Adaptive Security in the Threshold Setting: From Cryptosystems to Signature Schemes

Anna Lysyanskaya and Chris Peikert

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{anna,cpeikert}@theory.lcs.mit.edu

Abstract. Threshold cryptosystems and signature schemes give ways to distribute trust throughout a group and increase the availability of cryptographic systems. A standard approach in designing these protocols is to base them upon existing single-server systems having the desired properties.

Two recent (single-server) signature schemes, one due to Gennaro et al., the other to Cramer and Shoup, have been developed which are provably secure using only standard number-theoretic hardness assumptions. Catalano et al. proposed a statically secure threshold implementation of these schemes. We improve their protocol to make it secure against an adaptive adversary, thus providing a threshold signature scheme with stronger security properties than any previously known.

As a tool, we also develop an adaptively secure, erasure-free threshold version of the Paillier cryptosystem.

1 Introduction

The goal of threshold cryptography [14, 15] is to enable a cluster of cooperating servers to securely and efficiently implement such cryptographic tasks as signing and decrypting. A threshold cryptographic system should remain functional and secure even when a fraction (say, almost one half) of the servers become malicious. This problem is well-motivated in practice.

One of the strongest adversarial models in this setting is the so-called *adaptive erasure-free* model. In this setting (1) the adversary corrupts servers over time depending on its entire view of the computation; and (2) upon becoming corrupted, the players have to hand over to the adversary their entire computation history; i.e., nothing can be erased.

Although results in general multi-party computation guarantee feasibility [6, 5, 10], they cannot be directly applied without incurring a considerable computation penalty. In contrast, threshold protocols are tailor-made for a specific task at hand and are therefore much more practical.

Securing threshold cryptographic systems against adaptive attacks has been the subject of extensive recent research [7, 17, 21]. Erasure-free solutions have also been considered [21]. However, none of these papers considered the question of constructing adaptively secure threshold versions of signature schemes provably secure against adaptive chosen message attacks [18, 12].

On the other hand, statically secure threshold versions of the Gennaro et al. [18] and Cramer-Shoup [12] signature schemes has been proposed by Catalano et al. [8]. Unlike the adaptive adversary, the static adversary's corruption strategy is independent of the computation history and can be assumed to be fixed in advance. It is known that statically secure protocols are not necessarily adaptively secure [6, 5, 10]. While Catalano et al. suggest that it is possible to turn their statically-secure solution into an adaptively secure one, they do not give an explicit construction.

In this paper, we extend the protocol of Catalano et al. and obtain the first construction of erasure-free adaptively secure threshold version of the Cramer-Shoup signature scheme. Our results apply as well to the Gennaro et al. [18] signature scheme. Practical threshold signature schemes with this level of security have not been exhibited before.

The general structure of our results. The protocol of Catalano et al. is constructed as follows: first, they give a secure protocol for the honest-but-curious adversary, i.e. the adversary whose corruption strategy may be adaptive, but who cannot force the players he controls to deviate from their respective protocols. Then, they show how to secure it against an adversary that forces the players to deviate from their protocols arbitrarily. In this second step, they are only able to exhibit static security.

Our starting point is the honest-but-curious protocol of Catalano et al. In order to make it adaptively secure in the erasure-free model, we utilize the techniques due to Cramer et al. [11].

Cramer et al. show how, given a threshold cryptosystem with certain desirable properties, to securely (in the static model) compute any arithmetic circuit. The general structure of their construction is as follows: the inputs to the circuit are given in encrypted form. The circuit is evaluated gate-by-gate. To evaluate each gate, the players run a corresponding protocol. For example, for the multiplication gate, the players run a special protocol that, on input two ciphertexts $E(a)$ and $E(b)$, produces a ciphertext $E(ab)$. Once all the gates are evaluated, the players jointly decrypt the ciphertext that corresponds to the output of the last gate. Cramer et al. [11] provide a statically secure multiplication protocol and a statically secure threshold cryptosystem with the required properties.

We extend the results of Cramer et al. in two ways: (1) we observe that their multiplication protocol is adaptively secure under a weaker definition that allows probability of failure, provided that the underlying threshold cryptosystem is adaptively secure; and (2) we exhibit an adaptively secure threshold cryptosystem with the required properties, namely, we show how to secure the Fouque et al. [16] version of the Paillier cryptosystem [24] against the adaptive adversary.

We then plug the resulting adaptively secure multiplication protocol into a slight modification of the honest-but-curious protocol of Catalano et al.

This approach has more general applications. It is intuitively simpler to construct protocols secure against the honest-but-curious adversary than ones secure against the active adversary. In fact the first results in secure multi-party computation were of this flavor [19]. Guided by our example, one can convert a protocol for the honest-but-curious case into one that is secure against an active and adaptive adversary in the erasure-free model, at only a small cost in efficiency.

2 The adaptive adversary model

In this paper, we use a standard model [7] to describe the execution of protocols and the capabilities of the adversary. We assume the existence of l parties communicating over a synchronous broadcast channel with rushing, where up to a threshold $t < l/2$ of them may be corrupted. The value k will represent the security parameter.

A t-limited *adaptive adversary* may choose to corrupt any party at any point over the lifetime of the system, as long as it does not corrupt more than t parties in total. The choices may be based on everything the adversary has seen up to that point (all broadcast messages and the computation histories of all other corrupted parties). When an adaptive adversary corrupts a party, it is given the entire computation history of that party and takes control of its actions for the life of the system. Note that this prohibits the honest parties from making *erasures* of their internal states at any time.

Summary of definitions and techniques. As expected, security of a protocol is defined in the adaptive model using the simulation paradigm. For any adaptive adversary $\mathcal{A}$, there must exist a simulator $\mathcal{S}$ which interacts with $\mathcal{A}$ to provide a view which is computationally indistinguishable from the adversary's view of the real protocol. The main difficulty in designing secure protocols in the adaptive model is in being able to fake the messages of the honest parties such that there are consistent internal states that can be supplied to the adversary when it chooses to corrupt new parties. In fact, we will design the protocols such that the simulator can supply consistent states on behalf of all honest parties except one, which we call the "single inconsistent party," [7] and denote P_S. We stress that the inconsistent party is chosen at random by the global simulator, and remains the same throughout all simulator subroutines.

We will design simulators that supply a suitable view to the adversary *provided P_S is not corrupted*, or said another way, the adversary's view will be indistinguishable from a real invocation *up to the point at which P_S is corrupted* (if ever). Of course, if P_S does become corrupted, then we may assume that the adversary detects the simulation perfectly; we call the probability of this event the *error* of the protocol. In order to make a reduction from a single-server signature scheme to a threshold version, we will require that the error be non-negligibly smaller than one. In particular, because $2t < l$, the error of our protocols will be less than $1/2$.

In all of our protocols, any deviation that is detectable by all honest parties will cause the misbehaving party to be excluded from all protocols for the life of the system. Upon detecting a dishonest party, the others restart *only the current* *protocol* from the beginning. Intuitively, this strategy prevents an adversary from gaining some advantage by failing to open its commitments after witnessing the honest parties' behavior. This rule will apply in each round of every protocol, even when not stated explicitly. In our simulators, we will be explicit about when misbehaving parties cause the protocol to be restarted, and when they cause the adversary to be rewound.

A note about the round-efficiency of this rule: the number of rounds of a single protocol execution is bounded only by a constant multiple of the threshold t (since one corrupt party may force a restart every time). However, the adversary can force a total of only $O(t)$ extra rounds to be executed over all invocations, which is a negligible amortized cost over the life of the system. (This assumes that all protocols are constant-round when no malicious parties are present, which will be the case.)

