Secure Multi-Party Computations using Secret Sharing Schemes

A coding theoretical approach

Master's Thesis
Aalborg University

Johan M. Sorknæs
September 2016

AALBORG UNIVERSITY
STUDENT REPORT

Department of Mathematical Sciences
Aalborg University
Fredrik Bajers Vej 7G
Phone number 99 40 99 40
http://www.math.aau.dk

Title Secure Multi-Party Computations using Secret Sharing Schemes - A coding theoretical approach

Topic Secure multi-party computation

Project period
February 2016 - September 2016

Project group

Johan Milton Sorknæs

Supervisor Diego Ruano

Circulation 3

Total number of pages 88

Appendix None

Completed September 11, 2016

Synopsis

This master's thesis concerns secure multi-party computations (MPC) using secret sharing schemes (SSS) based on linear codes.
We introduce actively and passively secure protocols for MPC throughout the project, and give several examples using linear SSSs based on linear codes, particularly a ramp version of Massey's SSS.
First we consider an actively secure protocol for MPC using linear SSS, however, this only allows for addition and scalar multiplication. Therefore, we introduce multiplicative and strongly multiplicative SSSs, which allows us to construct passively and actively secure protocols for any function. Since multiplicative SSSs, and their associated matrices, are difficult to find we give several examples of multiplicative SSSs in the project.
Lastly, we introduce a passively secure protocol for MPC, which reduces the communication between participants by grouping multiplications.

Preface

This report serves as the master's thesis of Johan Milton Sorknæs, and has been composed from February 2016 to September 2016 as part of the masters program in applied mathematics, at the department of Mathematical Sciences at Aalborg University.

The topic of the report is secure multi-party computations. In this report both actively and passively secure multi-party computation protocols are presented, each using linear secret sharing schemes constructed from linear codes. The thesis is a continuation of work from the preceding semester of the masters program, in which the basics of secret sharing schemes and their security were studied. Due to this, the chapter on secret sharing schemes has been condensed to the essentials.

Bibliographical references in the report are noted with [number]. The number refers to the numbered bibliography which can be found at the end of the report. The page numbers in the bibliography refers to the pages in the report, where the given source is cited.

It is expected that the reader has basic knowledge of coding theory, information theory, and general abstract algebra, including finite fields.

I would like to thank my supervisor, Associate Professor Diego Ruano, whom has been a great help during the thesis.

Danish Abstract

Det generelle formål med denne specialeafhandling er at konstruere protokoller til sikkert distribueret beregninger ved brug af secret sharing ordninger baseret på lineære koder i endelige legemer, heriblandt en udvidet udgave af Masseys secret sharing. Specialet starter med en introduktion af secret sharing ordninger generelt, efterfulgt af to konstruktion af lineære secret sharing ordninger baseret på lineære koder. En secret sharing ordning opdeler en hemmelighed i flere delinger, således at visse delmængder af disse delinger kan rekonstruere hemmeligheden. Motivationen for at betragte disse typer af secret sharing ordninger er at kunne bruge teoretiske kodnings begreber til rekonstruktion i ordningerne, altså korrektion af sletning og fejl. Denne form for rekonstruktion i disse ordninger gennemgås sideløbende med sikkerheden af en secret sharing ordning.

Herefter introduceres sikkert distribueret beregninger, som er en beregning af en offentlig funktion afhængig af privat data uden at noget information om de private data lækker. Der gives desuden en aktivt sikker protokol for funktioner bestående af addition og skalar multiplikation ved brug af lineære secret sharing ordninger. Det bevises, vha. Lagrange interpolation, at enhver funktion i et endeligt legeme er ækvivalent med et polynomium af grad mindre end antallet af elementer i legemet. Altså, kan enhver funktion i et endeligt legeme løses med blot addition og multiplikation. Derfor introduceres multiplikative secret sharing ordninger, for hvilke produktet af hemmeligheders delinger kan rekonstruere produktet af de tilsvarende hemmeligheder. Ved brug af denne egenskab præsenteres en passivt sikker protokol for alle funktioner ved brug af multiplkative secret sharing ordninger.

Da det at finde en multiplikativ secret sharing ordning kan være besværligt, gennemgås der for kodetyperne Reed-Solomon og Reed-Muller, hvornår de udvidede Massey secret sharing ordninger baseret på disse koder er multiplikative. For Reed-Solomon koder bevises det, at hvis dimensionen af koden er under en hvis grænse, så vil secret sharing ordningen være multiplikativ. For Reed-Muller koder bevises det, hvilke koder er selv-ortogonale, hvilket

det bevises, at hvis en kode er selv-ortogonal, så er Masseys secret sharing ordning, som er baseret på koden, multiplikativ. For begge kodetyper vises ligeledes deres tilhørende rekombinations matricer.

For aktivt sikkert distribueret beregninger, der bygger på secret sharing ordninger, introduceres en distribution af delinger i en secret sharing ordning, hvor alle deltager i secret sharing ordningen er i stand til at verificere de delinger, som de modtager. Med denne distribution af delinger præsenteres også en aktivt sikker protokol for alle funktioner ved brug af stærk multiplikative secret sharing ordninger.

Slutteligt reduceres mængden af kommunikation mellem deltagerne i en sikker distribueret beregning ved at bruge en tungere definition af multiplikative secret sharing ordninger. Med denne definition introduceres en passivt sikker protokol for alle funktioner ved brug af secret sharing ordninger, der opfylder denne tungere definition af multiplkativ. Denne protokol tillader gruppering af flere multiplikationer, hvilket reducerer mængden af gange, som deltagerne behøver at kommunikere. Der gives ligeledes en kodetype, som konstrueres ud fra binære Reed-Muller koder. For denne kodetype vises der, at ved visse valg, kan koderne bruges til en secret sharing ordning, som opfylder denne tungere definition af multiplikativ. Det vises desuden også, hvordan man finder disse valg.

Contents

1 Secret Sharing Schemes

A secret sharing scheme (SSS) is a method for sharing a secret value among n participants by constructing shares from the secret and distributing them among the participants, such that only specific subsets of the participants are able to reconstruct the secret using their shares.

A SSS can only share a secret that is an element of its secret space, noted $\mathbb{S}$. The secret is a string of length ℓ, and in this project we generally use $\mathbb{S} = \mathbb{F}_q^\ell$. If $\ell > 1$ the scheme is called a ramp SSS, and if $\ell = 1$ it is called a non-ramp SSS. Given a secret $s \in \mathbb{S}$, the set X_s is the set of possible share vectors for the secret s, each element in a share vector is equal to a participant's share of the secret. In this project we generally use $X_s \subset \mathbb{F}_q^n$.

Definition 1.1. *A SSS has t-privacy and r-reconstruction if*

- *t is largest possible, such that any subset of t or less participants are unable to recover any information about the secret,*
- *and r is smallest possible, such that any subset of r or more participants can reconstruct the secret.*

A SSS with t-privacy and r-reconstruction is called a (t, r)-SSS. If $r = t - 1$ it is called a t-threshold SSS. For ramp schemes we introduce a more general definition of privacy and reconstruction.

