# Space Efficient Computational Multi-Secret Sharing and Its Applications

**Abstract.** In a $(t_1, \ldots, t_\ell)$-*multi-secret sharing scheme* (MSSS), $\ell$ independent secrets $s_1, \ldots, s_\ell$ are shared with $n$ parties in such a way that at least $t_i$ parties are required to recover the secret $s_i$ (while $s_i$ remains hidden with fewer shares). We consider the problem of minimizing the share size of MSSS in the challenging setting when there are many secrets to be shared among many parties. To circumvent the information-theoretic lower bound (e.g., Blundo [4]), we focus on the computational setting. A simple generalization of computational secret sharing (Krawczyk [17]) to multi-secret sharing yields a scheme with share size/overhead scaling linearly in $\ell$, the total number of secrets.

To beat this linear scaling, we consider constructing MSSS based on a related notion of encryption—dynamic threshold public key encryption (DTPKE)—that enables a sender to dynamically specify a threshold for each ciphertext. None of the existing DTPKE is well-suited for our purpose. Thus, we propose a new construction of a dynamic threshold public key encryption scheme with improved efficiency characteristics. We then give a recursive application of our construction that yields an efficient MSSS with share size only logarithmic in the number of secrets (thus effectively $O(\log \ell)$ as in the common cases, where $\ell, n$ are polynomially related).

Finally, we describe an application of our space efficient $(1, 2, \ldots, n-1)$-MSSS to a special tool called gradual verifiable secret sharing which is the fundamental building block for general multiparty computation (MPC) with $n$ players that provides fairness without honest majority.

## 1 Introduction

Many cryptographic systems rely on a trusted authority that maintains and employs a secret to carry out basic tasks such as encryption, decryption, or key generation. Reliance on a single authority, however, raises immediate security and reliability concerns. Secret sharing is the classical cryptographic primitive that can be used to overcome these obstacles. Given a set of $n$ parties and a collection $\mathcal{A}$ of subsets of the parties, a *secret sharing scheme* for $\mathcal{A}$ is a method of distributing shares of a secret among the parties so that any subset of parties from $\mathcal{A}$ can reconstruct the secret, but any subset of parties which is not in $\mathcal{A}$ cannot. Such schemes were introduced and studied by Blakley [3] and Shamir [25] for threshold structures: in such "threshold schemes," any subset of the parties can recover the secret precisely when the cardinality of that particular subset is above a certain threshold. Ito et al. [21] later generalized the notion to more general access structures.

A fundamental efficiency concern for secret sharing schemes is the size of the share that must be distributed to each participant. Karnin [16] showed that in any unconditionally secure secret sharing scheme, the length of each share must be at least that of the secret. To achieve better efficiency, Krawczyk [17] relaxed the security guarantee by introducing a natural computational notion: while $t$ shares can be used to efficiently recover the secret, any fewer than $t$ give "no information" (in a precise sense) about the secret to a computationally bounded adversary. This permits striking improvements in share size: each share has size $(m/t + \lambda)$ instead of $m$, where $m$ is the size of the shared secret and $\lambda$ is the security parameter.

An important and well-studied variant arises when multiple secrets must be distributed with different thresholds to the parties [1,4,26,27]; such schemes are known as *multi-secret sharing schemes (MSSS)*. Jackson et al. [15] considered the most natural case—called multi-secret threshold schemes—in which each secret $s_i$ is associated with a threshold $t_i$, and only more than $t_i$ participants can recover $s_i$ from their shares. Blundo et al. [4] established that in order to achieve unconditional security for multi-secret threshold schemes, the size of the share that each participant holds must be, at least, linear in the number of secrets. These lower bounds have been complemented by more recent work in the unconditional setting: Masucci [22] presented a weaker notion of security for multi-secret sharing schemes in unconditional settings, and gave some lower bounds for the size of the shares of each participant. Herranz et al. [12] then proved that the size of the share for each participant in multi-secret threshold schemes must indeed be linear in the number of secrets, even if the scheme enjoys only the weaker unconditional security introduced by [22].

Motivated by Krawczyk's success in the single secret case, computationally secure MSSSs can be considered for improved efficiency [12]. However, straightforward application of Krawczyk's scheme [17] for a MSSS still incurs a linear (in the number of secrets) size share for each player: e.g., in [12], $\ell$ instances of [17] are run and the resulting share size is $m(1/t_1 + \cdots + 1/t_\ell) + \ell\lambda$ (if there are $\ell$ secrets, and $\lambda$ denotes the security parameter, and $m$ denotes the shared secret size). The first term is clearly unavoidable; however, the second term yields an overhead that grows linearly in the number of secrets. In many applications, the number of shares can be very large. This raises our central question:

*Is there a computationally-secure multi-secret sharing scheme providing share size that grows sub-linearly with the number of shares?*

In this paper, we answer this question in the affirmative: we give a new MSSS construction that achieves overhead that scales only logarithmically in $\ell$. Aside from their intrinsic interest, we remark that such efficient MSSSs that do not rely on an external public storage (see below for more elaborations).

**The interesting cases when both $\ell, n$ are $\mathsf{poly}(\lambda)$.** For multi-secret sharing, there are two simple cases; the total number of user $n$ is small (however large is the number of shares $\ell$), or the total number of shares $\ell$ is small. In the first case, it follows that the range of all different thresholds would be small (smaller than

$n$). It is easy to see that we can simply group secrets with the same threshold into one bigger secret chunk, and run the trivial application of Krawczyk's scheme [17] on the chunks, which yields an MSSS with each share size overhead at most $n\lambda$ for a very small $n$. Similarly, we can directly apply the Krawczyk's scheme [17] for the latter case, and the resulting $\ell\lambda$ overhead is fine now as $\ell$ is small.

For many of the interesting cases, the number of shares and number of parties are both significantly large. In particular, we will consider a general setting where both $n, \ell$ are $\mathsf{poly}(\lambda)$. For example, in our multiparty computation application, we will need to consider a $(1, 2, 3, \ldots, n)$-MSSS (here $\ell = n$). We are aiming to achieve an asymptotically small overhead for each share size ($O(\log \ell) \cdot \lambda$). And note that in this case, $O(\log n)$ is the same as $O(\log \ell)$, thus sometimes in the paper, we used $n, \ell$ interchangeably for measuring space efficiency.

**Regarding a public bulletin board.** As a counterpoint to our results, Herranz et al. [12] observe that in the computational setting one can "shift" private storage to public storage via encryption. In particular, an information theoretic MSSS can be used to generate the shares first, which are then encrypted and stored on a public bulletin board. Thus, each player must only store a short key locally and retrieve all other material from the bulletin board. In this paper, we focus on constructing a space efficient MSSS without relying a public "bulletin board". There are a couple of obvious reasons that motivate us: reducing trust for space efficient MSSS scheme (without needing an online party who faithfully holds all the data). Moreover, sometimes such public bulletin board service may be impreferable or even unavailable as in case of common deployment of secure multiparty computation or decentralized applications. We remark, also, that even though we focused on the setting without a bulletin board, our techniques can actually provide efficiency savings also in the setting of having a public bulletin board. In particular, treating public storage as outsourced storage associated with each player, existing schemes use as much storage as in the information theoretic case ($O(m \cdot \ell)$ for each player at least [12]), while applying our technique achieves $O(m \cdot \log \ell)$ for each player. See the efficiency discussion in section 4.2 for more details. [1]

## 1.1 Our Results

We present a new construction for threshold MSSS (in this paper, we focus entirely on threshold MSSSs, so we drop the adjective "threshold") that achieves share size $O(m(1/t_1 + \cdots + 1/t_\ell) + \log \ell \cdot \lambda)$ which is nearly optimal for large $\ell$. [2] As

[1] One may wonder whether simply applying the tool of information dispersal (IDS) on the data stored in the public bulletin board solves the problem, However, doing so is highly non-trivial. As we will show later, in order to make the IDS effective, we first need to carefully design the scheme so that the public bulletin board size scales only linear with each threshold, not always $\ell$ as in [12].

[2] Note that there are three possible scenarios for an MSSS: few secrets to be shared that can easily be realized using an efficient single secret sharing scheme, few people

the first term is inherent for carrying the information of the message, the second term is the overhead that we try to minimize. Concretely, our contributions are as follows:

**Construction of space efficient computational multi-secret sharing.** We begin with a generic construction, providing the basic relationship between MSSS and DTPKE, and then focus on efficient constructions of DTPKEs. In more detail:

*A generic construction.* To warm up, we first point out a generic construction for computational MSSS. Inspired by [17], we use an encryption scheme as a building block. To capture the various threshold requirements, we rely on a *dynamic threshold public key encryption scheme (DTPKE)* [6]. Intuitively, given the secrets $(s_1, .., s_\ell)$, we apply a DTPKE scheme to encrypt each secret $s_i$ with its threshold $t_i$; thus, the resulting ciphertext $c_i$ requires at least $t_i$ users in order to be decrypted. Then, following [17], we apply an efficient *information dispersal scheme (IDS)* [17, 23] to distribute the ciphertexts together with all the public keys among users in a way that $t_i$ users can construct $c_i$ from their shares. This yields a generic construction that transforms a DTPKE scheme into an MSSS. The resulting construction is computationally secure if the given DTPKE is semantically secure. This simple construction enables us to focus on the challenge of designing an efficient DTPKE.