3 Tools

In this section we address a special kind of zero-knowledge proof called a *Σ-protocol* [11]. First we summarize Σ-protocols in the two-party setting, then we demonstrate how to implement them in a multiparty setting using *trapdoor commitments*. A reader familiar with the work of Cramer et al. [11] can skip to section 4.

Two-party Σ-protocols. The two-party Σ-protocols we use here are, in summary, honest-verifier perfect zero-knowledge proofs of knowledge with perfect completeness in which the knowledge extractor needs only two different conversation in order to extract a witness. We refer the reader to Cramer et al. [11] for formal definitions.

Trapdoor commitments. A *trapdoor commitment scheme* is much like a regular commitment scheme: a party P commits to a value by running some probabilistic algorithm on the value. The commitment gives no information about the committed value. At some later stage, P *opens* the commitment by revealing the committed value and the random coins used by the commitment algorithm. P must not be able to find a different value (and corresponding random string) that would yield the same commitment.

Trapdoor commitment schemes have one additional property: there exists a *trapdoor value* which allows P to construct commitments that he can open arbitrarily, such that this cheating is not detectable. Cramer et al. [11] provide a formal definition.

Multiparty Σ-protocols. The goal of a multiparty Σ-protocol is for many parties to make claims of knowledge such that all parties will be convinced. If all players are honest-but-curious, a naive way of achieving this goal is to make each prover participate in a separate (two-party) Σ-protocol with each of the other players. However, this approach incurs significant communication overhead, and it is not secure against an active adversary, since Σ-protocols are only honest-verifier zero-knowledge.

Part of an efficient multiparty Σ-protocol involves choosing a shared k-bit challenge string, though no particular distribution is required[1]. We simply need two different invocations to generate different challenge strings, except with negligible probability. The challenges are generated as follows: in a preprocessing phase, generate a key for a collision-resistant hash function from $\{0,1\}^l$ to $\{0,1\}^k$. To generate a challenge, each party contributes one random bit, then the hash function is applied to the concatentation of these bits. If two identical challenges are created, then either the inputs to the hash function were identical (which happens with probability at most $1/2^{l/2}$ since at least half the parties are honest), or a collision in the hash function is found.

The complete description of a multiparty Σ-protocol is as follows: in a preprocessing phase, a public key k_i for a trapdoor commitment scheme is generated for each P_i, and is distributed to all the parties by a key-distribution protocol which hides the trapdoor values. In a single proof phase, some subset P' of parties contains the parties who are to prove knowledge.

1. Each $P_i \in P'$ computes a_i, the first message of the two-party Σ-protocol. It then broadcasts a commitment $c_i = C(a_i, r_i, k_i)$, where r_i is chosen randomly by P_i.
2. A challenge r is generated by the parties, as described above. This single challenge will be used by all the provers.
3. Each $P_i \in P'$ computes the answer z_i to the challenge r, and broadcasts a_i, r_i, z_i.
4. Every party can check every proof by verifying that $c_i = C(a_i, r_i, k_i)$ and that (a_i, r, z_i) is an accepting conversation in the two-party Σ-protocol.

Cramer et al. [11] prove the security of this protocol against a static adversary. We have shows that it is also secure in the adaptive setting, using the single inconsistent party technique. We refer the reader to the full version [22] of this paper for the proof.

4 Threshold signatures using a threshold cryptosystem

Suppose an adaptive-chosen-message-secure signature scheme (such as Cramer-Shoup [12]) and a semantically secure cryptosystem (such as Paillier [24]) are given. Our signature scheme will be constructed as follows: besides the key pair (PK, SK) for a secure signature algorithm, the key generation algorithm also generates the key pair (E, D) for some semantically secure cryptosystem. The public key for the resulting signature scheme will be $(PK, E, E(SK))$, while the secret key is simply the secret key of the underlying signature scheme, i.e., SK. The signature and verification algorithms are the same as in the signature scheme given. It is easy to see, by a hybrid argument, that this resulting signature scheme is secure against the adaptive chosen message attack.

In the following sections, we will describe secure protocols for key generation and signing, and give proofs of security for these protocols.

[1] Thanks to an anonymous referee for suggesting this improvement, due to Nielsen [23].

4.1 Key generation

Recall the Cramer-Shoup signature scheme. The public key of the signer is a tuple (N, h, x, e', H), where: $N = pq$ is an RSA modulus such that $p = 2p' + 1$, $q = 2q' + 1$, and p',q' are both primes (p and q are called *safe* primes); the values $h, x \in \mathbb{Z}_n^*$ are both quadratic residues modulo N; e' is a random prime number; H is a collision-resistant hash function. The signature on a message m is a tuple (e, y', y) such that: $e \neq e'$ is a random prime number; y' is a random quadratic residue modulo N; $y^e = xh^{H(x')}$ where $x' = \frac{{y'}^{e'}}{h^{H(m)}} \bmod N$.

The key generation algorithm for the Cramer-Shoup cryptosystem generates a public key $PK = (N, h, x, e', H)$ and a secret key $SK = \phi(N)$. Our public key will also include a Paillier public key (g, n) (where n is a product of two safe primes, and g has order n modulo n^2), and a ciphertext $E(\phi(N)) = g^{\phi(N)} r^n \bmod n^2$ for a random $1 \leq r \leq n^2$.

Our key generation protocol will not be efficient, but since it is only carried out once, this efficiency penalty can be ignored. It will proceed in two steps. In Step 1, the parties will run a general multi-party computation to generate the public key $(PK, E, E(SK))$, as follows: each party P_i will contribute a random string r_i. The resulting key will be computed using the circuit for single-server key generation with randomness obtained by the exclusive-or of the r_is: $R = \oplus_{i=1}^{l} r_i$. The inputs to Step 2 are the values $\{r_i\}$ (i.e., the coins from Step 1) and fresh random bits $\{r'_i\}$. Then, using general MPC, the parties will compute the auxiliary information, emulating the one-server algorithm provided in section 5.2. For the underlying general MPC we can use the protocol due to Cramer et al. [10], which is adaptively secure and tolerates any number of corruptions below one half of the servers. In order to implement secure channels required by Cramer et al. [10], we use the non-committing encryption technique due to Canetti et al. [6] and Damgård and Nielsen [13].

The protocol described above will be secure: suppose we are given a target public key $(PK, E, S(SK))$. Our goal is to construct a simulator S which, on input the identity of an inconsistent party P_S, simulates the adversary's view of the computation provided the adversary does not corrupt P_S. We can use the simulator S_{MPC} as a subroutine. For Step 1, S will give S_{MPC} the value $(PK, E, S(SK))$ as the target output. We will also supply it with some random coins for parties that are corrupted at the beginning. As more parties get corrupted over time, S_{MPC} will request that we provide it with their inputs. S will just provide some more random coins each time that happens. If the adversary ever tries to corrupt the inconsistent party P_S, S aborts. Since the actual randomness of the algorithm is the exclusive-or of the coins of all parties, the resulting view will be correct. For Step 2, we will first run the one-server algorithm for generating the simulated auxiliary information and the simulated secret information for all but one party P_S. This one-server algorithm is described in section 5.2. We will then run the simulator S_{MPC} in the same way as for Step 1.

