Algorithms for Detecting Cheaters in Threshold Schemes

Douglas R. Stinson and Sheng Zhang
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo Ontario N2L 3G1
Canada

Abstract

In a (k, n)-threshold scheme, a secret key K is split into n shares in such a way that the K can be recovered from k or more shares, but no information about K can be obtained from any $k-1$ or fewer shares. We are interested in the situation where there are some number of incorrect (i.e., faulty) shares. When there are faulty shares, we might need to examine more than k shares in order to reconstruct the secret correctly. Given an upper bound, namely t, on the number of faulty shares, we focus on finding efficient algorithms for reconstructing the secret in a (k, n)-threshold scheme. We call this the *threshold scheme with cheaters* problem.

We first review known combinatorial algorithms that use covering designs, as presented in Rees et al [11] and Tso et al [13]. Then we extend the ideas of their algorithms to a more general one. We also link the threshold scheme with cheaters problem to decoding generalized Reed-Solomon codes. Then we adapt two decoding algorithms, namely, the Peterson-Gorenstein-Zierler Algorithm and Gao's Algorithm, to solve our problem. Finally, we contribute a general algorithm that combines both the combinatorial and decoding approaches, followed by an experimental analysis of all the algorithms we describe.

1 Introduction

Efficient threshold schemes are very helpful in the management of cryptographic keys. Informally, a (k, n)-*threshold scheme* is a method of sharing

a secret key K among a finite set S of n participants, such that

- with the knowledge of any k or more shares ($k \leq n$), the secret K can be computed; and
- with the knowledge of any $k-1$ or fewer shares, it is impossible to derive any information about the secret K.

The problem of constructing threshold schemes was independently introduced and solved by Blakley [3] and Shamir [12] in 1979. Blakley's solution uses finite geometries, while Shamir's is based on polynomial interpolation and it is a bit more efficient.

1.1 Shamir's Threshold Scheme

Suppose q is a prime or prime power. In a (k, n) Shamir threshold scheme, we let

$$\mathcal{S} = \{(x_i, y_i) : 1 \leq i \leq n\} \subseteq (\mathbb{F}_q \backslash \{0\}) \times \mathbb{F}_q$$

be the set of n shares. Let $K \in \mathbb{F}_q$ be the secret key.

In the scheme setup stage, the share distributor, who is denoted by D, secretly chooses $k-1$ not necessarily distinct numbers from $\mathbb{F}_q$, namely, $a_1, a_2, \ldots, a_{k-1}$, to generate a (secret) polynomial

$$P_0(x) = K + \sum_{j=1}^{k-1} a_j x^j,$$

where K is the secret key. D also chooses n distinct elements of $\mathbb{F}_q$, namely, $x_1, x_2, \ldots, x_n$. Then, for $1 \leq i \leq n$, D computes $y_i = P_0(x_i)$, and he gives the *share* (x_i, y_i) to participant P_i.[1]

Notice that $K = P_0(0)$ is the secret. In order to compute the secret, it suffices to reconstruct the secret polynomial. Suppose we collect a subset of k shares, say $T = \{(x_{i_j}, y_{i_j}) : 1 \leq j \leq k\}$. We know that $y_{i_j} = P_0(x_{i_j})$ for $1 \leq j \leq k$. Hence, we can construct a system of k linear equations to solve for the coefficients of P_0. Once the system is solved, $P_0(0) = K$ is the desired secret key.

Alternatively, one can use Lagrange interpolation over $\mathbb{F}_q$ to find the unique polynomial P_T passing through the k points in T. The Lagrange

[1] Note that the x_i's can be public, while the y_i's must be secret. Thus we sometimes refer to y_i as the share, rather than the pair (x_i, y_i).

interpolation formula for P_T is given by

$$P_T(x) = \sum_{1 \le j \le k} y_j \prod_{1 \le l \le k, l \ne j} \frac{x - x_{i_l}}{x_{i_j} - x_{i_l}}, \tag{1}$$

and then $P_T(0) = P_0(0)$ is the desired key.

1.2 Threshold Scheme with Cheaters

In practice, it often happens that some shares are faulty, due to miscommunication or misconduct. In this case, even if k shares are pooled together, we might not be able to compute the correct value of the secret.

In a (k, n) Shamir threshold scheme implemented in $\mathbb{F}_q$, let's assume that at most t shares are faulty. For any k-subset $T \subseteq N$, there is a unique polynomial P_T of degree at most $k-1$ such that, for each $i \in T$, $P_T(x_i) = y_i$. (This polynomial can be constructed using the formula (1).) Therefore, we assume $n \ge k + t$, since we need k correct shares to construct the secret polynomial.

We now define some notation used throughout this paper:

- Denote $P_0(x) = K + \sum_{j=1}^{k-1} a_j x^j$ (the secret polynomial);
- Denote $N = \{1, 2, ..., n\}$ (the set of indices of all n shares);
- Denote $G = \{i \in N : y_i = P_0(x_i)\}$ (the set of indices of the good shares)
- Denote $B = N \backslash G$ (the set of indices of the bad shares);
- Denote $C_T = \{i \in N : P_T(x_i) = y_i\}$ (the set of indices of the shares that lie on the polynomial P_T);
- Denote $NC_T = N \backslash C_T$ (the compliment of C_T).

It is easy to prove (see [11]) that the following facts hold:

(1) If $T \subseteq G$, then $P_T(x) = P_0(x)$ and $|C_T| \ge n - t$;

(2) If $T \cap B \ne \emptyset$, then $P_T(x) \ne P_0(x)$ and $|C_T| \le k + t - 1$.

We will assume that $n \geq k + 2t$ in the rest of the paper; this ensures that cases **(1)** and **(2)** enumerated above can be distinguished by the value of $|C_T|$.[2]

Now, it is obvious that **(1)** and **(2)** imply the following two additional facts:

(3) If $|C_T| \geq k + t$, then $P_T(x) = P_0(x)$; and

(4) If $|NC_T| \geq t + 1$, then $P_T(x) \neq P_0(x)$.

If we can somehow find a k-subset T containing no indices of bad shares, then the above facts tell us how to solve the threshold scheme with cheaters problem.[3] In the next section, we use a combinatorial structure, called a *covering*, to accomplish this goal.