*A new dynamic threshold public key encryption scheme.* The generic construction is useful only when it is applied to an efficient instantiation of DTPKE, and obtaining such a scheme appears to be challenging. The only existing construction that offers a constant-size ciphertext for DTPKE is given by [6] (without this property the encryption itself already precludes the possibility of our efficiency target). However, the scheme requires $O(n)$ public group elements, called the *combining key*, and all elements in the combining key are used even for decrypting a ciphertext with the threshold 1 (here $n$ is the number of users). This immediately prevents the above reduction from yielding an efficient MSSS. Requiring some kind of combining key seems to be unavoidable in the construction of DTPKE with small ciphertexts and, moreover, this particular inefficiency appears to be an inherent aspect of the existing system.

Taking all these facts into consideration, we propose a new DTPKE scheme that has an allowed *threshold range* in the parameters. We then show how to apply a degree reduction technique on the exponent that can be tuned to the threshold range to yield our new construction. The result is a combining key that scales linearly only in the size of the threshold range rather than the number

to share the secrets for which the information dispersal algorithms will not be useful since the threshold values are small, and many secrets to be shared among many users that we consider in this work.

of users $n$.[3] Specially, for a particular threshold range $[t, 2t)$, if the combining keys are distributed using an information dispersal scheme for the threshold $t$, then the size of the share will be constant. Moreover, the structure of our new scheme is algebraically simpler than the original scheme of [6]. These features enable us to recursively apply our general construction to get an efficient MSSS. Finally, we remark that the basic technical structure of this new DTPKE is rather different from the one in [6] which might be of independent interest.

*A space efficient construction of MSSS.* Direct application of our new DTPKE still cannot give us an efficient MSSS. As we emphasized above, we need to distribute all combining keys among the participants. Evenmore, since all components of the combining key are required in decrypting even a ciphertext with threshold 1, all combining keys should be given to each participant. It is clear that, even with our new construction, applying our general reduction naively results in a linear size share for each participant.

To circumvent this obstacle, we divide the threshold range into $\log n$ parts: $\mathit{\Delta}_1, \ldots, \mathit{\Delta}_{\log n}$, where $\mathit{\Delta}_u = [2^{u-1}, 2^u)$ (w.l.o.g, we assume $n$ is a power of 2). We then run independent DTPKE instances together with an IDS for each threshold range $\mathit{\Delta}_u$ in order to share the secrets $\{s_i \mid t_i \in \mathit{\Delta}_u\}$. For each window $\mathit{\Delta}_u$, there are at most $2^u$ elements for the combining key, while the minimum threshold for this range is $2^{u-1}$. Thus we can now apply the information dispersal scheme [17,23] to efficiently distribute the combining key; thus each share has size $O(1)$. Since both $n, \ell$ are $\mathsf{poly}(\lambda)$, the final share size is $O(\log \ell)$, where $\ell$ is the number of secrets.

**Applications to fair MPC with smaller private state.** We show an interesting application of our MSSS to get an efficient building block for MPC protocol that guarantees fairness when there is no honest majority. Hirt et al. [13] introduced a primitive called gradual verifiable secret sharing as the main building block of an MPC protocol that first splits a secret $s$ into $n-1$ pieces $s_i$, i.e., $s = \sum_{i=1}^{n-1} s_i$, and then jointly reveals $s_i$ using a secret sharing scheme with threshold $i$ one by one. Thus, the secret sharing scheme prevents the adversary who aborts the protocol to learn the output. This primitive, in our terminology, is a $(1, 2, \ldots, n-1)$-MSSS in which the shares are reconstructed one by one. Instantiating this primitive using our efficient MSSS scheme, yields an MPC protocol satisfies fairness with each party having only $O(\log n \cdot \lambda)$ storage for the gradual VSS part, while [13] only achieves linear size $O(n\lambda)$.

**Paper overview.** We recall the basic definitions and security requirements in Section 2. To establish the basic relationship between MSSS and DTPKE, we propose a generic construction for the computational MSSS in Section 3. In Section 4, we first present a new construction of DTPKE since non of the existing

[3] Note that this change of having an allowed range is a strict generalization of the previous definition of DTPKE rather then a weakening. To see this, we can simply set the allowed range to be $[1, n]$ to instantiate the previous schemes, e.g., [6].

construction is well-suited for our purpose, then we propose an efficient MSSS that employs the new construction of DTPKE. We also show an application of our efficient MSSS to secure multi-party computation in Section 5.

## 2 Preliminaries

**Notations:** Let $\ell$ denote the number of secrets to be shared and $n$ denote the number of users, which will also be the upper bound for the threshold values. We use $(s_1, ..., s_\ell)$ to denote the secrets to be shared, and the letter $t$ to denote threshold values, i.e. each $t_i$ is associated with the secret $s_i$. The letter $m$ is used to denote the size of the secret. We let $\lambda$ denote the security parameter.

### 2.1 Definitions

In this section, we give some definitions and the security requirements of the multi-secret threshold schemes.

**Definition 1.** *A $(t_1, \ldots, t_\ell)$-multi-secret threshold sharing scheme is defined by two protocols as follows:*

- ***Share:** This protocol takes the security parameter $\lambda$, the set of $n$ players $\mathcal{P}$, the global secret $\overrightarrow{s} = (s_1, ..., s_\ell)$ to be shared, and $\ell$ threshold values $(t_1, ..., t_\ell)$ where $t_i \in [n]$. It outputs the set of shares $\{sh_i\}_{P_i \in \mathcal{P}}$.*
- ***Reconstruct:** This is a protocol executed among a subset of users: each user $i$ takes the index $j \in [\ell]$ and his secret share $sh_j$ as input, and they jointly output the corresponding secret $s_j$ or $\perp$.*

*Remark.* We assume that the share protocol also outputs some public parameters required for the reconstruction protocol (such as description of the underlying group system). For simplicity, we consider these as part of share for each user.

A multi-secret sharing scheme has to satisfy the following two properties:

*Correctness:* Enough shares are able to reconstruct the secret. Formally, if $S = \{sh_{i_k}\}_{k=1}^{t_j} \subseteq \{sh_i\}_{P_i \in \mathcal{P}}$ where $\text{Share}(\lambda, \mathcal{P}, \overrightarrow{s}, \{t_i\}) = (\{sh_i\}_{P_i \in \mathcal{P}})$, then

$$s_j \leftarrow \text{Reconstruct}(j, \{sh_{i_k}\}_{k=1}^{t_j}).$$

However, if $|S| < t_j$, then $\text{Reconstruct}(j, S) = \perp$.

*Security*: We define the computational security of a multi-secret threshold sharing scheme similar to the regular security notion of semantic security that is used in encryption schemes. Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- The challenger gives the set $\mathcal{P}$ of users and the thresholds values $(t_1, ..., t_\ell)$ to the adversary.

- The adversary $\mathcal{A}$ submits the challenge threshold $t_i$.
- The adversary also submits two global secrets $\overrightarrow{s}^{(0)}, \overrightarrow{s}^{(1)}$ where $s_i^{(0)} \neq s_i^{(1)}$ and $s_j^{(0)} = s_j^{(1)}$ , $\forall j \neq i$.
  For simplicity, we just require the adversary to submit two global secrets $\overrightarrow{s}^{(0)}, \overrightarrow{s}^{(1)}$ such that they differ only at the index $i$. It can be easily seen to imply the general case.
- The challenger chooses a random bit $b \in \{0,1\}$, runs the share protocol $(\{sh_k\}_{P_k \in \mathcal{P}}, params) \leftarrow \text{Share}(\overrightarrow{s}^{(b)}, \mathcal{P}, \{t_k\}, \lambda)$, and gives $(\{sh_k\}_{k<t_i}, params)$ to the adversary.
- Finally, $\mathcal{A}$ outputs a guess $b'$.

The advantage of the adversary in breaking the security of the scheme is determined as

$$\mathsf{Adv}_{\mathcal{A}}^{\text{MSSS}} = |\Pr[b' = b] - 1/2|.$$

We say that the scheme is computationally secure if $\mathsf{Adv}_{\mathcal{A}}^{\text{MSSS}}$ is negligible.

We require, also, the notion of an information dispersal scheme, defined below.

**Definition 2 (Rabin [23]).** *A $(t,n)$-information dispersal scheme is a protocol that distributes a file $F$ among $n$ parties in such a way that the recovery of the file is possible in the presence of up to $(n-t)$ failed (inactive) parties at the time of reconstruction.*

Information dispersal is very like secret sharing without the security requirement. Indeed, the file is transformed into $n$ "shares" with the property that any $t$ can entirely reconstruct the file. The major parameter of interest is the size of the "shares." The protocol given by [23] achieves optimal share size of length $|F|/t + O(1)$.

### 2.2 Multi-Sequence of Exponents Diffie-Hellman Assumption

We here set down the cryptographic assumption—essentially identical to the assumption used in [6]—on which the security of our new construction for the dynamic threshold public key encryption is based.