4.2 Computing a signature

Signature generation is done in three steps: (1) generation of (y', e); (2) generation of a verifiable additive sharing of $e^{-1} \bmod \phi(N)$; and (3) computation of y such that $y^e = xh^{H(x')}$ where $x' = \frac{{y'}^{e'}}{h^{H(m)}} \bmod N$, i.e. computation of $(xh^{H(x')})^{1/e} \bmod N$.

Adaptively secure erasure-free threshold protocols for selecting a random number already exist (see, for example, the one due to Jarecki and Lysyanskaya [21]). These can be employed for performing Step 1.

Suppose a secure protocol for computing an additive sharing of $e^{-1} \bmod \phi(N)$ has been performed. Let d_i denote the share held by player P_i. Suppose it is backed up with a public ciphertext $E(d_i)$. Each player P_i computes $x' = \frac{{y'}^{e'}}{h^{H(m)}} \bmod N$ and reveals $(xh^{H(x')})^{d_i}$, and proves that this was done correctly by invoking the Σ-protocol for proving equality of discrete logarithms. (If a player fails, his d_i is decrypted so that the other players can compute whatever is needed without him.) This takes care of Step 3.

Therefore, the only challenging piece is the computation of a verifiable additive sharing of e^{-1}, i.e., Step 2. Following the example of Catalano et al., we cast this as the *modular inversion problem*. The problem is as follows: suppose $E(\phi)$ is public. On input e, the task is to compute an additive sharing of the value $d \equiv e^{-1} \bmod \phi$, with public backup encryptions of each share. For the problem at hand, the value ϕ is, of course, $\phi(N)$.

Simulating signature generation given a signature. Suppose the simulators for each specific steps are given (simulators for steps 1 and 3 are known; the one for step 2 is given below). Here is how we simulate the signature generation. Our input is a signature on message m: (e, y', y). First, we run the simulator for Step 1 and simulate the distributed generation of (e, y'). Then we run the simulator for Step 2 and arrive at a verifiable additive sharing of e^{-1}. Finally, we run the simulator for Step 3 to simulate raising the value $xh^{H(x')}$ to the power e^{-1} to obtain y.

Background for the modular inversion protocol. Catalano et al. [8] present two versions of a modular inversion protocol which are secure against a static adversary. The first is private but not robust, while the second adds robustness at the cost of more complexity. Here we give an adaptively secure version, based on their simpler protocol. Our protocol requires $O(lk)$ bits to be broadcast, which is the same cost as the protocol of Catalano et al.

We assume the existence of a homomorphic threshold cryptosystem, defined in Appendix B. We denote an encryption of a message x as $\overline{x}$ when the public key is clear from the context. We also assume a trapdoor commitment scheme as described in section 3.

Using an adaptively secure multiparty Σ-protocol, the MULT protocol from Cramer et al. [11] is secure against an adaptive adversary as well, because its simulator only uses a single inconsistent party.

A preliminary subprotocol. First we assume existence of a secure protocol MAD (meaning "multiply, add, decrypt") which has the following specification:

(1) public inputs $w, \overline{x}, \overline{y}, \overline{z}$ to all parties, (2) public output $F = wx + yz$ for all parties.

Given a suitable homomorphic threshold cryptosystem, MAD can be implemented using the secure MULT and DECRYPT protocols. We give the protocol and a proof of security in Appendix A.

Two preliminary Σ-protocols. In the inversion protocol, each party provides a ciphertext and must prove that it is an encryption of zero. In addition, each party must publish a ciphertext and prove that the corresponding plaintext lies within a specified range. We describe both of these proofs for the Paillier cryptosystem in section 5.1.

The inversion protocol. The INVERT protocol has the following specification:

- common public input $(pk, e, N, \overline{\phi}, \{k_i\})$. Here pk is the public key of the homomorphic cryptosystem, e is a prime to be inverted modulo the secret ϕ, N is an upper bound on the value of ϕ, and $\{k_i\}$ is the set of all public trapdoor commitment keys.
- secret input sk_i, the ith secret key share, to party P_i.
- common public output $\overline{d_i}$ where d_i is described below.
- secret output d_i from each party. The $\{d_i\}$ constitute an additive sharing of the inverse, i.e. $\sum_{P_i \in P} d_i = e^{-1} \bmod \phi$.

The protocol proceeds as follows:

1. Each P_i publishes a random encryption $\overline{0_i}$ of zero, and proves that it is valid (see section 4.2). All parties internally compute $\overline{\phi_B} = (\boxplus \overline{0_i}) \boxplus \overline{\phi}$.
2. Each P_i chooses random λ_i from the range $[0 \ldots N^2]$, and random r_i from the range $[0 \ldots N^3]$, and encrypts them to get $\overline{\lambda_i}$ and $\overline{r_i}$, respectively.
3. Each P_i broadcasts a commitment to his ciphertexts $\overline{\lambda_i}$ and $\overline{r_i}$.
4. **Step R.** Each P_i decommits by broadcasting $\overline{\lambda_i}$ and $\overline{r_i}$, and the random strings used to generate the commitments.
5. Each party proves that its λ_i and r_i values are within the proper respective intervals: each party first publishes commitments to both values, then proves that the committed values are the same as their respective plaintexts, and finally proves that the committed values are within range.
6. Each party proves knowledge of its plaintexts λ_i and r_i using a multiparty Σ-protocol. Let $\lambda = \sum_{i \in P} \lambda_i$, $R = \sum_{i \in P} r_i$, and $F = Re + \lambda\phi$.
7. The parties run the MAD protocol on e, $\overline{R}$, $\overline{\lambda}$, and $\overline{\phi_B}$, where $\overline{R} = \boxplus_{i \in P} \overline{r_i}$, $\overline{\lambda} = \boxplus_{i \in P} \overline{\lambda_i}$ by addition of ciphertexts. This protocol securely computes the value $F = Re + \lambda\phi$ as the common output.
8. Each party determines whether $(e, F) = 1$. Because e is prime, $(e, F) \neq 1$ only if e divides λ, which happens with probability about $1/e$ because at least one λ_i is chosen at random. If $(e, F) \neq 1$, the parties repeat the protocol. Otherwise, all parties compute a, b such that

$$\begin{aligned} aF + be = 1 &\iff aRe + a\lambda\phi + be = 1 \\ &\iff (aR + b) \equiv e^{-1} \bmod \phi. \end{aligned}$$

P_i's share is $d_i = ar_i$ for $i > 1$, and $d_1 = ar_1 + b$ for $i = 1$. Note that any party can use the homomorphic properties of the cryptosystem to compute an encryption $\overline{d_i}$ for any $i \in P$, because the values of a and b are known to all parties, as well as encryptions $\overline{r_i}$ for all $i \in P$.

Theorem 1 (Security of inversion protocol). *For $t < l/2$,* Invert *is an adaptively t-secure protocol for computing an additive sharing of $e^{-1} \bmod \phi$.*

Proof: We will assume a secure key-generation protocol for the homomorphic cryptosystem. We describe the construction of such a protocol for a threshold Paillier cryptosystem in section 5. We will also assume a secure key-generation protocol for the trapdoor commitment scheme.

Let k_P be the public commitment keys for all the parties. Let P_S be the inconsistent party and t_S be its trapdoor value determined by the simulator for the key-generation protocol. Given $\mathcal{A}$, we will construct a simulator subroutine $\mathcal{S}_{Invert}$ which takes input $(\mathcal{A}, pk, e, N, \overline{\phi}, k_P, P_S, t_S)$.