Definition 1.2. *A SSS has $(t_1, \ldots, t_\ell)$-privacy and $(r_1, \ldots, r_\ell)$-reconstruction if*

- *t_m is largest possible, such that any subset of t_m or less participants are unable to recover m or more bits of information about the secret,*
- *and r_m is smallest possible, such that any subset of r_m or more participants are able to recover m or more bits of information,*

where $1 \leq m \leq \ell$.

Note, that in this notation $t_1 = t$ and $r_\ell = r$. In this project we will work in $\mathbb{F}_q$, hence the bits of information mentioned above are q-bits. Since the privacy and reconstruction only give the size of subsets that are or are not able to recover m q-bits of information of the secret, we introduce the access structure for a complete picture of the information that a subset is able to recover.

Definition 1.3. *The access structure of a SSS with secret length ℓ is $\Gamma = \{\mathbb{A}_0, \ldots, \mathbb{A}_\ell\}$, where $\mathbb{A}_m$ is the family of subsets of participants able to recover exactly m q-bits of information, for $1 \leq m \leq \ell$.*

Note that any set $A \in \mathbb{A}_\ell$ can reconstruct the secret. These sets are called accepted sets. Any set $A \in \mathbb{A}_0$ are unable to recover any information about the secret, these sets are called rejected sets.

Using the notation of the access structure we can write the t_m and r_m as

$$t_m = \min_{A \subseteq \mathcal{I}} \left\{ |A| : A \in \bigcup_{i=m}^{\ell} \mathbb{A}_i \right\} - 1,$$
$$r_m = \max_{A \subseteq \mathcal{I}} \left\{ |A| : A \notin \bigcup_{i=m}^{\ell} \mathbb{A}_i \right\} + 1,$$

for $1 \leq m \leq \ell$, where $\mathcal{I} = \{P_1, \ldots, P_n\}$, which is the set of participants. Generally we will use the notation $\mathcal{I}$ for the set of participants going forward.

A set $A \subseteq \mathcal{I}$ being able to recover m q-bits of information about the secret, is equivalent to reducing the number of possible secrets from q^ℓ to $q^{\ell-m}$. This is a reduction of uncertainty of the secret, and using notation from information theory we can write this reduction as

$$I_q\left(\vec{S}, \phi(A)\right) = H_q\left(\vec{S}\right) - H_q\left(\vec{S}|\phi(A)\right) = \ell - (\ell - m) = m,$$

where $\vec{S}$ is the discrete random variable for the secret, and $\phi(A)$ is the shares of the participants in A. Since we work in $\mathbb{F}_q$, we generally omit the q when using entropy and mutual information, i.e. $H_q(\vec{S}) = H(\vec{S})$. Using mutual information we can define the sets $\mathbb{A}_m$, for $0 \leq m \leq \ell$, as

$$A \in \mathbb{A}_m \Leftrightarrow I\left(\vec{S}, \phi(A)\right) = m, \quad 0 \leq m \leq \ell,$$

where $A \subseteq \mathcal{I}$.

In general, if the SSS is not designed after a certain access structure, the access structure is difficult to compute, hence the privacy and reconstruction (or often bounds hereof) are used instead.

For certain applications of SSSs we want the properties of linearity, hence we want the SSS to be linear.

Definition 1.4. *A SSS is linear if for any $s, s' \in \mathbb{S}$ and $\lambda \in \mathbb{N}$*

$$X_s + X_{s'} \subseteq X_{s+s'}, \quad \textit{and}$$
$$\lambda X_s \subseteq X_{\lambda s}.$$

1.1 $LSSS(C)$

To be sure that the chosen SSS is always linear, we introduce a linear SSS obtained by a linear code. This type of SSS, as well as an improved version, will be used throughout the project.

Let C be an $[n+\ell, k, d]$ code over $\mathbb{F}_q$, with $\ell < d^\perp$, where $d^\perp$ is the minimum weight of the dual code. The code C, is generated by the rows of a generator matrix

$$G = \left[\boldsymbol{e}_1^k, \ldots, \boldsymbol{e}_\ell^k, g_{\ell+1}, \ldots, g_{n+\ell}\right]$$

where $\boldsymbol{e}_i^k$ is the ith standard vector of $\mathbb{F}_q^k$, for $1 \leq i \leq \ell$, and $g_j \in \mathbb{F}_q^k$, for $\ell+1 \leq j \leq n+\ell$. Note, that since $\ell < d^\perp$ a generator matrix for C can always be given on this form by row operations and permutations.

The generator matrix G motivates a simple construction of a linear SSS with n participants, where $\mathbb{S} = \mathbb{F}_q^\ell$. Namely, a SSS where the secret is equal to the first ℓ coordinates of a codeword, and the shares are equal to the remaining n coordinates. This is a ramp version of Massey's secret sharing scheme [11].

Definition 1.5. *[3] Given an $[n+\ell, k, d]$ code C over $\mathbb{F}_q$, with $\ell < d^\perp$, where $d^\perp$ is the minimum weight of the dual code $C^\perp$.*

The linear SSS with $\mathbb{S} = \mathbb{F}_q^\ell$ and $X_s = \left\{\tilde{c} \in \mathbb{F}_q^n : (s, \tilde{c}) \in C\right\}$, for any $s \in \mathbb{S}$ is denoted $LSSS(C)$.

The reason for using linear codes for SSSs is not only that it is always linear, but also that we gain the properties of linear codes, e.g. error and erasure correction. Hence, if a subset of participants is trying to reconstruct a secret in $LSSS(C)$, they would simply use erasure correction.

Therefore, the following methodology for reconstruction will recover any information that the subset of participants trying to reconstruct the secret can recover. Note, that the solution of the secret, s, might include some free variables, and the number of q-bits of information recovered is equal to $\ell - m$, where m is the number of free variables in s.

For the methodology to be possible a generator matrix for C is needed. Since the generator matrix used for the SSS does not contain any information about the secret, it can be considered public information in $LSSS(C)$.

Let $A \subseteq \mathcal{I}$ be the set of participants trying to reconstruct the secret.

1. Construct vector $y \in \mathbb{F}_q^n$, where y_i is the share of P_i, if $P_i \in A$, and let y_i be unknown otherwise, for $1 \leq i \leq n$.
2. Construct a parity check matrix H for C. Solve the equation system $(s, y)H^T = 0$, where $s = (s_1, \ldots, s_\ell) \in \mathbb{F}_q^\ell$ is unknown.
3. Output any solutions found for $(s_1, \ldots, s_\ell)$.

Note, that the time complexity of this methodology is dominated by the construction of the parity check matrix for C, which in turn is dominated by Gaussian elimination, hence the methodology has time complexity $\mathcal{O}(k^3)$. Furthermore, if the SSS is used more than once, then the parity check matrix H, computed during reconstruction, can be stored as public information for later use.