**Definition 3.** *Let $G_1$, $G_1$, and $G_T$ be three groups of prime order $p$. We say a map $e : G_1 \times G_2 \rightarrow G_T$ is a* bilinear *map if it satisfies the following properties:*

- *Bilinearity. For all $a, b \in Z_p$, $g_1 \in G_1$, and $g_2 \in G_2$; $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$,*
- *Non-degeneracy. If $g_1$ and $g_2$ are generators for $G_1$, $G_2$, respectively, then $e(g_1, g_2)$ is a generator for $G_T$,*
- *Efficiency. There exists an efficient algorithm [8] to compute $g, h \mapsto e(g,h)$ for elements $g, h \in G$.*

A bilinear map group system is a tuple $\mathcal{B} = (p, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, e)$ that consists of elements as described above.

**Definition 4.** *The decisional problem $(\ell, n, t)$-MSE-DDH is defined as follows: let $\mathcal{B} = (p, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, e)$ be a bilinear group systems, and $\ell$, $n$, and $t$ be three integers. Let $g_0$ be a generator of $\mathcal{G}_1$ and $h_0$ be a generator of $\mathcal{G}_2$. Given two random coprime polynomials $f$ and $g$, of respective orders $\ell$ and $t$, with the roots $y_1, \ldots, y_\ell$ and $x_1, \ldots, x_t$ respectively, where $x_i \neq y_j$, and several sequences of exponentiations*

$$\begin{array}{ll} g_0, g_0^{\gamma}, \ldots, g_0^{\gamma^{\ell+n-1}}, & g_0^{k.\gamma.f(\gamma)}, \\ g_0^{\alpha}, g_0^{\alpha.\gamma}, \ldots, g_0^{\alpha.\gamma^{\ell+n-1}}, & \\ h_0, h_0^{\gamma}, \ldots, h_0^{\gamma^{n-1}}, & \\ h_0^{\alpha}, h_0^{\alpha.\gamma}, \ldots, h_0^{\alpha.\gamma^{n-1}}, & h_0^{k.g(\gamma)} \end{array}$$

*and also $T \in \mathcal{G}_T$, decide whether $T$ is equal to $e(g_0, h_0)^{k.f(\gamma)}$ or some random element of $\mathcal{G}_T$ where $k, \alpha, \gamma \in Z_p^*$.*

Delerablee et al. [6] justified the intractability of the decisional problem $(\ell, n, t)$-MSE-DDH in the generic group model. As they point out, it is non-interactive and falsifiable.

## 3 Generic Construction of Computational Multi-Secret Sharing Schemes

To circumvent the lower bound restriction in the perfect secret sharing scheme [16], Krawczyk [17] proposed a computational $(t, n)$-secret sharing for one secret. Specifically, after encrypting the secret, the ciphertext is distributed via an information dispersal scheme to the users, while a regular Shamir secret sharing scheme is applied to distribute the secret key. It can achieve essentially optimal share size: $m/t$, where $m$ is the size of the data to be shared and $t$ is the threshold value. As a secret key is only $\lambda$ bits long, the final share size for each user is only $m/t + \lambda$. This elegant framework inspires us to consider the efficiency benefits of a computational notion of security in the setting of multi-secret sharing as well.

A natural way to generalize Krawczyk's idea to multi-secret sharing (assuming there are $\ell$ independent secrets to share) is to run the Krawczyk scheme $\ell$ times. Briefly, each secret can be encrypted under a different key. Then, the resulting ciphertexts will be dispersed as in [17], and $\ell$ independent Shamir secret sharing schemes are then carried out to distribute the secret keys. Thus, the total size will be $\sum_{i=1}^{\ell}(m_i/t_i + \lambda) = \sum_{i=1}^{\ell} m_i/t_i + \ell\lambda$. This strategy was observed in [11] and analyzed there in detail. As discussed in the introduction, our goal in this paper is to construct a multi-secret sharing scheme with at most sub-linear overhead in the number of secrets. Furthermore, we adopt a model where there is no available public bulletin board.

The above strategy inherently requires overhead linear in $\ell$, as each secret and key are treated independently. We need to deviate from this natural construction and generalize Krawczyk's idea in a fashion that meaningfully combines information across secrets. To realize this plan, we apply a generalized version of (threshold) encryption scheme so that the ciphertext can be created with flexible thresholds while each user has only a single secret key.

We first give the formal definition of such a dynamic threshold public key encryption (DTPKE), and describe a generic construction of MSSS using DTPKE as a building block. This generic construction will serve as the stepping stone for us to achieve an efficient MSSS.

### 3.1 Dynamic threshold public key encryption

The notion of dynamic threshold public key encryption (DTPKE) can be considered as a refinement of the regular threshold encryption scheme. In a DTPKE, each user/receiver has a secret key, and a sender can specify a threshold $t$ so that the ciphertext requires $t$ secret keys to jointly recover the plaintext. Moreover, the choice of the threshold $t$ is not fixed in advance—it can be determined during encryption by the sender. We present here a simplified definition which will be enough for our use in this paper. In more detail, a DTPKE is composed with the following algorithms:

- **D.Setup**$(\lambda)$**:** This algorithm takes a security parameter $\lambda$ as input, and generates a master secret key $MK$, an encryption key $EK$, and a combining key $CK$. It then outputs the public parameters $\mathit{params} = (EK, CK)$, and gives $MK$ to the registration authority.
- **D.KeyGen**$(MK, i)$**:** This algorithm takes $MK$ as input, and outputs the user's secret key $usk_i$ and the user's public key $upk_i$ for each user $i$.
- **D.Enc**$(EK, t, m)$**:** This algorithm takes the encryption key $EK$, a threshold $t$, and a message $M$ as inputs. It then outputs a ciphertext $C$.
- **D.ShareDecrypt**$(usk_i, C)$**:** It takes the user's secret key $usk_i$ and a ciphertext $C$ as inputs, and outputs a decryption share $\sigma_i$.
- **D.Combine**$(CK, T, \sigma_1, ..., \sigma_t, C)$**:** The algorithm takes the combining key $CK$, a ciphertext $C$, a subset $T$ of $t$ user public keys, and a list $(\sigma_1, ..., \sigma_t)$ of $t$ decryption shares as inputs. It either outputs a message $M$ or $\perp$.

Note that Delerablee et al. [6] included two more algorithms **ValidateCT** and **VerifyShare** as parts of usual DTPKE scheme. Since those algorithms are not essential for our purpose, and can easily be obtained using traditional techniques, we omit them here.

*Correctness*: The correctness of a dynamic threshold public key encryption scheme requires that for any ciphertext $C = D.Enc(EK, t, M)$ with a threshold $t$, if $t$ users correctly produced the partial decryption shares $\sigma_i$, then the **Combine** algorithm on the set $\{\sigma_1, ..., \sigma_t\}$ correctly outputs the message $M$. Formally,

$$\Pr[\mathbf{D.Combine}(CK, T, \{\sigma_i\}_{i \in [t]}, C) = M] = 1$$

if $C = \mathbf{D.Enc}(EK, t, M),$ and $\forall i \in [t], \sigma_i = \mathbf{D.ShareDecrypt}(usk_i, C)$.

*Security*: We here give the security requirements of the primitive. Briefly, the semantic security of the scheme guarantees that fewer than $t$ users cannot decrypt a ciphertext with the threshold $t$. Even if we allow an adversary to corrupt $t-1$ users, we claim that the content of the ciphertext with the threshold $t$ will still be semantically secure. Consider the following game between a P.P.T adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

- The adversary $\mathcal{A}$ submits a threshold $t$.
- The challenger simulates the **Setup** algorithm and gets the system parameters $(MK, CK, EK)$. It then gives the public parameters $\mathit{params} = (EK, CK)$ to the adversary.
- $\mathcal{A}$ is allowed to make corrupt queries. To respond the queries, the challenger runs **D.KeyGen** to get secret key-public key pairs and sends the corresponding keys $(upk_1, usk_1), \ldots, (upk_q, usk_q)$ to $\mathcal{A}$.
- $\mathcal{A}$ submits two messages $M_0$ and $M_1$ with the threshold $t$. The challenger randomly chooses $b \in \{0,1\}$, and sends $C_b$ as an encryption of $M_b$ to $\mathcal{A}$.
- $\mathcal{A}$ can adaptively make extra $q'$ corrupt queries.
- Finally, $\mathcal{A}$ outputs a guess $b'$ and wins the game if $b' = b$.

We require in the above game that $q + q' < t$. The advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{cpa} = \left| \Pr[b' = b] - \frac{1}{2} \right| .$$

We say a dynamic threshold public key encryption scheme is semantically secure if for any polynomial time adversary $\mathcal{A}$ the advantage $\mathsf{Adv}_{\mathcal{A}}^{cpa}$ is a negligible function of $\lambda$.

*Remark.* The adversary can also make "ShareDecrypt" queries, and get some decryption shares of a ciphertext for some particular users. However, the "Corrupt" queries give more advantages in comparison to ShareDecrypt queries, and even we allow the adversary to corrupt up to $t$ users in proving the security of the ciphertext with the threshold $t$. Thus, allowing only corrupt queries would be enough for the security definition.

