An Efficient *Threshold* Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack

(Extended Abstract)

Ran Canetti[1] and Shafi Goldwasser[2⋆]

[1] IBM T. J. Watson Research Center, Yorktown Height, NY, 10598,
canetti@watson.ibm.com
[2] Laboratory for Computer Science, Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139,
shafi@theory.lcs.mit.edu

Abstract. This paper proposes a simple *threshold* Public-Key Cryptosystem (PKC) which is secure against adaptive chosen ciphertext attack, under the Decisional Diffie-Hellman (DDH) intractability assumption.

Previously, it was shown how to design non-interactive threshold PKC secure under chosen ciphertext attack, in the random-oracle model and under the DDH intractability assumption [25]. The random-oracle was used both in the proof of security and to eliminate interaction. General completeness results for multi-party computations [6,13] enable in principle converting any single server PKC secure against CCA (e.g., [19,17]) into a threshold one, but the conversions are inefficient and require much interaction among the servers for each ciphertext decrypted.

The recent work by Cramer and Shoup [17] on single server PKC secure against adaptive CCA is the starting point for the new proposal.

1 Introduction

A threshold public-key cryptosystem (PKC) [18] extends the idea of a PKC as follows: instead of a single party holding the private decryption key, there are n *decryption servers,* each of which hold a piece of the private decryption key. When a user receives a ciphertext c to be decrypted, she sends c to each decryption server, receives a piece of information from each, and recovers the cleartext from the collected pieces.

Semantic security of encryption schemes [28] can be easily extended to the threshold PKC case. A threshold PKC is called *t-secure* if a coalition of t curious but honest servers cannot distinguish between ciphertexts of different messages, yet sufficiently many servers can jointly reconstruct the cleartext. A threshold

⋆ Supported by DARPA grant DABT63-96-C-0018.

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 90–106, 1999.

PKC is called *t-robust* if it meets these requirements even when up to t servers are maliciously faulty.

Secure and robust threshold PKC's can be designed, under general assumptions such as the existence of trapdoor permutations and using multi-party computation completeness theorems [6,26], to convert any centralized semantically secure PKC into a threshold one. More efficient threshold PKC's have been designed based on the RSA and DH intractability assumptions [24,18,35]. All of these proposals require interaction among the servers and the user, in order to achieve robustness for a linear fraction of faults. The general conversions require interaction to achieve both security and robustness. In the work of [24] the presence of a trusted dealer, which distributes verification data for each pair of server and user in a pre-processing stage, is proposed as a way to eliminate interaction and yet achieve robustness for linear number of faults (they actually address RSA signatures but the work can be easily reformulated for RSA decryption).

Stronger notions of security of centralized encryption schemes, namely security against 'Lunch-time Attacks' and 'chosen ciphertext attacks' (CCA) were defined, constructed, and studied in [33,38,19,17,4]. These notions capture additional security concerns when using encryption within a general security application. CCA security of *threshold* PKC has been recently defined in [25]. In principal the Dolev-Dwork-Naor PKC secure against CCA (using non-interactive zero knowledge) can be converted, using multi-party completeness theorems, into a threshold PKC secure against CCA if trapdoor functions exist, but the resulting scheme is inefficient and requires much interaction among the servers. *Efficient* CCA-secure threshold PKC schemes were proposed in [25], in the *random oracle model* under the DDH intractability assumption. The use of random oracles was essential for proving security against CCA. Once the random oracle was present it was also used to eliminate interaction to achieve robustness of the scheme against a linear number of faulty servers. Our goal is to design an efficient threshold PKC secure against CCA *not* in the random oracle model.

A threshold decryption service has several applications. Let us sketch a few. One application (suggested in [25]) is for distributing the escrow service in a *key recovery* mechanism and allowing it to decrypt only specific messages rather than entirely recover the key. Another attractive application is for having public encryption keys associated with an *organization.* Here messages directed to the organization are encrypted with the organization's public key; the organization's decryption servers now direct the decrypted plaintext to an appropriate organization member. Another application is for a decryption service that 'sits on the net' and offers decryption services for customers who do not have their own certified public keys. This service can also be part of an 'electronic vault' application (e.g., [23]). Here it may be important that the decryption be done so that no one except some specified party, not even the decryption servers themselves, will learn anything about the plaintext. (Our security requirements from a threshold PKC take these scenarios into consideration, in an explicit way.)

1.1 New Results

In this paper we present a new threshold PKC, which is provably secure against CCA based on the DDH intractability assumption. Our scheme makes no use of random oracles. The scheme achieves security against a coalition of t honest but curious servers upto $t < \frac{n}{2}$.

The starting point for our scheme is the recent attractive result of Cramer and Shoup [17] which proposed (using techniques reminiscent of those of [25]) an efficient *centralized* PKC secure against adaptive CCA, under the DDH intractability assumption.

The idea of the Cramer-Shoup scheme is that the ciphertext carries with it a *tag*, which the decryption algorithm checks for validity before computing the cleartext. If the tag is valid then the cleartext is output, else the decrypting algorithm outputs 'invalid'. Simplistically stated, unless the legal encryption algorithm was used to produce the ciphertext, it is computationally hard to come up with anything but an invalid tag, and thus it is safe for the server to decrypt ciphertexts carrying a valid tag.

Differently from previous PKC's proved secure against lunch-time attacks and CCA [33,19], this scheme is **not** publicly verifiable. That is, deciding whether the tag is valid or not requires the knowledge of the private key. In particular, this knowledge enables computing from the ciphertext a tag' which should equal tag when the ciphertext is valid.

We now turn our attention to trying to make Cramer-Shoup into a threshold PKC system. First note that if one is willing to increase the size of the ciphertext (and of the public encryption key) proportionally to the number of servers then achieving threshold CCA security is very simple: Let each server have a separate public key of the Cramer-Shoup scheme, and modify the encryption algorithm to that it first generates a Shamir secret sharing $m_1 ... m_n$ of the message m, and then each m_i is separately encrypted using the public key of the ith server. Each server decrypts its share as usual and hands it to the decrypting user.