1.3 Coverings

Definition 1.1. *A collection $\mathcal{T}$ of k-subsets of $\{1, 2, \ldots, n\}$ (called* blocks*) is an (n, k, t)-*covering *if every t-subset of $\{1, 2, \ldots, n\}$ is contained in at least one block.*

For example,

$$\mathcal{T} = \{\{1, 2, 3, 4\}, \{1, 4, 5, 6\}, \{1, 5, 6, 7\}, \{2, 3, 4, 7\}, \{2, 3, 5, 6\}\}$$

forms a $(7, 4, 2)$-covering.

The following elementary lemma is the underlying idea of Rees et al's deterministic algorithm ([11]).

Lemma 1.1. *Suppose that $\mathcal{T}'$ is the set of blocks in an $(n, n-k, t)$-covering. Define*

$$\mathcal{T} = \{\{1, 2, \ldots, n\} \setminus T : T \in \mathcal{T}'\}.$$

Then, for any t-subset of $\{1, 2, \ldots, n\}$, there exists a block (of size k) in $\mathcal{T}$ that is disjoint from the given t-subset.

[2] If $n < k + 2t$, then it is in fact easy to see that the threshold scheme with cheaters problem cannot be solved.

[3] By "solving" the threshold with cheaters problem, we mean reconstructing the correct secret, even if there are t or fewer faulty shares. We are not required to identify all the incorrect shares.

Remark: The converse assertion also holds.

In the next section of this paper, we review three existing algorithms for solving the threshold scheme with cheaters problem. In later sections, we present several new approaches. Our new contributions are summarized as follows. We show the relation between *generalized Reed-Solomon codes* and the threshold schemes with cheaters problem. This allows us to apply two decoding algorithms, the *Peterson-Gorenstein-Zierler Decoding Algorithm* (*PGZ*) and *Gao's Decoding Algorithm*, to solve our problem. As well, we combine both the decoding approach and the covering approach to yield a generalized combined algorithm for solving the threshold schemes with cheaters problem.

2 Previous Algorithms

In this section, we review three existing algorithms for solving the threshold scheme with cheaters problem. Two of these algorithms are contributed by Rees et al in [11]: **Algorithm 1** is a deterministic algorithm which is based on coverings and **Algorithm 2** is a randomized algorithm. **Algorithm 3** was given by Tso et al in [13]; it is a modified version of **Algorithm 1** that introduces a *break-point* variable to decrease the number of iterations of the algorithm. We then present a new algorithm, **Algorithm 4**, by further generalizing the break-point technique used in **Algorithm 3**.

2.1 Rees et al.'s Deterministic Algorithm

Let $S = \{(x_i, y_i) : 1 \leq i \leq n\}$ be a set of shares; and let $\mathcal{T}$ be a collection of k-subsets of $N = \{1, 2, \ldots, n\}$, where $\{N \backslash T : T \in \mathcal{T}\}$ forms a $(n, n-k, t)$-covering. By Lemma 1.1, it follows that, for every t-subset $Y \subseteq N$, there exists a block $T \in \mathcal{T}$ such that $N \cap Y = \emptyset$.

Here is the algorithm from [11]:

Algorithm 1.
Input: $\mathcal{T}, S, n, k, t$
1. FOR EACH $T \in \mathcal{T}$ DO
2. compute $P_T(x)$
3. compute C_T
4. IF $|C_T| \geq n - t$ THEN $P_0(x) \leftarrow P_T(x)$ and QUIT

It is easy to see that **Algorithm 1** will solve the threshold scheme with cheaters problem. There exists at least one block $T \in \mathcal{T}$ such that $T \cap B = \emptyset$. For any such block T, we have that $|C_T| \geq n - t$ by fact **(1)**. On the other hand, if $T \cap B \neq \emptyset$, then $|C_T| \leq k + t - 1 < n - t$ by fact **(2)**. Therefore the algorithm will compute the correct polynomial $P_0(x)$.

The number of iterations of the FOR loop in **Algorithm 1** is bounded above by the number of blocks in $\mathcal{T}$. Let $C(n, k, t)$ denote the minimum number of blocks in an (n, k, t)-covering; then the number of iterations is at most $C(n, n - k, t)$.

If n is sufficiently large compared to k and t, then it is easy to construct an $(n, n - k, t)$-covering having $C(n, n - k, t)$ blocks. Below we recall two theorems from Mills [9].

Theorem 2.1. *If $n \geq k(t + 1)$, then $C(n, n - k, t) = t + 1$.*

For $n \geq k(t+1)$, we can take $\mathcal{T}$ to consist of $t+1$ disjoint blocks of size k.

Theorem 2.2. *If $k(t + 1/2) \leq n \leq k(t + 1)$, then $C(n, n - k, t) = t + 2$.*

For $k(t + 1/2) \leq n \leq k(t + 1)$, we can take $\mathcal{T}$ to consist of $t - 1$ disjoint blocks of size k, together with three additional blocks such that $3k/2$ additional points are each contained in two of these three blocks. For example, if $n = 14$, $k = 4$ and $t = 3$, then we can take

$$\mathcal{T} = \{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{9, 10, 11, 12\}, \{9, 10, 13, 14\}, \{11, 12, 13, 14\}\}.$$

From the above theorems, we see that the number of iterations can be bounded above by $t + 2$ provided that $n \geq k(t + 1/2)$. On the other hand, if n is "close to" $k + 2t$, then an $(n, n - k, t)$-covering becomes difficult to construct and the required number of blocks is increased.

2.2 Rees et al.'s Randomized Algorithm

Let $S = \{(x_i, y_i) : 1 \leq i \leq n\}$ be a set of shares. We proceed to describe the second algorithm.

Algorithm 2.
Input: S, n, k, t
1. REPEAT the following steps:
2. Let T be a random k-subset of $\{1, 2, \ldots, n\}$
3. Compute $P_T(x)$
4. Compute C_T
5. IF $|C_T| \geq n - t$, THEN $P_0(x) \leftarrow P_T(x)$ and QUIT

Algorithm 2 is a Las Vegas [1] type algorithm; it terminates when we eventually choose a set T that contains no indices of bad shares. An analysis of the average-case complexity of the algorithm is given in [11].

2.3 Tso et al's Algorithm

Tso et al's algorithm from [13], which we present as **Algorithm 3**, is a modified version of **Algorithm 1**. One contribution of [13] is to modify the algorithm so that we require the input of only $k + 2t$ shares (instead of all n shares, as was the case in **Algorithm 1**). Another feature of this algorithm is to keep track of the number of shares that lie on the interpolation polynomial $P_T(x)$, as well as the shares that do not, for each interpolation polynomial $P_T(x)$ determined by a block T. This allows **Algorithm 3** to potentially accept or reject a block T in the set system more quickly than **Algorithm 1** does.