### 3.2 Constructing MSSS from DTPKE.

We now show how to use the DTPKE to construct a MSSS that provides computational security. Note that this generic construction itself does not give efficiency guarantee, but it will be a critical component in the final construction. It will also allow us to treat the security analysis in a modular fashion.

First, consider an intermediate case where there exists a public bulletin board through which all data can be accessed by any user. Assume there are $\ell$ secrets $s_1, ..., s_\ell$ to be shared with corresponding thresholds $t_1, ..., t_\ell$. These secrets can simply be encrypted using the DTPKE with the corresponding thresholds, and the resulting ciphertexts—along with the combining keys—can be placed on the bulletin board. In addition, each user is given a secret key as a share. To reconstruct the secret $s_i$, $t_i$ users jointly decrypt the corresponding ciphertext and get the plaintext as the secret $s_i$. To construct an actual MSSS and remove the reliance on a bulletin board, we need to further distribute the bulletin board data among users. The ciphertexts can be distributed via an information dispersal scheme (IDS) without affecting the security. But all the combining keys are required to be able to decrypt even a ciphertext with the threshold 1. Thus, all of them should be given to each user.

Formally, given an information dispersal algorithm IDS, and a DTPKE scheme **D=(D.Setup, D.KeyGen, D.Enc**, **D.ShareDecrypt, D.Combine**), a multi-secret threshold sharing scheme can be obtained as follows:

**Share:** The algorithm takes the security parameter $\lambda$, the set of users $\mathcal{P} = \{1, \ldots, n\}$, the secrets $s_1, ..., s_l$ to be distributed, and the threshold values $\{t_i\}_{i \in [\ell]}$. It first calls **D.Setup**($\lambda$) to get the parameters $(MK, EK, CK)$.
It then runs **D.KeyGen** to get the secret keys and the public keys

$$\{(upk_j, usk_j)\}_{j=1,\ldots,n}$$

for the users.
The algorithm then encrypts each secret and computes $c_i = \mathbf{D.Enc}(EK, t_i, s_i)$ for the secrets $s_i$ using its threshold $t_i$.
Next, the algorithm distributes each $c_i$ by running an IDS on it using threshold $t_i$. Suppose the shares are $\{c_{i,j}\}_{i \in [\ell], j \in [n]}$.
This algorithm ends with each user $j$ receiving his key pair $(usk_j, upk_j)$, the combining key $CK$, and $\ell$ shares $(c_{1,j}, ..., c_{\ell,j})$ for ciphertexts.

**Reconstruct:** Assume there is a set $S$ of users that jointly run the protocol in order to get the corresponding secret $s_k$, and each user holds a share $(usk_j, upk_j, CK, \{c_{i,j}\})$ where $j \in S$, $i \in [\ell]$, and $|S| \geq t_k$. They first jointly reconstruct $c_k$ using the IDS reconstruct algorithm on the shares $\{c_{k,j}\}_{j \in S}$. Then each user in the set $S$ runs **D.ShareDecrypt** on $c_k$ to get a decryption share $\sigma_{k,j}$. Finally, they jointly compute the secret as $s_k = \mathbf{D.Combine}(\{\sigma_{k,j}\}, c_k, CK)$.

*Remark.* Regarding efficiency, the size of each user's share will be $|CK| + |usk| + |upk| + \sum_{i=1}^{l} |c_{i,j}|$. In order to guarantee sub-linear size share, we need to ensure the ciphertext size to be small. Furthermore, we demand a dynamic threshold encryption scheme such that either (i) the combining key of the scheme is constant or (ii) the structure of the scheme enables us to employ IDS on $CK$ such that each share size of $CK$ will be at most sub-linear.

*Correctness.* According to the IDS property, when $|S| \geq t_k$, the ciphertext $c_k$ can be recovered. Thus, following the correctness of the underlying DTPKE, $|S|$ secret keys together with the combing key can jointly decrypt $c_k$, and get the corresponding secret $s_k$.

*Security.* The security of the generic MSSS construction follows from the semantic security of the DTPKE scheme. Formally,

**Theorem 1.** *If* **D** *is a semantically secure DTPKE scheme, then the construction given above is a computationally secure MSSS.*

*Proof.* Assume there is an adversary that breaks the security of the multi-secret threshold scheme; then we can build an algorithm $\mathcal{B}$ that breaks the security of the scheme $\mathcal{D}$. Consider the following game between the algorithm $\mathcal{B}$ and the adversary $\mathcal{A}$.

- $\mathcal{B}$ submits the set of users $\mathcal{P}$ and $\ell$ threshold values $(t_1, \ldots, t_\ell)$.
- $\mathcal{A}$ submits the threshold $t_{i^*}$ and two global secrets $\overrightarrow{s}^{(0)}$, $\overrightarrow{s}^{(1)}$ where $\overrightarrow{s}^{(0)}$ and $\overrightarrow{s}^{(1)}$ differ only at the index $i^*$.
- The algorithm $\mathcal{B}$ invokes the challenger of the scheme $\mathcal{D}$ in order to get the public parameters $(EK, CK)$.
  $\mathcal{B}$ then makes corrupt queries for any $q$ users, and gets $q$ secret key-public key pairs $\{usk_j, upk_j\}_{j=1}^{q}$ from the challenger.
  $\mathcal{B}$ gives two secrets $s_{i^*}^{(0)}$ and $s_{i^*}^{(1)}$ together with the threshold value $t_{i^*}$ to the challenger, and receives the challenge ciphertext $c_{i^*}^{(b)}$ of the secret $s_{i^*}^{(b)}$.
  $\mathcal{B}$ makes extra corrupt queries for any $q'$ more users and gets $q'$ secret key-public key pairs $\{usk_j, upk_j\}_{j=q+1}^{q+q'}$ where $q + q' = t_{i^*} - 1$.
  $\mathcal{B}$ also computes the ciphertexts $c_i$ of the secrets $s_i^{(0)}$ for all $i \neq i^* \in [\ell]$. It then produces the shares $\{c_{i,j}\}_{i\in[\ell], P_j\in\mathcal{P}}$ for the ciphertexts

$$(c_1, \ldots, c_{i^*-1}, c_{i^*}^{(b)}, c_{i^*+1}, \ldots, c_l)$$

  by running IDS protocol.
  Finally, $\mathcal{B}$ gives all $(t_{i^*} - 1)$ shares $\{(usk_{j_k}, upk_{j_k}, CK, c_{i,j_k})\}_{i\in[\ell], k<t_{i^*}}$ to the adversary.
- The adversary $\mathcal{A}$ submits a guess $b'$. The algorithm $\mathcal{B}$ gives this guess $b'$ to the challenger of DTPKE as a guess for the challenge ciphertext $c_{i^*}^{(b)}$.

Assume $Adv_{\mathcal{A}}^{MSSS}$ is the advantage of $\mathcal{A}$ in breaking the security of the construction, and $Adv_{\mathcal{B}}^{DTPKE}$ is the advantage of $\mathcal{S}$ in breaking the security of DTPKE. Since $\mathcal{A}$' s view will be identical to a real attack, if $Adv_{\mathcal{A}}^{MSSS} > \epsilon$, then $Adv_{\mathcal{B}}^{DTPKE} \geq \epsilon$. Thus, we don't have any security loss.

### 3.3 Obstacles to efficient DTPKE

Conventional threshold encryption schemes insist on a fixed threshold determined when the system is initialized, e.g., [25]. It is not clear how to advantageously apply such a scheme in the MSSS case, as each threshold would—at least naively—require its own scheme. So, we consider the DTPKE schemes in which the threshold is not fixed during setup, i.e., the scheme allows one to dynamically set a threshold for each ciphertext. To the best of our knowledge, there is only one construction for DTPKE [6]. Unfortunately, it has a constant size ciphertexts but calls for a large combining key (linear in the number of users), so that we cannot directly use it for an efficient MSSS. Let us first recall the construction of [6] to approach the challenge of providing an efficient DTPKE.

The construction employs an arbitrary bilinear map group system. Let $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map, and $g$ and $h$ be two randomly selected generators for the groups $G_1$,$G_2$, respectively. In the scheme, the encryption key $EK$ consists of $(2n+2)$ group elements $(u, v, \{h^{\alpha\cdot\gamma^i}\}_{i=0}^{2n-1}, D)$ where $u = g^{\alpha\cdot\gamma}$, $v = e(g,h)^{\alpha}$, and D consists of $(n-1)$ random integers $(d_1, ..., d_{n-1})$ from $Z_p$;[4] and the combining key consists of $(n-1)$ group elements $(h, h^{\gamma}, ..., h^{\gamma^{n-2}})$ and $D$. The secret key of user $j$ is $g^{\frac{1}{\gamma+j}}$.