$\mathcal{S}_{Invert}$ operates as follows:

1. For each honest P_i except P_S, honestly publish and prove validity of a random encryption of zero. For P_S, publish a blinding of $\overline{N} \boxminus \overline{\phi}$ and run $\mathcal{E}_\Sigma$ (see section 3) with the trapdoor value t_S to give a false proof of validity (do not extract witnesses from the corrupt parties, however). If any parties gave invalid proofs, restart the protocol and exclude them. At this stage, $\phi_B = N$, and all of the parties hold an encryption of N instead of an encryption of ϕ.
2. Through the decommitment phase, behave honestly. That is, choose random λ_i and r_i for each honest party, commit to their ciphertexts, and decommit honestly. If any parties fail to decommit, restart the protocol and exclude them.
3. During the round in which the parties prove plaintext knowledge, use the subroutine $\mathcal{E}_{POPK}$ to determine the values λ_i and r_i for all corrupted parties P_i who supplied valid proofs. If any parties fail to give valid proofs, restart the protocol and exclude them.
4. Set $R' = \sum_{i \in P} r_i$ and $\lambda' = \sum_{i \in P} \lambda_i$. Run $\mathcal{S}_{Mad}$ on $e, \boxplus_{i \in P} \overline{r_i}, \boxplus_{i \in P} \overline{\lambda_i}, \overline{\phi_B} = \overline{N}$, and $F' = R'e + \lambda' N$.
5. Proceed exactly according to the protocol, repeating if $(e, F') \neq 1$.

It is clear that the simulator runs in expected polynomial time. It remains to be shown that the output of the simulator is computationally indistinguishable from the output of a real run of the protocol. Let us assume for now that this is not the case, and that there is an adversary $\mathcal{A}$ which can distinguish between a real-life execution of Invert and an interaction with $\mathcal{S}_{Invert}$ with non-negligible advantage. We will provide a reduction that uses $\mathcal{A}$ to break the semantic security of the cryptosystem, thus establishing a contradiction. The reduction will employ a *hybrid simulator* interacting with $\mathcal{A}$.

Consider the simulator $\mathcal{S}_{Hybrid}$ which receives the public key pk of the homomorphic cryptosystem, the public commitment keys k_P, the identity of the inconsistent party P_S, and its commitment trapdoor value t_S. In addition, it is supplied with $N, e, \phi, \overline{\phi}$, a ciphertext $\overline{b}$ where b is either 0 or 1, and an auxiliary input representing the state of the adversary $\mathcal{A}$. The adversary's interaction with $\mathcal{S}_{Hybrid}$, and its resulting decision (whether the interaction was real or simulated), will determine with non-negligible probability whether b was 0 or 1. The hybrid simulator works as follows:

1. For each $P_i \neq P_S$, publish a random encryption of zero and proves its validity. For P_S, publish $(N - \phi) \boxdot \overline{b}$ and give a false proof of validity using $\mathcal{E}_\Sigma$ and the trapdoor t_S. Let $\overline{\phi_B}$ be as in the INVERT protocol.
2. For all uncorrupted $P_i \neq P_S$, honestly choose λ_i and r_i and commit to their ciphertexts. For P_S, choose λ_S and r_S from the proper ranges, but use the commitment trapdoor t_S to create cheating commitments. Receive all commitments from the corrupt parties. Let the set of corrupt parties at this point be called C. Let $\lambda_H = \sum_{i \notin C} \lambda_i$ and $R_H = \sum_{i \notin C} r_i$.
3. For all honest $P_i \neq P_S$, decommit the ciphertexts honestly. For P_S, open the cheating commitments as $\overline{\lambda_S}$ and $\overline{r_S}$. If any parties fail to open their commitments, restart the protocol (excluding those parties forever).
4. Honestly prove plaintext knowledge for all honest parties, and use $\mathcal{E}_{POPK}$ to extract λ_i and r_i for all $P_i \in C$ who provided valid proofs. If any parties fail to give valid proofs, restart the protocol (excluding those parties forever). Let $\lambda_C = \sum_{i \in C} \lambda_i$, and $R_C = \sum_{i \in C} r_i$. Solve for λ'_H and R'_H such that $F = (\lambda_C + \lambda_H)\phi + (R_C + R_H)e = (\lambda_C + \lambda'_H)N + (R_C + R'_H)e$. We shall prove that such λ'_H and R'_H are easy to compute, and are statistically indistinguishable from λ_H and R_H (respectively). Then execute the following loop:
   (a) Rewind the adversary to **Step R** in the protocol. For all honest parties $P_i \neq P_S$, again honestly decommit to their ciphertexts $\overline{\lambda_i}$, $\overline{r_i}$. For P_S, open the cheating commitments as blinded ciphertexts $\overline{\lambda'_S}$ and $\overline{r'_S}$, where $\overline{\lambda'_S} = \overline{\lambda_S} \boxplus ((\lambda'_H - \lambda_H) \boxdot \overline{b})$, and $\overline{r'_S} = \overline{r_S} \boxplus ((R'_H - R_H) \boxdot \overline{b})$. Receive decommitments from each corrupt party (which, if valid, must be the same as the earlier valid decommitment). If any parties refuse to open their commitments, go to the beginning of the loop.
   (b) Honestly prove plaintext knowledge on behalf of all honest parties $P_i \neq P_S$, and use $\mathcal{E}_{POPK}$ and the commitment trapdoor t_S to provide fake proofs of plaintext knowledge for $\overline{\lambda'_S}$ and $\overline{r'_S}$. Also receive proofs of plaintext knowledge from the corrupt parties. If any parties give invalid proofs, go to the beginning of the loop.
   (c) Exit the loop.
5. Run $\mathcal{S}_{Mad}$ on e, $\overline{R}$ and $\overline{\lambda}$ (as in the real protocol), $\overline{\phi_B}$, and the value F. Finish according to the protocol.

First we must analyze the running time of the hybrid simulator. It suffices to show that the loop is executed a polynomial number of times in expectation. Note that the loop is only reached if all parties open their commitments and prove plaintext knowledge correctly. Let ϵ_0 be the probability that the the adversary behaves in this way, given that P_S publishes random encryptions $\overline{\lambda_S}$, $\overline{r_S}$, and let ϵ_1 be defined similarly, given that P_S publishes random encryptions $\overline{\lambda'_S}$, $\overline{r'_S}$. By semantic security, ϵ_0 is negligibly close to ϵ_1. The contribution of the loop to the expected running time, therefore, is negligibly more than ϵ_0 times the expected number of times the loop is executed (which is $1/\epsilon_0$), so the contribution of the loop is $O(1)$.

We now prove the correctness of the reduction. Certainly if the adversary corrupts any party besides P_S, the hybrid can supply a valid computation history because it is acting honestly on behalf of that party. We now show that if $b = 0$, the output of $\mathcal{S}_{Hybrid}$ is indistinguishable from a real run of the INVERT protocol. Similarly, if $b = 1$, the output is indistinguishable from the output of $\mathcal{S}_{Invert}$. Therefore an adversary that can detect a simulation of INVERT can be used to break the semantic security of the underlying cryptosystem.

First, assume that $b = 0$. Then it is easy to verify that the hybrid acts honestly on behalf of all the uncorrupted parties, and in the first round P_S indeed publishes a random encryption of zero, so $\phi_B = \phi$. The only deviation from the real protocol occurs in the creation of cheating commitments for P_S and in the proofs of plaintext knowledge, but these commitments are computationally indistinguishable from honest commitments. Because $\lambda'_S = \lambda_S$ and $r'_S = r_S$, the behavior of P_S is indistinguishable from an honest party's behavior in the real protocol.

