How to Share a Function Securely

ALFREDO DE SANTIS[*] YVO DESMEDT[†] YAIR FRANKEL[‡] MOTI YUNG[§]

(extended summary)

Abstract

We define the primitive of *function sharing*, a functional analog of secret sharing, and employ it to construct novel cryptosystems.

The basic idea of function sharing is to split a hard to compute (trapdoor) function into *shadow functions* (or *share-functions*). The intractable function becomes easy to compute at a given point value when given any threshold (at least t out of l) of shadow functions evaluations at that point. Otherwise, the function remains hard. Furthermore, the function must remain intractable even after exposing up to $t-1$ shadow functions and exposing values of all shadow functions at polynomially many inputs.

The primitive enables the distribution of the power to perform cryptography (signature, decryption, etc.) to agents. This enables the design of various novel cryptosystems with improved integrity, availability and security properties.

Our model should be contrasted with the model of secure function evaluation protocols. We require no channels between agents holding the shadow functions, as the agents act non-interactively on a publicly available input. Our security solely relies on secure memories (and results) as in regular cryptosystems. In secure function evaluation, on the other hand, it is necessary to have private/ secured bilateral channels, interactive protocol, and security of all inputs – in addition to secure memories.

[*]Dip. di Informatica ed Applicazioni Università di Salerno, Baronissi (SA), Italy.

[†]Dept. of EE&CS, Univ. of Wisconsin Milwaukee, WI. Partially supported by NSF Grant NCR-9106327.

[‡]GTE Laboratories Incorporated, Waltham, MA.

[§]IBM T. J. Watson Research Center, Yorktown Heights, NY.

STOC 94- 5/94 Montreal, Quebec, Canada
© 1994 ACM 0-89791-663-8/94/0005..$3.50

1 Introduction

As cryptographic techniques mature and security applications grow, the fundamental need to distribute "cryptographic capabilities" amongst a multitude of agents (rather than relying on a single agent) has been recognized. A multi-agent operation, as typically required, should be based simply on outputs of a quorum (threshold) of agents which can be *easily* combined to the desired output. The combination should anot require security constraints (such as private or oblivious-transfer channels) beyond what is required from a single agent, and should be efficient. It should produce a cryptographic result (*i.e.*, a publicly available output such as a signature, or a private output such as a decryption value of a publicly available ciphertext). Various heuristics have been employed to assure distribution of power, but no systematic secure methods have been developed for this set of tasks.

In this work we suggest a basic primitive which enables rigorous sharing of cryptographic power. We call the mechanism *shareable functions*; these are the functional analog of the fundamental primitive of secret sharing [33, 5] (where a piece of information is distributed). A secret sharing scheme is a one-time operation, in the sense that the shadows (shares) are revealed when the secret is reconstructed. In function sharing, the shadow functions are never revealed to anyone and therefore the secret (*i.e.*, the function's intractability) is maintained and the function is reuseable many times. After reviewing its properties and sufficient (algebraic and complexity) conditions, we give concrete implementations based on exponentiation over groups (in particular the RSA function). We then suggest a variety of new applications to cryptosystems.

Motivation: The need for shareable functions was motivated by various recent efforts to distribute the functionality of a party performing cryptography.

(a) One such effort is obtaining a digital signature of an organization rather than of individuals. The shadow functions are employed to split the *organization sig-*

nature. This exhibits strong integrity and security properties against large internal collusions. The recipient of a signature does not learn, or need to know, the organization's internal structure and operations. A seemingly alternative method that requires explicit signatures of individuals has two major drawbacks: (1) it puts the burden to validate integrity on the receiver; its task complexity thus relies on the number of individual signatories, and (2) it exposes the organization internal regulations (identities / number of authorized signatories), which may be unacceptable (from security perspective). With shared functions, the current public-key methods need not change, yet large organizations can flexibly regulate their internal authorization policy. Previous methods to generate organization signatures have been based on heuristic schemes or tamper-resistant devices. Our secure method does not require such a strong assumption. For heuristics, references and recent methods, see [7, 9, 25, 29].

(b) Various sensitive cryptographic operations require the participation of more than one party; e.g., certifying a signature (signing the signature function of a party). The commercial RSA Data Security Inc.'s Certificate Issuing System (CIS) uses a tamper-proof device which requires t-out-of-l individuals to generate (sign) an X.509 digital signature certificates [22] in order to certify a user's public key [32]. Our methods for assuring multi agent approval do not require such a centralized tamper-proof device and can be performed distributedly (by communication).

(c) Another effort requiring distribution of cryptographic power is the balance of individual privacy with society-protection in encryption systems (see [24, 31, 8]). The idea is to have escrow agents who are able to reveal an individual's key when requested by the court. The philosophy behind key-escrow systems like the Clipper and fair cryptosystems is to share the key using traditional secret sharing. This may not be sufficient– as opening of *keys* compromises **all** traffic encrypted with these keys, which may not always be effective or satisfactory policy. (See also [24] where this problem is discussed and added-on suggestions (which change keys from time to time) are given as a partial remedy). With shareable functions we can take a step forward in balancing individual privacy and society protection. That is, the decryption of specific messages can be opened *without* reducing the strength of the key involved– thus any granularity of legal compromise of privacy is made possible. This provides better privacy for individuals. Another advantage of such granularity is that, when authorized, ciphertext messages can be sporadically sampled and decrypted to assure that the encryption system is employed over meaningful cleartext messages and not over already otherwise encrypted messages (a possibility which is the major criticism of key-escrow systems). We call our systems – based on shareable functions – *balanced cryptosystems.*

1.1 Our results

Motivated by the above scenarios, we investigate secure sharing of functions.

- We define the notion of shareable-function and its computational security.
- We give sufficient algebraic conditions and complexity assumptions for implementions.
- We present algebraic domains where the conditions can be applied, and concretely construct a shareable variant of RSA.
- Finally, we apply shareable functions to design multi-agent cryptosystems (just as trapdoor functions are used in designing single-agent cryptosystems).

1.2 Function sharing vs. secure function evaluation– the difference

Our model should be contrasted with the model of secure function evaluation protocols (sfe) [16]. The security of function sharing is similar to traditional cryptosystems; it relies on the adversary not being able to access enough of the agents' internal memories (and outputs when they are private). The agents have no bilateral communication channels whatsoever and the inputs are public. In resilient sfe the source of security are the secured channels among agents (in addition to memories), see [27, 26, 2].

Considering efficiency, in our model the agents produce one output per each input and pass it to the receiver. On the other hand, the communication complexity of resilient sfe protocols– if employed to produce cryptographic transformation over public channels– will depend on the computational complexity (*i.e.*, the circuit size) of cryptographic functions. Two classes of cryptographic protocols are distinguished in [13]: inefficient ones where the communication depends on the computational complexity of the party (size of circuits rather than on their result, as in sfe), and efficient ones where it does not, as in our model. The former class (which contains sfe) is considered inefficient in practice.

2 Background, Definitions and Notations

Threshold Schemes: With a threshold scheme a secret is shared via shadows during the *shadow generation phase* such that information theoretically less than quorum of shadows reveals nothing about the secret. During the *secret reconstruction phase* any quorum of shadows may be used to efficiently reconstruct the secret.

Definition 1 *Let $\Gamma = \{1, \ldots, l\}$. A (t, l)-threshold scheme over a set $\mathcal{K}$ ($t \leq l$) is two probabilistic polynomial-time algorithms* ($\mathrm{share}_{t,l}, \mathrm{rec}_{t,l}$) *where 1) given a secret $k \in \mathcal{K}$,* $\mathrm{share}_{t,l}$ *returns l shadows (shares) $\{s_1, \ldots, s_l\}$ such that less than t shadows reveal nothing about the secret, and 2) the class of polynomial time reconstruction algorithms* $\mathrm{rec}_{t,l}$ *indexed by subsets of Γ with cardinality t recover the secret k given as input t shadows (i.e., $\mathrm{rec}_{t,l,\Lambda}(s_{i_1}, \ldots, s_{i_t}) = k$ for $\Lambda = \{i_1, \ldots, i_t\} \subset \Gamma$).*

Shamir [33] developed a threshold scheme based on Lagrange Interpolation over a finite field. A random polynomial $r(x)$ of degree $t-1$ (over a finite field) is chosen such that $r(0) = k$, the secret. Each shareholder $i \in \Gamma$ is assigned a unique public number x_i and a secret share $s_i = r(x_i)$. Due to Lagrange interpolation, the set Λ of cardinality t determine

$$k = \sum_{i \in \Lambda} y_{i,\Lambda} \cdot s_i \qquad (1)$$

where $y_{i,\Lambda} = \prod_{\substack{v \in \Lambda \\ v \neq i}} (x_i - x_v)^{-1}(0 - x_v)$.