The encryption algorithm computes the header and the session key for a threshold $t$ of users as $(Hdr = (C_1 = u^{-k}, C_2 = h^{k\alpha P(\gamma)}), K = v^k)$

$$P(\gamma) = \prod_{x=1}^{n}(\gamma + x) \cdot \prod_{x \in D_1}(\gamma + x)\,,$$

where $D_1 \subset D$, and $|D_1| = t-1$. Note that for a threshold $t$, the algorithm adds $(t-1)$ dummy users to fix the degree of the polynomial $P(\gamma)$ as $(n + t - 1)$. Since the decryption procedure applies bilinear maps to $C_2$ and $usk$, a collection of $t$ secret keys can be used collectively to reduce the degree of $P(\gamma)$ by $t$ as they will have $t$ different $(\gamma + x_i)$ in the denominator of the exponent.[5] Thus, the combining key $\{h^{\gamma^i}\}_{i=0}^{n-2}$ will be enough to complete the decryption of the ciphertext. However, with fewer than $t$ secret keys, the polynomial in the exponent of $e(g,h)$ must have degree no smaller than $n$. In this case, the combining key will not be sufficient to complete the decryption of the given ciphertext since it is impossible to reconstruct $e(g,h)^{\gamma^n}$. This works for any $t$, which is the core idea allowing a dynamic threshold.

In the above construction, all the $(n-2)$ group elements of the combining key must be used in the decryption process, even if the threshold value $t$ of the ciphertext was set as 1. This fact creates an efficiency problem when one utilizes this DTPKE to instantiate our generic construction (of a MSSS) since all the $(n-2)$ group elements of the combining key must be given to each user. Using information dispersal to distribute the combining key (as it is public information anyway) can alleviate this problem when the threshold is very large. However, this idea fails for small thresholds. Furthermore, reliance on the full combining key appears to be inherent in the construction strategy.

## 4 Main Construction

In this section, we will give our main construction of MSSS with nearly optimal share size. We first propose a new construction of DTPKE for a given threshold range $[\delta_0, \delta_1)$ which is more efficient than that of [6] in the sense that the combining (and encryption) key grows linearly only in the *range* of the threshold values rather than the total number of users. Such a DTPKE scheme has an important

[4] $D$ forms a set of *dummy users*, which will be combined with the set of users in order to be consistent with the threshold values.

[5] The decryption procedure uses a special aggregate technique, which employs a simple fact that a product of inverses of coprime polynomials can be written as a sum of inverses of affine polynomials. For the detailed description, we refer the paper [6].

property in our context: for threshold range $[\delta_0, 2\delta_0)$, if one distributes combining keys via an IDS using threshold $\delta_0$, then each share size will be constant. We then show how our generic construction can be adapted to exploit this efficiency property of the new DTPKE to get our actual space efficient multi-secret sharing scheme.[6]

### 4.1 New construction of DTPKE

As described in Sect. 3.3, the construction of [6] requires a linearly-size (in total number of users $n$) "combining key," which prevents the scheme from yielding an efficient MSSS. We here propose a new construction for DTPKE so that the combining (and encryption) key grows linearly in the *range* of the threshold $t$. The new scheme enables senders to specify an arbitrary threshold for the ciphertext in a given range $[\delta_0, \delta_1)$, and—as in [6]— achieves optimal size ciphertexts (a constant number of group elements). Additionally, our scheme is algebraically simpler than the original scheme of [6].

In more detail, the new scheme creates the secret key of a user $i$ as $usk = g^{P(\gamma)}$ where $P(\gamma)$ is the evaluation of a $(\delta_1 - 1)$-degree polynomial $P(x)$ at a secret point $\gamma$. For a random $k, \alpha$, the encryption algorithm of the scheme encodes the evaluation of a $(t - \delta_0)$-degree polynomial $A(x)$ into $C_2 = h^{k\alpha A(\gamma)}$ as a part of ciphertext. During the decryption, each user pairs a secret key with the ciphertext $C_2$ in order to form a decryption share $e(g,h)^{k\alpha A(\gamma)P(\gamma)}$ which will have the evaluation of a $(\delta_1 - \delta_0 + t - 1)$-degree polynomial at the secret point $\gamma$ in the exponent. The essential point for correctness here is to remove the dependence between the decryption share and the threshold $t$ in the exponent. We observe that using simple linear algebra, $t$ users can find $t$ coefficients to remove $(t-1)$ higher terms of $P(x)A(x)$, arriving at a polynomial with degree $(\delta_1 - \delta_0)$. With one more small trick, we can bring it down to $\delta_1 - \delta_0 - 1$. Thus, the combining key containing $(\delta_1 - \delta_0 - 1)$ corresponding powers will be enough to complete the decryption. Security will follow because when there is a deficit in the number of decryption shares, then there will necessarily be a leftover term in the exponent that masks the session key. Formally, given a threshold interval $[\delta_0, \delta_1)$, our DTPKE is as follows:

- **Setup**: The algorithm first chooses parameters defining a bilinear map system for a given security parameter $\lambda$: $\mathcal{B} = (p, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, e)$ where $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$ is a bilinear map and $|p| = \lambda$. The algorithm then randomly chooses two integers $\gamma, \alpha \in \mathcal{Z}_p^*$, and two generators $g \in \mathcal{G}_1$ and $h \in \mathcal{G}_2$. It also randomly selects $(\delta_1 - \delta_0 - 1)$ integers $x_i \in \mathcal{Z}_p^*$. The algorithm then outputs the master secret key $msk = (g, \alpha, \gamma)$, the encryption key

$$\mathrm{EK} = \left(u, v, \{h^{\alpha.\gamma^i}\}_{i=0}^{\delta_1-\delta_0-1}, \{x_i\}_{i=1}^{\delta_1-\delta_0-1}\right)$$

[6] Note that giving a construction for a specific threshold range is a generalization of the previous DTPKE. If the range can be simply defined as $[1, n]$, our new scheme implies the regular DTPKE given by [6].

where $u = g^{\alpha\cdot\gamma}$ and $v = e(g,h)^{\alpha}$, and the combining key

$$\text{CK} = \left( \{h^{\gamma^i}\}_{i=0}^{\delta_1-\delta_0-1}, \{x_i\}_{i=1}^{\delta_1-\delta_0-1} \right) .$$

- **KeyGen**:$(i, msk)$ The algorithm takes the master secret key $msk$ as input. It assigns an integer $i \in Z_p^*$ to each user as his $upk$, and constructs the $(\delta_1 - 1)$-degree polynomial $P_i(x) = i^{\delta_1-1}x^{\delta_1-1} + \cdots + i^2x^2 + ix + 1$. It then outputs the user's key pairs as $usk = g^{P_i(\gamma)}$ and $upk = i$ for all users.
- **Enc**$(t, \delta_0, \delta_1, EK, M)$: The algorithm takes as input the encryption key EK, a threshold value $t \in [\delta_0, \delta_1)$, and a message $M$. It randomly chooses an integer $k \in Z_p^*$ and calculates the ciphertext $C = (C_1, C_2, C_3)$ as

  $$C_1 = u^{-k}, \qquad C_2 = h^{k\alpha A(\gamma)}, \qquad C_3 = K \cdot M,$$

  where the polynomial $A(x) = (x + x_1) \ldots (x + x_{t-\delta_0})$ and $K = v^k$ is used as a session key. Since $A(\cdot)$ is $(t - \delta_0)$-degree polynomial, and $t \in [\delta_0, \delta_1)$, the second component $C_2 = h^{k\alpha A(\gamma)}$ of the ciphertext can be computed from $EK$ (see the explanation below).
- **ShareDecrypt**$(usk_i, C)$: The algorithm takes a ciphertext $(C_1, C_2, C_3)$ and secret key of a user as inputs. It computes the decryption share as

  $$\sigma_i = e(usk_i, C_2) = e(g,h)^{k.\alpha\cdot P_i(\gamma)\cdot A(\gamma)}.$$

- **Combine**$(CK, \{\sigma_i\}), \{upk_i\})$: This algorithm takes the set of decryption shares $\{\sigma_1, \ldots, \sigma_t\}$ and the identities $\{1, \ldots, t\}$ as inputs (w.l.o.g, we assume that the shares are from users $1, \ldots, t$, and the $P_i(\cdot)$ are defined using the coefficients $\langle 1, i, i^2 \ldots, i^{\delta_1-1}\rangle$ for $i \in [t]$).