However, we are interested in schemes where the ciphertext and the encryption key are small, and in particular independent of the number of servers. A straightforward approach would thus be to distribute the private key among all the decryption servers. When a ciphertext arrives, the servers distributively compute whether the tag is valid or not and if it is valid each server outputs a piece of the cleartext. The user then uses the pieces to recover the cleartext. The basic problem of this approach is: *how to distributively implement the check that the tags are valid?* General completeness results for multi-party computation indicate that this is of course possible in principle, but requires interaction between servers for every ciphertext received. More efficient, DDH based protocols seem to require interaction as well.

The Main Idea: Avoiding the Validity Check The new idea is to first modify the PKC scheme so as to avoid an explicit validity check. Instead, the decrypting algorithm (still in the standard PKC case) will output the cleartext

if the ciphertext is valid, and otherwise a value indistinguishable from random[1]. Thus, when the ciphertext is invalid (as defined in [17]) the user will get essentially 'random garbage' computationally unrelated to the ciphertext. Such a modified scheme (which we label M-CS) enjoys a very similar security proof to the original scheme, but it is now possible to turn it into a threshold PKC scheme avoiding the distributed validity check.

In our threshold PKC scheme, each of the n servers will output a piece of information with the following property $\mathcal{P}$:

• if the ciphertext was *valid,* then the cleartext can be recovered from the pieces sent by the decryption servers; but

• if the ciphertext was *invalid,* then the collection of all the pieces is indistinguishable from random garbage.

How is this achieved? Let *tag, tag'* be as discussed above. We come up with a function f such that (1) f(*tag,tag'*) = 1 if *tag=tag'* ; (2) f(*tag,tag'*) = *rval* if *tag≠tag'* (where *rval* is indistinguishable from random); and (3) f is easy to distribute in the sense that it is easy to compute a share of f(*tag,tag'*) from a share of the secret key. Condition (3) is necessary for threshold PKC whereas any function with input/output behavior as specified in conditions (1)-(2) suffices for M-CS . Using such f, each server will compute from the ciphertext and its share of the private key, a share of $f(tag, tag')$ and send to the user a share of $cleartext \cdot f(tag, tag)$.[2] The user will combine the shares to obtain $cleartext \cdot f(tag, tag')$. This choice of f guarantees property $\mathcal{P}$.

We propose to use $f(tag, tag') = (tag/tag')^s$ where s is a random exponent. In order to implement f, at system startup the servers will agree on a sequence of random numbers s shared between them using some secret sharing method such as polynomial secret sharing, and will use these numbers for f as ciphertexts arrive.

Where Does the Randomness in Decoding Come from? The idea described above requires that for *each* ciphertext, the servers will use a new random number that is shared among them using a secret sharing method such as polynomial secret sharing. How are these numbers chosen and shared? We suggest the following method.

A straightforward implementation would be that before the start of the system the servers agree using standard methods (e.g [39,6,20,35,22]) on m random numbers $r_1, ...r_m$ each of which is shared using a polynomial secret sharing among the n players. These are used for decrypting m ciphertexts, after which time a new set of random numbers will be chosen. This means, that each server must store in local memory m shares of m random numbers in addition to his secret

[1] This value does not have to be random. It would actually be sufficient to output, in case of invalid tag, a value which is unrelated to the ciphertext.

[2] This is a slight over simplification for purpose of exposition in the introduction. In the actual scheme the server sends a share of $mask \cdot f(tag, tag')$ where $mask \cdot cleartext$ is part of the ciphertext. Receiving *mask* will enables the user to compute the cleartext. See exact details within.

key. (Alternatively, the servers may generate these random numbers every time a ciphertext appears. However, this method requires interaction among servers at the time of decryption, and is thus not recommended.)

This implementation may encounter synchronization problems when the servers receive the ciphertexts in different orders. We suggest solutions within.

We do not know how to keep the memory requirements of the servers independent of the number of decryptions to be performed, without interaction among the servers. This is left as an interesting open problem. (See more details in Section 3.1.)

Robustness Suppose now that some of the decryption servers are maliciously faulty. To achieve t-robustness we propose several variants of our basic scheme, all of which use [39,20] style polynomial secret sharing as a building block. Our solutions use standard tools which have been used in the literature to address robustness of threshold signature and encryption schemes such as the prover proving in zero-knowledge to the user that the share provided is proper; we come up with efficient instantiations of such tools tailored to the tasks at hand. We stress that in all methods the public encryption key and the encryption algorithm are identical to those of Cramer-Shoup, and in particular are independent of the number of servers. We sketch these methods, all of which achieve t-robustness for up to $t < \frac{n}{3}$ malicious server faults.

- A first method is fully non-interactive, and is efficient when $t = O(\sqrt{n})$. Practically speaking, when, say, $n = 7$ and $t = 2$ this method is quite efficient.
- A second method requires a simple four-round interactive proof between the user and the decryption servers (no interaction between the decryption servers themselves is necessary). Here each server proves to the user that the piece of decryption information provided is correct. The interactive protocol can be avoided when sufficient number of decryption servers do not act in a faulty fashion. The user first runs a local detection-algorithm to see if she can use the pieces of information she received from the servers to decrypt the ciphertext. Only when the user detects that too many pieces were faulty, should she carry out the interactive-proofs to determine which pieces were faulty and should be discarded. Here the decrypting user needs some verification information for each of the servers. Thus the size of the public file grows by a factor of n. Yet, it is stressed that the encryption algorithm remains identical to that of Cramer-Shoup, and the public key needed for encryption remains small.
- A third method uses the technique of check-vectors introduced by [37] for VSS implementation and used by [24] to achieve robustness of threshold RSA signatures. The idea of [24] was that at the time of key generation, a trusted dealer generates additional *verification data* for every pair of user-server, and gives part of this data to the user and part to the server. At the time of signature verification, the user uses her verification data to verify non-interactively that each piece of RSA signature she received from each server is non-faulty. A slight modification of the idea of [24] can be applied to our scheme as well to make it non-interactive and t-robust for $t < \frac{n}{3}$. It will however require each potential

decrypting user to have some secret information, and increase the size of each server's key proportionally to the number of potential decrypting users. Thus, this variant is adequate when the number of decrypting users is small, or a 'decryption gateway' is available. (For lack of space, this method is deleted from the proceedings version. It is described in [12].)

