcal{A}, \alpha]$ and $[\mathcal{I}(\mathcal{A}), \beta]$. □