  It first finds $t$ values $a_1, \ldots, a_t$, (not all zero) that satisfy

  $$\sum_{i=1}^{t} a_i P_i(x) = Q(x)\,,$$

  where $Q(x)$ is $(\delta_1 - t)$-degree polynomial.[7] This can be done by finding a nontrivial solution to the following linear system:

  $$\begin{bmatrix} 1 & 2^{\delta_1-1} & \cdots & t^{\delta_1-1} \\ 1 & 2^{\delta_1-2} & \cdots & t^{\delta_1-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 2^{\delta_1-t+1} & \cdots & t^{\delta_1-t+1} \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1}$$

[7] Note that $a_1, \ldots, a_t$ are also applied to calculate the value $H = \prod_{i=1}^{t-\delta_0} x_i(\sum a_i)$, which is used in the denominator in the decryption. Thus, not to get zero in the denominator, $a_1, \ldots, a_t$ must be chosen such that $\sum a_i \cdot \neq 0$

(Observe that the system has $t$ unknowns and $t-1$ constraints; hence the nullspace has dimension at least 1.) It follows that:

$$\prod_{i=1}^{t}(\sigma_i^{a_i}) = e(g,h)^{k.\alpha\cdot A(\gamma)\cdot\sum_{i=1}^{t} a_i Pi(\gamma)} ,$$

and it is equal to $e(g,h)^{k\alpha\cdot Q(\gamma)A(\gamma)}$ for a $(\delta_1-\delta_0)$-degree polynomial $Q(x)A(x)$. We further define:

$$R(x) = \frac{Q(x)\cdot A(x) - H}{x}, \text{ where } H = \prod_{i=1}^{t-\delta_0} x_i \left(\sum a_i\right) .$$

In one formula, the combine algorithm then extracts the key as:

$$\begin{aligned} K &= [e(C_1, h^{R(\gamma)}) \prod_{i=1}^{t} \sigma_i^{a_i}]^{1/H} \\ &= [e(g^{-k\alpha\gamma}, h^{R(\gamma)}) e(g,h)^{k\alpha Q(\gamma)\cdot A(\gamma)}]^{1/H} \\ &= e(g,h)^{k\alpha[Q(\gamma)A(\gamma)-\gamma R(\gamma)]/H} \\ &= e(g,h)^{k\alpha} \end{aligned}$$

(All the above value can be computed from public values; see the details below about correctness.)
Finally, it outputs $C_3/K$ as the plaintext.

*Efficiency:* We can see that the construction has secret key and ciphertext both given by a constant number of group elements; the encryption key and combining key size grow linearly in the size of the range where the thresholds fall.

*Correctness:* Note that all the coefficients of the polynomial $A(x)$ are public and $A(x)$ is with degree $t-\delta_0 < \delta_1 - \delta_0$, thus the ciphertext $C_2 = (h^{\alpha A(\lambda)})^k$ is computable from the public values $h^{\alpha}, h^{\alpha\gamma}, \ldots, h^{\alpha\gamma^{\delta_1-\delta_0-1}}$ contained in $EK$.

Furthermore, since the degree the coefficients of the polynomials $\{P_i(x)\}$ are all public, the equations for the linear system can easily be formed. Observe that the coefficient matrix of the linear system has rank $(t-1)$ and there are $t$ unknowns for the equations. Thus, there are always a solution set for $a_1, \ldots, a_t$. More importantly, since the solution set of the linear system aims at reducing the degree of $\sum_i a_i P_i(x)$, i.e., to kill the higher degree terms, the resulting polynomial $Q(x)$ will have degree $[\delta_1 - 1 - (t-1)] = \delta_1 - t$.

Finally, the polynomial $R(x)$ is equal to

$$\frac{Q(x)\cdot A(x) - H}{x}, \qquad \text{where} \qquad H = \prod_{i=1}^{t-\delta_0} x_i \left(\sum a_i\right)$$

is the constant term of $Q(x)A(x)$. Thus, $R(x)$ will have degree $\delta_1 - \delta_0 - 1$, and all the coefficients are known (or defined by the values of $\langle a_1, \ldots, a_t\rangle$). It follows that $h^{R(\gamma)}$ can be computed from the public values $h, h^{\gamma}, \ldots, h^{\gamma^{\delta_1-\delta_0-1}}$ that are contained in the combining key.

*Security:* We first briefly explain the intuition for the security. In the **KeyGen** algorithm, the rows of the Vandermonde matrix are used as the coefficients for generating the polynomial $P_i(x)$ in the secret keys. So, if we remove any column from the coefficient matrix of the linear system, the resulting matrix will be a Vandermonde matrix which has full rank. Thus, the system (1) with $(t-1)$ equations will have only one solution, in which all $a_i$ are zero. However, the system (1) with at most $(t-1)$ equations and $t$ coefficients will have many solutions. Thus, when $t < \delta_1$, using one such solution in which not all $a_i$ are zero and $\sum a_i \cdot \neq 0$, $t$ users can decrease the degree on the exponent of the decryption shares by at most $t-1$. If the ciphertext is assigned threshold $t' > t$, then the leftover degree (on the exponent hiding $K$) will be larger than $\delta_1 - \delta_0$. Thus, the combining key will not be enough to compute the leftover term, and finish the decryption. (Otherwise, the adversary can reduce degrees on the exponent which violates the underlying assumption.)

**Theorem 2.** *The dynamic threshold public key encryption scheme given above is IND-CPA secure under MSE-DDH assumption.*

*Proof.* Assume there is an adversary $\mathcal{A}$ that breaks the security of DTPKE, then we claim that we build a simulator $\mathcal{S}$ that breaks the security of the assumption MSE-DDH. Both the adversary and the simulator are given the threshold interval $[\delta_0, \delta_1)$ and a threshold value $t$. The simulator $\mathcal{S}$ interacts with the adversary $\mathcal{A}$ as follows:

- The simulator is given a group system $\mathcal{B} = (p, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, e)$ and an MSE-DDH instance in $\mathcal{B}$. Thus, the simulator has two coprime polynomials $f$ and $g$ with the degree $(t-\delta_0)$, and with their distinct roots $y_1, \ldots, y_{(t-\delta_0)}$ and $z_1, \ldots, z_{(t-\delta_0)}$. $\mathcal{S}$ has also a sequence of exponentiations

$$\begin{array}{ll} g_0, g_0^{\gamma}, \ldots, g_0^{\gamma^{\delta_1+t-\delta_0-1}}, & g_0^{k.\gamma.f(\gamma)}, \\ g_0^{\alpha}, g_0^{\alpha.\gamma}, \ldots, g_0^{\alpha.\gamma^{\delta_1+t-\delta_0+1}}, & \\ h_0, h_0^{\gamma}, \ldots, h_0^{\gamma^{\delta_1-\delta_0-1}}, & \\ h_0^{\alpha}, h_0^{\alpha.\gamma}, \ldots, h_0^{\alpha.\gamma^{2.\delta_1-2.\delta_0+1}}, & h_0^{k.g(\gamma)} \end{array}$$

  as well as $T \in \mathcal{G}_T$ which is either equal to $e(g_0, h_0)^{k.f(\gamma)}$ or some random element of $\mathcal{G}_T$.
- To produce the public parameters, $\mathcal{S}$ first sets

$$\begin{array}{ll} g = g_0^{f(\gamma)} \quad , & u = g_0^{\alpha.\gamma.f(\gamma)} = g^{\alpha.\gamma} \\ h = h_0 \quad , & v = e(g_0, h_0)^{\alpha.f(\gamma)} = e(g,h)^{\alpha}. \end{array}$$

  Since $f$ is $(t-\delta_0)$-degree polynomial, $u$ and $v$ can be calculated from the given instance.

The simulator $\mathcal{S}$ then forms the set $\{x_i\}_{i=1}^{\delta_1-\delta_0-1}$ as follows: it sets the first $(t-\delta_0)$ elements as $x_i = z_i$ where $z_1, \ldots, z_{t-\delta_0}$ are the roots of the polynomial $g$; and the remaining elements $x_{t-\delta_0+1}, \ldots, x_{\delta_1-\delta_0-1}$ are chosen as random integers in $\mathcal{Z}_p^*$. Finally, it gives the encryption key

$$\mathrm{EK} = (u, v, \{h^{\alpha.\gamma^i}\}_{i=0}^{\delta_1-\delta_0-1}, \{x_i\}_{i=1}^{\delta_1-\delta_0-1})$$

and the combining key

$$\mathrm{CK} = (\{h^{\gamma^i}\}_{i=0}^{\delta_1-\delta_0-1}, \{x_i\}_{i=1}^{\delta_1-\delta_0-1})$$

to the adversary.

- When the adversary makes a corrupt query for a user with the identity $id$, the simulator first picks a random integer $i \in [n]$, and gives the corresponding secret key $usk_i = g^{P_i(\gamma)}$ and public key $upk_i = i$ to the adversary. Note that since $P_i(x) = i^{\delta_1-1}.x^{\delta_1-1} + \cdots + i^2.x^2 + i.x + 1$ is an $(\delta_1 - 1)$-degree polynomial, the secret key $usk_i = g_0^{f(\gamma).P_i(\gamma)}$ can be computed from the given instance.

- The adversary submits two messages $M_0, M_1$. The simulator chooses a bit $b \in \{0, 1\}$, and sets the ciphertext $C_b = (C_1, C_2, C_3)$ as

$$C_b = (g_0^{-k.\gamma.f(\gamma)}, h_0^{k.g(\gamma)}, T \cdot M_b)$$

and returns $C_b$ to $\mathcal{A}$. It can be easily proved that if we set $k' = k/\alpha$, then

$$C_1 = u^{-k'}, \quad C_2 = h^{k'.\alpha.g(\gamma)}.$$

Note that if $T = e(g_0, h_0)^{k \cdot f(\gamma)}$, then $C_3 = v^{k'}.M_b$, and $C_2 = h^{k'.\alpha.g(\gamma)} = h^{k'.\alpha.A(\gamma)}$ as in the original scheme.

- The adversary outputs a guess $b'$ and the simulator gives $b'$ to the challenger.