Now assume that $b = 1$. Then P_S publishes a random encryption of $N - \phi$ as in the simulation, and $\phi_B = N$. Note that all λ_i, r_i belonging to honest parties are chosen uniformly except for λ'_S and r'_S. But as we will show, the distributions of those variables are statistically indistinguishable from the respective uniform distributions. So in fact the behavior of P_S in the hybrid is indistinguishable from its behavior under $\mathcal{S}_{Invert}$.

It only remains to be proven that λ_S, r_S are similarly-distributed with λ'_S, r'_S (respectively), which we do here. We assume for simplicity that $N - \phi = O(\sqrt{N})$, as is the case when $\phi = \phi(N)$ and N is the product of two large primes of approximately equal size. First we state the following lemma:

Lemma 1 ([8]). *Let x, y be two integers such that $(x, y) = 1$ and A, B two integers such that $A < B$, $x, y < A$, and $B > Ax$. Then every integer z in the closed interval $[xy - x - y + 1, Ax + By - xy + x + y - 1]$ can be written as $ax + by$ where $a \in [0, A]$ and $b \in [0, B]$. Furthermore, there exists a polynomial-time algorithm that on input x, y, and z, outputs such a and b.*

Let us denote λ as $\lambda_C + \lambda_H$, λ' as $\lambda_C + \lambda'_H$, R as $R_C + R_H$, and R' as $R_C + R'_H$. We apply this lemma twice, first with $x = \phi$, $y = e$, and again with $x = N$, $y = e$ to conclude that any integer F in the interval $[\delta, \Delta]$ can be written both as $\lambda\phi + Re$, and as $\lambda' N + R' e$, where $\lambda, \lambda' \in [0, nN^2]$ and $R, R' \in [0, nN^3]$. Here $\delta = Ne - e + 1$, and $\Delta = n(N^2\phi + N^3 e) - \phi e + \phi + e - 1$.

Now, given any fixed λ_C, R_C (the sums of the adversaries' chosen values in the protocol) and any λ_H (respectively, R_H) distributed as the sum of at least $n/2$ honestly-chosen uniform values from $[0, N^2]$ (respectively, $[0, N^3]$), it is easy to see by Chernoff bounds that the probability that F falls outside the range $[\delta, \Delta]$ is negligible since both bounds fall far away from the mean of F.

Now suppose $F \in [\delta, \Delta]$ and λ_C, R_C are fixed as in the protocol. Given a pair λ_H, R_H such that $F = (\lambda_C + \lambda_H)\phi + (R_C + R_H)e$, we present an efficient mapping that produces λ'_H, R'_H such that $F = (\lambda_C + \lambda'_H)N + (R_C + R'_H)e$. That is, $\lambda\phi - \lambda' N = (R' - R)e$. Since $(N, e) = 1$, for any given λ there exists a unique and efficiently-computable $\lambda' \in [\lambda, \lambda + e - 1]$ such that $\lambda\phi - \lambda' N$ is a multiple of e. This determines the value $\lambda'_H - \lambda_H + \lambda_S = \lambda'_S$ (one of the values published by the first honest party in the hybrid simulator), and from that we can solve for $R' - R + r_S = r'_S$ (the other published value).

We need only show that λ_S, λ'_S and r_S, r'_S are close enough in a statistical sense, i.e. that their differences are small relative to the sizes of the intervals from which they are drawn. Indeed, $\frac{|\lambda' - \lambda|}{N^2} \le \frac{e}{N^2} \le \frac{1}{N}$ and

$$|r_1 - r'_1| = \frac{|\lambda\phi - \lambda' N|}{e} = \left| \frac{(\lambda - \lambda')\phi}{e} + \frac{\lambda'(\phi - N)}{e} \right| \le \left| \phi - \frac{nN^2\sqrt{N}}{e} \right| \le nN^2\sqrt{N}$$

Thus $\frac{|r_1 - r'_1|}{N^3} \le \frac{n}{\sqrt{N}}$ which again is negligible. This completes the proof.

5 An adaptively secure threshold Paillier cryptosystem

We introduce the following notation: for any $n \in \mathbb{N}$, $\lambda(n)$ denotes Carmichael's lambda function, defined as the largest order of the elements of $\mathbb{Z}_n^*$. It is known that if the prime factorization of an odd integer n is $\prod_{i=1}^k q_i^{f_i}$, then $\lambda(n) = \mathrm{lcm}_{i=1\ldots k}(q_i^{f_i - 1}(q_i - 1))$.

Our protocols will make use of two tools: Shamir secret sharing over the integers [26], and proofs of discrete log equality in groups of unknown order [9, 4].

5.1 The Paillier cryptosystem

The Paillier cryptosystem [24] is based on *composite-degree residuosity classes*, and has the desired homomorphic properties. It is based upon the Carmichael lambda function in $\mathbb{Z}_{n^2}^*$ and two useful facts regarding it: for all $w \in \mathbb{Z}_{n^2}^*$, $w^{\lambda(n)} = 1 \bmod n$, and $w^{n\lambda(n)} = 1 \bmod n^2$. Here we recall the cryptosystem.

Key generation. Let $n = pq$ where p, q are primes. Let $g = (1+n)^a b^n \bmod n^2$ for random $a, b \in \mathbb{Z}_n^*$. The public key is (n, g) and the secret key is $\lambda(n)$.

Encryption. To encrypt a message $M \in \mathbb{Z}_n$, randomly choose $x \in \mathbb{Z}_n^*$ and compute the ciphertext $c = g^M x^n \bmod n^2$.

Decryption. To decrypt c, compute $M = \frac{L(c^{\lambda(n)} \bmod n^2)}{L(g^{\lambda(n)} \bmod n^2)} \bmod n$ where the domain of L is the set $S_n = \{u < n^2 \ : \ u = 1 \bmod n\}$ and $L(u) = \frac{u-1}{n}$.

Other useful properties. The Paillier cryptosystem is *homomorphic*, in the sense of the definition in Appendix B. Cramer et al. [11] provide Σ-protocols for proof of plaintext knowledge and proof of correct multiplication. We also require a proof that a ciphertext is an encryption of zero; is merely a proof of nth residuosity modulo n^2. Such a proof and is virtually identical to a zero-knowledge proof of quadratic residuosity mod n as given by, for example, Goldwasser et al. [20] Finally, we require a proof that, given a ciphertext, the corresponding plaintext lies within a specified range. Boudot [2] describes such a proof for committed values, and a proof of equality between a committed value and a ciphertext in the Paillier cryptosystem can be constructed using standard techniques (see, for example, Camenisch and Lysyanskaya [3]).

The security of the scheme is based upon the composite residuosity class problem, which is exactly the problem of decrypting a ciphertext. Semantic security can be proven based on the hardness of detecting nth residues mod n^2.

Fouque et al. [16] present a threshold version of the Paillier cryptosystem, using techniques developed by Shoup [27] for threshold RSA signatures. The version presented there is known to be secure only in the static adversary model, assuming the semantic security of the non-threshold version.

5.2 An adaptively secure threshold version

Here we present the novel result of a threshold Paillier cryptosystem which is secure in the adaptive adversary model, based upon the security of Paillier's cryptosystem and the existence of trapdoor commitment schemes. This cryptosystem is inspired by the statically-secure threshold version presented in Fouque et al. [16]

Description of the protocols. Recall $\Delta = l!$, where l is the number of parties.
Key generation. We first describe key generation in terms of an l-party function on input k, the security parameter. This function is evaluated by a trusted party, who distributes the proper values to the parties.