Remark: The question of whether it is possible to achieve robustness efficiently against a linear number of faults without interaction (either among the servers or between the servers and the user) or a trusted dealer is an interesting open problem for threshold PKC regardless of which security is desirable, be it semantic security or CCA.

1.2 Additional Contributions of This Paper

A New Definition of Security for Threshold PKC. Another contribution of our work is proposing an alternative definition of security for threshold PKCs. than the definition of [17]. (The definition of [17] is stated in the random-oracle model; yet it can be readily transformed to protocols that do not use the random oracle.) An attractive feature of the new definition (which follows a standard methodology for defining security of cryptographic protocols [27,31,1,9]) is that it is geared towards defining the security of the threshold PKS as a component within larger systems. In particular, on top of guaranteeing CCA security it addresses issues like non-malleability [19], plaintext awareness [5,4] and security against dynamic adversaries [3,10].

Remote Key Encryption. One of the by-products of our method is yet another variant of our PKC (for the single or multiple server case) such that the user can send the ciphertext to a decryption server(or several servers) on line and receive information which allows the user to recover the cleartext. Yet, neither the servers nor anyone else listening on line can get any information about the cleartext. This functionality has been introduced and (very different) constructions were given in [7]. This variant is secure against lunch-time attacks only.

Proactiveness. Our techniques can be 'proactivized' (i.e., modified to withstand mobile faults, as suggested in [34,11]) in standard ways [29]. See more discussion in [12].

2 Security of Threshold Cryptosystems

We present a measure of security of threshold PKCs. Our formalization is geared towards capturing the security requirements that emerge when using the system as a "service" in a complex and unpredictable environment. In a nutshell, the definition here requires that the system behaves like an "ideal encryption service" under *any usage pattern.* Indeed, this requirement incorporates known security measures like CCA security, non-malleability, plaintext awareness, and security against dynamic adversaries.

The definition here takes a different approach than that of [25], where threshold CCA-security is regarded as a natural extension of the standard definition of CCA-security to the context of threshold cryptosystems. In particular, security according to the definition here implies security according to the definition of [25]. (The converse does not hold in general.)

For lack of space we only sketch the definition in this extended abstract. See [12] for full details on the definition and the relations with that of [25].

Outline of our definition. Following the approach used for defining security of general cryptographic protocols [27,31,1,9], we proceed in three steps. First we formalize the model of computation and specify a syntax for threshold PKCs. Next we specify an idealized model where a threshold PKC is replaced with an "ideal encryption service". Finally, we say that a threshold PKC is secure if it *emulates* the ideal service in a way described below.

THE COMPUTATIONAL MODEL AND THRESHOLD PKCS. There are n decryption servers $S_1 ... s_n$, an encrypting user E and a decrypting user U. A threshold PKC consists of:
A key generation module, that given the security parameter generates a public key, pk, known to all parties, and some secret key, sk_i, known to each server S_i;
An encryption algorithm (run by E) that, given pk, a message m to be encrypted, and random input ρ, outputs a ciphertext $c = \text{ENC}_{\mathsf{pk}}(m, \rho)$;
A server decryption module that, when operating within server S_i and given sk_i and a ciphertext c, possibly interacts with D and other servers and eventually generates a decryption share μ_i;
A user decryption module (run by D) that, given a ciphertext c, interacts with the servers, and eventually outputs $m = \text{DEC}_{\mathsf{pk}}(c, \rho_i, \mu_1 ... \mu_n)$.

A run of the system consists of an invocation of the key generation module (at the end of which pk is made public and the secret keys are given to the corresponding servers), followed by an interaction among the parties via some standard model of distributed communication. (For simplicity assume ideally secure and authenticated communication links). The interaction is orchestrated by an adversary $\mathcal{A}$ who can invoke E and D on cleartexts and ciphertexts of its choice; in addition, $\mathcal{A}$ can corrupt D and up to t servers. (The corruptions are either static or dynamic. Corrupting D gives $\mathcal{A}$ access to the decrypted data.) We augment the model by allowing the adversary to freely interact with an additional entity, called an environment machine $\mathcal{Z}$. This (Turing) machine models the external environment, and in particular provides the adversary with arbitrary and interactive 'auxiliary input' throughout the run of the system. In particular, $\mathcal{Z}$ learns all the information learned by $\mathcal{A}$ (and, in general, can have additional information that is unknown to $\mathcal{A}$.)

We let the global output $\text{EXEC}_{\tau,\mathcal{A},\mathcal{Z}}$ of a run of a threshold PKC τ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ be the concatenation of the output of all the parties, the adversary, and the environment machine. In particular, the global output reg-

isters all the encryption requests made to E, all the decryption requests made to D and each S_i, and the resulting ciphertexts and cleartexts.

THE IDEAL ENCRYPTION MODEL. The ideal model consists of replacing the four modules of a threshold PKC with a trusted service T, parameterized by a threshold t, and a security parameter k. First T receives a description of a distribution Γ from the adversary (who is now called an ideal model adversary, $\mathcal{S}$).[3] Next, the trusted party provides the following services:
Ideal Encryption, where E hands T a message m to encrypt. In response, E receives a receipt c, chosen from distribution Γ *independently of* m.
Ideal Decryption, where The servers can hand a receipt c to T. Once t servers have handed c to T, and if c was previously generated by T, then T hands D the message m that corresponds to c. Otherwise T ignores the request.
A run of the system in the ideal model is orchestrated by the adversary in the same way as described above.

Let the ideal global output $\text{IDEAL}_{t,\mathcal{S},\mathcal{Z}}$ be defined analogously to $\text{EXEC}_{\tau,\mathcal{A},\mathcal{Z}}$ with respect to parties running in the ideal encryption model with ideal-model adversary $\mathcal{S}$, where t is the trusted party's threshold.

SECURITY OF THRESHOLD PKCS. A threshold PKC τ is called t-secure if it *emulates* the ideal encryption service, in the following way. For any adversary $\mathcal{A}$ there should exist an ideal model adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ the global outputs $\text{IDEAL}_{t,\mathcal{S},\mathcal{Z}}$ and $\text{EXEC}_{\tau,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable (when regarded as distribution ensembles). We stress that the environment $\mathcal{Z}$ is the same in the real-life and ideal executions; that is, $\mathcal{S}$ can "mimic" the behavior of $\mathcal{A}$ in *any* environment.