In **Algorithm 3**, $\mathcal{R}$ is a $(k+2t)$-subset of N, $S_{\mathcal{R}} = \{(x_i, y_i) : i \in \mathcal{R}\} \subseteq S$ is a subset of $k + 2t$ shares, and $\mathcal{T}$ is a set of k-subsets of $\mathcal{R}$ such that $\{\mathcal{R} \backslash T : T \in \mathcal{T}\}$ forms a $(k + 2t, 2t, t)$-covering.

Algorithm 3.
Input: $\mathcal{R}, \mathcal{T}, S_{\mathcal{R}}, k, t$
1. FOR EACH $T \in \mathcal{T}$ DO
2. Compute $P_T(x)$
3. $NC_T \leftarrow \emptyset$
4. $C_T \leftarrow T$
5. FOR j FROM 1 TO $2t$ DO
6. IF $y_{i_j} = P_T(x_{i_j})$ THEN $C_T = C_T \cup \{x_{i_j}\}$
7. ELSE $NC_T \leftarrow NC_T \cup \{x_{i_j}\}$
8. IF $|C_T| \geq k + t$ THEN $P_0(x) \leftarrow P_T(x)$ and QUIT
9. ELSE IF $|NC_T| \geq t + 1$ THEN BREAK

By keeping track of $|NC_T|$ and $|C_T|$ as we go along, **Algorithm 3** makes it possible to accept or to discard a given block T more quickly than

was the case in **Algorithm 1**. In step 9, if $|NC_T| \geq t+1$, then we are able to terminate the inner FOR loop early and proceed to the next block T in the outer FOR loop. In step 8, if $|C_T| \geq k+t$, then we are able to terminate the entire algorithm because we know that we have the correct polynomial $P_0(X)$.

A major drawback of **Algorithm 3** is that the construction of the necessary covering can be quite difficult. That is, there is no efficient systematic way known to construct $(k+2t, 2t, t)$-coverings. However, we note that the use of $|NC_T|$ could also be applied to **Algorithm 1**. Therefore, we present an algorithm that combines **Algorithms 1** and **3** by relaxing the restriction on n, to achieve a faster running time and cheaper covering computation overhead. In this algorithm, we assume that we have at least $k(t+1/2)$ shares, so we can use one of the efficient covering constructions given in Theorems 2.1 and 2.2.

2.4 An Improved Algorithm

In the next algorithm, we assume that $\mathcal{R}$ is a v-subset of N, where $v \geq min\{k(t+1/2), k+2t\}$, $S_{\mathcal{R}} = \{(x_i, y_i) : i \in \mathcal{R}\} \subseteq S$ is a subset of v shares, and $\mathcal{T}$ is a set of k-subsets of $\mathcal{R}$ such that $\{\mathcal{R} \backslash T : T \in \mathcal{T}\}$ forms a $(v, v-k, t)$-covering.

Algorithm 4.
Input: $\mathcal{R}, \mathcal{T}, S_{\mathcal{R}}, k, t$
1. FOR EACH $T \in \mathcal{T}$, perform the following steps:
2. compute $P_T(x)$
3. $S_T \leftarrow \{(x_i, y_i) : i \in T\}$
4. $NC_T \leftarrow \emptyset$
5. $C_T \leftarrow T$
6. FOR EACH (x_i, y_i) in $(S_{\mathcal{R}} \setminus S_T)$ DO
7. IF $y_i = P_T(x_i)$ THEN $C_T = C_T \cup \{x_{i_j}\}$
8. ELSE $NC_T \leftarrow NC_T \cup \{x_{i_j}\}$
9. IF $|C_T| \geq k+t$ THEN $P_0(x) \leftarrow P_T(x)$ and QUIT
10. ELSE IF $|NC_T| \geq t+1$ THEN BREAK

3 Decoding Algorithms

In this section, we will show a close relation between threshold schemes and generalized Reed-Solomon codes. It turns out that that solving the threshold scheme with cheaters problem is equivalent to decoding a certain

generalized Reed-Solomon code. Then we present and discuss two suitable decoding algorithms: the Peterson-Gorenstein-Zierler Algorithm and Gao's Algorithm.

Remark 3.1. It was observed by McEliece and Sarwate [8] that Shamir's threshold scheme [12] is closely related to Reed-Solomon codes [10, 2]. Reed-Solomon codes are widely used in many applications (e.g., CD and DVD players) due to their burst error-correction capabilities, and there are various (fast) decoding algorithms for Reed-Solomon codes.

3.1 Generalized Reed-Solomon Codes and Threshold Scheme

We now review some standard concepts from coding theory; we mainly follow the presentation in Huffman and Pless [6].

Definition 3.1. *Suppose q is a prime power and $\mathcal{C}$ is a k-dimensional subspace of $(\mathbb{F}_q)^n$ such that any two vectors in $\mathcal{C}$ have hamming distance at least d. Then $\mathcal{C}$ is called an* (n, k, d)-code *over $\mathbb{F}_q$.*

Definition 3.2. *[6] Let β be a primitive element of $\mathbb{F}_q$. For $0 \leq k \leq n = q - 1$, let $\mathcal{P}_k$ denote the set of polynomials whose degree does not exceed $k - 1$ (including the zero polynomial). Define*

$$\mathcal{C} = \{(f(1),\ f(\beta),\ f(\beta^2),\ \ldots,\ f(\beta^{n-1}))\ :\ f \in \mathcal{P}_k\}$$

*Then $\mathcal{C}$ is an $(n, k, n-k+1)$-code over $\mathbb{F}_q$, which is known as an $(n, k, n-k+1)$-*Reed-Solomon code *(or,* RS code*).*

Definition 3.3. *[6] Let n be an arbitrary integer with $1 \leq n \leq q - 1$. Let $\gamma = (\gamma_0,\ \gamma_1,\ \ldots,\ \gamma_{n-1})$ be an n-tuple of distinct elements over $\mathbb{F}_q$, and let $\nu = (v_0,\ v_1,\ \ldots,\ v_{n-1})$ be an n-tuple of nonzero elements over $\mathbb{F}_q$, not necessarily distinct. Finally, let k be an integer such that $1 \leq k \leq n$. Define*

$$\mathcal{C} = \{(v_0 f(\gamma_0),\ v_1 f(\gamma_1),\ \ldots,\ v_{n-1} f(\gamma_{n-1}))\ :\ f \in \mathcal{P}_k\} \tag{2}$$