Choose an integer n, the product of two strong primes p, q of length k such that $p = 2p' + 1$ and $q = 2q' + 1$, and $\gcd(n, \phi(n)) = 1$. Set $\lambda = 2p'q' = \lambda(n)$. Choose random $(a, b) \leftarrow \mathbb{Z}_n^* \times \mathbb{Z}_n^*$, and let $g = (1+n)^a b^n \bmod n^2$. The secret key is the value $\beta\lambda$ for a random $\beta \leftarrow \mathbb{Z}_n^*$, which is shared additively as follows: for all parties P_i but one, choose random $s_i \leftarrow \mathbb{Z}_{n\lambda}$, and choose the last s_i such that $\sum_{i \in P} s_i = \beta\lambda \bmod n\lambda$. The public key is the triple (g, n, θ), where $\theta = a\beta\lambda \bmod n$. To compute public verification keys, choose a random public square v from $\mathbb{Z}_{n^2}^*$, and let $v_i = v^{s_i} \bmod n^2$. In addition, compute polynomial backups for each s_i as follows: let $a_{i,0} = \Delta s_i$, and choose random $a_{i,j} \leftarrow [-\Delta^2 n^3/2, \ldots, \Delta^2 n^3/2]$, then define a polynomial *over the integers* $f_i(X) = \sum_{j=0}^{t} a_{i,j} X^j$ (so that $f_i(0) = \Delta s_i$). To each party P_j, give the values $f_i(j)$ for all i. Finally, compute public commitments for these backup shares using any perfectly-hiding commitment scheme, such as Pedersen's [25]. Let the public value $w_{i,j}$ be a commitment to $f_i(j)$ under public key k_j and random string $r_{i,j}$, and give $r_{i,j}$ to party P_j.

It is well known [19, 1, 6, 10] that for any l-party function, there is an adaptively secure protocol which evaluates it. Therefore there is a simulator which, given all the outputs of the function (excluding any values belonging only to P_S), interacts with the adversary and gives it a suitable view of the key generation protocol. In section 5.2 we describe how to provide suitably-distributed inputs to this simulator. This key generation protocol may be very inefficient, but it is only executed once to initialize the threshold cryptosystem.

Encryption. To encrypt a message $M \in \mathbb{Z}_n$, pick random $x \leftarrow \mathbb{Z}_n^*$ and compute the ciphertext $c = g^M x^n \bmod n^2$.

Computing decryption shares. Player P_i computes his decryption share $c_i = c^{s_i} \bmod n^2$, and proves via a Σ-protocol that c_i^2 (in base c^2) and v_i (in base v) have the same discrete log s_i in $\mathbb{Z}_{n^2}^*$.

Combining shares. If any party P_i refuses to publish his c_i, or gives an invalid proof, then the other parties reconstruct his secret share s_i as follows. Each party P_j publishes its backup share $f_i(j)$ and random string $r_{i,j}$, and all parties verify that $w_{i,j}$ matches those values. Because there are at least $t+1$ honest parties, each party may pick some $t+1$ honestly-published values $f_i(j)$, and by interpolation, discover $s_i = f_i(0)/\Delta$ and compute $c_i = c^{s_i} \bmod n^2$.

Now each party has a correct value $c_i = c^{s_i} \bmod n^2$, for all i. The message can be computed by each party as follows:

$$\frac{L\left(\prod_{i\in P} c_i\right)}{\theta} = \frac{L\left(c^{\sum s_i \bmod n\lambda}\right)}{\theta} = \frac{L(c^{\beta\lambda})}{\theta} = \frac{L(g^{\beta\lambda M})}{\theta} = \frac{a\beta\lambda M}{\theta} = M \bmod n$$

since the value $\theta = a\beta\lambda \bmod n$ is part of the public key.

Simulating decryption. The input to the decryption simulator is a tuple $(\{s_i\}, \{v_P\}, \{a_{i,j}\}, \{w_{i,j}\}, \{k_i\}, g, n, \theta, v, c, M, P_S, t_S, \mathcal{A})$. The sets and (g, n, θ) are simulated values corresponding to those in the real protocol; c is the ciphertext to be decrypted and M is its decryption; P_S is the identity of the single inconsistent party and t_S is its commitment trapdoor; $\mathcal{A}$ is an arbitrary input corresponding to the state of the adversary before the protocol execution. In the next section, we describe how these simulated values can be generated from only a public key from the single-server Paillier cryptosystem.

The simulator acts honestly on behalf of all uncorrupted parties P_i (excluding P_S) by publishing $c_i = c^{s_i} \bmod n^2$ and proving correctness of the decryption shares. On behalf of P_S, the simulator publishes $c_S = (1 + M\theta n) \prod_{i\neq S} c^{-s_i} \bmod n^2$ and provides a false proof of correctness using t_S. If any corrupted party fails to provide a correct decryption share, the simulator honestly interpolates that party's secret share as in the decryption protocol, and proceeds normally. The simulator then honestly computes the plaintext by multiplying the published shares, yielding $(1+M\theta n) \bmod n^2$, applying L, and dividing by θ to get common output M.

The view of the adversary under the simulation is statistically indistinguishable from a real run of the protocol, provided that all public inputs are suitably simulated. If the adversary corrupts any party P_j (other than P_S), that party's behavior over every invocation of the protocol is consistent with the secret s_j revealed to the adversary. In addition, the adversary is entitled to see $f_i(j)$ and $r_{i,j}$, for all i. When $j \neq S$, the values are consistent with anything else the adversary has seen. For $i = S$, we prove below that with high probability, any set of at most t values $f_S(j)$ is distributed similarly regardless of the value being shared, and therefore the simulated values $f_i(j)$ are statistically indistinguishable from those in a real run.

Simulating key generation. We now show that the outputs of the key generation function can be simulated (up to statistical closeness), given a public key (g', n) and the identity of the single inconsistent party P_S. (It is sufficient to simulate every value produced by the key generation function, except the secret share s_S belonging to P_S. This is because the entire simulation is aborted if the adversary ever attempts to corrupt P_S, so we need not simulate its private data.) When these values are given to the simulator for the key-generation protocol, it generates a suitable view for the adversary.

Choose random $(x, y, \theta) \leftarrow \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ and let $g = (g')^x y^n \bmod n^2$. Choose random $\alpha \leftarrow \mathbb{Z}_n^*$, and let $v = g^{2\alpha}$. Then for each player, choose random $s_i \leftarrow [0, \ldots, \lfloor n/2 \rfloor - 1]$, and create verification shares $v_i = v^{s_i} \bmod n^2$ for all parties but P_S. For P_S, set $v_S = (1 + 2\alpha\theta n)v^{-\sum_{i \neq S} s_i} \bmod n^2$. Finally, create commitments $w_{i,j}$ honestly (from polynomials with free terms Δs_i and random coefficients) for all i and j, and random $r_{i,j}$.

First, note that the statistical difference between the uniform distributions on $[0, \ldots, \lfloor n/2 \rfloor - 1]$ and $\mathbb{Z}_{n\lambda}$ is $O(n^{-1/2})$, so any set of at most $l-1$ secret keys s_i is statistically indistinguishable between a real and simulated run. Both g and θ are uniformly chosen from their respective domains, and are identically-distributed with their respective values in the real protocol. In addition, v is a random element of Q_{n^2}, the cyclic group of squares mod n^2. Because $|Q_{n^2}| = pqp'q'$, and $\phi(pqp'q') = (p-1)(q-1)(p'-1)(q'-1)$, v is a generator of Q_{n^2} with high probability, and is identically-distributed with its value in the real protocol.