Definition 2 *A* **trapdoor permutation** *family [11] $\mathcal{F}_h$ consists of a polynomial time generator G given as input 1^h (h the security parameter) returns (wlog) a randomly drawn tuple consisting of two h bit strings* (pu, se) *where* pu *is a public key and* se *is the secret key. The keys define permutations, $f_{\mathrm{pu}}(\cdot)$ and $f_{\mathrm{se}}^{-1}(\cdot)$, over the sampleable message space $\mathcal{D}_{\mathrm{pu}}$ such that $f_{\mathrm{se}}^{-1}(\cdot)$ is the inverse function for $f_{\mathrm{pu}}(\cdot)$ and given* pu *(se) applying $f_{\mathrm{pu}}(\cdot)$ ($f_{\mathrm{se}}^{-1}(\cdot)$) is polynomial time.*

Finding inverses of random values of a randomly chosen permutation without having the secret key se *is hard. That is, for all probabilistic polynomial time algorithms A, for any polynomial* poly(·), *for all h large enough,*[1]
$P[f_{pu}(u) = w : (\mathrm{pu}, \mathrm{se}) \leftarrow G(1^h); w \in_R \{0,1\}^h; u \leftarrow A(1^h, \mathrm{pu}, w)] < \frac{1}{\mathrm{poly}(h)}$.

[1]Recall that, $P[x : e_1; \ldots; e_j]$ is the standard notation for the probability of predicate x after the events e_i occur (are executed) sequentially.

We use the notion of **indistinguishable distributions** [18, 34] (used to define security of encryption and protocols [19, 15]). It is known [17, 14] that a sequential composition of computable indistinguishable distributions is indistinguishable.

3 Function sharing

In this section we introduce the function sharing primitive (Definition 3) and its security (Definition 4).

3.1 The function sharing primitive

The function sharing primitive has two phases: the *shadow function generation phase* (an initialization) where programs $\{P_1, \ldots, P_l\}$ are generated by a key generator for the input function f_{se}^{-1} represented by tuple (pu, se) (randomly chosen by the function generator G), and the *function reconstruction phase* where a threshold (at least t) of *partial results* $P_i(\alpha)$ are created from a public input α and combined to construct $f_{\mathrm{se}}^{-1}(\alpha)$. Note that the two phases are analogous to the two phases of a threshold scheme. The main difference is that the reconstruction phase of a threshold scheme is only performed once while the function reconstruction phase of the function sharing primitive may be performed polynomially many (in the security parameter h) times.

Definition 3 *The function sharing primitive consists of the following:*

Input: (pu, se) $\in \mathcal{F}_h$ *is a random instance of the trapdoor function family.*
Output:

- **Shadow function generation phase:** *A probabilistic polynomial time algorithm* $\mathrm{share}'_{t,l}$ *given* (pu, se) *computes* $(P_1, \ldots, P_l)$.
- **Function reconstruction phase:** *A probabilistic polynomial time algorithm* $\mathrm{rec}_{t,l,\Lambda}$ *given* pu *and any t out of l evaluations $P_j(\alpha)$ (called partial results) for any $\alpha \in \mathcal{D}_{\mathrm{pu}}$ computes $f_{\mathrm{se}}^{-1}(\alpha)$.*

Evaluations: *A bound b =* poly(h) *on the number of evaluations (i.e., the times $\mathrm{rec}_{t,l,\Lambda}$ is employed) is given. Each evaluation m is associated with an arbitrary subset of t or more programs P_i's, (that is i is in an index set computed by a probabilistic polynomial-time $\Lambda_m \in \Gamma = \{1, .., l\}$ and $|\Lambda_m| \geq t$). An evaluations may fail with a negligible probability.*

We now define what is meant by a *secure* function sharing primitive. Intuitively the definition states that an attacker given up to $t-1$ programs P_i *and* a history tape of partial results for polynomial many random input/output pairs (of evaluations of the function via combining partial results) learns no more about the function (in a computational point of view) than it would have learned without this extra information.

Definition 4 *A function sharing primitive is said to be (t, l)-secure (l fixed polynomial in the security parameter h) with respect to $\mathcal{F}_h$, when for all $\{i_1, \ldots, i_j\} \subset \Gamma$ where $0 \leq j < t \leq l$, for all probabilistic polynomial time algorithms A, for any polynomial* $\text{poly}(\cdot)$, *for all h large enough*

$P[f_{\text{pu}}(u)) = w : (\text{pu}, \text{se}) \leftarrow G(1^h); (P_1, \ldots, P_l) \leftarrow \text{share}'_{t,l}((\text{pu}, \text{se})); w \in_R \{0,1\}^h; u \leftarrow A(1^h, w, H, P_{i_1}, \ldots, P_{i_j})] < \frac{1}{\text{poly}(h)}$

where H is a history tape of length polynomial in h whose m^{th} entry contains $\alpha_m \in_R \mathcal{D}_{\text{pu}}, f_{\text{se}}^{-1}(\alpha_m)$ and partial results generated by each member of the set indexed by the set $\Lambda_m \in \Gamma$ ($|\Lambda_m| \geq t$), namely it is the tuple $\langle \alpha_m, f_{\text{se}}^{-1}(\alpha_m), P_{i_{m,1}}(\alpha_m), \ldots, P_{i_{m,|\Lambda_m|}}(\alpha_m) \rangle$. (Note that $f_{\text{se}}^{-1}(\alpha_m)$ is there, but can be omitted wlog as it can be generated given the $P_{i_{m,j}}(\alpha_m)$'s in polynomial time.)

3.2 Technical requirements for (t, l)- security

The following technical Lemma breaks down the security requirements to two conditions.

Lemma 1 *Let l be polynomial in h, $\mathcal{F}_h$ be a family of trapdoor (one-way) permutations which are the input to a function sharing primitive and* $(\text{pu}, \text{se}) \in_R \mathcal{F}_h$. *Let* $\{P_1, \ldots, P_l\} \leftarrow \text{share}'_{t,l}((\text{pu}, \text{se}))$ *and H be a history tape as in Definition 4. If*

1. **Simulateable shadow function generation:** *for any $\Lambda \subset \Gamma$ with $|\Lambda| = t - 1 < l$ there exists a probabilistic polynomial time algorithm $SIM_{\text{share}'_{t,l}}$ which given* pu *and* Λ *as input draws $t-1$ random shadow functions (respective to Λ) with a distribution indistinguishable from the $t-1$ random shadow functions (respective to Λ) drawn by* $\text{share}'_{t,l}$.
2. **Simulateable function reconstruction:** *there exists a probabilistic polynomial time simulator SIM_{rec} given as input* pu, $\{P_{i_1}, \ldots, P_{i_j}\}$ *($0 < j < t$) and all the Λ_m from H creates a simulated history tape H' which is indistinguishable from H by any probabilistic polynomial time distinguisher which takes as input* pu, $\{P_{i_1}, \ldots, P_{i_j}\}$, *all the Λ_m from H and a history tape.*

then the function sharing primitive is (t, l)-secure.

Proof. We show that if $(\text{pu}, \text{se}) \in \mathcal{F}_h$ is an input to a shareable family of trapdoor permutations and if the function sharing primitive is not (t, l)-secure then either 1) or 2) is false. Suppose that 1) and 2) hold but the function sharing primitive is not (t, l)-secure.

Let $\Lambda' = \{i_1, \ldots, i_j\}$ and Λ be a superset of Λ' with cardinality $t - 1$. Due to 1) and 2) it is possible to simulate the function sharing protocol as viewed by Λ'. The shadow function generation phase as viewed by Λ' can be simulated by running $\text{SIM}_{\text{share}'_{t,l}}$ with input pu and Λ, Using these simulated shadow functions, pu, and H simulates the polynomially many evaluations by creating a simulated history tape H' which is indistinguishable from H (due to 2)) by employing SIM_{rec}.