*Then $\mathcal{C}$ is an $(n, k, n-k+1)$-code over $\mathbb{F}_q$, which is known as a $(n, k, n-k+1)$-*generalized-Reed-Solomon code *(or,* GRS code*). We refer to this code as $GRS_k(\gamma, \nu)$.*

Theorem 3.1. *[6] A generator matrix for $GRS_k(\gamma, \nu)$ is*

$$G = \begin{bmatrix} v_0 & v_1 & \cdots & v_{n-1} \\ v_0\gamma_0 & v_1\gamma_1 & \cdots & v_{n-1}\gamma_{n-1} \\ v_0\gamma_0^2 & v_1\gamma_1^2 & \cdots & v_{n-1}\gamma_{n-1}^2 \\ & & \vdots & \\ v_0\gamma_0^{k-1} & v_1\gamma_1^{k-1} & \cdots & v_{n-1}\gamma_{n-1}^{k-1} \end{bmatrix}. \tag{3}$$

Now, suppose we define

$$\nu = (1, 1, \ldots, 1) \quad \text{and} \quad \gamma = (x_1, \ldots, x_n).$$

Let $(K, a_1, a_2, \ldots, a_{k-1})$ be the vector of coefficients of the polynomial $P_0(x)$ in a (k, n) Shamir threshold scheme. Consider the codeword

$$(y_1, \ldots, y_n) = (K, a_1, a_2, \ldots, a_{k-1})G$$

in the code $GRS_k(\gamma, \nu)$. It is easy to see that this codeword is just the vector of n shares $(y_1, \ldots, y_n)$ in the threshold scheme. That is, the process of encoding using the above-described generating matrix G is the same thing as computing shares in a (k, n) Shamir threshold scheme.

We can also interpret the faulty shares in a Shamir threshold scheme in the context of of GRS codes. Suppose t shares are faulty among the n distributed shares. Then, correspondingly, a received vector in the GRS code contains t errors. Identifying the t faulty shares is the same thing as correcting the t errors in the received vector.

An (n, k)-GRS code can correct up to $\lfloor (n-k)/2 \rfloor$ errors. In order to correct t errors, we require $t \leq (n-k)/2$, *i.e.,* $n \geq k + 2t$. This is the same assumption that we made in Section 1.2.

Before discussing decoding algorithms for GRS codes, we present the parity check matrix of $GRS_k(\gamma, \nu)$.

Theorem 3.2. *[6] Let $\gamma = (\gamma_0, \gamma_1, \ldots, \gamma_{n-1})$ be an n-tuple of distinct elements over $\mathbb{F}_q$, and let $\nu = (v_0, v_1, \ldots, v_{n-1})$ be an n-tuple of nonzero elements over $\mathbb{F}_q$, not necessarily distinct. Then there exists an n-tuple $w = (w_0, w_1, \ldots, w_{n-1})$ of nonzero elements over $\mathbb{F}_q$, such that $GRS_k(\gamma, \nu)^\perp = GRS_{n-k}(\gamma, w)$ for all $0 \leq k \leq n-1$. Furthermore, the n-tuple w is a nonzero codeword in the 1-dimensional code $GRS_{n-1}(\gamma, \nu)^\perp$ and it satisfies*

$$\sum_{i=0}^{n-1} w_i v_i h(\gamma_i) = 0 \tag{4}$$

for any $h \in \mathcal{P}_{n-1}$.

Theorem 3.3. *[6] The generator matrix of $GRS_{n-k}(\gamma, w)$, where w is given in (4), is the parity check matrix H of $GRS_k(\gamma, \nu)$. The matrix H is*

as follows:

$$H = \begin{bmatrix} w_0 & w_1 & \dots & w_{n-1} \\ w_0\gamma_0 & w_1\gamma_1 & \dots & w_{n-1}\gamma_{n-1} \\ w_0\gamma_0^2 & w_1\gamma_1^2 & \dots & w_{n-1}\gamma_{n-1}^2 \\ & & \vdots & \\ w_0\gamma_0^{n-k-1} & w_1\gamma_1^{n-k-1} & \dots & w_{n-1}\gamma_{n-1}^{n-k-1} \end{bmatrix}. \tag{5}$$

3.2 Peterson-Gorenstein-Zierler Decoding Algorithm

The Peterson-Gorenstein-Zierler (or PGZ) Algorithm, is designed for BCH codes and it can correct up to $t = \lfloor \frac{d-1}{2} \rfloor$ errors. where d is the distance of the code. In this section, we will adapt the PGZ Algorithm to decoding GRS codes. The description of the next algorithm is modified from [6].

Algorithm 5 (PGZ Decoding Algorithm).
Input: A generating matrix G for $GRS_k(\gamma, \nu)$, and a received vector $y = (y_1,\ y_2,\ \dots,\ y_n) \in \mathbb{F}_q^n$.
Step 1: Find the vector w using the method described in Theorem 3.2 and then construct a parity check matrix H using the method described in Theorem 3.3.
Step 2: Compute the syndrome $S = (S_1,\ S_2,\ \dots,\ S_{n-k}) = yH^T$, where y is the received vector.
Step 3: FOR μ FROM t DOWNTO 1, construct the matrix

$$M_\mu = \begin{bmatrix} S_1 & S_2 & S_3 & \dots & S_{\mu-1} & S_\mu \\ S_2 & S_3 & S_4 & \dots & S_\mu & S_{\mu+1} \\ S_3 & S_4 & S_5 & \dots & S_{\mu+1} & S_{\mu+2} \\ & & & \vdots & & \\ S_\mu & S_{\mu+1} & S_{\mu+2} & \dots & S_{2\mu-2} & S_{2\mu-1} \end{bmatrix}.$$

Stop at the first value of μ such that M_μ is nonsingular. Then set $v = \mu$.
Step 4: Solve the matrix equation

$$\begin{bmatrix} S_1 & S_2 & S_3 & \dots & S_{v-1} & S_v \\ S_2 & S_3 & S_4 & \dots & S_v & S_{v+1} \\ S_3 & S_4 & S_5 & \dots & S_{v+1} & S_{v+2} \\ & & & \vdots & & \\ S_v & S_{v+1} & S_{v+2} & \dots & S_{2v-2} & S_{2v-1} \end{bmatrix} \begin{bmatrix} \sigma_v \\ \sigma_{v-1} \\ \sigma_{v-2} \\ \vdots \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ -S_{v+3} \\ \vdots \\ -S_{2v} \end{bmatrix}$$

for the σ_i's $(1 \le i \le v)$.
Step 5: Construct the polynomial $\sigma(x) = 1 + \sum_{i=1}^{v} \sigma_i x^i$.