Assume $\mathsf{Adv}_{\mathcal{A}}^{DTPKE}$ is the advantage of $\mathcal{A}$ in breaking the security of the construction, and $\mathsf{Adv}_{\mathcal{S}}^{MSE-DDH}$ is the advantage of $\mathcal{S}$ in breaking the security of the assumption. If $T = e(g_0, h_0)^{k \cdot f(\gamma)}$, then $\mathcal{A}$' s view will be identical to a real attack. Thus $\mathsf{Adv}_{\mathcal{A}}^{DTPKE} = |\Pr[b' = b] - 1/2| > \epsilon$. However, if $T$ is just random group element, then $\Pr[b' = b] = 1/2$. Therefore

$$\mathsf{Adv}_{\mathcal{S}}^{MSE-DDH} \geq \mid \frac{1}{2} \pm \epsilon - \frac{1}{2} \mid = \epsilon.$$

### 4.2 Computational multi-secret sharing with small share size

With the new construction of DTPKE developed above, we are now ready to construct our efficient multi-secret sharing scheme. Assume there are $n$ users and $\ell$ secrets. As briefly explained in section 4.2, we divide the secrets into $\log n$ parts according to their threshold ranges, i.e., $[1, 2)$,$[2, 4)$, $\ldots$, $[n/2, n)$. For the secrets in the range of $[\delta/2, \delta)$, we will run an independent instance of our general construction of multi-secret sharing, instantiating the underlying DTPKE with the new scheme we introduce in section 4.2.

Note that the generic construction will be slightly altered: for each threshold range $[\delta/2, \delta)$, the combining key $CK$ is dispersed using an IDS with threshold $\delta/2$. Since our new DTPKE for threshold range $[\delta/2, \delta)$ has the combining key with the size $O(\delta/2)$ (growing linearly with the size of the range instead of the largest threshold value $n$), running an IDS on this CK will yield share size $O(\delta/2)/(\delta/2) = O(1)$. Thus, we will have final share size $|C|(1/t_1 + 1/t_2 + \ldots + 1/t_\ell) + O(\log n)$ where $|C|$ is the ciphertext size of each message and $t_1, \ldots, t_\ell$ are the threshold values for the corresponding secrets. The first term is essentially the smallest size corresponding to the amount of information in the secrets, and the second part is the overhead we are aiming to minimize in this paper.

Given an efficient IDS (**IDA.Disperse, IDA.Rec**), and our new DTPKE (**PD.Setup, PD.KeyGen, PD.Enc, PD.ShareDecrypt, PD.Combine**), the details of our final construction of multi-secret sharing are presented as follows:

- **Share**: The algorithm takes the security parameter $\lambda$, the set $\mathcal{P}$ of $n$ users, and the secrets $s_1, \ldots, s_\ell$ to be distributed as inputs. It first divides these secrets into $\log n$ parts $\Delta_1, \ldots, \Delta_{\log n}$, according to the threshold values, where $\Delta_u := [2^{u-1}, 2^u)$ for $u = 1, \ldots, \log n$. Here $s_i$ with threshold $t_i$ belongs to $\Delta_u$ if $t_i \in \Delta_u$. For each group $\Delta_u$, the algorithm runs **PD.Setup**$(\lambda)$ to generate parameters $\{(MK_u, EK_u, CK_u)\}_{u=1,\ldots,\log n}$.
  The algorithm then encrypts all the secrets using the corresponding master public keys and the threshold values to generate the ciphertext $\{c_1, \ldots, c_\ell\}$, where $c_i = \textbf{PD.Enc}(EK_u, t_i, s_i)$ if $s_i$ is in group $\Delta_u$.
  Now the algorithm runs the IDS on each ciphertext $c_i$ using thresholds $t_i$, i.e., the shares of $c_i$ are generated as $\{c_{i,j}\}_{j=1,\ldots,n} \leftarrow \textbf{IDA.Disperse}(c_i, t_i)$. It also runs the IDS on each combining key $CK_u$ using threshold $2^{u-1}$ and gets the shares $\{CK_{u,j}\}_{j=1,\ldots,n}$.
  For each user $j$, the algorithm runs **PD.KeyGen** on each range of thresholds to generate key pairs $(upk_{j,u}, usk_{j,u}) \leftarrow \textbf{PD.KeyGen}(MK_u, j)$ for $j = 1, \ldots, n$. Also, it distributes the corresponding shares of ciphertexts and combining keys to this user.
  This algorithm ends with each user $j$ receiving his shares of ciphertexts $\{c_{i,j}\}_{i=1,\ldots,\ell}$, his shares of combining keys $\{CK_{u,j}\}_{u=1,\ldots,\log n}$, and his key pairs $\{(upk_{j,u}, usk_{j,u})\}_{u=1,\ldots,\log n}$.
- **Reconstruct**: A set $S$ of $t_k$ users jointly run this protocol. Each user $j \in S$ takes the index $k$ and his share $\{(usk_{j,u}, upk_{j,u}), CK_{u,j}, c_{i,j}\}_{u\in[\log n]}$ as inputs. The users first jointly run the **IDA.Rec** protocol to reconstruct the ciphertext $c_k$ and the combining key $CK_u$ (assuming $t_k \in [2^{u-1}, 2^u)$) that were dispersed using thresholds $t_k, 2^{u-1}$ respectively, i.e., they jointly reconstruct

$$c_k \leftarrow \textbf{IDA.Rec}(\{c_{k,j}\}_{j\in S}); \qquad \text{and} \qquad CK_u \leftarrow \textbf{IDA.Rec}(\{CK_{u,j}\}_{j\in S})\,.$$

Next, each user $j$ in the set $S$ partially decrypts the ciphertext, i.e.,

$$\sigma_{k,j} \leftarrow \textbf{PD.ShareDecrypt}(usk_{j,u}, c_k)$$

to get decryption share $\sigma_{k,j}$.

Finally, they jointly compute the corresponding secret

$$s_k \leftarrow \textbf{PD.Combine}(\{\sigma_{k,j}\}_{j \in S}, CK_u, c_k)\,,$$

and each party outputs the secret $s_k$.

*Efficiency:* For each secret $s_i$ with threshold $t_i$ such that $t_i \in [\delta/2, \delta)$ where $\delta = 2^{u-1}$ for some $u \in [\log n]$, the corresponding ciphertext $c_i$ will be dispersed using threshold $t_i$ which will have size $|C|/t_i$ (using, e.g., Rabin IDS [23], where $|C|$ is the ciphertext size). Thus, all the ciphertext shares have size $|C|/t_1 + |C|/t_2 + \cdots + |C|/t_\ell$ in total assuming all the ciphertexts are the same length. Regarding the combining key shares, there are $\log n$ combining keys, each with size $|CK_u| = O(2^u)$. They will be dispersed using threshold $2^{u-1}$ so that each contributes $O(1)$ to the share size. Hence, the total size from combining keys is $O(\log n)$. Additionally, there are $\log n$ user secret keys with constant size (or simply security parameter $\lambda$) each; thus the size from this part is $O(\log n)$. Since we consider the case where both $n, \ell$ are $\mathsf{poly}(\lambda)$, each share size is bounded by

$$|C|\Big(\frac{1}{t_1} + \cdots + \frac{1}{t_\ell}\Big) + O(\log \ell).$$

Note that our ciphertexts have 2 more group elements than the plaintext, as we consider the case that $\ell$ is large, this little overhead is subsumed by the dominating terms. Here we ignore the $\lambda$ factor at the overhead term.

As noted in the Introduction, we here also briefly compare the efficiency of our construction with other schemes in the literature that utilize a public bulletin board. Remark that our scheme achieves $O(\log \ell)$ share size for each user. If we encrypt the shares of users with a secret key and put the encryptions in a public bulletin board similar to [12, 18], then we achieve $O(\log \ell)$ size public storage together with constant size share for each user, which is the secret key. Thus, the total size of the bulletin board will be $O(\log \ell \cdot n)$. However, [12, 18] achieve only $O(\ell \cdot n)$ size bulletin board.

*Correctness:* This is very similar to the correctness of the generic construction, with the only difference being that the combining keys are dispersed; however, as long as there are enough shares, the combining key can be reconstructed trivially from the IDS property.

*Security:* The only difference with the generic construction is that the combining keys are also dispersed here. Note that, they are given out directly in the generic construction, thus the security is not influenced. Combining Theorem 1 and Theorem 2, we conclude the section with the following theorem:

**Theorem 3.** *Suppose $\lambda$ is the security parameter, under the MSE-DH assumption, there exists a computational secure $(t_1, \dots, t_\ell)$-multi-secret sharing scheme distributing $\ell$ secrets with length $m$ each, having share size at*

$$O\left(m \cdot (\frac{1}{t_1} + \cdots + \frac{1}{t_\ell}) + \log \ell\right).$$

## 5 A fair MPC protocol with smaller private state

A multi-party computation (MPC) protocol [9, 28] allows a set of parties to securely compute a function $f$ in a distributed way while ensuring the privacy of the users' input and the correctness of the output, in the presence of dishonest users. Besides these basic *correctness* and *privacy* requirements, a secure multiparty computation protocol is highly desirable to achieve *robustness* (the honest parties learn their output even some parties play maliciously), *fairness* (if the honest parties do not learn their output, then the corrupted parties also cannot learn their output). MPC protocols can be divided into two groups : ones that require an honest majority and ones that can achieve security even when there is no honest majority. Goldreich et al. [9] constructed two different secure MPC protocols : one achieves security in the passive model even when there is only one honest user, and one achieves security in the active model under the condition of honest majority. After this seminal work, most of the works on MPC focus on these two settings individually.