The simulation of the two stages creates a Polynomial-size Attacker A given only simulated tape H' (based on evaluation of the function at random points) and the j simulated functions shadows can not compute $f_{\text{se}}^{-1}(\alpha)$ for a random point $\alpha \in \mathcal{D}_{\text{pu}}$ with probability greater than $\frac{1}{\text{poly}(h)}$ (for large enough security parameter) since the combined simulation is a polynomial-time computation, and that would contradict the fact that f^{-1} is a trapdoor permutation.

On the other hand, if the attacker A can invert when given H and up to $t-1$ real shares with some probability $\frac{1}{\text{poly}(h)}$ for infinitely many h's, then A can be turned into a distinguisher between the "real view" and the "simulated view" using by now standard techniques. Simply, run A on a view, and if inversion occurs, output 1, otherwise output 1 with probability 1/2. This is turned into a distinguisher for the combined "simulated view" of the two stages. This implies (by indistinguishability of sequential compositions of indistinguishable computed distributions) that one of the stages' view 1) or 2) is not simulateable.

□

4 Shareable Functions: Sufficient Conditions

The next two sections define three conditions for secure shareable functions based on trapdoor permutations. In this section we show two conditions for function sharing: an algebraic condition of "share-translation" (a homomorphism of the function composition– different from usual homomorphism associated with certain trapdoors), and an efficiency condition.

4.1 Function sharing via share translation

We would like to share f_k^{-1} via function shadows $P_1 = f_{s_1}^{-1}, \ldots, P_l = f_{s_l}^{-1}$ where the s_i is the ith shadow generated by the Shamir threshold scheme for the secret k and have partial result $P_i(\alpha) = f_{s_i}^{-1}(\alpha)$. To simplify our preliminary discussion we will first use "partial results" $f_{y_{i,\Lambda}\cdot s_i}^{-1}(\alpha)$ (where $y_{i,\Lambda}$ is defined in (1) section 2). With some wishful thinking about the algebraic structure of the key space (of the trapdoor functions) we would like that
$f_k^{-1}(\alpha) = f^{-1}_{(\sum_{i\in\Lambda} y_{i,\Lambda}\cdot s_i)}(\alpha) = \prod_{i\in\Lambda} f^{-1}_{(y_{i,\Lambda}\cdot s_i)}(\alpha)$.
The above suggests working with functions which exhibit the share-translation property $f_{k_1+k_2}^{-1}(\alpha) = f_{k_1}^{-1}(\alpha) * f_{k_2}^{-1}(\alpha)$. An example of such a function is an exponentiation function (*e.g.*, $f_k^{-1} = x^k \bmod n$) often used in cryptography as in [30, 12]. The above wishful thinking is in general not possible, due to several algebraic and computational constraints. For instance, the Shamir threshold scheme [33] requires that the share space be a field. This algebraic constraint is unlikely to exist for many trapdoor permutations. We solve this problem by only requiring the share space to be a finite Abelian group. For another difficulty, note that the above requires the knowledge of Λ before the creation of a partial result. We also demonstrate how to dispose of this knowledge by generating the "partial result" $f_{y_{i,\Lambda}\, s_i}^{-1}(\alpha)$ easily from i, $P_i(\alpha)$ and Λ. Therefore allowing partial results to be created without knowledge of the function shadows that will be used to combine $f_k^{-1}(\alpha)$.

To resolve some of the algebraic and computational problems we define an Abelian group[2] $\mathcal{K}_{\mathrm{se}}(+)$ which contains the secret key se, a group $\mathcal{D}_{\mathrm{pu}}(*)$ which is the domain of the trapdoor function acted upon by a binary operation "$*$", and an "almost everywhere" (with respect to $\mathcal{K}_{\mathrm{se}} \times \mathcal{D}_{\mathrm{pu}}$) function[3] g^{-1}. The function g^{-1} behaves similarly to f^{-1} but it has the share-translation property and it may not be defined for some inputs. That is $g^{-1} : \mathcal{X} \to \mathcal{D}_{\mathrm{pu}}$ where $\mathcal{X} \subset \mathcal{K}_{\mathrm{se}} \times \mathcal{D}_{\mathrm{pu}}$, $g_{\mathrm{se}}^{-1} = f_{\mathrm{se}}^{-1}$ and $g_{k_1+k_2}^{-1}(\alpha) = g_{k_1}^{-1}(\alpha) * g_{k_2}^{-1}(\alpha)$ for $k_1, k_2 \in \mathcal{K}_{se}$. The g^{-1} and $\mathcal{K}_{\mathrm{se}}$ are created because it may not be easy (or even possible) to make elements in $\mathcal{F}_h$ share-translateable. But other functions which do have the desired property may be easy to find. For instance, the function $f_{\mathrm{se}}^{-1}(x) \equiv x^k \bmod p$, for se $= (k,p)$ and p a prime, is a permutation only if k is relatively prime to $p-1$. Thus $Z_{p-1}(+)$ can not be the key space input to trapdoor *permutation* f^{-1} since it contains even numbers. Introducing function (not necessarily a permutation) $g_{\mathrm{se}'}^{-1}(x) \equiv x^k \bmod p$, for se$' = (k,p)$ and p a prime, simplifies our results. This, essentially, will be our sufficient share-translation condition[4] described as follows.

Definition 5 (Share-Translation) *Let the secret key be* se $= k$. *We say that a trapdoor (hard to compute) permutation f_k^{-1} is* share-translateable *when 1) there exists an Abelian* share space group $\mathcal{K}_{\mathrm{se}}(+)$ *where $k \in \mathcal{K}_{\mathrm{se}}$, 2) there exists a polynomial time "mapping" $g^{-1} : \mathcal{K}_{\mathrm{se}} \times \mathcal{D}_{\mathrm{pu}} \to \mathcal{D}_{\mathrm{pu}}$ such that $g_k^{-1} = f_k^{-1}$, 3) the probability that g^{-1} is* not *defined for one or more randomly chosen* poly(h) *elements in $\mathcal{D}_{\mathrm{pu}}$ is negligible, and 4) there exists an associated binary operation "$*$" for the set $\mathcal{D}_{\mathrm{pu}}$ such that $g_{k_1}^{-1}(\alpha) * g_{k_2}^{-1}(\alpha) = g_{k_1+k_2}^{-1}(\alpha)$ for any $k_1, k_2 \in \mathcal{K}_{\mathrm{se}}$ and $\alpha \in \mathcal{D}_{\mathrm{pu}}$.*

We say that a family of trapdoor permutations $\mathcal{F}_h$ is share-translateable if almost all elements in it are share-translateable.

We will show that the above is a sufficient algebraic condition to assure that a function is shareable (perhaps not efficiently). First, we discuss how to generalize the Shamir threshold scheme to work over any group.

4.1.1 Generalized Shamir threshold scheme

As will be proven in Lemma 2 the Shamir threshold scheme can be generalized to work over any finite module[5] where $\mathcal{K}(+)$ is a group and the scalars x_i and $x_i - x_j$ ($i, j \in \Gamma, i \neq j$ and x_i is defined in Section 2) have inverses. To implement share-translation we overview how to create such a threshold scheme as was developed in [10] (and simplify the proof). The approach is similar to performing Lagrange interpolation with the key space GF(2) and $l > 1$ (note if $l > 1$ then it is not possible to use Lagrange with sharespace GF(2)). The idea is to extend GF(2) to GF(2^n) where $2^n - 1 > l$ and perform Lagrange interpolation in GF(2^n). The secret in GF(2^n) is the first component of a new "secret key" (*i.e.*, if k is the secret in GF(2) then make the new "secret key" in $GF(2^n)$ the element $[k, r_1, \ldots, r_{n-1}]$ where the $r_i \in$ GF(2) are not necessarily secret). A similar method is used to find an appropriate module by utilizing algebraic extensions over the integers.

[2]WLOG we assume the group is additive.

[3]It is almost a function since with a negligible probability (for some inputs) it is not defined.

[4]One notes that this is similar to group homomorphisms, however, we avoid the terminology since share homomorphism has other meanings [4].

[5]A module is analogous to a vector space, where the scalar operation is over a ring rather than a field.