Step 6: Find the roots of $\sigma(x)$.
Step 7: If there are v distinct roots, say $\delta_1,\ \delta_2\ \ldots,\ \delta_v$ then compute the error locations $\delta_1^{-1},\ \delta_2^{-1}\ \ldots,\ \delta_v^{-1}$ and construct the codeword y' from the co-ordinates in y that are not in error. If there do not exist v distinct roots, then output "Decoding failure".

3.3 Gao's Decoding Algorithm

Gao's Algorithm is based on the extended Euclidean algorithm. In this section, we only consider $GRS_k(\gamma, 1)$ codes (*i.e.,* all v_i's are ones), in order to simplify the notation.

We use polynomial notation: To encode a tuple of k message symbols, $m = (m_1,\ m_2,\ \ldots,\ m_k)$, we define the message polynomial

$$f(x) = m_1 + m_2 x + \ldots + m_k x^{k-1}\ \in \mathcal{P}_k. \tag{6}$$

The corresponding codeword is

$$(c_1,\ c_2,\ \ldots,\ c_n) = (f(\gamma_1),\ f(\gamma_2),\ \ldots,\ f(\gamma_n)), \tag{7}$$

which we write in polynomial form as

$$c(x) = c_1 + c_2 x + \ldots + c_n x^{n-1}.$$

Further, let $y = (y_1,\ y_2,\ \ldots,\ y_n) \in \mathbb{F}_q^n$ be a received vector. The corresponding received polynomial is denoted $y(x)$. Gao's Algorithm will compute the message polynomial $f(x)$ in (6) that defines a codeword $c(x)$ whose distance from $y(x)$ is at most t (provided that such a codeword exists).

Algorithm 6 (Gao's Algorithm).
Input: A vector $\gamma = (\gamma_1,\ \gamma_2,\ \ldots,\ \gamma_n)$ and a received vector

$$y = (y_1,\ y_2,\ \ldots,\ y_n) \in \mathbb{F}_q^n.$$

Output: The message polynomial $f = m_1 + m_2 x + \ldots + m_k x^{k-1}$, or "Decoding failure".
Step 0: (Pre-computation) Compute the polynomial

$$g_0 = \prod_{i=1}^{n} (x - \gamma_i)\ \in \mathbb{F}_q[x]. \tag{8}$$

Step 1: (Interpolation) Find the unique polynomial $g_1(x) \in \mathbb{F}_q[x]$ of degree $\leq n-1$ such that $g_1(\gamma_i) = y_i$ for all i, $1 \leq i \leq n$.

Step 2: (Partial gcd) Apply the extended Euclidean algorithm to $g_0(x)$ and $g_1(x)$. Stop when the remainder, namely, $g(x)$, has degree less than $(n+k)/2$. Upon termination, we have

$$u(x)g_0(x) + v(x)g_1(x) = g(x). \tag{9}$$

(The polynomial $v(x)$ is the *error locator polynomial*, whose roots indicate the error coordinates.)
Step 3: (Long division) Divide $g(x)$ by $v(x)$ obtained in step 2, obtaining

$$g(x) = f_1(x)v(x) + r(x), \tag{10}$$

where $deg(r(x)) < deg(v(x))$. If $r(x) = 0$ and $f_1(x)$ has degree $< k$, then output $f_1(x)$; otherwise output "Decoding failure" (which implies that more than t errors occurred).

4 Combined Algorithms

Both decoding algorithms introduced in last section require solving a system of n equations. In particular, we need to construct the kernal space of an $(n-1) \times n$ matrix in the PGZ Algorithm, and we construct an interpolation polynomial for the n pairs (γ_i, y_i) in Gao's Algorithm. These are expensive computations, which make the entire algorithms slow and impractical if n is large.

Recall that, in each step of the combinatorial approach to the problem, we consider a *block* of shares that contains fewer than n shares. We use additional shares (up to n or $k + 2t$, depending on the algorithm) only to verify the correctness of the solution. In this section, we combine this cover design methodology with the error-correcting code approach, in order to improve the overall running time.

We first illustrate the approach by reducing the problem of correcting t errors to a sequence of smaller subproblems, each of which involves correcting one error. Then we present a generalized version of the algorithm that combines both the error-correcting codes approach and the block construction technique.

4.1 A Combined Algorithm with Block-Size $k + 2$

Suppose we have a (k, n)-threshold scheme with at most t faulty shares. We partition all n shares into disjoint blocks of $k + 2$ shares (see Figure 1).

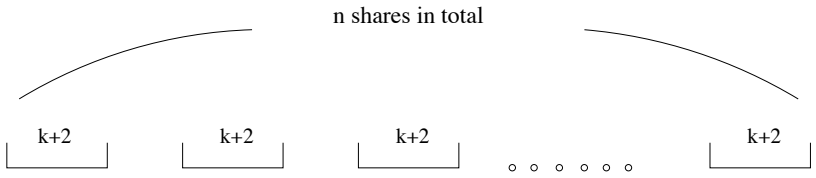

Figure 1: A partition of v blocks with block size $k+2$.

Let v denote the number of blocks. Assume that $v \geq \lceil \frac{t+1}{2} \rceil$, or equivalently, $n \geq (k+2)\lceil \frac{t+1}{2} \rceil$. Under this assumption, there exists a block that contains at most one faulty share. (Otherwise, if all blocks contain at least two faulty shares, then the total number of faulty shares is at least

$$2v = 2\left\lceil \frac{t+1}{2} \right\rceil \geq t+1 > t,$$

which exceeds the actual number of faulty shares.)

Suppose that the block b contains at most one faulty share. Denote the share indices for block b by $(b_1,\ b_2,\ \ldots,\ b_{k+2})$, and the corresponding share values by $(y_{b_1},\ y_{b_2},\ \ldots,\ y_{b_{k+2}})$. We can regard the shares in block b as shares in a $(k, k+2)$-threshold scheme, at most one of which is faulty, and apply any GRS decoding algorithm (e.g., either PGZ or Gao's Algorithm) to this block of shares to retrieve the secret polynomial. Then the resulting polynomial is the secret polynomial for the block b, as well as for the entire (k, n)-threshold scheme.