Replacing an ideal service. The quantification over all environments $\mathcal{Z}$ provides a powerful guarantee. In particular, it captures the interaction of *any application protocol* with the PKC in question. Consequently, this definition can be used to show the following attractive property of PKCs. Consider an arbitrary, multi-party 'application protocol' π where, in addition to communicating over the specified communication channels, the parties have access to an ideal encryption service similar to the one described above. Let τ be a PKC that meets the above definition, and let π^τ be the protocol where each call to the ideal service is replaced, in the natural way, with an invocation of the corresponding module of τ. Then π^τ *emulates* π, where the notion of emulation is similar to the one used above. See more details in [12].

3 A Threshold Cryptosystem

Our threshold cryptosystem is based on the Cramer-Shoup cryptosystem [17]. We first briefly review the (basic variant of the) Cramer-Shoup scheme, denoted

[3] Typically, Γ will be the distribution of an encryption of a random message in the domain, under a randomly chosen public key.

CS, and modify it as a step towards constructing the distributed scheme. Next we present the basic scheme and its extensions.

The Cramer-Shoup scheme. Given security parameter k, the secret key is $(p, g_1, g_2, x_1, x_2, y_1, y_2, z, H)$ where p is a k-bit prime, g_1, g_2 are generators of a subgroup of $\mathbb{Z}_p$ of a large prime order q, function H is a hash function chosen from a collision-resistant hash function family and $x_1, x_2, y_1, y_2, z \stackrel{R}{\leftarrow} \mathbb{Z}_q$.[4] The public key is (p, g_1, g_2, c, d, h) where $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$.

It is assumed that messages are encoded as elements in $\mathbb{Z}_q$. To encrypt a message m choose $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and let $\text{ENC}(m, r) = (g_1^r, g_2^r, h^r m, c^r d^{r\alpha})$, where $\alpha = H(g_1^r, g_2^r, h^r m)$. Decrypting a ciphertext (u_1, u_2, e, v) proceeds as follows. First compute $v' = u_1^{x_1 + y_1 \alpha} \cdot u_2^{x_2 + y_2 \alpha}$. Next, perform a validity check: if $v \neq v'$ then output an error message, denoted '?'. Otherwise, output $m = e/u_1^z$. Security of this scheme against CCA is proven, based on the decisional Diffie-Hellman assumption (DDH), in [17].

Towards a threshold scheme. We first observe that this scheme can be slightly modified as follows, without losing in security. If the decryptor finds $v \neq v'$ then instead of outputting '?' it outputs a random value in $\mathbb{Z}_q$. In a sense, the modified scheme is even "more secure" since the adversary does not get notified by the decryptor whether a ciphertext is valid.

Next, modify this scheme further, as follows. The decryption algorithm now does not explicitly check validity. Instead, given (u_1, u_2, e, v) it outputs $e/u_1^z \cdot (v'/v)^s$, where v' is computed as before and $s \stackrel{R}{\leftarrow} \mathbb{Z}_q$. (Note that now the decryption algorithm is *randomized.*) To see the validity of this modification, notice that if $v = v'$ then $(v/v')^s = 1$ for all s, and the correct value is output. If $v \neq v'$ then the decryption algorithm outputs a uniformly distributed value in $\mathbb{Z}_q$, independent of m, as in the previous scheme. Call this scheme M-CS.

Claim. If scheme CS is secure against CCA then so is scheme M-CS.

Proof. Correctness of M-CS (i.e., correct decryption of correctly generated ciphertexts) clearly holds. To show security against CCA, consider an adversary $\mathcal{A}$ that wins in the 'CCA-game' (see [17]) against M-CS with probability non-negligibly more than one half. Construct the following adversary, $\mathcal{A}'$ that operates against CS. $\mathcal{A}'$ runs $\mathcal{A}$, with the exception that whenever $\mathcal{A}'$ receives an answer '?' from the decryption oracle it chooses $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and gives r to $\mathcal{A}$. Finally $\mathcal{A}'$ outputs whatever $\mathcal{A}$ does. The view of $\mathcal{A}'$ is distributed identically to its view in an interaction with M-CS, thus it predicts the bit b chosen by the encryption oracle of $\mathcal{A}'$ with probability non-negligibly more than one half.

Verifying Validity of Ciphertexts. An apparent disadvantage of M-CS is that even a legitimate user of the decryption algorithm does not learn whether a

[4] In fact, H can be a target-collision-resistant hash function. The notation $e \stackrel{R}{\leftarrow} D$ means that element e is drawn uniformly at random from domain D.

ciphertext was valid. However, this information may be obtained in several ways: First, when applying the decryption algorithm twice to an invalid ciphertext, two independent random numbers are output, but if the ciphertext is valid then both applications output the same cleartext. Alternatively, valid cleartexts can be assumed to have a pre-defined format (say, a leading sequence of zeros). The output of the decryption algorithm on an invalid ciphertext, being a random number, has the right format with probability that can be made negligibly small.

On Remotely Keyed Encryption: As a side remark, one can trivially change CS and M-CS to qualify as a remotely-keyed-encryption scheme [7] secured against lunch-time attacks. Simply, drop d from the public key, and let $\text{ENC}(m,r) = (g_1^r, g_2^r, h^r m, c^r)$.[5] Then the user sends to be decrypted remotely only (g_1^r, g_2^r, c^r), dropping the third component of the ciphertext. To decrypt, the server who gets (u_1, u_2, v) computes $v' = u_1^{x_1} u_2^{x_2}$ and sends $p = \frac{(v'/v)^s}{u_1^z}$ back to the user. The user sets $m = e \cdot p$. Clearly, the server got no information about m. A similar modification can be applied to the threshold PKC coming up in the next section, to obtain a remotely keyed threshold PKC secure against lunch time attacks.

3.1 An Threshold Cryptosystem for Passive Server Faults

The basic threshold scheme, denoted T-CS, distributes scheme M-CS in a straightforward way. Let p, q, g_1, g_2 be as in the original scheme, and let t be the threshold. The scheme requires an additional parameter, L, specifying the number of decryption performed before the public and secret keys need to be refreshed. We first describe the scheme for the case where all serves follow their protocol.