Definition 6 (Extension) *Let $Z[u]$ be an algebraic extension over the integers of degree m. The extended share space is the module $\mathcal{K}_{\text{se}} \times \cdots \times \mathcal{K}_{\text{se}}$ (i.e., the direct product of m copies of $\mathcal{K}_{\text{se}}$) over $Z[u]$.*

Let us now create an extension of $\mathcal{K}_{\text{se}}$ which has the appropriate algebraic structure to perform Lagrange interpolation over any finite Abelian group uniformly for any l. Let q be a prime greater than or equal to $l+1$ and u be a root of the cyclotomic polynomial $p(x) = (x^q - 1)/(x-1) = \sum_{j=0}^{q-1} x^j$. This defines an extended key space $\mathcal{K}^{q-1} = \mathcal{K}_{\text{se}} \times \cdots \times \mathcal{K}_{\text{se}} \cong \mathcal{K}_{\text{se}}[x]/(p(x))$, the direct product of $q-1$ copies of $\mathcal{K}_{\text{se}}$. We consider $\mathcal{K}^{q-1}$ as a module over $Z[u] \cong Z[x]/(p(x))$ (i.e., the extension of integers with the element u). Observe that $(u^i)^{-1} = u^{q-i}$ and using the extended Euclidean algorithm [21] $(x_i)^{-1}$, $1 \le i \le l$ may be computed since $\gcd(u^q - 1, u^i - 1) = u^{\gcd(q,i)} - 1$ in $Z[u]$, $u - 1 \mid u^i - 1$ for $1 \le i$, and q is a prime. Therefore, this module has a scalar in which the x_i and $x_i - x_j$ $(i \ne j)$ have inverses when $x_v = (u^v - 1)/(u-1) = \sum_{v'=0}^{v-1} u^{v'}$.

We write elements of $\mathcal{K}^{q-1}$ as $[k_0, \ldots, k_{q-2}]$ and the identity element as $[0, \ldots, 0]$, where 0 is the identity element of $\mathcal{K}_{\text{se}}(+)$. Addition in $\mathcal{K}^{q-1}(+)$ is defined as $[k_0, \ldots, k_{q-2}] + [k'_0, \ldots, k'_{q-2}] = [k_0 + k'_0, \ldots, k_{q-2} + k'_{q-2}]$. The scalar operation $(b_0 + \cdots + b_{q-2}u^{q-2}) \cdot [k_0, \ldots, k_{q-2}]$ is defined recursively from $b \cdot [k_0, \ldots, k_{q-2}] = [b \cdot k_0, \ldots, b \cdot k_{q-2}]$ for $b \in Z$ and $u \cdot [k_0, \ldots, k_{q-2}] = [0, k_0, \ldots, k_{q-3}] + [-k_{q-2}, \ldots, -k_{q-2}]$.

Let us now show how to perform the extended Shamir threshold scheme for any finite Abelian group $\mathcal{K}^{q-1}(+)$.

Protocol 1 Extended Shamir Threshold Scheme

Let $q > l$ be a prime and u be a root of the cyclotomic polynomial $p(x) = \sum_{j=0}^{q-1} x^j$. Let $x_i = \sum_{j=0}^{i-1} u^j$ and any input $k \in \mathcal{K}_{\text{se}}$ be the secret.

Shadow generation phase: The generator chooses a random "polynomial" r over the module $\mathcal{K}^{q-1}$ of degree $t-1$ such that $r([0, \ldots, 0]) = [k, 0, \ldots, 0]$ and gives $s_i = r(x_i)$ to shareholder i.

Secret Regeneration phase Perform (1) from Section 2 over $\mathcal{K}^{q-1}$ to regenerates $r([0, \ldots, 0]) = [k, 0, \ldots, 0]$.

The reader should note that the above protocol does not require any knowledge about $\mathcal{K}_{\text{se}}$ other than how to compute inverses, perform the group operation and choose random elements in the group. We next prove that the above is a threshold scheme over $\mathcal{K}_{\text{se}}$

Lemma 2 *The Shamir (t, l)-threshold scheme works over any finite module $\mathcal{K}$ where the scalars x_i and $x_i - x_j$ have inverses for all $i, j \in \Gamma$ and $i \ne j$.*

Proof. Let[6] $r(x) = x^{t-1}a_{t-1} + \ldots + xa_1 + a_0$ where $a_0 = k$ is the secret and $a_i \in_R \mathcal{K}$. Let shadow $s_j = r(x_j)$. We first prove that it is possible to regenerate k given t shadows. A set of t shadows indexed by Λ define t unknowns $(a_{t-1}, \ldots, a_0)$ and the equations $\sum_{i=0}^{t-1} x_j^i a_i = r(x_j)$ for each $j \in \Lambda$. These equations and unknowns give rise to a Vandermonde matrix with a determinant $\prod_{\substack{i,j \in \Lambda \\ i<j}} (x_i - x_j)$. Since the $x_i - x_j$ are invertible, this matrix is invertible. Thus it is possible to calculate uniquely $a_0 = k$ given t shadows.

We now prove that less than $t-1$ shadows provide nothing from an information theoretical point of view about the secret (*i.e.*, this shows that the shadows are t-wise independent). A set of $t-1$ shadows indexed by Λ' define t unknowns $(a_{t-1}, \ldots, a_0)$ and the equations $\sum_{i=0}^{t-1} x_j^i a_i = r(x_j)$ for each $j \in \Lambda'$. The equation $a_0 = k$ for any $k \in \mathcal{K}$ and these other equation and unknowns give rise to a Vandermonde matrix with determinant $\prod_{i \in \Lambda'} x_i \prod_{\substack{i,j \in \Lambda' \\ i<j}} (x_i - x_j)$. Since the x_i and $x_i - x_j$ are invertible, this matrix is invertible. Thus a_0 can be any $k \in \mathcal{K}$ (*i.e.*, the uncertainty about the secret when given up to $t-1$ shadows is the same as when zero shadows are given). □

4.1.2 Constructing a Function Sharing Primitive

Given Protocol 1, a function sharing primitive is now developed for functions which are share-translateable.

We first define a module $\mathcal{D}^{q-1} = \mathcal{D}_{\text{pu}} \times \cdots \times \mathcal{D}_{\text{pu}}$ over $Z[u]$ with similar module operations as $\mathcal{K}^{q-1}$. Therefore since $\mathcal{D}_{\text{pu}}(*)$ is *wlog* multiplicative, $\mathcal{D}^{q-1}(*)$ has identity element $[1, \ldots, 1]$ and addition in the module $\mathcal{D}^{q-1}$ is defined as $[a_0, \ldots, a_{q-2}] + [a'_0, \ldots, a'_{q-2}] = [a_0 * a'_0, \ldots, a_{q-2} * a'_{q-2}]$ for $a_i, a'_i \in \mathcal{D}_{\text{pu}}(*)$. Recursively the scalar operation is defined by $u \cdot [a_0, \ldots, a_{q-2}] = [1, a_0, \ldots, a_{q-3}] + [a_{q-2}^{-1}, \ldots, a_{q-2}^{-1}]$ and $b \cdot [a_0, \ldots, a_{q-2}] = [a_0^b, \ldots, a_{q-2}^b]$ for $b \in Z$. We can now define the function sharing primitive.

[6] We put the indeterminate on the left since it is traditional for the scalars to act from the left.

Protocol 2 Function sharing primitive

Let (pu, se) be the input to the function sharing primitive and $k = \mathrm{se}$. Create $\mathcal{K}_{\mathrm{se}}(+), \mathcal{D}_{\mathrm{pu}}(*)$ and g^{-1} so that they satisfy conditions in Definition 5.

Let $q > l$ be prime and u be a root of the cyclotomic polynomial $p(x) = \sum_{j=0}^{q-1} x^j$. Let $x_i = \sum_{j=0}^{i-1} u^j$.

Shadow function generation phase: Perform the shadow generation phase from Protocol 1 to generate s_i for each $i \in \Gamma$ for the secret $[k, 0, \ldots, 0] \in \mathcal{K}^{q-1}$. The function shadows are $P_i(\cdot) = [g^{-1}_{s'_{i,0}}(\cdot), \ldots, g^{-1}_{s'_{i,q-2}}(\cdot)]$ where $s_i = [s'_{i,0}, \ldots, s'_{i,q-2}]$.