The following example illustrates the $(k+2)$-block construction.

Example 4.1. *Suppose we have a* $(5, 22)$*-threshold scheme over* $\mathbb{F}_{29}$ *in which at most four shares are faulty. First, we check that*

$$n = 22 \geq 21 = (5+2)\left\lceil \frac{4+1}{2} \right\rceil = (k+2)\left\lceil \frac{t+1}{2} \right\rceil. \tag{11}$$

Hence we can apply the $(k+2)$*-block construction with disjoint blocks of size* 7*.*

Suppose the secret polynomial is $P_0(x) = 1+2x+4x^2+8x^3+16x^4$*. The* 22 *shares ar:e*

$$(1,2),\ (2,22),\ (3,18),\ (4,12),\ (5,4),\ (6,1),\ (7,17),$$
$$(8,15),\ (9,23),\ (10,18),\ (11,13),\ (12,28),\ (13,3),\ (14,1),$$
$$(15,5),\ (16,5),\ (17,27),\ (18,17),\ (19,15),\ (20,10),\ (21,27),$$
$$(22,11).$$

Assume that four of them are changed, as follows:

$$(2, 28),\ (9, 14),\ (12, 27),\ (21, 22).$$

Observe that the 2^{nd}, 9^{th}, 12^{th} *and* 21^{st} *shares are faulty.*

To solve the threshold scheme with cheaters problem, we first construct four blocks of shares with block size 7. *(The fourth block has fewer than* 7 *shares, but the problem will be solved before hitting the last block.)*

Block $1 : (1, 2),\ (2, 28),\ (3, 18),\ (4, 12),\ (5, 4),\ (6, 1),\ (7, 17),$

Block $2 : (8, 15),\ (9, 14),\ (10, 18),\ (11, 13),\ (12, 27),\ (13, 3),\ (14, 1),$

Block $3 : (15, 5),\ (16, 5),\ (17, 27),\ (18, 17),\ (19, 15),\ (20, 10),\ (21, 22),$

Block $4 : (22, 11).$

We scan the shares block by block. For the first block, the share indices are $(1, 2, 3, 4, 5, 6, 7)$, *and the corresponding shares are* $(2, 28, 18, 12, 4, 1, 17)$. *We can use Gao's Algorithm, for example, to decode this particular block, which yields*

$$f_1(x) = 1 + 2x + 4x^2 + 8x^3 + 16x^4 = P_0(x).$$

The correctness of this polynomial can be verified by checking that it passes through all but three of the remaining shares. Therefore, we've found the secret polynomial, $f_1(x) = P_0(x)$, *by "solving" only one block of shares.*

If, on the other hand, we started with the second block, we would obtain a decoding failure because this block contains two faulty shares. However, as long as there is at least one block containing at most one faulty share, the algorithm succeeds. We ensure that this is the case by verifyting that (11) holds.

In the above example, we see that the original $(5, 22)$-threshold scheme is reduced to disjoint blocks of $(5, 7)$-threshold schemes, *i.e.,* the problem size shrinks from size 22 to subproblems of size 7.

We summarize this $(k+2)$-block construction as the following algorithm.

Algorithm 7 (A Combined Algorithm with Block-size $k + 2$).
Input: A (k, n)-threshold scheme over $\mathbb{F}_q$ with at most t faulty shares $(where n \geq (k + 2)\lceil \frac{t+1}{2} \rceil)$, with $\mathcal{S} = \{(x_i, y_i) : 1 \leq i \leq n\}$.
Output: The secret polynomial $P_0(x) \in \mathbb{F}_q[x]$.
Step 1: Let $v = (k+2)\lceil \frac{t+1}{2} \rceil$ denote the number of $(k+2)$-blocks. Construct a block system

$$\mathcal{T} = \{T_1,\ T_2,\ \ldots,\ T_v\}, \tag{12}$$

where $T_i = \{(x_j, y_j) : \ (i-1)(k+2)+1 \leq j \leq i(k+2)\} \subseteq \mathcal{S}$.
Step 2:

1. FOR i FROM 1 TO v DO
2. Retrieve block T_i
3. Try to solve the $(k, k+2)$-threshold scheme T_i, assuming at most one share is faulty
4. IF the decoding algorithm returns a polynomial $f(x)$ THEN
5. IF $|\{i : \ f(x_i) = y_i\}| \geq n - t$, THEN
6. RETURN f and QUIT
7. RETURN "Decoding Failure"

By our early analysis, it is guaranteed that there exists at least one block, namely, T_b $(1 \leq b \leq v)$, containing at most one faulty share. Hence the algorithm will terminate after processing the block T_b, and it finds the desired secret polynomial.

4.2 Generalized Combined Algorithm

In **Algorithm 7**, we construct v blocks, where each block contains $k+2$ shares, and search for a block that contains at most one faulty share. Let's define $t' = 1$, indicating the maximum number of faulty shares the algorithm can correct within a block.

What if we instead choose $t' = 2$? Then we need to find a block of shares that can contains at most two faulty shares. In order to correct up to two faulty shares in a block, we require the block size to be at least $k + 2 \times 2 = k + 4$. Therefore, we construct disjoint blocks of size $k + 4$. In order to ensure that there exists at least one block of size $k + 4$ that contains at most two faulty shares, we require v, the number of blocks, to be at least $\lceil \frac{t+1}{3} \rceil$. Under this block construction, we partition the original problem into v smaller schemes, at least one of which is a $(k, k+4)$-threshold scheme having at most two faulty shares.

This methodology can be generalized to arbitrary $t' \leq t$. In general, the block size needs to be $k + 2t'$. Solving the (k, n)-threshold scheme with at most t faulty shares reduces to solving a series of $(k, k + 2t')$-threshold schemes, at least one of which has at most t' faulty shares. The following easy lemma gives a sufficient condition for this method to succeed if we partition the shares into v disjoint blocks of size $k + 2t'$.

Lemma 4.1. *Suppose that* $n \geq (k + 2t')(\lceil \frac{t-t'}{t'+1} \rceil + 1)$. *Suppose we partition the shares of a* (k, n)*-threshold scheme having at most* t *faulty shares into* $\lceil \frac{t-t'}{t'+1} \rceil + 1$ *disjoint blocks of size* $k + 2t'$. *Then there exists at least one block*

that contains at most t' faulty shares.