Example 1.6

We want to show how to share a secret s in an $LSSS(C)$ over $\mathbb{F}_q$, and how some subsets of participants will attempt to reconstruct the secret. Therefore, we first need a linear code C. Let C be an $[8,4]$ code over $\mathbb{F}_7$, with generator matrix

$$G = \begin{bmatrix} 1 & 0 & 6 & 2 & 0 & 4 & 4 & 4 \\ 0 & 1 & 1 & 6 & 3 & 3 & 3 & 1 \\ 0 & 0 & 1 & 3 & 0 & 2 & 3 & 1 \\ 0 & 0 & 6 & 2 & 1 & 3 & 6 & 0 \end{bmatrix}.$$

Note, that we have chosen a generator matrix, where the choice of ℓ is easy, hence we let $\ell = 2$.

Let $s = (5,5) \in \mathbb{F}_7^2$ be the secret we want to share. In order to construct a share vector we choose $x \in \mathbb{F}_7^{\ell-k=2}$ uniformly at random, and compute a share vector $\tilde{c}$, where $(s,x)G = (s,\tilde{c})$.

In this case we use $x = (3,2)$, so we get

$$(s,x)G = (5,5,3,2)\begin{bmatrix} 1 & 0 & 6 & 2 & 0 & 4 & 4 & 4 \\ 0 & 1 & 1 & 6 & 3 & 3 & 3 & 1 \\ 0 & 0 & 1 & 3 & 0 & 2 & 3 & 1 \\ 0 & 0 & 6 & 2 & 1 & 3 & 6 & 0 \end{bmatrix} = (5,5,1,4,3,5,0,0),$$

and hence the share vector is $\tilde{c} = (1,4,3,5,0,0)$.

We will now show two attempts of reconstruction using the methodology above. Therefore, we need to produce a parity check matrix for the code C. By Gaussian elimination we get the parity check matrix

$$H = \begin{bmatrix} 1 & 6 & 2 & 4 & 1 & 0 & 0 & 0 \\ 6 & 2 & 1 & 6 & 0 & 1 & 0 & 0 \\ 2 & 4 & 1 & 1 & 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 4 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Let $A = \{P_1, P_3, P_6\}$ attempt to reconstruct the secret. Using the methodology we get the vector $y = (1, y_2, 3, y_4, y_5, 0)$, and thus

$$(s, y)H^T = (s_1, s_2, 1, y_2, 3, y_4, y_5, 0) \begin{bmatrix} 1 & 6 & 2 & 3 \\ 6 & 2 & 4 & 2 \\ 2 & 1 & 1 & 1 \\ 4 & 6 & 1 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = 0.$$

Solving the equation system we get

$$s_1 = 5 + 2y_5, \quad s_2 = 5 + y_5, \quad y_2 = 4 + 5y_5, \quad \text{and} \quad y_4 = 5 + 5y_5.$$

Notice, that the solution for the secret has one free variable when the set A attempts reconstruction. Therefore, $A \in \mathbb{A}_1$, since A achieves 1 q-bit of information about the secret.

Now let the set of participants $B = \{P_1, P_3, P_4, P_6\}$ attempt to reconstruct the secret. We get the equation system

$$(s, y)H^T = (s_1, s_2, 1, y_2, 3, 5, y_5, 0) \begin{bmatrix} 1 & 6 & 2 & 3 \\ 6 & 2 & 4 & 2 \\ 2 & 1 & 1 & 1 \\ 4 & 6 & 1 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = 0.$$

Solving the system we get

$$s_1 = 5, \quad s_2 = 5, \quad y_2 = 4, \quad \text{and} \quad y_5 = 0.$$

Hence, B is able to reconstruct the secret, i.e. $B \in \mathbb{A}_2$. $\triangle$

We give a proof for a lower bound for the privacy and an upper bound for the reconstruction in $LSSS(C)$. These bounds are useful to prove, since they are sharp for some types of linear codes.

Theorem 1.7. *[3] $LSSS(C)$ has $(\geq d^{\perp}-\ell-1)$-privacy and $(\leq n+\ell-d+1)$-reconstruction.*

Proof. Let C be an $[n+\ell, k, d]$ code over $\mathbb{F}_q$. Assume $c, c' \in C$, such that c and c' agree in at least $n+\ell-(d-1)$ coordinates. Since C is linear $c-c' \in C$, however

$$w_H(c-c') \leq n+\ell-(n+\ell-d+1) = d-1,$$

thus no two codewords can agree in $n+\ell-d+1$ coordinates. Therefore, for any $n+\ell-d+1$ shares, there exists exactly one unique $c \in C$. Hence, the reconstruction of $LSSS(C)$ is $\leq n+\ell-d+1$.

Since any $d^{\perp}-1$ columns of the generator matrix G are linearly independent, the ℓ columns generating s and any other $d^{\perp}-\ell-1$ columns are linear independent. Hence, since each column generates either an element of the secret or a share, it follows that any $d^{\perp}-\ell-1$ shares are independent of the secret, and thus give no information. Therefore, the privacy of $LSSS(C)$ is $\geq d^{\perp}-\ell-1$. □

A maximum distance separable (MDS) $[n+\ell, k, d]$ code C has $d = n+\ell-k+1$, and the dual code, $C^{\perp}$, of an MDS code is also an MDS code, hence $d^{\perp} = n+\ell-(n+\ell-k)+1 = k+1$. Therefore, the following lemma follows from theorem 1.7.

Lemma 1.8. *Let C be an $[n+\ell, k]$ MDS code, then $LSSS(C)$ has $(k-\ell)$-privacy and k-reconstruction.*

Proof. Let C be an $[n+\ell, k]$ MDS code, then $d = n+\ell-k+1$ and $d^{\perp} = k+1$. Let $LSSS(C)$ have t-privacy and r-reconstruction, from theorem 1.7 we get the bounds

$$\begin{aligned} t &\geq d^{\perp}-\ell-1 = k+1-\ell-1 = k-\ell, \\ r &\leq n+\ell-d+1 = n+\ell-(n+\ell-k+1)+1 = k. \end{aligned}$$

Since the amount of information given by a share can never be more than one q-bit the bounds has to be sharp, hence the lemma follows. □

1.2 $LSSS(\hat{C}, C)$

The following SSS is constructed using an $[n, \hat{k}, \hat{d}]$ code $\hat{C}$ and a non-empty subcode $C \subset \hat{C}$. The secret is then hidden in another subcode $S \subset \hat{C}$, such that $S \oplus C = \hat{C}$. The advantage of this construction, compared to $LSSS(C)$, is that the dimension of the secret space is bounded by the dimension of the code, rather than the minimum weight of its dual code. Additionally, since the secret is not directly hidden in the codewords, the codelength is shorter, since each coordinate of a codeword is a share.

Definition 1.9. *[3] Given two linear codes $C \subset \hat{C} \subseteq \mathbb{F}_q^n$, with $\dim C > 0$ and $\dim \hat{C} - \dim C = \ell$, and an arbitrary isomorphism $\psi : \mathbb{F}_q^\ell \to S$, where $S \oplus C = \hat{C}$.*

The linear SSS with $\mathbb{S} = \mathbb{F}_q^\ell$ and $X_s = \psi(s) + C$, for any $s \in \mathbb{S}$, is denoted $LSSS(\hat{C}, C)$.