Shared function evaluation phase: First, at least t partial results $P_i(\alpha)$ are created for a given $\alpha \in \mathcal{D}_{\mathrm{pu}}$. Then calculate $y_{i,\Lambda}$ as in (1) (section 2) and compute $[f_k^{-1}(\alpha), \diamond, \ldots, \diamond] = \sum_{i\in\Lambda} y_{i,\Lambda} \cdot P_i(\alpha)$ where $\diamond$ represents an element in $\mathcal{D}_{\mathrm{pu}}$ which is not relevant.

The above protocol allows us to prove.

Lemma 3 *If a family of trapdoor permutations $\mathcal{F}_h$ is share-translateable (Definition 5) then with an overwhelming probability it is possible to use Protocol 2 to share a random function in $\mathcal{F}_h$ via shadow functions.*

Proof. Fix $\Lambda \subset \Gamma$ such that $|\Lambda| = t$ and let $s_{i_j} = [s'_{i_j,0}, \ldots, s'_{i_j,q-2}]$ be generated by the shadow function generation phase of Protocol 1.

First we prove that $\nu_\alpha : \mathcal{K}^{q-1} \to \mathcal{D}_{\mathrm{pu}} : [s_0, \ldots, s_{q-2}] \to [g^{-1}_{s_0}(\alpha), \ldots, g^{-1}_{s_{q-2}}(\alpha)]$ is an "almost everywhere" module homomorphism[7]. Observe that $\nu_\alpha(s_{i_1}) + \nu_\alpha(s_{i_2}) = [g^{-1}_{s'_{i_1,0}}(\alpha), \ldots, g^{-1}_{s'_{i_1,q-2}}(\alpha)] + [g^{-1}_{s'_{i_2,0}}(\alpha), \ldots, g^{-1}_{s'_{i_2,q-2}}(\alpha)] = [g^{-1}_{s'_{i_1,0}+s'_{i_2,0}}(\alpha), \ldots, g^{-1}_{s'_{i_1,q-2}+s'_{i_2,q-2}}(\alpha)] = \nu_\alpha(s_{i_1} + s_{i_2})$. Similarly for $b \in Z$, $b \cdot \nu_\alpha(s_{i_j}) = \sum_{v=1}^{b}[g^{-1}_{s'_{i_j,0}}(\alpha), \ldots, g^{-1}_{s'_{i_j,q-2}}(\alpha)] = \nu_\alpha(b \cdot s_{i_j})$ and $u \cdot \nu_\alpha(s_i) = [1, g^{-1}_{s'_{i_j,0}}(\alpha), \ldots, g^{-1}_{s'_{i_j,q-3}}(\alpha)] + [(g^{-1}_{s'_{i_j,q-2}}(\alpha))^{-1}, \ldots, (g^{-1}_{s'_{i_j,q-2}}(\alpha))^{-1}] = \nu_\alpha(u \cdot s_{i_j})$.

The proof of the lemma is completed by observing that $\nu_\alpha([k, 0, \ldots, 0]) = \nu_\alpha(\sum_{i\in\Lambda} y_{i,\Lambda} \cdot s_i) = \sum_{i\in\Lambda} y_{i,\Lambda} \cdot P_i(\alpha)$ unless $g^{-1}_{s'_{i,j}}(\alpha)$ is not defined for some α which only occurs with a negligible probability. □

[7] We say "almost everywhere" because $g^{-1}_{k'}(\cdot)$ may not be defined for all elements in $\mathcal{D}_{\mathrm{pu}}$.

4.2 Polynomial efficiency

For feasibility, we require that all of the operations in a function sharing primitive are efficient:

Definition 7 (Share Efficiency) *We say that a trapdoor permutation is efficiently (polynomial time) share-translateable when 1) given* pu, *multiplying elements and computing inverses in the message space $\mathcal{D}_{\mathrm{pu}}$ is polynomial time in h, 2) share space $\mathcal{K}_{\mathrm{se}}$ is a sampleable set, 3) the share space $\mathcal{K}_{se}(+)$ is an Abelian group such that given* se *one can perform the "+" and inverses in polynomial time in h, and 4) $\mathcal{K}_{\mathrm{se}}$ may be found in probabilistic polynomial time, given* se.

A family of trapdoor permutations $\mathcal{F}_h$ is efficiently share-translateable if almost every element in it is.

Theorem 1 *Let l be polynomial in h and let $\mathcal{F}_h$ be a family of trapdoor permutations which is share translateable (Definition 5) and share efficient (Definition 7). Then there exists a polynomial time function sharing primitive for $\mathcal{F}_h$.*

Proof. Due to Lemma 3 it is possible to share random elements in $\mathcal{F}_h$. To complete the proof we demonstrate that the primitive is polynomial time. First $(u^i)^{-1} = u^{q-i}$ and $(x_i)^{-1}$, for $0 < i < q$, can be computed in polynomial time using the extended Euclidean algorithm and it is of the form $b_0 + b_1 u + b_{q-2}u^{q-2}$ where $b_v \in \{-1, 0, 1\}$ [10]. Since t and q (due to density of primes [21]) are polynomial in h and the coefficients of $(x_i)^{-1}$ and $(x_i - x_j)^{-1}$ are small, $y_{i,\Lambda} = y_{i,0} + y_{i,1}u + \cdots + y_{i,q-2}u^{q-2}$ can be computed in polynomial time. Therefore using Definition 7 we note that all operations needed to perform Protocol 2 for both phases can be performed in polynomial time. □

5 Secure Function Sharing via Sampleability

Protocol 1 works universally over any sampleable finite Abelian group (*i.e.*, given the public information, a random key/share can be drawn) where the group operation and computing inverses are in polynomial time. Thus it leaks minimal information about the group. However the share space is often not sampleable as in the RSA function where the share space of the trapdoor permutation could be $\mathcal{K}_{\mathrm{se}} = Z_{\phi(n)}(+)$. Leaking *any* information about $\phi(n)$ may compromise security. The technical tool we use to ensure security is simulation (Goldwasser, Micali, and Rackoff [19]), formally:

Definition 8 (Simulateable) *Let* (pu, se) $\in \mathcal{F}_h$ *be the input to a function sharing primitive. We say that*

the shadow function generation phase is simulateable for $\mathcal{F}_h$ when 1) there exists a probabilistic polynomial time algorithm S_1 which when given pu *returns polynomial space description of $\mathcal{Z}_{\text{pu}}$ (defined below) and a polynomial time algorithm $z^{-1} : \mathcal{Z}_{\text{pu}} \times \mathcal{D}_{\text{pu}} \longrightarrow \mathcal{D}_{\text{pu}}$, 2) there exists a probabilistic polynomial time algorithm $S_2 : \{0,1\}^h \times \{0,1\}^h \times \mathcal{K}_{\text{se}} \longrightarrow \mathcal{Z}_{\text{pu}}$, such that $g_{k'}^{-1} = z_k^{-1}$ for $k \leftarrow S_2(\text{pu}, \text{se}, k'))$ and 3) distributions $\{x \mid x \in_R \mathcal{Z}_{\text{pu}}\}$ and $\{x \mid y \in_R \mathcal{K}_{\text{se}}; x \leftarrow S_2(\text{pu}, \text{se}, y)\}$ are statistically indistinguishable.*
A family of trapdoor permutations $\mathcal{F}_h$ is simulateable if almost every element in it is.

From the above observe that given public information only, one can in polynomial time simulate the actions of g^{-1} (the shared function) and $\mathcal{K}_{se}$ via a function z^{-1} (the simulateable function) and $\mathcal{Z}_{\text{pu}}$ (the simulateable share space), respectively.

5.1 Implementing Function Sharing

We only change the shadow function generation phase of Section 4.1.2 so that the function generation is over the simulateable share space:

Protocol 3 (t, l)-Secure function sharing primitive implementation

Let (pu, se) be the input to the function sharing primitive and $k = \text{se}$. Create $\mathcal{K}_{\text{se}}(+), \mathcal{D}_{\text{pu}}$ and g^{-1} to satisfy conditions in Definition 5.

Let $q > l$ be a prime and u be a root of the cyclotomic polynomial $p(x) = \sum_{j=0}^{q-1} x^j$. Let $x_i = \sum_{j=0}^{i-1} u^j$ and $k \in \mathcal{K}_{\text{se}}$ be the secret.