Proof. Let v be the minimum number of blocks required. In the worst case, exactly one block contains t' shares and all other blocks have $t'+1$ faulty shares. Therefore, all we need is to ensure the following:

$$\begin{aligned}(v-1)(t'+1)+t' &\geq t \\ v-1 &\geq \frac{t-t'}{t'+1} \\ v &\geq \left\lceil \frac{t-t'}{t'+1} \right\rceil + 1.\end{aligned}$$

This in turn implies the lower bound for n,

$$n \geq (k+2t')\left(\left\lceil \frac{t-t'}{t'+1} \right\rceil + 1 \right). \tag{13}$$

□

Now we present **Algorithm 8**, our general combined algorithm.

Algorithm 8 (Generalized Combined Algorithm).
Input: A (k,n)-threshold scheme $\mathcal{S} = \{(x_i, y_i) : 1 \leq i \leq n\}$ over $\mathbb{F}_q$, which has at most t faulty shares, and an integer $t' \leq t$ such that (13) is satisfied.

Output: The secret polynomial $P_0(x) \in \mathbb{F}_q[x]$.
Step 1: Let $v = (k+2t')(\lceil \frac{t-t'}{t'+1} \rceil + 1)$ indicate the number of blocks. Construct a system of v disjoint blocks, with each block size $\lceil \frac{t-t'}{t'+1} \rceil + 1$. Namely, let

$$\mathcal{T} = \{T_1,\ T_2,\ \ldots,\ T_v\},$$

where $T_i = \{(x_j, y_j) : \ (i-1)(k+2t')+1 \leq j \leq i(k+2t')\} \subseteq \mathcal{S}$.
Step 2:

1. FOR i FROM 1 TO v DO
2. Retrieve block T_i
3. Try to solve the $(k, k+2t')$-threshold scheme T_i,
   assuming at most t' shares are faulty
4. IF the decoding algorithm returns a polynomial $f(x)$ THEN
5. IF $|\{i : \ f(x_i) = y_i\}| \geq n-t$, THEN
6. RETURN f and QUIT
7. RETURN "Decoding Failure"

Notice that when $t' = t$, we are solving the system of one block of $k+2t$ shares to discover the secret polynomial. When $t' = 0$, the combined algorithm corresponds to the original covering design approach (**Algorithm 1**). Finally **Algorithm 7** is the special case of **Algorithm 8** with $t' = 1$.

5 Experimental Analysis

In this section, we provide experimental results on the actual running times of most of the algorithms discussed in previous seections. Some papers ([11] and [13]) have addressed the complexity of the combinatorial covering approach, based on the number of iterations in the relevant algorithms. We decided to develop an implementation of threshold schemes as well as the algorithms used to solve the threshold scheme with cheaters problem, in order to evaluate their actual performance in practice.

In our experiment, we constructed (k, n)-threshold schemes, which included:

- construction of secret polynomial $P_0(x) \in \mathbb{F}_q[x]$;
- construction of n shares $\{(x_i, y_i) \ : \ 1 \leq i \leq n\}$;
- random selection of t shares which become faulty.

We ran each algorithm multiple times and we compared the average running time of various algorithms. All the algorithms were implemented in Maple.

5.1 Results

All the experiments were done on a P4 2.2 Ghz windows-based machine. To achieve better results, we compared each algorithm by using the same "scheme setup", and we ran the algorithm 20 times, computing the average running time. Since the running time of the algorithms might be affected by the "random" polynomial generated by the "scheme setup" file, we repeated this process four times, *i.e.,* we constructed four different threshold schemes, in order to make the results more independent of the choice of schemes.

We are interested mostly in the relative difference between the running time of the different algorithms. Throughout the experiments, k, t and n correspond to the parameters of a (k, n)-threshold scheme having at most

t faulty shares, q refers to the field $\mathbb{F}_q$, and the running time is recorded in seconds.

In summary, here are the algorithms considered in this paper:

label	algorithm	section
A1	$(n, n-k, t)$-covering	§2.1
A2	randomized algorithm	§2.2
A3	$(k+2t, 2t, t)$-covering (Tso et al)	§2.3
A4	$(v, v-k, t)$-covering $(v \geq min\{k(t+1/2), k+2t\})$	§2.4
A5	PGZ decoding algorithm	§3.2
A6	Gao's decoding algorithm	§3.3
A7/PGZ	combined algorithm with PGZ (≤ 1 error/block)	§4.1
A7/Gao	combined algorithm with Gao (≤ 1 error/block)	§4.1
A8/PGZ	combined algorithm with PGZ ($\leq t'$ errors/block)	§4.2
A8/Gao	combined algorithm with Gao ($\leq t'$ errors/block)	§4.2

We did not run implement A3 or A4, but we tested all the other algorithms. Or first results are recorded in Table 1.

From Table 1, we see that the running time of A1 is very close to the running time of A2. Especially when n becomes large, A2 achieves very good performance because the probability of including a faulty share in k selected shares is extremely small.

Another interesting point is that when n gets large, A5 and A6 are very slow, due to the required computations of a system of linear equations in n unknowns or interpolating a polynomial of degree $n-1$. The performance of A7 is much better, but it still is not competitive with A1 and A2 for the parameters considered in Table 1.

It seems that A6 usually is at least 4–5 times faster than A5. However, we suspect that this is probably due to the way certain operations are implemented in Maple, rather than being an intrinsic propery of the algorithms.

Notice that we tested the running times of the algorithms using different values of q. The results show that the computation time is affected by the value of q, but the relative difference between the running time of different algorithms remains the same.