An exciting property of the $LSSS(\hat{C}, C)$ construction of linear SSSs is, that, as shown in the following theorem, any linear SSS is equivalent to a SSS on the form $LSSS(\hat{C}, C)$. The theorem was first stated in [3] and then elaborated in [7], here we present a detailed proof for completeness.

Theorem 1.10. *Any linear SSS over $\mathbb{F}_q$ with $\mathbb{S} = \mathbb{F}_q^\ell$ and n participants is equivalent to $LSSS(\hat{C}, C)$ for some linear codes $C \subset \hat{C} \subseteq \mathbb{F}_q^n$, where $\dim \hat{C} - \dim C = \ell$.*

Proof. Let $s, s' \in \mathbb{F}_q^\ell$ be two secrets in a linear SSS, with X_s and $X_{s'}$ as the sets of possible share vectors for each secret respectively. Then, since the SSS is linear the following holds for all $\alpha, \beta \in \mathbb{F}_q$

$$\alpha X_s + \beta X_{s'} \subseteq X_{\alpha s + \beta s'}.$$

Let $\hat{C} = \cup_{s \in \mathbb{F}_q^\ell} X_s$ and $C = X_0$, it is clear that $\hat{C}$ and C are linear since the SSS is linear, and that $C \subset \hat{C} \subseteq \mathbb{F}_q^n$. Moreover, since $|X_s| = |X_{s'}|$ for all $s, s' \in \mathbb{F}_q^\ell$ due to linearity, and $X_s \cap X_{s'} = \emptyset$ due to reconstruction in a SSS, it follows that

$$|\hat{C}| = q^\ell |C|.$$

Since the SSS is over $\mathbb{F}_q$ and the codes are linear, $\dim \hat{C} = \log |\hat{C}|$ and $\dim C = \log |C|$, hence

$$\begin{aligned} & |\hat{C}| = q^\ell |C| \\ \Leftrightarrow & \log |\hat{C}| = \log(q^\ell |C|) \\ \Leftrightarrow & \dim \hat{C} = \ell + \dim C \\ \Leftrightarrow & \dim \hat{C} - \dim C = \ell. \end{aligned}$$

Let $S \subset \hat{C}$, where $S \oplus C = \hat{C}$. Note that $\dim S = \ell$, hence $|S| = q^\ell$, and from $S \oplus C = \hat{C}$ it then follows that if $x, y \in S$ and $x, y \in X_s$, for some $s \in \mathbb{F}_q^\ell$, then $x = y$. Therefore, we can construct the following isomorphism.

Let ψ be an isomorphism $\psi : \mathbb{F}_q^\ell \to S$, such that $\psi(s) \in X_s$, for any $s \in \mathbb{F}_q^\ell$. Then $\psi(s) + c$, where $c \in C = X_0$ is chosen at random, is a share vector for the secret s, since, from linearity of the SSS, $\psi(s) + c \in X_{s+0} = X_s$.

Hence, from the choice of ψ any secret $s \in \mathbb{F}_q^\ell$ has the same possible share vectors in the original linear SSS as in $LSSS(\hat{C}, C)$ for the linear codes $\hat{C} = \cup_{s \in \mathbb{F}_q^\ell} X_s$ and $C = X_0$, and equivalence follows. □

We now introduce a general methodology that can be used for reconstruction of secrets in an $LSSS(\hat{C}, C)$. Recall that not all subsets of participants are able to reconstruct the secret, i.e. if a subset of participants $A \in \mathbb{A}_m$ tries to reconstruct the secret, they will recover m q-bits of information about the secret, for $1 \leq m \leq \ell$.

The methodology uses coding theory to reconstruct the secret. Therefore, the shares are considered as a received word, where any missing shares are seen as erasures. These missing shares are those of participants whom are not involved in the reconstruction.

The methodology first reduces the number of erasures, if possible, and through puncturing of the received word and a generator matrix then solve a linear equation system in order to find s.

The solution of s might include some free variables. The amount of q-bits recovered about the secret is equal to $\ell - m$, where m is the number of free variables in s. Note, that if $m \geq \ell$ then no q-bits of information are recovered.

Let $A \subseteq \mathcal{I}$ be the set of participants trying to reconstruct the secret.

1. Construct vector $y \in \mathbb{F}_q^n$, where y_i is the share of P_i, if $P_i \in A$, and let y_i be unknown otherwise, for $1 \leq i \leq n$.
2. Construct a parity check matrix H for $\hat{C}$. Solve the equation system $yH^T = 0$ in order to decrease the number of erasures. Replace the unknowns in y with any solutions found.
3. Construct a generator matrix $G = \begin{bmatrix} G_C \\ G_S \end{bmatrix}$, where G_C is any generator matrix for C and $G_S^T = [u_1, \ldots, u_\ell]$ is the generator matrix for S, such that $\psi(s) = s_1 u_1 + \cdots + s_\ell u_\ell$.
4. Let y^* and $\mathcal{P}(G)$ be the generator matrix G and the vector y, respectively, each punctured on the indexes corresponding to the unknowns in y. Solve the equation system $(x, s)\mathcal{P}(G) = y^*$, where $x \in \mathbb{F}_q^{\dim C}$.

The methodology's time complexity is dominated by the Gaussian elimination used in the construction of the parity check matrix. Hence, the time complexity is $\mathcal{O}\left(\hat{k}^3\right)$.

Note, that for the methodology to be possible, bases for C and $\hat{C}$ has to be known, as well as the isomorphism ψ. Since none of this contains information about the secret, these are considered public information in $LSSS(\hat{C}, C)$.

Example 1.11

We want to show how to share a secret s in an $LSSS(\hat{C}, C)$ over $\mathbb{F}_q$, and how some subsets of participants will attempt to reconstruct the secret. Firstly, we set $q = 13$, $n = 11$, $\ell = 2$, and $\dim \hat{C} = 6$. Now we need to choose the codes $\hat{C}$ and C. We define the code $\hat{C}$ as the $[11, 6]$ code generated by the matrix $\hat{G}$, and C as the $[11, 6-2]$ subcode generated by the submatrix G.

$$\hat{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 \\ 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 1 & 8 & 12 \\ 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 \\ 1 & 6 & 10 & 8 & 9 & 2 & 12 & 7 & 3 & 5 & 4 \end{bmatrix},$$

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 \\ 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 1 & 8 & 12 \end{bmatrix}.$$