Note that any set of at most $l-1$ simulated verification keys v_i is statistically close to a real set. However, in the real protocol with a fixed v, the values of $l-1$ verification shares induce a distribution upon the last (because the values of $l-1$ secret shares s_i induce a distribution upon the last). That is, it is necessary and sufficient that $\prod_{i \in P} v_i = v^{\beta\lambda} \bmod n^2$ for some uniformly-chosen β from $\mathbb{Z}_n^*$. In the simulation, we choose $\prod_{i \in P} v_i = v^{\beta\lambda} = (1 + 2\alpha\theta n) \bmod n^2$ without knowing λ but just by randomly choosing θ, which induces a uniform distribution upon β as desired.

Finally, we note that the simulated set $\{w_{i,j}\}$ is identically-distributed to its counterpart in the real protocol, by the perfect-hiding of the commitment scheme.

It remains to be shown that the simulated values $f_i(j)$ for all i and for the adversary's chosen j are indistinguishable from those in a real run. It is clear that the $f_i(j)$ are identically distributed for $i \neq S$, because the simulator behaves honestly. It is also obvious that the points of different polynomials are independent. We therefore show that with high probability, the values $f_S(j)$ seen by the adversary are consistent with a polynomial having free term $\Delta \hat{s}_S$ and coefficients from the proper range, for any value of $\hat{s}_S$.

Let $f_S(X)$ be the polynomial used in the simulation, that is, $f_S(X) = \Delta s_S + \sum_{i=1}^{t} a_{S,j} X^j$ where the $a_{S,j}$ are randomly chosen. Say that the adversary has corrupted a set of parties C, with $|C| \leq t$. We wish to find a polynomial $\hat{f}_S(X)$ such that $\hat{f}_S(0) = \Delta \hat{s}_S$ for an arbitrary $\hat{s}_S$, and $\hat{f}_S(i) = f_S(i)$ for $i \in C$. Consider a polynomial $h(X)$ such that $h(0) = \Delta(\hat{s}_S - s_S)$, and $h(i) = 0$ for $i \in C$. Then we have $\hat{f}(X) \equiv f(X) + h(X)$. By interpolation,

$$h(X) = \sum_{i \in C} h(i) \cdot \prod_{j \neq i, j \in \{0\} \cup C} \frac{z-j}{i-j} = \Delta(\hat{s}_S - s_S) \prod_{j \in C} \frac{z-j}{-j}$$

so the coefficient of X^i in $h(X)$ is: $\Delta(\hat{s}_S - s_S) \sum_{B \subseteq C, |B|=i} \frac{\prod_{j \in B}(-j)}{\prod_{j \in C}(-j)} \in \mathbb{Z}$ which is bounded in absolute value by

$$\sum_{B \subseteq C, |B|=i} \Delta(\hat{s}_S - s_S) \leq \Delta(\hat{s}_S - s_S) \binom{t}{i} \leq \frac{\Delta(\hat{s}_S - s_S)t!}{i!(t-i)!} \leq \Delta(\hat{s}_S - s_S)t! \leq \Delta^2 n^2/2$$

since $\hat{s}_S, s_S \in \{0, \ldots n^2/2\}$.

Now the coefficients of $\hat{f}(X)$ are outside of the desired range only if any of the coefficients of $f(X)$ are outside of $[-\Delta^2(n^3 - n^2)/2, \ldots, \Delta^2(n^3 - n^2)/2]$. By the union bound, this happens with probability at most t/n, which is negligible. In addition, there is a bijection between the coefficients of f and the coefficients of $\hat{f}$ when $s_S, \hat{s}_S$, and C are fixed. Therefore the distribution of the coefficients of $\hat{f}$ is statistically close to uniform, as desired.

A reduction from the original cryptosystem. With these simulations in hand, the reduction from one-server semantic security to threshold semantic security is straightforward. Assume there is an adversary that can break the security of the threshold cryptosystem. Given a public key (g', n) for the single-server Paillier cryptosystem, we first simulate the key generation protocol and any decryptions as described above. (Recall that the public key of the threshold cryptosystem is $(g = (g')^x y^n \bmod n^2, n, \theta)$ for some uniformly-chosen x, y, θ.) The adversary then outputs two messages m_0, m_1, which we send to an oracle, who responds with a random encryption c of m_b for some random bit b. We compute $\chi = c^x \bmod n^2$ (where x is the value chosen by the key generation simulator) and give it to the adversary. By assumption, the adversary can distinguish with non-negligible advantage whether χ is an encryption of m_0 or m_1 under (g, n, θ). This is equivalent to whether c is an encryption of m_0 or m_1 under (g', n), hence we have broken the semantic security of the original cryptosystem. This completes the reduction.

Acknowledgements We are indebted to Ron Rivest for valuable discussions. We would also like to thank the anonymous referees for their detailed and thoughtful comments. Anna Lysyanskaya acknowledges the support of an NSF graduate fellowship, the Lucent Technologies GRPW program, and the Merrill-Lynch grant given to R. L. Rivest. Chris Peikert is supported by an MIT Presidential Fellowship, sponsored by Akamai Technologies.

References

1. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.

2. Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.
3. Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 388–407. Springer Verlag, 2001.
4. Jan Camenisch and Markus Michels. A group signature scheme based on an RSA-variant. Technical Report RS-98-27, BRICS, Departement of Computer Science, University of Aarhus, November 1998. Preliminary version in:*Advances in Cryptology — ASIACRYPT '98*, vol. 1514 of *LNCS*.
5. Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
6. Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 639–648, 1996.
7. Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology—CRYPTO 99*. Springer-Verlag, 1999.
8. Dario Catalano, Rosario Gennaro, and Shai Halevi. Computing inverses over a shared secret modulus. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 190–206. Springer Verlag, 2000.
9. David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.
10. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology—EUROCRYPT 99*, pages 311–326. Springer-Verlag, 1999.
11. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, Lecture Notes in Computer Science. Springer Verlag, 2001.
12. Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 46–52. ACM press, nov 1999.
13. Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 432–450. Springer Verlag, 2000.
14. Yvo Desmedt. Society and group oriented cryptography. In *Advances in Cryptology—CRYPTO 87*. Springer-Verlag, 1987.
15. Yvo Desmedt and Yair Frankel. Threshold cryptography. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1990.
16. P. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography 2000*, Lecture Notes in Computer Science. Springer Verlag, 2000.
17. Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. In *Advances in Cryptology—ASIACRYPT 99*. Springer-Verlag, 1999.

18. Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer Verlag, 1999.
19. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
20. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 27th Annual Symposium on Foundations of Computer Science*, pages 291–304, 1985.
21. Stanisław Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: introducing cocurrency, removing erasures. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 190–206. Springer Verlag, 2000.
22. Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. Manuscript. Available from `http://eprint.iacr.org`.
23. Jesper Buus Nielsen. Personal communication.
24. Pascal Paillier. Public-key cryptosystems based on composite residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–239. Springer Verlag, 1999.
25. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.
26. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
27. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer Verlag, 2000.