KEY GENERATION. For simplicity we assume a trusted dealer for this stage. This simplifying assumption can be replaced by an interactive protocol performed by the servers. This can be done using general multi-party computation techniques [26,6,13] or more efficiently using techniques from [35]. Say that a polynomial $P(\xi) = \sum_{i=0}^{d} a_i \xi^i \pmod q$ is a random polynomial for a if $a_0 = a$ and $a_1 ... a_d \stackrel{R}{\leftarrow} \mathbb{Z}_q$. The dealer generates:

- $x_1, x_2, y_1, y_2, z \stackrel{R}{\leftarrow} \mathbb{Z}_q$ as in the original CS, and random degree t polynomials $P^{x_1}(), P^{x_2}(), P^{y_1}(), P^{y_2}(), P^{z}()$ for x_1, x_2, y_1, y_2, z, respectively.
- L values $s_1 ... s_L \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and random degree t polynomials $P^{s_1}() ... P^{s_L}()$ for them.
- L random degree $2t$ polynomials $P^{o_1}() ... P^{o_L}()$ for the value 0.[6]

Let $x_{j,i} = P^{x_j}(i)$. Let $y_{j,i}, z_i, s_{l,i}, o_{l,i}$ be defined similarly. The secret key of server S_i is now set to $\mathsf{sk}_i = (p, q, g_1, g_2, x_{1,i}, x_{2,i}, y_{1,i}, y_{2,i}, z_i, s_{1,i} ... s_{L,i}, o_{1,i} ... o_{L,i})$. The public key is identical to that of CS: $\mathsf{pk} = (p, q, g_1, g_2, c, d, h)$ where $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$.

[5] This simplification was suggested in [17].

[6] Looking ahead, we note that these values are needed to make sure that the partial decryptions are computed based on a *random* degree $2t$ polynomial. More specifically, these shares make sure that polynomial $Q()$ defined in Equation (2) below is a *random* degree $2t$ polynomial for the appropriate value.

ENCRYPTION is identical to CS: $\text{ENC}_{\mathsf{pk}}(m,r) = (g_1^r, g_2^r, h^r m, c^r d^{r\alpha})$, where $\alpha = H(g_1^r, g_2^r, h^r m)$.

DECRYPTION. Each server S_i proceeds as follows, to decrypt the lth ciphertext, (u_1,u_2,e,v). First it computes a share v_i' of v', by letting $v_i' = u_1^{x_{1,i}+y_{1,i}\alpha} u_2^{x_{2,i}+y_{2,i}\alpha}$. Then it computes a share $u_1^{z_i}$ of u_1^z and a share $g_1^{o_{l,i}}$ of the value '1'. Next it computes and outputs the partial decryption:[7]

$$f_i = u_1^{z_i} \cdot (v/v_i')^{s_{l,i}} \cdot g_1^{o_{l,i}}.$$

The user module collects the partial decryptions $f_1 ... f_n$ and computes the value $f_0 = \Pi_{i=1}^n f_i^{\lambda_i}$, where the λ_i's are the appropriate Lagrange interpolation coefficients; that is, the λ_i's satisfy that for any degree $2t$ polynomial $P()$ over $\mathbb{Z}_q$ we have $P(0) = \sum_{i=1}^n \lambda_i P(i)$. Next, the user outputs $m = e/f_0$.

Theorem 1. *If the DDH assumption holds then* T-CS *is a t-secure threshold cryptosystem for any $t < \frac{n}{2}$, provided that even corrupted servers follow their protocol.*

Proof. See proof in [12]. Here we only verify that the output of the user's decryption module is identical to the output of the decryption module in M-CS. Each partial decryption f_i can be written as follows:

$$f_i = u_1^{z_i - s_{l,i}(x_{1,i}+y_{1,i}\alpha)} u_2^{-s_{l,i}(x_{2,i}+y_{2,i}\alpha)} v^{s_{l,i}} g_1^{o_{l,i}} \tag{1}$$

Let $r_1 = \log_{g_1} u_1$ (i.e., r_1 satisfies $g_1^{r_1} = u_1$), let $r_2 = \log_{g_2} u_2$, and let $r_3 = \log_{g_1} v$. Then we have

$$f_i = g_1^{r_1 \cdot z_i - r_1 \cdot s_{l,i} x_{1,i} - r_1 \alpha \cdot s_{l,i} y_{1,i} - r_2 \cdot s_{l,i} x_{2,i} - r_2 \alpha \cdot s_{l,i} y_{2,i} + r_3 \cdot s_{l,i} + o_{l,i}}.$$

Consequently, $f_i = g_1^{Q(i)}$ where $Q()$ is the degree $2t$ polynomial:

$$\begin{aligned} Q() = {} & r_1 P^z() - r_1 P^{s_l}() P^{x_1}() - r_1 \alpha P^{s_l}() P^{y_1}() - r_2 P^{s_l}() P^{x_2}() \\ & - r_2 \alpha P^{s_l}() P^{y_2}() + r_3 P^{s_l}() + P^{o_l}() \end{aligned} \tag{2}$$

It follows that

$$f_0 = g_1^{Q(0)} = g_1^{r_1 \cdot z - r_1 \cdot s_l x_1 - r_1 \alpha \cdot s_l y_1 - r_2 \cdot s_l x_2 - r_2 \alpha \cdot s_l y_2 + r_3 \cdot s_l + 0} = u_1^z \cdot (v/v')^{s_l}$$

therefore $e/f_0 = m \cdot (v'/v)^{s_l}$.

How to synchronize the s's. The above scheme may encounter synchronization problems when the servers receive the ciphertexts in different orders, and consequently associate shares of different s's with the same ciphertext. A way for solving this problem is to have the servers agree on a bivariate polynomial $H(x,y)$

[7] Once the partial decryption is generated, the server *erases* the shares $o_{l,i}, s_{l,i}$. This provision is important for proving security of the scheme against *dynamic* adversaries that may corrupt parties during the course of the computation.

of degree t in x and degree L in y, where each server P_i holds the degree-m univariate polynomial $H_i(y) = H(i, y)$. The value $s_{i,c}$ associated with the ciphertext c is computed as $s_{i,c} = H(i, h(c)$ where $h()$ is a collision-resistant hash function that outputs numbers in $\mathbb{Z}_q$. It now holds that the first L ciphertexts will be associated with L independent s's, regardless of the relative order of arrival at the servers. (Ciphertext c will be associated with the value $s_c = H(0, h(c))$. Using bivariate polynomials for related purposes is common in the literature. The use here is similar to the one in [32].