Note that the codes $\hat{C}$ and C are Reed-Solomon codes, hence we know that the minimum weight of $\hat{C}$ is $\hat{d} = 11 - 6 + 1 = 6$, and the minimum weight of C is $d = 11 - 4 + 1 = 8$. Furthermore, in [8] it is shown for the reconstruction and privacy of $LSSS(\hat{C}, C)$, where $\hat{C}$ and C are MDS codes, that $r_m = \dim C + m$ and $t_m = r_m - 1$, for $1 \leq m \leq \ell$. Hence, our SSS has $(4, 5)$-privacy and $(5, 6)$-reconstruction, i.e. the access structure, $\Gamma = \{\mathbb{A}_0, \mathbb{A}_1, \mathbb{A}_2\}$, is given as

$$\begin{aligned} \mathbb{A}_0 &= \{A \subset \mathcal{I} : |A| \leq 4\}, \\ \mathbb{A}_1 &= \{A \subset \mathcal{I} : |A| = 5\}, \\ \mathbb{A}_2 &= \{A \subseteq \mathcal{I} : |A| \geq 6\}. \end{aligned}$$

From definition 1.9 we have that $S \oplus C = \hat{C}$, hence

$$S = \langle (1, 3, 9, 1, 3, 9, 1, 3, 9, 1, 3), (1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4) \rangle .$$

Therefore, we choose our isomorphism $\psi : \mathbb{F}_{13}^2 \to S$ as

$$\psi(s) = s_1(1, 3, 9, 1, 3, 9, 1, 3, 9, 1, 3) + s_2(1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4).$$

Note that for any secret $s \in \mathbb{F}_{13}^2$, a share vector, $\psi(s) + c$, where $c \in C$, can be computed as $(x, s)\hat{G}$, where $x \in \mathbb{F}_{13}^4$ is chosen uniformly at random.

Now that we have our SSS, we need a secret that we want to share, here we let $s = (3, 12)$. In order to compute a share vector we choose uniformly at random $x \in \mathbb{F}_{13}^4$, in this case $x = (11, 10, 4, 6)$. Hence, the share vector is

$$\hat{c} = (x, s)\hat{G} = (7, 7, 9, 8, 5, 2, 3, 0, 0, 12, 3).$$

The shares are distributed such that participant P_i receives the share $\hat{c}_i$, for $1 \leq i \leq 11$. Note, that the index of each participant is public information, e.g. it is known that P_3's share is $\hat{c}_3$, but the value of the share is private.

We now look at two different attempts to reconstruct the secret, using the methodology listed earlier. In the following the steps referred to, are the steps of the methodology for reconstruction.

First, the subset of participants $A = \{P_3, P_{10}, P_{11}\}$ try to reconstruct the secret. From step 1, we get the vector

$$y = (y_1, y_2, 9, y_4, y_5, y_6, y_7, y_8, y_9, 12, 3).$$

In order to construct a parity check matrix for $\hat{C}$, we use Guassian elimination on $\hat{G}$ to get a generator matrix for $\hat{C}$ on the form $[I\ P]$, where I is the identity matrix and P is some 6×5 matrix. Then the 5×11 matrix $[-P^T\ I]$ is a parity check matrix for $\hat{C}$, i.e. we get the parity check matrix

$$H = \begin{bmatrix} 8 & 7 & 12 & 7 & 2 & 2 & 1 & 0 & 0 & 0 & 0 \\ 10 & 7 & 9 & 11 & 3 & 11 & 0 & 1 & 0 & 0 & 0 \\ 3 & 11 & 5 & 10 & 2 & 7 & 0 & 0 & 1 & 0 & 0 \\ 9 & 6 & 5 & 8 & 9 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 2 & 7 & 11 & 6 & 7 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Solving the equation system $yH^T = 0$ we get the following solutions for the unknowns

$$\begin{aligned} y_1 &= y_1, \\ y_2 &= 7y_1 + 7y_4 + 8y_5 + 5, \\ y_4 &= y_4, \\ y_5 &= y_5, \\ y_6 &= y_1 + 2y_4 + 8y_5 + 4, \\ y_7 &= 6y_1 + 5y_4 + 4y_5 + 5, \\ y_8 &= 8y_1 + 9y_4 + 9y_5 + 9, \\ y_9 &= 4y_1 + 3y_4 + 10y_5 + 2. \end{aligned}$$

Since we do not have a clear solution to any of the unknowns we cannot replace any unknown in y.

The generator matrix $\hat{G}$ has all the properties required of the matrix constructed in step 3, hence we simply use the matrix $\hat{G}$. We puncture the vector y and generator matrix $\hat{G}$ on the indexes corresponding to the unknowns in y, and define them as y^* and $\mathcal{P}(\hat{G})$, respectively, hence we get the following

$$y^* = (9, 12, 3),$$
$$\mathcal{P}(\hat{G}) = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 5 & 10 \\ 3 & 12 & 9 \\ 12 & 8 & 12 \\ 9 & 1 & 3 \\ 10 & 5 & 4 \end{bmatrix}.$$

Solving the system $(x, s)\mathcal{P}(\hat{G}) = y^*$ we get

$$\begin{aligned} x_1 &= 8x_4 + 9s_1 + 6s_2 + 7, \\ x_2 &= 6x_4 + 5s_1 + 7s_2 + 5, \\ x_3 &= 7x_4 + 9s_1 + 7s_2 + 7, \\ x_4 &= x_4, \\ s_1 &= s_1, \\ s_2 &= s_2. \end{aligned}$$

It is clear that since s_1 and s_2 are free variables, the participants in A are not able to recover any information about the secret. Note that it is not necessary to solve $x_1, \ldots, x_4$ in order to reconstruct the secret.

Now let the set $A = \{P_3, P_5, P_9, P_{10}, P_{11}\}$ attempt to reconstruct the secret. In step 1 we get the vector

$$y = (y_1, y_2, 9, y_4, 5, y_6, y_7, y_8, 0, 12, 3).$$

Using the parity check matrix H from before we solve the system $yH^T = 0$, and get the equations

$$\begin{aligned} y_1 &= 9y_4, \\ y_2 &= 5y_4 + 6, \\ y_4 &= y_4, \\ y_6 &= 11y_4 + 5, \\ y_7 &= 7y_4 + 12, \\ y_8 &= 3y_4 + 2. \end{aligned}$$

Again we do not have a clear solution for any of the unknowns in y. However, note that since there is only one free variable there are only 13 possible solutions for the vector y, and since $X_s \cup X_{s'} = \emptyset$ for any two secrets s, s' in any SSS, it follows that there are at most 13 possible secrets.

Now we solve the equation system $(x, s)\mathcal{P}(\hat{G}) = y^*$, where

$$y^* = (9, 5, 0, 12, 3),$$
$$\mathcal{P}(\hat{G}) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 4 & 3 & 9 & 5 & 10 \\ 3 & 9 & 3 & 12 & 9 \\ 12 & 1 & 1 & 8 & 12 \\ 9 & 3 & 9 & 1 & 3 \\ 10 & 9 & 3 & 5 & 4 \end{bmatrix},$$

and get $s_1 = 8s_2 + 11$ and $s_2 = s_2$. Hence, we get only 13 possible secrets, rather than 13^2 as when no information is given about the secret. This means that the participants have recovered 1 q-bit of information about the secret, i.e. $I\left(\vec{S}, \phi(A)\right) = 1$, which matches the privacy and reconstruction of the SSS. △

1.3 Security of Secret Sharing Schemes

In general a SSS shares a secret such that no single participant should be able to reconstruct the secret, however, some subsets of participants are able to reconstruct the secret, or to recover some partial information hereof. The access structure gives these subsets.