Next, we compared the running times of algorithms in the case when $n = k+2t$. As mentioned earlier, when n is close to $k+2t$, the construction of a covering design becomes extremely difficult in general. Therefore, only A2, A5 and A6 are suitable (in general) when $n = k+2t$. These results are

n	k	t	q	A1	A2	A5	A7/PGZ	A6	A7/Gao
10	3	2	331	0.00213	0.00409	0.05233	0.02578	0.01074	0.01211
11	3	2	331	0.00173	0.00156	0.05060	0.02049	0.01251	0.00939
50	3	2	331	0.00235	0.00235	1.78900	0.01715	0.49600	0.00630
100	3	2	331	0.00705	0.00545	24.06175	0.01795	9.59375	0.02270
10	3	2	1009	0.00234	0.00179	0.02713	0.01505	0.01013	0.00900
11	3	2	1009	0.00176	0.00195	0.03460	0.01523	0.00879	0.00469
50	3	2	1009	0.00470	0.00625	3.43775	0.03360	0.53500	0.01090
100	3	2	1009	0.01090	0.00935	24.78875	0.03360	9.29275	0.02035
12	4	2	17	0.00334	0.00585	0.04960	0.02266	0.01388	0.00975
15	5	2	17	0.00391	0.00548	0.05469	0.02518	0.01544	0.00878
16	5	2	17	0.00470	0.00449	0.07815	0.02616	0.02149	0.01056
12	4	2	331	0.00330	0.00430	0.05703	0.02694	0.01233	0.01229
15	5	2	331	0.00271	0.00646	0.07420	0.02734	0.01641	0.01331
16	5	2	331	0.00608	0.00546	0.09045	0.02764	0.02225	0.01154
12	4	2	1009	0.00176	0.01114	0.04688	0.02364	0.01153	0.00545
15	5	2	1009	0.00275	0.00430	0.20585	0.02111	0.01228	0.00763
16	5	2	1009	0.00488	0.00588	0.09296	0.02850	0.02168	0.01151
12	4	2	100000007	0.00275	0.00701	0.06018	0.02306	0.01603	0.01035
15	5	2	100000007	0.00409	0.00918	0.08909	0.03146	0.02461	0.00781
16	5	2	100000007	0.00324	0.00371	0.09906	0.03123	0.02226	0.01406
100	4	2	331	0.00775	0.00470	27.71800	0.03985	8.64450	0.01175
100	6	4	331	0.01580	0.01020	25.27850	0.03520	9.06625	0.01325
100	6	4	1009	0.00855	0.01015	29.45600	0.02970	8.41400	0.01330
100	6	4	100000007	0.00945	0.01325	41.68725	0.04385	8.94900	0.01875
132	12	10	331	0.26010	0.11485	91.00825	0.11720	159.79625	0.20475

Table 1: Comparison of algorithms when $n \geq k(t+1)$

n	k	t	q	A2	A5	A6
7	3	2	331	0.00763	0.09669	0.01075
8	4	2	331	0.00859	0.03280	0.00881
9	5	2	331	0.01309	0.03456	0.01328
10	6	2	331	0.01074	0.02385	0.00605
12	6	3	331	0.03030	0.05019	0.01328
14	6	4	331	0.05160	0.06523	0.01366
16	6	5	331	0.03575	0.10724	0.01660
18	6	6	331	0.04548	0.10509	0.01796
20	6	7	331	0.06346	0.16579	0.03538

Table 2: Comparison of three algorithms when $n = k + 2t$

presented in Table 2.

From Table 2, we see that all three algorithms perform reasonably well. However, as the number of faulty shares, t, becomes larger, A6 seems to have the best performance.

Lastly, we compared the running times of A7 using different values of t'; see Table 3.

In Table 3, we see that, for the same (n, k, t)-triples, a smaller value of t' usually results in a faster performance. In general, it seems to be a good strategy to choose t' to be the smallest integer such that the inequality (13) is satisfied. For example, when $(n, k, t) = (84, 20, 10)$, then we cannot apply the combined algorithm with $t' < 4$. So we only consider $t' = 4, 5, \ldots$ for this parameter triple. Here, $t' = 4$ gives the fastest running times.

Analogous to the results in our other comparisons, A7 seems to be faster when used in conjunction with Gao's Algorithm than it is with the PGZ algorithm. However, as mentioned above, we are hesitant to conclude that means that Gao's Algorithm is inherently faster, as the results depend strongly on the efficiency of the underlying Maple computations.

6 Conclusion

In this paper, we studied several approaches to the threshold scheme with cheaters problem. We provided three basic strategies to solve it:

- the combinatorial approach using covering designs;

n	k	t	q	t'	A8/PGZ	A8/Gao
132	20	10	10007	1	0.19571	0.11128
132	20	10	10007	2	0.24181	0.09666
132	20	10	10007	3	0.26171	0.10839
132	20	10	10007	4	0.37846	0.11603
132	20	10	10007	5	0.39748	0.13536
132	20	10	10007	9	0.80195	0.21191
104	20	10	10007	3	0.25818	0.08290
104	20	10	10007	4	0.37559	0.11600
104	20	10	10007	5	0.41233	0.14863
104	20	10	10007	9	0.86555	0.22539
84	20	10	10007	4	0.41798	0.11180
84	20	10	10007	5	0.50775	0.16095
84	20	10	10007	9	1.09471	0.20076
76	20	10	10007	9	0.84545	0.19380

Table 3: Comparison of the Combined Algorithm using different t'

- the error-correcting code approach using generalized Reed-Solomon code decoding; and
- methods that combine the presious two approaches.

The strengths and weaknesses of the various approaches were discussed and analyzed.

Acknowledgements

The research of the first author is supported by NSERC discovery grant NSERC-RGPIN #203114-02.

References

[1] Mikhail J. Atallah. *Algorithms and Theory of Computation Handbook*. CRC Press LLC, 1999.

[2] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[3] G. R. Blakley. Safeguarding cryptographic keys. In *AFIPS Conference Proceedings*, vol. 48, pp. 313–317, 1979.

[4] Shuhong Gao. A new algorithm for decoding Reed-Solomon codes. *Communications, Information and Network Security* **712** (2003), 55–68.

[5] D. C. Gorenstein and N. Zierler. A class of error-correcting codes in p^m symbols. *SIAM Journal on Applied Mathematics* **9** (1961), 207–214.

[6] W. C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[7] C. D. Mayer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000.

[8] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM* **24** (1981), 583–584.

[9] W. H. Mills. Covering designs I: coverings by a small number of subsets. *Ars Combinatoria* **8** (1979), 199–315.

[10] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM Journal on Applied Mathematics*, **8** (1960), 300–304.

[11] R. S. Rees, D. R. Stinson, R. Wei, and G. H. J. van Rees. An application of covering designs: determining the maximum consistent set of shares in a threshold scheme. *Ars Combin.* **53** (1999), 225–237.

[12] A. Shamir. How to share a secret. *Communications of the ACM* **22** (1979), 612–613.

[13] R. Tso, Y. Miao, and E. Okamoto. A new algorithm for searching a consistent set of shares in a threshold scheme with cheaters. *Lecture Notes in Computer Science*, **2971** (2004), 377–385.