This method does not reduce the memory requirements from the servers, since each H_i has $L + 1$ coefficients. Furthermore, our proof of security against *dynamic* adversaries does not go through when this method is used. (Security against static adversaries remains unchanged.)

In an alternative method (that allows the proof against dynamic adversaries to go through) the servers use a universal hash function h (not cryptographic, just avoiding collisions with high probability) to map the ciphertext c to an index i. Once an s_l has been utilized, erase it from the list.[8] Note that universal hash functions suffice here, as it is in the interest of the encrypting party to prevent collisions in hashed ciphertexts. However, only a fraction of the s's are used before collisions become frequent.

On pseudorandomly generated s's. The need to prepare in advance the s's and the o's (i.e., the shared random values and the shares of the value 0) is a drawback of our scheme. It raises an interesting open problem of whether it is possible to construct a non-interactive and efficient implementation of a threshold pseudorandom function (TPRF), namely a PRF family $\{f_k\}$ where the secret key k is shared by a number of servers so that the servers can jointly evaluate the function, yet the function remains pseudorandom to an adversary who may control a coalition of some of the servers. For our scheme, we would need in addition that the shares of the servers of $f_k(c)$ would correspond to the values of a degree-t polynomial whose free term is $f_k(c)$. If such function family would exist, then instead of pre-sharing the random s's, each server S_i will, given a ciphertext c, set s_i to be the ith share of $f_k(c)$. (The shares of the value '0' can be pseudorandomly generated using similar methods.)

In fact, a threshold pseudorandom generator (TPRG) will suffice for us and could possibly be easier to implement. In a TPRG suitable for our purpose, the seed to the generator would be shared among the servers. Each server would compute a point on a degree t random polynomial whose free term is the ith output block of the generator.

[8] In the event that a c' arrives s.t. $h(c') = i$ for an s_i that was previously used and erased, the server alerts the user to replace c' with c'' (a perturbed c') and $s_{h(c'')}$ is used instead.

3.2 Achieving Robustness

This section deals with protecting against actively faulty decryption servers. Notice that since scheme T-CS is non-interactive then actively faulty servers cannot help the adversary in compromising the secrecy of encrypted messages that were not explicitly decrypted by the non-corrupted servers. The only damage that actively faulty servers can cause is denial of service. This is a lesser concern than secrecy, and in particular can usually be dealt with using external methods, such as notifying a higher-layer protocol or an operator. Still, we describe three methods for dealing with such active faults, as sketched in the Introduction.

Local error correcting. The first method uses the fact that, as long as $t < \frac{n}{3}$, the correct value f_0 is uniquely determined. This holds even if up to t of the f_i's are arbitrary elements in $\mathbb{Z}_q$: Let $Q()$ be the polynomial defined in Equation (2). Then at least $n - t$ of the partial decryptions $f_1 ... f_n$ satisfy $f_i = g_1^{Q(i)}$. Furthermore, there exists only a single degree $2t$ polynomial that agrees with $n - t$ of the f_i's.

We describe below a method for finding $f_0 = g_1^{Q(0)}$. This method is efficient only when $t = O(\sqrt{n})$. We do not know how to efficiently find f_0 for larger values of t; this 'error correction in the exponent' is an interesting and general open problem with various applications for cryptographic protocols. In particular, standard error correction algorithms for Reed-Solomon codes [30,41], which work when the perturbed $Q(i)$'s are explicitly given, do not seem to work here.

Our simplistic method for finding the value $f_0 = g_1^{Q(0)}$ proceeds as follows. We first pick at random a set $G = \{f_{i_1} ... f_{i_d}\}$ of $d = 2t + 1$ f_i's, and check its validity using the appropriate Lagrange coefficients. That is, let $\lambda_1^x ... \lambda_d^x$ be such that $P(x) = \sum_{k=1}^{d} \lambda_k^x P(i_k)$ for all polynomials $P()$ of degree $2t$. (These λ_k^x's are specific for x and for the set G.) Then, for each $j = 1..n$ we test whether $f_j = \Pi_{k=1}^{d} f_{i_k}^{\lambda_k^j}$. Say that s is valid if the test fails for at most t f_j's. We are guaranteed by the uniqueness of $Q()$ that if G is valid then letting $f_0 = \Pi_{k=1}^{d} f_{i_k}^{\lambda_k^0}$ yields the correct value. Furthermore, if S_i is uncorrupted (and thus $f_i = g_1^{Q(i)}$) for all $i \in G$ then G is valid.

We thus repeatedly choose random sets of size $2t + 1$ and check for validity. Each trial succeeds with probability $\Omega(e^{-2t^2/n})$. Thus when $t = O(\sqrt{n})$ we are guaranteed that a valid set G is found within a constant number of trials. (A similar argument is used in [2]). When n is small — as would be the case for practical applications — this method is quite efficient.

Interactive proofs of validity of partial decryptions. This method calls for the decrypting user to perform a (four-move) Zero Knowledge interaction with each of the servers to verify the validity of the partial decryptions. While making sure that neither corrupted servers nor a corrupted user gather more information (or, rather, more computational ability) than in the basic scheme (T-CS), these interactions guarantee that the user will almost never accept an invalid partial

decryption as valid. Once the interactions are done, the user interpolates f_0 as in the basic scheme, based on the acceded partial decryptions. We remark that the user need not always perform these interactions. It can first locally check validity (using terminology from the previous method) of the entire set $\{f_1...f_n\}$ and interact with the servers only if $\{f_1...f_n\}$ is found invalid.