Therefore, security in SSSs is defined by the attacks on participants. These attacks are done by adversaries. An adversary corrupts a subset of the participants, and acquires any information that the participants might have. Corruption can be put into two possible categories; passive corruption, where the adversary only acquires the participants' information, and active corruption, where the adversary can choose the participants' actions. Note, that in both cases of corruption the adversary acquires any information held by the corrupted participants.

Through passive corruption an adversary can simply recover the same information about the secret as the set of corrupted participants would be able to. In other words, we can think of passive corruption as an attempt of reconstruction by an unauthorized source. Note, that an adversary corrupting t or less participants cannot recover any information about the secret, if the SSS has t-privacy.

In active corruption the adversary can also influence what values the corrupted participants give when reconstruction is attempted. E.g. if all the participants try to reconstruct the secret, and four participants are actively corrupted, four of the shares might be the wrong values.

Note, that these types of corruption give rise to mainly two security problems. Firstly, someone might be able to reconstruct the secret by obtaining information from participants. Secondly, someone might be able to invalidate the SSS by changing the information held by participants so that the shares reconstruct the wrong secret.

Since the information held by the participants, i.e. the shares, have to be enough for reconstruction, there is no way to completely avoid these security problems. However, we can try to minimize their impact. First, to avoid adversaries reconstructing the secret, we choose a SSS with t-privacy, where t is large. Hence, any adversary corrupting t or less participants cannot recover any information about the secret, and with a large t the adversary needs to corrupt a large subset of participants. In order to avoid invalidation of the SSS, we look at our methodology for reconstruction.

In the methodologies for reconstruction we presented earlier for both $LSSS(C)$ and $LSSS(\hat{C}, C)$, reconstruction is solved as a coding theoretic problem. Missing shares are equal to erasures, and likewise wrong values of shares, as can be received from an actively corrupted participant, are equivalent to errors. Hence, we can update the methodologies from before to include error correction, in order to obtain some security towards active corruption.

For $LSSS(C)$

1. Construct vector $y \in \mathbb{F}_q^n$, where y_i is the share of P_i, if $P_i \in A$, and let y_i be unknown otherwise, for $1 \leq i \leq n$.
2. Let y' be the vector y punctured on the unknowns, and $\mathcal{P}(C)$ be the code C punctured on the corresponding indexes. Use a relevant error correction algorithm for the code $\mathcal{P}(C)$ to correct any errors in y'.
3. Update the vector y by correcting any errors found in y'.
4. Construct a parity check matrix H for C. Solve the equation system $(s, y)H^T = 0$, where $s = (s_1, \ldots, s_\ell) \in \mathbb{F}_q^\ell$ is unknown.
5. Output any solutions found for $(s_1, \ldots, s_\ell)$.

Note, that we simply add error correction before erasure correction. It is clear that the time complexity of the methodology, including error correction, is now dominated by either the Gaussian elimination used to construct the parity check matrix or the error correction algorithm. Therefore, the time complexity is $\geq \mathcal{O}(k^3)$.

For $LSSS(\hat{C}, C)$

The methodology for $LSSS(\hat{C}, C)$ is updated almost identically to that for $LSSS(C)$, and the time complexity follows in the same way.

1. Construct vector $y \in \mathbb{F}_q^n$, where y_i is the share of P_i, if $P_i \in A$, and let y_i be unknown otherwise, for $1 \leq i \leq n$.

2. Let y' be the vector y punctured on the unknowns, and $\mathcal{P}(\hat{C})$ be the code $\hat{C}$ punctured on the corresponding indexes. Use a relevant error correction algorithm for the code $\mathcal{P}(\hat{C})$ to correct any errors in y'.

3. Update the vector y by correcting any errors found in y'.

4. Construct a parity check matrix H for $\hat{C}$. Solve the equation system $yH^T = 0$ in order to decrease the number of erasures. Replace the unknowns in y with any solutions found.

5. Construct a generator matrix $G = \begin{bmatrix} G_C \\ G_S \end{bmatrix}$, where G_C is any generator matrix for C and $G_S^T = [u_1, \ldots, u_\ell]$ is the generator matrix for S, such that $\psi(s) = s_1u_1 + \cdots + s_\ell u_\ell$.

6. Let y^* and $\mathcal{P}(G)$ be the generator matrix G and the vector y, respectively, each punctured on the indexes corresponding to the unknowns in y. Solve the equation system $(x, s)\mathcal{P}(G) = y^*$, where $x \in \mathbb{F}_q^{\dim C}$.

Note, that this approach will only work as long as the number of errors from actively corrupted participants is small. More precisely correcting errors in linear codes are twice the cost of correcting erasures, hence we can correct errors and reconstruct the secret if

$$r \leq n - 2e - \epsilon,$$

where n is the number of participants, r is the reconstruction of the SSS, e is the number of errors, and ϵ is the number of erasures. Therefore, in order to avoid invalidation of the SSS from active corruption we want a SSS with as low a reconstruction as possible. Hence, for optimal security of a (t, r)-SSS we want high privacy, t, and low reconstruction, r.

Example 1.12

We try to reconstruct the secret in example 1.11, however, this time we include an error.

The participants $A = \{P_1, P_2, P_3, P_5, P_6, P_9, P_{10}, P_{11}\}$ attempt to reconstruct the secret, but P_2 sends the wrong share, and we receive the vector

$$y = (7, 12, 9, y_4, 5, 2, y_7, y_8, 0, 12, 3).$$

First, we puncture the vector on the unknowns and get $y' = (7, 12, 9, 5, 2, 0, 12, 3)$. The code $\hat{C}$ punctured on the same indexes is an $[11 - 3, 6]$ Reed-Solomon code. We therefore apply an error correcting algorithm for Reed-Solomon codes in order to correct up to $\tau = \left\lfloor \frac{n-k}{2} \right\rfloor = 1$ errors, and get the vector $y' - e = (7, 7, 9, 5, 2, 0, 12, 3)$. Updating the vector y we get

$$y = (7, 7, 9, y_4, 5, 2, y_7, y_8, 0, 12, 3).$$

Solving $yH^T = 0$, we get $y_4 = 8$, $y_7 = 3$, and $y_8 = 0$, and by replacing them with the unknowns in y, we get the vector $y = (7, 7, 9, 8, 5, 2, 3, 0, 0, 12, 3)$.

Lastly, we need to solve the linear equation system $(x, s)\hat{G} = y$. Note that the generator matrix and vector are left un-punctured since there are no unknowns in y. We get that $s = (3, 12)$, which the secret shared in example 1.11. △

An adversary is defined by the possible subset of participants that it is able to corrupt, called an adversary structure. An adversary structure is a family of subsets of participants, noted $\mathcal{A}$. An adversary with adversary structure $\mathcal{A}$ is called an $\mathcal{A}$-adversary.