Shadow function generation phase: Perform the shadow generation phase from Protocol 1 to generate s_i for each $i \in \Gamma$ for the secret $[k, 0, \ldots, 0] \in \mathcal{K}^{q-1}$. Then calculates $(z^{-1}, \mathcal{Z}_{\text{pu}}) \leftarrow S_1(\text{pu})$ and generates $P_i(\cdot) = [z_{s''_{i,0}}^{-1}(\cdot), \ldots, z_{s''_{i,q-2}}^{-1}(\cdot)]$ where $s_i = [s'_{i,0}, \ldots, s'_{i,q-2}]$ and $s''_{i,j} = S_2(\text{pu}, \text{se}, s'_{i,j})$.

Shared function evaluation phase: Same as Protocol 2.

5.2 Security Proof

Given the above implementation we can now prove:

Theorem 2 *If there exists a trapdoor permutation family $\mathcal{F}_h$ which is share-translateable, share efficient and also simulateable then there exists a (t, l)-secure function sharing primitive for $\mathcal{F}_h$.*

Proof. First we demonstrate that Protocol 3 is a function sharing primitive. Since $g_{s'}^{-1} = z_s^{-1}$ for $s \leftarrow S_2(\text{pu}, \text{se}, s'))$ due to Definition 8, $P_i(\alpha)$ in Protocol 2 is the same as $P_i(\alpha)$ in Protocol 3 with an overwhelming probability. Due to Lemma 3, Protocol 3 performs the evaluations correctly except that it may fail with a negligible probability. Moreover all of the operations can be performed in polynomial time due to Definition 8 and Theorem 1. Thus it is a polynomial time function sharing primitive.

We now only need to prove that Protocol 3 is secure, namely it meets the sufficient conditions of Lemma 1. Let $\Lambda' = \{i_1, \ldots, i_j\}$ and $\Lambda = \{i_1, \ldots, i_{t-1}\} \in \Gamma$ be a superset of Λ'. Consider the distributions $\mathcal{A} = \{x \mid y \in_R \mathcal{K}_{\text{se}}; x \leftarrow S_2(\text{pu}, \text{se}, \cdot)\}$ and $\mathcal{B} = \{x \mid x \in_R \mathcal{Z}_{\text{pu}}\}$.

We first prove condition 1) of Lemma 1 is satisfied by proving a stronger result. We show that given function shadows for Λ' then it is possible in polynomial time to generate function shadows for Λ which are statistically indistinguishable from those generated during the function shadow generation phase. From Lemma 1 the shadows of the extended Shamir threshold scheme (Protocol 1) are t-wise independent; this is true even when the secret is of the special form $[k, 0, \ldots, 0]$. Thus any $t-1$ shadows $s_{i_v} = [s'_{i_v,0}, \ldots, s'_{i_v,q-2}]$ generated in Protocol 1 look like $t-1$ randomly chosen elements in $\mathcal{K}^{q-1}$. Namely, the $(t-1) * (q-1)$ components of the shares are random in $\mathcal{K}_{\text{se}}$. In Protocol 3, the set Λ holds function shadows $z_{s''_{v_1,v_2}}^{-1}(\cdot)$ where $s''_{v_1,v_2} \leftarrow S_2(\text{pu}, \text{se}, s'_{v_1,v_2})$ for $v_1 \in \Lambda$ and $0 \leq v_2 \leq q-2$. Observe that since t and q are polynomial in h, $(t-1) * (q-1) < h^K$, where $K = O(1)$. Now a polynomial time simulator after executing $S_1(\text{pu})$ chooses $(t-1-j) * (q-1)$ random elements from $\mathcal{Z}_{\text{pu}}$ and using z^{-1} computes $t-1-j$ simulated shadow functions for $\Lambda \setminus \Lambda'$ (*i.e.*, the shadow functions for Λ but not in Λ'). Due to Definition 8 condition 3), $\mathcal{A}$ and $\mathcal{B}$ are statistically indistiguishable. Thus for some $\epsilon = \omega(1)$, $\sum_\delta |\text{prob}(\mathcal{A}(x) = \delta) - \text{prob}(\mathcal{B}(x) = \delta)| = 1/h^\epsilon < 1/\text{poly}(h)$ where $\delta \in S_2(\text{pu}, \text{se}, \cdot) \cup \mathcal{Z}_{\text{pu}}$. Now $h^K/h^\epsilon < 1/\text{poly}(h)$. Thus a simulator randomly choosing $(t-1-j)*(q-1)$ elements in $\mathcal{Z}_{\text{pu}}$ can generate simulated function shadows which are statistically indistinguishable then those generated by Protocol 3. Since the shadows (after being mapped by S_2) and simulated shadows (in $\mathcal{Z}_{\text{pu}}$) are statistically indistinguishable and since there is a mapping from $\mathcal{Z}_{\text{pu}}$ to functions shadows, the function shadows and simulated function shadows are also statistically indistinguishable. Similarly for t-wise independence. Thus condition 1) of Lemma 1 is met by the function sharing primitive (*i.e.*, simulateable shadow function generation).

We now prove condition 2) of Lemma 1 is satisfied. Let $\Lambda_i'' = \Lambda \cup \{i\}$ for $i \notin \Lambda$. Observe that in Protocol 1, $s_i = y_{i,\Lambda_i''}^{-1} \cdot (k - \sum_{v \in \Lambda} y_{v,\Lambda_i''} \cdot s_v) \in \mathcal{K}^{q-1}$. Thus using Definitions 5 and 7 note that in polynomial time one can compute $P_i(\alpha) = y_{i,\Lambda_i}^{-1} \cdot ([g_k^{-1}(\alpha), g_0^{-1}(\alpha), \ldots, g_0^{-1}(\alpha)] - \sum_{v \in \Lambda} y_{v,\Lambda_i} \cdot P_v(\alpha))$. As mentioned before, $P_v(\alpha)$ in Protocol 2 is the same as $P_v(\alpha)$ in Protocol 3 with an overwhelming probability. Therefore in either Protocol 2 or 3 given $t-1$ shadow functions, α and $f_k^{-1}(\alpha)$ it is possible to generate deterministically in polynomial time the partial results that the other shadow functions would have generated. Thus since the space of $t-1$ simulated shadow functions ($j < t$ of which are the actual shares) are statistically indistinguishable from one generated by Protocol 3, the space of $t-1$ simulated shadows, the result $f_k^{-1}(\alpha)$ and the simulated partial results derived deterministically in polynomial time are also statisically indistinguishable. Thus for any history tape H (as described in Lemma 1) a polynomial time simulator given public information, up to j shadow functions, and the Λ_i's (for stages $i = 1, \ldots, m$) in H returns a simulated history tape H' which is indistinguishable from H and therefore Condition 2) of Lemma 1 is satisfied by Protocol 3 (*i.e.*, simulateable function reconstruction). □

6 RSA-based Secure Function Sharing

We now provide an example based on RSA [30] (it requires that the share space be kept secret). First we review the RSA algorithm. A key generator produces two large random primes p and q, and computes $n = p \cdot q$ and the Euler totent function $\phi(n) = (p-1)*(q-1)$. To compute se, the generator chooses a random d such that $\gcd(d, \phi(n)) = 1$. The public key pu is determined by finding an integer e such that $ed \equiv 1 \bmod \phi(n)$. Thus the trapdoor key is se $= (d, p, q)$, the public key is pu $= (e, n)$, the message space is $\mathcal{D}_{\text{pu}}(*) = Z_n^*(*)$ and the key space is $\mathcal{K}_{\text{se}}(+) = Z_{\phi(n)}(+)$. Lastly, $f_{se}^{-1}(x) \equiv x^d \bmod n$ and $f_{\text{pu}} = x^e \bmod n$.

Observe that knowing $\phi(n)$ for RSA implies computing the trapdoor key from the public key. Therefore it is not possible to have the simulateable key space be $Z_{\phi(n)}(+)$ since this may provide knowledge about $\phi(n)$. We now prove that the simulator can use Z_n as its simulateable share space.