We use standard techniques for discrete-log based ZK proofs of membership and knowledge [21,14,15,40,16,8]. First the following verification information is added to the public key. (We stress that this information is not needed for encrypting messages; it is used only by the decrypting users.) For each server S_i and each $l = 1..L$ we add:

$$g_1^{z_i},\ g_1^{s_{l,i}},\ g_2^{s_{l,i}},\ g_2^{o_{l,i}}.$$

Now, given the lth ciphertext (u_1, u_2, e, v) server S_i sends to the decrypting user, along with the partial decryption f_i (computed as in T-CS), also the values $\tilde{u}_1 = u_1^{s_{l,i}}$ and $\tilde{u}_2 = u_2^{s_{l,i}}$. Next, S_i and the user U engage in the following interaction, whose purpose can be informally described as follows. Recall that $\alpha = H(u_1, u_2, e)$. Server S_i proves to U that:

1. $\log_{u_1} \tilde{u}_1 = \log_{g_1} g_1^{s_{l,i}}$ and $\log_{u_2} \tilde{u}_2 = \log_{g_2} g_2^{s_{l,i}}$.
2. S_i "knows" values w_1, w_2, w_3, w_4, w_5 and $x_{1,i}, x_{2,i}, y_{1,i}, y_{2,i}$, such that:
   (a) $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5 = f_i$
   (b) $w_1^{-1} = \tilde{u}_1^{x_{1,i}} \tilde{u}_2^{x_{2,i}}$ and $w_2^{-1} = \tilde{u}_1^{y_{1,i}\alpha} \tilde{u}_2^{y_{2,i}\alpha}$
   (c) $\log_{u_1} w_3 = \log_{g_1} g_1^{z_i}$
   (d) $\log_v w_4 = \log_{g_1} g_1^{s_{l,i}}$
   (e) $\log_{g_1} w_5 = \log_{g_2} g_2^{o_{l,i}}$.

The proof proceeds as follows. We describe the proof in two parts. These parts are performed in parallel. (In fact, U can use the same challenge for both proofs.) First, to prove item (1) a standard ZK proof of equality of discrete-logs [16] is performed:

1. U commits to a challenge $c \stackrel{R}{\leftarrow} \mathbb{Z}_q$.[9]
2. S_i chooses $r_1, r_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and sends $b_1 = u_1^{r_1}, b_2 = g_1^{r_1}, b_3 = u_2^{r_2}, b_4 = g_2^{r_2}$ to U.
3. U de-commits to c
4. S_i sends $a_1 = r_1 + cs_{l,i}$ and $a_2 = r_2 + cs_{l,i}$ to U.
5. U accepts if $u_1^{a_1} = \tilde{u}_1^{cb_1}$ and $g_1^{a_1} = g_1^{s_{l,i}cb_2}$ and $u_2^{a_2} = \tilde{u}_2^{cb_3}$ and $g_2^{a_2} = g_2^{s_{l,i}cb_4}$.

The above interaction consists of two [16] proofs, that use the same challenge. It can be seen that using the same challenge does not significantly increase the probability of error for the user.

Next, to show item (2) above, server S_i and user U engage in the following interaction (which is a combination of the above proof of equality of discrete logs, and a proof of "knowledge of a representation" from [14,15] (we use the formulation of [8]).

[9] Specifically, we use the Pedersen commitment scheme [36]. Here the parties may use two predetermined generators g, h of the subgroup of size q in $\mathbb{Z}_p^*$. The user commits to c by sending $g^c h^s$ for a randomly chosen s in $\mathbb{Z}_q$.

1. U commits to a challenge $c \stackrel{R}{\leftarrow} \mathbb{Z}_q$, as before.
2. S_i chooses $r_1 ... r_7 \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and sends $b_1 = \tilde{u}_1^{r_1}\tilde{u}_1^{r_2}$, $b_2 = \tilde{u}_2^{r_3\alpha}\tilde{u}_2^{r_4\alpha}$, $b_3 = u_1^{r_5}$, $b_4 = g_1^{r_5}$, $b_5 = v^{r_6}$, $b_6 = g_1^{r_6}$, $b_7 = g_1^{r_7}$, $b_8 = g_2^{r_7}$ to U.
3. U de-commits to c.
4. S_i sends $a_1 = r_1 + x_{1,i}c$, $a_2 = r_2 + x_{2,i}c$, $a_3 = r_3 + y_{1,i}c$, $a_4 = r_4 + y_{2,i}c$, $a_5 = r_5 + z_i c$, $a_6 = r_6 + s_{l,i}c$, $a_7 = r_6 + o_{l,i}c$ to U.
5. U accepts f_i if $g_1^{a_5} = g_1^{z_i c b_4}$ and $g_1^{a_6} = g_1^{s_{l,i} c b_6}$ and $g_2^{a_7} = g_2^{o_{l,i} c b_8}$ and

$$\tilde{u}_1^{a_1}\tilde{u}_2^{a_2}\tilde{u}_1^{a_3\alpha}\tilde{u}_2^{a_4\alpha}u_1^{a_5}v^{a_6}g_1^{a_7} = f_i^c b_1 b_2 b_3 b_5 b_7. \quad (3)$$

The above interaction combines three [16] proofs of equality of discrete logarithms with two [14,15] proofs of knowledge of representation. In addition to using the same challenge, here the verifier's acceptance conditions of the five proofs are combined in a single product (3). This allows the verifier to check the validity of the product f_i without knowing the individual w_i's. Correctness of this interaction is based on the fact that if S_i 'knows' representations $w_1^{-1} = \tilde{u}_1^{x_{1,i}}\tilde{u}_2^{x_{2,i}}$ and $w_2^{-1} = \tilde{u}_1^{y_{1,i}\alpha}\tilde{u}_2^{y_{2,i}\alpha}$ then the values $x_{1,i}, x_{2,i}, y_{1,i}, y_{2,i}$ must be the ones from S_i's secret key (otherwise a knowledge extractor for S_i can be used to find the index of g_2 w.r.t. g_1).

User U decides that f_i is valid if it accepted f_i in both of the above interactions. Finally, U proceeds to compute f_0 and m based on the valid f_i's, as in the basic scheme. Let I-CS denote this interactive variant of T-CS.

Theorem 2. *If the DDH assumption holds then* I-CS *is a t-robust threshold cryptosystem for any* $t < \frac{n}{2}$.