Which subset $A \in \mathcal{A}$ that an $\mathcal{A}$-adversary corrupts is either chosen once before the SSS starts, or can be changed at any point during the SSS, depending on the type of adversary. An adversary that chooses whom to corrupt once is called a static adversary, and an adversary that can change whom it corrupts at any point is called an adaptive adversary. However, the complete subset of participants that an adaptive $\mathcal{A}$-adversary chooses to corrupt during a SSS has to be in $\mathcal{A}$.

2 Multi-Party Computation

Multi-party computation (MPC) is a computation of a public function involving private data of n participants, without revealing any information about the private data to the other participants. A simple example of MPC is the summation of private data.

Example 2.1

Three participants, P_1, P_2, and P_3, want to compute their combined wealth without revealing any information of their own wealth to the other participants. Let s_1, s_2, and s_3 be the wealth of each participant respectively.

A simple approach would be for each participant to split their wealth in three smaller chunks, i.e. $c_{i,1}+c_{i,2}+c_{i,3} = s_i$, for $1 \leq i \leq 3$, and then P_i gives $c_{i,j}$ to P_j, for $1 \leq i, j \leq 3$. Now P_i will have $c_{1,i}$, $c_{2,i}$, and $c_{3,i}$, and sums them, such that $C_i = c_{1,i} + c_{2,i} + c_{3,i}$, for $1 \leq i \leq 3$. It is easy to see that $\sum_{i=1}^{3} s_i = \sum_{i=1}^{3} C_i$, i.e.

$$\sum_{i=1}^{3} s_i = \sum_{i=1}^{3} \left(\sum_{j=1}^{3} c_{i,j} \right) = \sum_{i,j} c_{i,j} = \sum_{j=1}^{3} \left(\sum_{i=1}^{3} c_{i,j} \right) = \sum_{j=1}^{3} C_j.$$

Hence, to compute their combined wealth each P_i gives C_i, for $1 \leq i \leq 3$, and the function can be publicly computed without revealing any information about the individual participants wealth. △

Note, that in example 2.1 if we assume that each participants wealth is positive, then it is clear that the result of the public function will provide information about each participant's wealth, in the form of an upper bound. Hence, even though no information about the private data can be obtained during the MPC, the result will, in this case, always contain some information.

Furthermore, it is clear that in general if the computation only involves two participant, e.g. if the function is $f = s_1 + s_2$, where s_1 and s_2 is the private data of participants P_1 and P_2 respectively, then the two participants will be able to compute each other's private

data from the result. Therefore, any function involving the private data of two, or one, participants will leak information, hence the function of a MPC should always include at least three participants.

In general a MPC protocol securely computes a given function if the function computes correctly and no information about any participant's private data is revealed or can be recovered. However, a MPC protocol can only securely compute a given function, if the set of corrupted participants are in some adversary structure $\mathcal{A}$ to which the MPC protocol is secure.

We say that the MPC protocol is secure with respect to $\mathcal{A}$, where $\mathcal{A}$ is the largest adversary structure to which the MPC protocol is secure. Moreover, the security of a MPC protocol is given in two parts, for passive corruption and for active corruption.

A MPC protocol is passively secure with respect to $\mathcal{A}$, if any set of participants $A \in \mathcal{A}$ cannot recover any information about the secret. A MPC protocol is actively secure with respect to $\mathcal{A}$, if for any $A \in \mathcal{A}$, the protocol will compute the function correctly, even if the participants in A are actively corrupted.

2.1 MPC Protocol for Addition using Linear SSS

In the following section we will introduce an actively secure MPC protocol for functions consisting of addition and scalar multiplication, that uses linear SSSs. First we show how to apply linear SSSs to solve a sum, similar to the MPC in example 2.1.

If we look at the 'chunks' in example 2.1 as shares, and the private data as secrets, we see that each participant is using a SSS in order to distribute shares of their secret among the participants. Hence, we can solve the problem in example 2.1 with SSSs, more precisely any linear SSS will work.

Recall that in a linear SSS, any two secrets s, s' with share vectors $c \in X_s$ and $c' \in X_{s'}$ respectively, the vector $c+c'$ is a share vector for $s+s'$. Hence, we can solve the MPC with n participants and the public function $\sum_{i=1}^{n} s_i$, where s_i is the private data of participant P_i, for $1 \leq i \leq n$, using a linear SSS.

The n participants, each holding their private data s_i, each constructs shares according to a linear SSS for the secret s_i, such that c_i is a share vector for s_i, for $1 \leq i \leq n$. Participant P_i then distributes the shares $c_i = (c_{i,1}, \ldots, c_{i,n})$ among the n participants, for $1 \leq i \leq n$.

Now each participant P_i holds the shares $c_{1,i}, \ldots, c_{n,i}$. Note that since the SSS is linear, the sum of the shares held by any participant is equal to a share for the sum of the participants' private data. Hence, by summing their shares, each participant now holds a share for the secret that is equal to $\sum_{i=1}^{n} s_i$.

Lastly, each participant's computed sum of their received shares is then used for secret reconstruction, and the reconstructed secret is then equal to the result of the public function.

Example 2.2

Given 6 participants, P_i, with private data $s_i \in \mathbb{F}_{11}^2$, for $1 \leq i \leq 6$, we want to compute the public function $\sum_{i=1}^{6} s_i$. We will use a linear SSS to solve this MPC problem, specifically $LSSS(\hat{C}, C)$ over $\mathbb{F}_{11}$, where $\hat{C}$ is generated by the matrix $\hat{G}$,

$$\hat{G} = \begin{bmatrix} 1 & 2 & 2 & 1 & 2 & 2 \\ 3 & 3 & 3 & 1 & 3 & 2 \\ 10 & 9 & 9 & 8 & 7 & 6 \\ 10 & 2 & 2 & 10 & 2 & 10 \end{bmatrix},$$

the code $C = \langle (1,2,2,1,2,2), (3,3,3,1,3,2) \rangle$, hence, since $S \oplus C = \hat{C}$,

$$S = \langle (10,9,9,8,7,7), (10,2,2,10,2,10) \rangle,$$

and we define the isomorphism $\psi : \mathbb{S} \to s$, as

$$\psi(s) = s_1(10,9,9,8,7,7) + s_2(10,2,2,10,2,10).$$

Now each participant chooses an $x_i \in \mathbb{F}_{11}^2$ uniformly at random, for $1 \leq i \leq 6$, and generates the share vector $\psi(s_i) + c_i = (x_i, s_i)\hat{G} = \hat{c}_i$, for $1 \leq i \leq 6$. Note, that $c_i = (x_i, 0)\hat{G}$, for $1 \leq i \leq 6$. In table 2.1 is an overview of each participant's private data, choice of x_i, computed c_i and $\psi(s_i)$, as well as the corresponding share vector $\hat{c}_i$, for $1 \leq i \leq 6$.