Theorem 3 *Let $\mathcal{F}_h = \{(\text{pu}, \text{se}) \mid (\text{pu}, \text{se}) \leftarrow G(1^h)$ such that p, q are primes, $\text{se} = (d, p, q), \text{pu} = (e, n = pq), ed \equiv 1 \bmod \phi(n)$ and $\text{length}(p) = \text{length}(q) = \frac{h}{2}\}$ be a family of RSA functions with security parameter h. There exists a secure function sharing primitive for $\mathcal{F}_h$.*

Proof. Let $((e, n = pq), (d, p, q)) \in_R \mathcal{F}_h$. Conditions for Definition 5 and 7 are satisfied by $\mathcal{K}_{(d,p,q)} = Z_{\phi(n)}(+)$, $\mathcal{D}_{(e,n=pq)}(*) = Z_n^*(*)$ and $g_{(d,p,q)}^{-1}(x) = x^d \bmod n$.

We now prove that there exists S_1 and S_2 which satisfy conditions of Definition 8. Let $S_1((e, n = pq))$ return $\mathcal{Z}_{(e,n=pq)} = Z_n$ and $z^{-1} : Z_n \times Z_n^* \rightarrow Z_n^* : (k, x) \rightarrow x^k \bmod n$. This clearly can be executed in polynomial time given only pu. Let $S_2((e, n = pq), (d, p, q), k) = k$. Note that condition 2) is clearly true. Thus conditions 1) and 2) of Definition 8 are satisfied.

We now need to demonstrate that Protocol 3 satisfies condition 3) of Definition 8. Consider the distributions $\mathcal{A} = \{x \mid y \in_R \mathcal{K}_{\text{se}}; x \in_R S_2(\text{pu}, se, y)\}$ and $\mathcal{B} = \{x \mid x \in_R Z_n\}$. Now we prove that condition 3) is satisfied by demonstrating that for any polynomial poly$(\cdot)$ for h large enough, $\sum_\delta |\text{prob}(\mathcal{A}(x) = \delta) - \text{prob}(\mathcal{B}(x) = \delta)| < 1/\text{poly}(h)$ where $\delta \in Z_n$. That is we prove that the two distributions are statistically indistinguishable. We break this sum down into two parts and show that both are less than $\frac{1}{\text{poly}(h)}$ by using the fact that when n is chosen as above then $\phi(n) = n + 1 - (p+q) \geq n - n^{2/3}$. The first part of the summation is for the δ's which can be chosen by both $\mathcal{A}$ and $\mathcal{B}$ (*i.e.*, $\delta \in \{0, \ldots, \phi(n) - 1\}$). Thus the first summation is equal to $\phi(n)(\frac{1}{\phi(n)} - \frac{1}{n}) < 1 - \frac{n - n^{2/3}}{n} = \frac{1}{n^{1/3}} < 1/\text{poly}(h)$. The second part of the summation is for the δ's which be chosen by $\mathcal{B}$ but not $\mathcal{A}$ (*i.e.*, $\delta \in \{\phi(n), \ldots, n-1\}$). There are at most $n^{2/3}$ elements in this set. Thus the second summation is at most $n^{2/3} * \frac{1}{n} = \frac{1}{n^{1/3}} < \frac{1}{\text{poly}(h)}$. So the two distributions are statistically indistinguishable.

Since the conditions of Definitios 5, 7 and 8 are satisfied there exists a secure function sharing primitive for $\mathcal{F}_h$ (due to Theorem 2).

□

7 Multi-agent cryptosystems

We now construct a protocol model and protocols that employ the function sharing primitive. We assume that the notion of communicating Turing machines [19] is known. Essentially, a system of communicating machines defines individual machines and their input and output tapes (some accessible by all machines (public tapes) and some are private (accessible to a unique or a subset of the machines). The system also defines the order of taking steps in a synchronous fashion.

We can construct protocols in this model where the agents replace a participant: decryption public-key schemes, encryption and pseudo-random generation, signature and identification schemes.

Definition 9 *A multi-agent cryptosystem $\mathcal{MA}$ consists of the following:*

Inputs:

- **Public key and Input:** *A public tape with a security parameter h and an initial public key PK (of the size of the security parameter, i.e., drawn from $\mathcal{F}_h$). In addition there is $\mathcal{MA}$'s public input tape.*

Players:

- **The Sender:** *The sender is a machine with an output tape and it has read-write access to $\mathcal{MA}$'s input tape.*
- **The Receiver:** *There are two types. (1) A public receiver does not require that the $\mathcal{MA}$'s output be kept secret (such as the signature of a message). (2) A private receiver does require that the $\mathcal{MA}$'s output be kept secret (such as the decryption of ciphertext).*
- **Agents:** *A set of l random polynomial-time communicating Turing machines called agents with each agent i having a private function description tape.*
- **A combiner:** *A probabolistic polynomial time algorithm called a combiner which computes $\text{rec}_{t,l,\Lambda}$ and possibly other polynomial time computations using the agents' output tapes and public information. $\mathcal{MA}$'s output is written by the combiner on a public (private) output tape to be read by the public (private) receiver.*

Execution:

- **Non-interactive rounds:** *In a synchronous fashion in each round, an input parameter from the $\mathcal{MA}$'s input tape is given. Then the agents take a single step which is followed by the combiner taking a single step.*
- **Internal Communication:** *There exists an internal communication tape which is read-only for the combiner and write-only (or write-read in history dependent systems) for the agents. An agent in each round given the round input tape, round parameter, the system's input history tape and its private tape produces a partial result according to the function sharing primitive and perform possibly other computations. It then in some predefined order writes its partial result, its index, and possibly other information on its public (private) output tape if the receiver is public (private). The combiner then takes a step by, first, reading the agents' output tapes and possibly any internal public tapes. It then produces the result of the function reconstruction phase and other computations based on public inputs. These results are written on $\mathcal{MA}$'s output tape.*
- **Bound on Evaluations:** *A bound $b = \text{poly}(h)$ on the number of evaluations (i.e., the times $\text{rec}_{t,l,\Lambda}$ is employed) is given.*

Note that access to the communication tapes implies that the combiner "learns" the internal tapes and messages (knows who sends messages at which round), while the receiver only knows the result of the computation.

To avoid added complexity to the above definition we assumed that the multi-agent cryptosystem and the outsider are not performing an interactive protocol. One can also consider interactive protocols between the sender, the receiver and the $\mathcal{MA}$ cryptosystem.

Let us define what a secure multi-agent cryptosystem is. We allow for two types of adversaries to attack a multi-agent cryptosystem: *eavesdropping adversaries* and *destruction adversaries.* Both types of adversaries have access to all of the public tapes. However, the combiner's and an agent's output tape is public or private depending on what type of receiver the multi-agent cryptosystem has. An eavesdropping adversary has access to the private tapes of up to $t-1$ agents at the start of the system. A destruction adversary may at any round ommit some messages written by the agents on the internal communication tapes. We assume that all but t messages may be omitted and that the set of omitted messages are computed by the adversary. Because of the general nature of the definition we do not say what a valid "attack" is.

Meta Definition 1 *Let $\mathcal{A}$ be a polynomial time machine using a private tape which contains a description of a cryptographic function, a public initialization tape (i.e., the tape contains the security parameter and a public key) and a public input history tape performs a cryptographic task Tsk. Let Ad be a polynomial time adversary attacking $\mathcal{A}$ with capability Cap (such as chosen ciphertext, chosen plaintext) and let $\mathcal{MA}$ be a multiagent version of $\mathcal{A}$ where each agent i has shadow function of $\mathcal{MA}$'s cryptographic function on its private input tape. $\mathcal{MA}$ is said to be as secure as*

$\mathcal{A}$ with respect to Ad if the existence of a polynomial time Ad' which can perform a successful attack on $\mathcal{MA}$ having capability Cap and eavesdropping access to the internal computations of up to $t-1$ agents in $\mathcal{MA}$ (and destruction capabilities) implies Ad can perform the same attack on $\mathcal{A}$.

Using the above non-interactive model and informal definition of multi-agent security, we design multi-agent cryptosystems and argue about their availability and security.