The proof combines the simulation technique from the proof of Theorem 1 with the proofs of the protocols of [16,14,15]. We omit details from this version. (Here we only withstand static adversaries.) We remark that the protocols described here do not withstand *asynchronously concurrent* interactions between a corrupted user and the servers. This problem can be solved once general mechanisms for efficiently dealing with the concurrency problem are provided.

Acknowledgments

We thank Rosario Gennaro, Oded Goldreich, Shai Halevi, Tal Rabin, Omer Reingold, Ronitt Rubinfeld, Victor Shoup and Madhu Sudan for helpful discussions. The first author expresses special thanks to Rosario and Tal for very instructive tutorials. We also thank Moni Naor and Benny Pinkas for pointing out the bivariate-polynomial solution to the synchronization problem among the servers.

References

1. D. Beaver, "Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority", J. Cryptology (1991) 4: 75-122.

2. D. Beaver and J. Feigenbaum, "Hiding instances in multi-oracle queries", *STACS,* 1990.
3. D. Beaver and S. Haber, "Cryptographic Protocols Provably secure Against Dynamic Adversaries", *Eurocrypt,* 1992.
4. M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, "Relations among notions of security for public-key encryption schemes", *CRYPTO '98,* 1998, pp. 26-40.
5. M. Bellare and P. Rogaway, "Optimal Asymmetric Encryption", *Eurocrypt '94 (LNCS 950),* 92-111, 1995.
6. M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th STOC,* 1988, pp. 1-10.
7. M. Blaze, J. Feigenbaum and M. Naor, "A formal treatment of remotely keyed encryption", *Eurocrypt '98,* 1998, pp. 251-165.
8. S. Brands, "An efficient off-line electronic cash system based on the representation problem", CWI TR CS-R9323, 1993.
9. R. Canetti, "Security and composition of multi-party protocols", Available at the Theory of Cryptography Library, `http://philby.ucsd.edu`, 1998.
10. R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th STOC,* 1996. Fuller version in MIT-LCS-TR #682, 1996.
11. R. Canetti and A. Herzberg, "Maintaining security in the presence of transient faults", *CRYPTO'94*, 1994.
12. R. Canetti and S. Goldwasser, "A threshold public-key cryptosystem secure against chosen ciphertext attacks", Available at the Theory of Cryptography Library, http://philby.ucsd.edu, 1999.
13. D. Chaum, C. Crepeau, and I. Damgard, "Multi-party Unconditionally Secure Protocols", *20th STOC,* 1998, pp. 11–19.
14. D. Chaum, E. Everetse and J. van der Graaf, "An improved protocol for demonstrating possession of discrete logarithms and some generalizations", *Eurocrypt '87,*, LNCS 304, 1987, pp. 127-141.
15. D.Chaum, A. Fiat and M. Naor, "Untraceable electronic cash", *CRYPTO '88,* LNCS 403, 1988, pp. 319-327.
16. D. Chaum and T. Pedersen, "Wallet databases with observers", *CRYPTO '92,* 1992, pp. 89-105.
17. R. Cramer and V. Shoup, "A paractical public-key cryptosystem provably secure against adaptive chosen ciphertext attack", *CRYPTO '98,* 1998.
18. Y. Desmedt and Y. Frankel, "Threshold cryptosystems", *Crypto '89 (LNCS 435)*, 1989, pages 307–315.
19. D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *23rd STOC,* 1991.
20. P. Feldman, "A practical scheme for non-interactive Verifiable Secret Sharing", *28th FOCS,* 1987, pp. 427-437.
21. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", *CRYPTO'86 (LNCS 263),* 186-194, 1986.
22. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems", *these proceedings.*
23. J. Garay, R. Gennaro, C. Jutla and T. Rabin, "Secure Distributed Storage and Retrieval" Proceedings of 11th International Workshop on Distributed Algorithms (WDAG97) Lecture Notes in Computer Science 1320, pp. 275-289, 1997.
24. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust and efficient sharing of RSA functions, *CRYPTO '96,* 1996, pp. 157-172.

25. R. Gennaro and V. Shoup, "Securing threshold cryptosystems against chosen ciphertext attack", *Eurocrypt '98,* 1998, pp. 1-16.
26. O. Goldreich, S. Micali and A. Wigderson, "How to Play any Mental Game", *19th STOC,* 1987, pp. 218-229.
27. S. Goldwasser, and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO,* 1990.
28. S. Goldwasser and S. Micali, "Probabilistic encryption", *JCSS,* Vol. 28, No 2, April 1984, pp. 270-299.
29. A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung "Proactive Secret Sharing or: How to Cope with Perpetual Leakage", *CRYPTO '95*, LNCS 963, 1995. pp. 339–352.
30. F. J. Macwiliams and N. J. A. Sloane, *"The Theory of Error Correcting Codes"*, North-Holland, 1977.
31. S. Micali and P. Rogaway, "Secure Computation", unpublished manuscript, 1992. Preliminary version in *CRYPTO 91.*
32. M. Naor, B. Pinkas, and O. Reibgold "Distributed Pseudo-random Functions and KDCs", *these proceedings.*
33. M. Naor and M. Yung, "Public key cryptosystems provably secure against chosen ciphertext attacks", *22nd STOC,* 427-437, 1990.
34. R. Ostrovsky and M. Yung. "How to withstand mobile virus attacks". *10th PODC,* 1991, pp. 51–59.
35. T. Pedersen, "A threshold cryptosystem without a trusted party", *Eurocrypt '91,* 1991, pp. 522-526.
36. T. Pedersen. Distributed provers with applications to undeniable signatures. *Eurocrypt '91*, 1991.
37. T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multi-party Protocols with Honest Majority", *21st STOC,* 1989, pp. 73-85.
38. C. Rackoff and D. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack", *CRYPTO '91,* 1991.
39. A. Shamir. "How to Share a Secret", *Communications of the ACM*, 22:612–613, 1979.
40. C. Schnorr, "Efficient signature generation by smart cards", *J. Cryptology* 4:161-174, 1991.
41. M. Sudan, "Algorithmic issues in coding theorey", *17th Conf. on Foundations of Software Technology and Theoretical Computer Science*, Kharapur, India, 1997. Available on-line at `theory.lcs.mit.edu/~madhu/`