A The Mad protocol

The MAD protocol takes common inputs $w, \overline{x}, \overline{y}, \overline{z}$ and returns common output $F = wx + yz$. It is implemented as follows:

1. Each party publishes a trapdoor-commitment to a random string r_i for use in the multiplication-by-ring-element algorithm.
2. The parties open their commitments, and compute r as the exclusive-or of all properly-decommitted strings.
3. Each party runs the multiplication-by-ring-element algorithm on inputs w and $\overline{x}$ with random string r, yielding a common random ciphertext $\overline{wx}$.
4. The parties enter the MULT protocol on $\overline{y}, \overline{z}$, yielding common random ciphertext $\overline{yz}$.
5. Each party uses the deterministic addition-of-ciphertexts algorithm to compute a common input $\overline{wx + yz}$ to the DECRYPT protocol, yielding common output $F = wx + yx$, as desired.

For the proof of security, we refer the reader to the full version [22] of the paper.

B Homomorphic threshold encryption

Here, for self-containment, we provide a modification of the definitions given by Cramer et al. [11]. The modification is that we require security in the adaptive setting.

B.1 Threshold cryptosystems

Here we define threshold encryption schemes and their security properties.

Definition 1. *An* adaptively-secure threshold cryptosystem *for parties* $P = \{P_1, \ldots, P_l\}$ *with* threshold $t < l$ *and security parameter* k *is a 5-tuple* $(K, \mathrm{KG}, M, E, \textsc{Decrypt})$ *having the following properties:*

1. *(Key space) The* key space $K = \{K_{k,l}\}_{k,l \in \mathbb{N}}$ *is a family of finite sets of the form* $(pk, sk_1, \ldots, sk_l)$*. We call* pk *the* public key *and* sk_i *the* private key share *of party* P_i*. For* $C \subseteq P$ *we denote the family* $\{sk_i\}_{i \in C}$ *by* sk_C*.*
2. *(Key generation) There exists an adaptively* t*-secure* key generation l*-party protocol* KG *which, on input* 1^k*, computes, in probabilistic polynomial time, public output* pk *and secret output* sk_i *for party* P_i*, where* $(pk, sk_1, \ldots, sk_l) \in K_k$*. We write* $(pk, sk_1, \ldots, sk_l) \leftarrow \mathrm{KG}(1^k)$ *to represent this process.*
3. *(Message sampling) There exists some probabilistic polynomial-time algorithm which, on input* pk*, outputs a uniformly random element from a* message space M_{pk}*. We write* $m \leftarrow M_{pk}$ *to describe this process.*
4. *(Encryption) There exists a probabilistic polynomial-time algorithm* E *which, on input* pk *and* $m \in M_{pk}$*, outputs an encryption* $\overline{m} = E_{pk}(m)[r]$ *of* m*. Here* r *is a uniformly random string used as the random input, and* $E_{pk}(m)[r]$ *denotes the encryption algorithm run on inputs* pk *and* m*, with random tape containing* r*.*
5. *(Decryption) There exists an adaptively* t*-secure protocol* $\textsc{Decrypt}$ *which, on common public input* (c, pk) *and secret input* sk_i *for each uncorrupted party* P_i*, where* sk_i *is the secret key share of the public key* pk *(as generated by* KG*) and* $c = E_{pk}(m)[r]$ *is an encrypted message for some* r*, returns* m *as a common public output.*
6. *(Threshold semantic security) For all probabilistic circuit families* $\{S_k\}$ *(the* message sampler*) and* $\{D_k\}$ *(called the* distinguisher*), all constants* $c > 0$*, all sufficiently large* k*, and all* $C \subseteq P$ *such that* $|C| \leq t$*,*

$$\begin{aligned} \Pr[\ & (pk, sk_1, \ldots, sk_l) \leftarrow \mathrm{KG}(1^k); (m_0, m_1, s) \leftarrow S_k(pk, sk_C); i \stackrel{R}{\leftarrow} \{0,1\}; \\ & e \leftarrow E(pk, m_i); b \leftarrow D_k(s, e) : b = i] < 1/2 + 1/k^c \end{aligned}$$

B.2 Homomorphic properties

We also need the cryptosystem to have the following homomorphic properties:

1. (Message ring) For all public keys pk, the message space M_{pk} is a ring in which we can compute efficiently using the public key only. We denote the ring $(M_{pk}, \cdot_{pk}, +_{pk}, 0_{pk}, 1_{pk})$. We require that the identity elements 0_{pk} and 1_{pk} be efficiently computable from the public key.
2. ($+_{pk}$-homomorphic) There exists a polynomial-time algorithm which, given public key pk and encryptions $\overline{m}_1 \in E_{pk}(m_1)$ and $\overline{m}_2 \in E_{pk}(m_2)$, outputs a uniquely-determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} = \overline{m}_1 \boxplus \overline{m}_2$. Likewise, there exists a polynomial-time algorithm for performing subtraction: $\overline{m} = \overline{m}_1 \boxminus \overline{m}_2$.
3. (Multiplication of a ciphertext by a ring element) There exists a probabilistic polynomial-time algorithm which, on input pk, $m_1 \in M_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$, outputs a random encryption $\overline{m} \leftarrow E_{pk}(m_1 \cdot_{pk} m_2)$. We assume that we can multiply a ring element from both the left and right. We write $\overline{m} \leftarrow m_1 \boxdot \overline{m}_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$ and $\overline{m} \leftarrow \overline{m}_1 \boxdot m_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$. Let $(m_1 \boxdot \overline{m}_2)[r]$ denote the unique encryption produced by using r as the random coins in the multiplication-by-ring-element algorithm.
4. (Addition of a ciphertext and a ring element) There exists a probabilistic polynomial-time algorithm which, on input pk, $m_1 \in M_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$, outputs a uniquely-determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} = m_1 \boxplus \overline{m}_2$.
5. (Blindable) There exists a probabilistic polynomial-time algorithm B which, on input pk and $\overline{m} \in E_{pk}(m)$, outputs an encryption $\overline{m}' \in E_{pk}(m)$ such that $\overline{m}' = E_{pk}(m)[r]$, where r is chosen uniformly at random.
6. (Check of ciphertextness) By C_{pk} we denote the set of possible encryptions of any message, under the public key pk. Given $y \in \{0,1\}^*$ and pk, it is easy to check whether $y \in C_{pk}$.
7. (Proof of plaintext knowledge) Let $L_1 = \{(pk, y) : pk \text{ is a public key} \wedge y \in C_{pk}\}$. There exists a Σ-protocol for proving the relation R_{POPK} over $L_1 \times (\{0,1\}^*)^2$ given by $R_{POPK} = \{((pk, y), (x, r)) : x \in M_{pk} \wedge y = E_{pk}(x)[r]\}$. Let $\mathcal{E}_{POPK}$ be the simulator for this Σ-protocol, which is just a special case of $\mathcal{E}_\Sigma$ described in section 3.
8. (Proof of correct multiplication) Let $L_2 = \{(pk, x, y, z) : pk \text{ is a public key} \wedge x, y, z \in C_{pk}\}$. There exists a Σ-protocol for proving the relation R_{POCM} over $L_2 \times (\{0,1\}^*)^3$ given by $R_{POCM} = \{((pk, x, y, z), (d, r_1, r_2)) : y = E_{pk}(d)[r_1] \wedge z = (d \boxdot x)[r_2]\}$.

We call any such scheme meeting these additional requirements a *homomorphic threshold cryptosystem*.

From these properties, it is clear how to perform addition of two ciphertexts: use the $+_{pk}$ algorithm, following by an optional blinding step. The remaining operation to be supported is secure multiplication of ciphertexts. That is, given $\overline{a}$ and $\overline{b}$, determine a ciphertext $\overline{c}$ such that $c = a \cdot_{pk} b$, without leaking any information about a, b, or c. Cramer et al. [11] give the MULT protocol for multiplication, and prove its security against a static adversary.