7.1 Multi-agent public key cryptography

Based on shareable functions, and secure probabilistic encryption we can directly show:

Theorem 4 *There exists a multi-agent public-key decryption protocol (assuming shareable trapdoor permutations).*

7.1.1 Balanced public key cryptosystems

An escrow cryptosystem, of which both Fair cryptosystems [24] and the Clipper system (as described in [31]) are examples, allow for the user's **key** to be shared amongst trusted agents. When the government (through a court order) asks the agents to open the **key** of the user (to protect society), the agents give the key. The fairness is assured as long as the majority of agents are trusted the user's privacy is maintained and society (*i.e.*, the government) has a way to protect itself. In [24] it is mentioned that opening a key is too huge a penalty for individuls; a refinement which distributes temporal keys is suggested. This is indeed an improved situation, yet the goal (which it highlights) should be: solving the problem in *full granularity*. Namely, enabling each and every **individual message** to be revealed without affecting other messages. It seems that our *balanced cryptosystem* (based on shareable functions) may provide more acceptable solution in some cases. This can be done in two ways:

1. *Shared public-key*: Each user (key issuing authority) shares its shareable key among the trusted agents. Upon authorized request and given past record (ciphertext), the agents can be employed by society to enforce fairness.

2. *Generic agencies*: There are generic agencies with known public keys (the secret decryption key is shared among individual agents in the agency). Any sender is required to send a message encrypted twice: by the receiver's key and by the agency key. A non-interactive proof of consistency of the two encryptions is also sent (as in [28]). In this method the user's key can never be exposed.

Theorem 5 *There exists a balanced public-key cryptosystems, both, with shared private key and with generic agency key (assuming shareable trapdoor permutations exist).*

7.2 Multi-agent signatures

The most powerful attack on a signature scheme (formalized by Goldwasser, Micali, and Rivest [20]) is of a Forger who tries an adaptive chosen-plaintext attack.

We employ the secure Bellare-Micali [3] scheme (based on trapdoor permutations) where the agents sign using a decryption of a random string which are based on the message and a new random string. The system also employs generation of these agreed upon random strings which are "transparent trusted random bits" [23] generated by the agents (which, in previous stages, first encrypted random bits, then later decrypted all of them, using partial results and ex-ored the bits committed to by the agents). The trusted randomness at a stage is ready for signing at the next stage (and the first three random strings $\alpha_0, \alpha_1, \alpha_2$ are part of the initial public key).

We will prove that if $t-1$ agents can forge a signature with probability better than $1/\text{poly}(h)$ Bellare-Micali's scheme is not secure, or the encryption is not secure (*i.e.*, the trapdoor permutations can be attacked).

Theorem 6 *There is a secure multi-agent signature scheme (based on shareable trapdoor permutations).*

In future work we consider stronger adversaries and add validation proofs and other techniques to cope with them. Further cryptosystems and models will be described in the full version.

Acknowledgment

We would like to thank Ron Rivest for his encouragement regarding this research and his suggested applications, and Joan Feigenbaum for her helpful remarks.

References

[1] W. Alexi, B. Chor, O. Goldreich and C. Schnorr, *RSA/Rabin Bits are* $1/2+1/poly$ *Secure*, Siam Journal on Computing, 17(2) (1988), pp.194-209.

[2] M. Bellare, L. Cowen, and S. Goldwasser, *On the Structure of Secret Key Exchange*, Distributed Computing and Cryptography, DIMACS series in Disc. Math. and Th. Comp. SCi., v. 2, AMS and ACM, pp 79–92, (Eds. J. Feigenbaum and M. Merritt).

[3] M. Bellare and S. Micali, *How to Sign Given Any Trapdoor Function*, Journal of the ACM, (39), 1992, pp. 214–233.

[4] J. C. Benaloh. *Secret sharing homomorphisms: Keeping shares of a secret secret.* Advances in Cryptology, Proc. of Crypto'86 LNCS 263, 1987, pp 251–260.

[5] G.R. Blakley, *Safeguarding Cryptographic Keys*, AFIPS Con. Proc (v. 48), 1979, pp 313–317.

[6] M. Blum and S. Micali, *How to generate cryptographically strong sequences of psuedo-random bits*, SIAM *Journal of Computing*, 13(4), 1984.

[7] C. Boyd, *Digital Multisignatures*, Cryptography and Coding, Claredon Press, 241–246, (Eds. H. Baker and F. Piper).

[8] D.E. Denning et al., To Tap or Not To Tap. CACM 93.

[9] Y. Desmedt and Y. Frankel, *Shared generation of authenticators and signatures*, Advances in Cryptology– Proc. of Crypto 91, Springer-Verlag LNCS 576, 1992, pp. 307–315.

[10] Y. Desmedt and Y. Frankel, *Perfect Zero-Knowledge Sharing Schemes over Any Abelian Group*, Sequences II, Methods in Communication, Security and Computer Science, Springer-Verlag, 1993, pp. 369–378, (Eds. R. Capocelli, A. De Santis and U. Vaccaro). Journal version: SIAM J. on Disc. Math. (to appear).

[11] W. Diffie and M. Hellman, *New Directions in Cryptography* , IEEE Trans. on Information Theory 22 (6), 1976, pp. 644-654.

[12] T. El Gamal, *A Public key cryptosystem and a signature scheme based on discrete logarithm*, IEEE Trans. on Information Theory 31, 465-472, 1985.

[13] M. Franklin and M. Yung, *Secure and Efficient Off-Line Digital Money*, Proc. of the 20th Int. Col. on Automata, Languages and Programming (ICALP), 1993, LNCS 700, Springer Verlag, pp. 265-276.

[14] Z. Galil, S. Haber and M. Yung, *Minimum-Knowledge Interactive Proof for Decision Problems*, SIAM J. Comp., 18, 1989, pp 711–739.

[15] S. Goldreich, S. Micali and A. Wigderson, *Proofs that Yield Nothing But their Validity, and a Methodology of Cryptographic Protocol Design*, J. ACM, 38 (1), 1991, pp 691–729.

[16] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, Proceedings of the Nineteenth annual ACM Symp. Theory of Computing, 1987, pp 218–229.

[17] O. Goldreich and Y. Oren, *Definitions and Properties of Zero-Knowledge Proof Systems*, J. of Cryptology, 7(1), pp 1–32, 1994.

[18] S. Goldwasser and S. Micali, *Probabilistic Encryption*, J. Com. Sys. Sci. 28 (1984), pp 270-299.

[19] S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, Siam J. on Computing, 18(1) (1989), pp 186-208.

[20] S. Goldwasser, S. Micali and R. Rivest, *A Secure Digital Signature Scheme* , Siam Journal on Computing, Vol. 17, 2 (1988), pp. 281-308.

[21] G. Hardy and E. Wright *An introduction to the theory of numbers*, Oxford Science Publications, London, Great Britain, fifth ed., 1985

[22] *OSI Directory - Part 8: Authentication Framework.* ISO (Int. Standard Organization) 9594-8, Geneva (1988).

[23] M. Luby, *PseudoRandomness and Applications*, Princeton University Press, (to appear).

[24] S. Micali, *Fair public-key cryptosystems*, Crypto '92.

[25] T. Okamoto, *A digital multisignature scheme using bijective public-key cryptosystems*, ACM Trans. on Computer Systems, 6(8), 1988, pp. 432–441.

[26] R. Ostrovsky, R. Venkatesan and M Yung, *Fair Games against an All-Powerful Adversary*, Sequences II, Methods in Communication, Security and Computer Science, Springer-Verlag, 1993, pp. 418–429, (Eds. R. Capocelli, A. De Santis and U. Vaccaro).

[27] R. Ostrovsky and M Yung, *On Necessary Conditions for Secure Computations*, Distributed Computing and Cryptography, DIMACS series in Disc. Math. and Th. Comp. SCi., v. 2, AMS and ACM, pp 229–235, (Eds. J. Feigenbaum and M. Merritt).

[28] M. Naor and M. Yung, *Public-key cryptosytems provably secure against chosen ciphertext attack*, Proc. of the 22nd Annual Symposium on the Theory of Computing, 1990, pp. 427–437.

[29] M. Reiter and K. Birman, *How to securely replicate services*, TR92–1274, Cornell University, 1992.

[30] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signature and Public Key Cryptosystems*, Comm. of ACM, 21 (1978), pp 120-126.

[31] *R. L. Rivest and M. E. Hellman and J. C. Anderson*, Responses to NIST's proposal, Commun. ACM, 35(7), 1992, pp. 41–54.

[32] RSA Data Security Inc. publication. Certificate Issuing Unit, manual.

[33] A. Shamir. *How to share a secret*, Commun. ACM, 22 (1979), pp 612-613.

[34] A. C. Yao, *Theory and Applications of Trapdoor functions*, Proceedings of the 23th Symposium on the Foundation of Computer Science, 1982, pp. 80-91.