In [14], Ishai et al. aimed to achieve the best of both, and combined the two protocols into one that enjoys full security in the presence of any number of passive corruptions and a minority of active corruptions. Briefly, given the function $f$ and the inputs of $n$ parties $x_1, ..., x_n$, the parties follow a non-robust MPC protocol to compute their shares $y_i$. If this phase terminates successfully, then the parties collude their shares $y_i$ to output $y = f(x_1, ..., x_n)$. If the adversary aborts the computation in this phase, then the parties follow a robust MPC protocol to complete the computation of $y = f(x_1, ..., x_n)$. Even though combining two types of protocols into single one incorporates the relative advantages of both types, it possesses some drawbacks. If the first phase terminates with abort, the output might already been revealed to the adversary, thus, proceeding the protocol would violate the security.

Hirt et al. [13] circumvented the above restriction and provide a dynamic tradeoff between active and passive corruption. They proposed a protocol that enjoys security in the presence of $t < n$ passive adversaries, and full security in presence of $t < n/2$ active adversaries. Moreover, the protocol also guarantees full security in the presence of mix corruptions, as long as at most $k$ of them are actively corrupted and less than $n - k$ parties are corrupted in total. To this aim, they introduce a new primitive "gradual verifiable secret sharing (VSS)", which enables the protocol to release the secret gradually in the reconstruction phase. By this way, the protocol still guarantees enough secrecy for the honest parties even if it aborts during the reconstruction.

To be more specific, a gradual VSS is a secret sharing scheme in which instead of the secret $s$, the random secrets $s_1, \ldots, s_{n-1}$ where $s_1 + \ldots + s_{n-1} = s$, are shared among users in a way that each $s_i$ is shared with threshold $i$. [8] In reconstruction protocol, the shares are released one by one in decreasing order of thresholds. Particularly, in stage $i$ of the reconstruction protocol, if at least $(i+1)$ users provide correct shares, then the protocol outputs the corresponding secret $s_i$. Otherwise, the protocol is aborted, and each user outputs a set of corrupted users that did not provide correct shares. If there is no abort, each user will eventually get the secret $s = s_1 + \ldots + s_{n-1}$. Remark that an abort at stage $i$ prevents the adversary from learning $s_i, \ldots, s_1$ and thus the secret $s$. From this fact, the gradual VSS enables the protocol to preserve as much secrecy as possible.

The gradual VSS can be seen as a $(1, 2, \ldots, n-1)$-MSSS that reconstructs the secrets one by one in a decreasing order of the thresholds. In the distribution of each secret, Hirt et al. [13] utilize $(n-1)$ independent Shamir secret sharing schemes to implement gradual VSS. Thus, the first phase of the protocol results in linear size shares for users in the terms of the number users. However, if the protocol employs our construction given in section 4 as gradual secret sharing, then the size of the shares will be only logarithmic in the number of users.

The only task left after replacing the "trivial" secret sharing scheme with our efficient MSSS is that we have to make it verifiable. In other words, the users should be convinced that the shares output by the protocol are valid. Besides, they have to prove the validity of their shares when they pool them to get the secrets $s_i$. Fortunately, it is not hard to turn our construction to a verifiable one. In brief, in the **Share** protocol, each user receives a secret key, ciphertext shares and combining key shares generated by the IDS. For the verifiability, the dealer has to prove that the secret key is in the form $g^{P_i(\gamma)}$ which can be done easily by checking some bilinear relation. The validity of the shares can be inherited from a verifiable IDS [5] (or directly applying zk-SNARK [2, 10, 19, 20]). While in the **Reconstruct** protocol, each user proves in zero-knowledge that the decryption share is in the form of $\sigma = e(g, h)^{k\alpha \cdot A(\gamma) \cdot P_i(\gamma)}$ which could be done efficiently (e.g., $\sigma$-protocol type of ZK proof [24]). At last, each user proves the validity of the shares of the ciphertexts and the combining keys (which can always be done via zk-SNARK for practical instantiation) [9]. Note that for verifiability, some extra communication is necessary, however, they do not belong to the private states that each user has to store. For the details of each underlying tools, we refer to the references [5, 7, 24].

[8] The shares are the outputs of a general MPC protocol by running [9] in the first phase. Note that even though the GMW protocol does not provide strong enough robustness or fairness, however, malicious parties do not gain any advantage aborting in this phase as they only see shares.

[9] Here, it will be helpful to distinguish between a one-time public parameter-generation and a full-time public bulletin board: the former is the one-time setup; more importantly, it can be performed by the data owners for our setting. However, the latter requires the maintenance of the bulletin board during the life-time of the shares

## 6 Conclusions and Open Problems

In this paper, we constructed a share size (near) optimal multi-secret sharing scheme in the computational setting. Our construction achieves a share/overhead size grows logarithmically in the number of secrets, which significantly improves the previously known results about multi-secret sharing that the share size grows linear in the number of secrets. We use a tool of dynamic threshold public key encryption scheme as the underlying building block. In order to achieve our goal of share size efficiency, we constructed a new DTPKE that has a constant size ciphertext and shorter combining key, which might be of independent interests. We also demonstrate two interesting applications of our MSSS scheme to decentralized storage network and to the setting of fair secure multiparty computation.

There are also some open problems remain. Since a MSSS has a unique feature that multiple independent secrets can be shared without the need of sharing them one by one, an interesting question could be finding other applications to distributive cryptography that inherits our share size efficiency. Furthermore, our main building block of a new, efficient DTPKE (and the underlying technique) might also be applicable to e.g., constructions of attribute based encryption. Also, showing some lower bound on the parameter efficiency on the DTPKE is intriguing, intuitively, it seems hard to get a DTPKE scheme with all parameters sublinear, since there is not enough space to encode the corresponding information according to all the possible thresholds, but formally showing such a result will be quite interesting.

## References

1. A. Beimel, A. Ben-Efraim, C. Padró, and I. Tyomkin. Multi-linear secret-sharing schemes. In *Theory of Cryptography-TCC 2014*, pages 394–418, 2014.
2. N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 315–333, 2013.
3. G. R. Blakley. Safeguarding cryptographic keys. *Managing Requirements Knowledge, International Workshop on*, page 313, 1979.
4. C. Blundo, A. D. Santis, G. D. Crescenzo, A. G. Gaggia, and U. Vaccaro. Multi-secret sharing schemes. In *CRYPTO '94, 1994*, pages 150–163, 1994.
5. C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *DISC 2005*, pages 503–504, 2005.
6. C. Delerablée and D. Pointcheval. Dynamic threshold public-key encryption. In *Advances in Cryptology - CRYPTO 2008*, pages 317–334, 2008.
7. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
8. S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the tate pairing. In *Algorithmic Number Theory 2002*, pages 324–337, 2002.
9. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC 87*, pages 218–229.

10. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 321–340, 2010.
11. J. Herranz, A. Ruiz, and G. Sáez. Sharing many secrets with computational provable security. *Inf. Process. Lett.*, 113(14-16):572–579, 2013.
12. J. Herranz, A. Ruiz, and G. Sáez. New results and applications for multi-secret sharing schemes. *Des. Codes Cryptography*, 73(3):841–864, 2014.
13. M. Hirt, C. Lucas, and U. Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In *CRYPTO 2013,*, pages 203–219, 2013.
14. Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology - CRYPTO 2006*, volume 4117, pages 483–500, 2006.
15. W.-A. Jackson, K. M. Martin, and C. M. O'Keefe. Multisecret threshold schemes. In *CRYPTO 1993*, pages 126–135, 1993.
16. E. D. Karnin, S. Member, J. W. Greene, S. Member, and M. E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29:35–41, 1983.
17. H. Krawczyk. Secret sharing made short. In *CRYPTO '93*, pages 136–146, 1993.
18. H. Lin and Y. Yeh. Dynamic multi-secret sharing scheme. Int. J. Contemp. Math. Sciences, pages 37–42, 2008.
19. H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 169–189, 2012.
20. H. Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, pages 41–60, 2013.
21. T. N. M. Itoh, A. Saito. Secret sharing scheme realizing general access structure. *IEEE Globecom*, page 99102, 1987.
22. B. Masucci. Sharing multiple secrets: Models, schemes and analysis. *Des. Codes Cryptography*, 39(1):89–111, 2006.
23. M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36:335–348, 1989.
24. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.
25. A. Shamir. How to share a secret. *Communications of the ACM*, 22(22):612–613, 1979.
26. C. Tartary, J. Pieprzyk, and H. Wang. Verifiable multi-secret sharing schemes for multiple threshold access structures. In *Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers*, pages 167–181, 2007.
27. M. van Dijk, W.-A. Jackson, and K. M. Martin. A general decomposition construction for incomplete secret sharing schemes. *Des. Codes Cryptography*, (3):301–321, 1998.
28. A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS 82*, pages 160–164, 1982.