P_i	s_i	x_i	$\psi(s_i)$	$c_i = (x_i, 0)\hat{G}$	$\hat{c}_i = \psi(s_i) + c_i$
P_1	$(1,2)$	$(3,2)$	$(8,2,2,6,0,4)$	$(9,1,1,5,1,10)$	$(6,3,3,0,1,3)$
P_2	$(4,7)$	$(3,9)$	$(0,6,6,3,9,6)$	$(8,0,0,1,0,2)$	$(8,6,6,4,9,8)$
P_3	$(8,10)$	$(3,5)$	$(4,4,4,10,10,5)$	$(7,10,10,8,10,5)$	$(0,3,3,7,9,10)$
P_4	$(2,2)$	$(10,0)$	$(7,0,0,3,7,10)$	$(10,9,9,10,9,9)$	$(6,9,9,2,5,8)$
P_5	$(6,3)$	$(2,2)$	$(2,5,5,1,4,0)$	$(8,10,10,4,10,8)$	$(10,4,4,5,3,8)$
P_6	$(3,1)$	$(5,10)$	$(7,7,7,1,1,6)$	$(2,7,7,4,7,8)$	$(9,3,3,5,8,3)$

Table 2.1: The private data, random x_i, computed c_i and $\psi(s_i)$, and share vector $\hat{c}_i$ for each participant P_i.

Each participant P_i now distributes their share vector, $\hat{c}_i = (\hat{c}_{i,1}, \ldots, \hat{c}_{i,6})$, among the participants, such that P_i sends $\hat{c}_{i,j}$ to P_j, for $1 \leq i, j \leq 6$.

In order to visualize which shares are hold by which participant after the distributions of shares, we construct a matrix D, where row i are the shares distributed by P_i and column j are the shares received by P_j, for $1 \leq i, j \leq 6$. Note that the matrix D will never exist in a MPC, unless an adversary has corrupted all participants and chooses to construct it.

$$D = \begin{bmatrix} 6 & 3 & 3 & 0 & 1 & 3 \\ 8 & 6 & 6 & 4 & 9 & 8 \\ 0 & 3 & 3 & 7 & 9 & 10 \\ 6 & 9 & 9 & 2 & 5 & 8 \\ 10 & 4 & 4 & 5 & 3 & 8 \\ 9 & 3 & 3 & 5 & 8 & 3 \end{bmatrix}.$$

Let δ_i be the sum of the shares held by P_i, for $1 \leq i \leq 6$, then $\delta = (6, 6, 6, 1, 2, 7)$. Reconstructing the secret with share vector δ is equivalent to solving the public function. Since we have every share, and assume no errors, we can skip most of the methodology and simply solve $(x, s)\hat{G} = \delta$, where we get $(x, s) = (4, 6, 2, 3)$, hence

$$\sum_{i=1}^{6} s_i = (2, 3).$$

△

We have seen how we can use linear SSSs to solve MPC problems with addition. Recalling the other part of the linearity of SSSs; given a share vector $\hat{c}$ for a secret s, then $\lambda\hat{c}$ is a share vector for the secret λs, we see that it is also possible to solve MPC problems with scalar multiplication of private data using linear SSSs. This is simply done by each participant multiplying the corresponding shares they hold by the scalar.

The MPC protocol for functions consisting only of addition and scalar multiplication using a linear SSSs, as used in example 2.2, can be divided into four steps.

1. Each participant distributes shares of their private data according to the SSS.
2. Each participant computes the function using his or her corresponding shares.
3. Each participant outputs the result of step 2., which due to the linearity of the SSS is a share of the result of the given function.
4. The outputs of the participants are used as shares for reconstruction of the functions result. See section 1.3 for the methodology.

The passive security of this protocol follows from the SSS, since the distributions of shares are independent of each other. Hence, if an adversary passively corrupts t or less participants, then the adversary cannot recover any information about any of the participants' private data.

Likewise, since an adversary can only change the shares that an actively corrupted participant is holding, it is clear that only the outputs of actively corrupted participants are affected by active corruption in this protocol, and hence active security also follows from the SSS.

A more formal description of the actively secure MPC protocol described above, is presented as protocol 1.

Input
Each participant generates shares of their private data according to the SSS, and distributes the shares among the participants.

Computation
The following steps are run for each computation in the function, until a share of f is stored for each participant.

- **Addition** If $s_m + s_h$ is the computation, where s_m and s_h are secrets, then P_j computes $d_{m,j} + d_{h,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ and $d_{h,j}$ are P_j's shares of s_m and s_h respectively. The result is stored privately as P_j's share of the secret $s_m + s_h$.
- **Scalar multiplication** If λs_m is the computation, where s_m is a secret and λ is a scalar, then P_j compute $\lambda d_{m,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ is P_j's share of the secret s_m. The result is stored privately as P_j's share of λs_m.

Output and reconstruction
Each participant outputs their share of f. The shares are then used for reconstruction using the methodology outlined in section 1.3 that corresponds to the linear SSS applied in this protocol.

Protocol 1: Actively secure MPC protocol for addition and scalar multiplication using a linear SSS.

Note, that for active security we have assumed that no participant is corrupted before or during the initial distribution of shares. In chapter 5 we present a methodology for distribution, which verifies the shares distributed by each participant. This methodology is not restricted to MPC protocols, but can be used for distribution of shares in any linear SSS.

2.2 Multiplicative Secret Sharing Schemes

For any function $f : \mathbb{F}_q^\ell \to \mathbb{F}_q^\ell$, the domain of f is of size q^ℓ, which is finite, hence by Lagrange interpolation, we can construct a polynomial $p \in \mathbb{F}_q^\ell[x]$ using the points $(x, f(x))$, for all $x \in \mathbb{F}_q^\ell$. Thus, $\deg(p) < q^\ell$, and $p(x) = f(x)$, for all $x \in \mathbb{F}_q^\ell$, i.e. any function $f : \mathbb{F}_q^\ell \to \mathbb{F}_q^\ell$ is equivalent to a polynomial $p \in \mathbb{F}_q^\ell[x]$ with $\deg(p) < q^\ell$. It follows that the public function of any MPC problem in $\mathbb{F}_q^\ell$ can be expressed as a polynomial of degree $< q^\ell$, and hence only addition and multiplication is necessary for any function in $\mathbb{F}_q^\ell$.

Earlier in this chapter we have shown how to compute public functions consisting of addition and scalar multiplication in MPC problems using linear SSSs. However, we are missing multiplication of two secrets from their shares in order to solve any MPC problem using linear SSSs.

We introduce the following notations. Let $v * u = (v_1u_1, \ldots, v_nu_n)$, where $v, u \in \mathbb{F}_q^n$, i.e. $v*u$ is the coordinatewise product of the vectors v and u. Let $v_A = (v_i)_{P_i \in A}$, where $A \subseteq \mathcal{I}$, i.e. v_A is the vector $v \in \mathbb{F}_q^n$ punctured in the coordinates corresponding to $\bar{A}$.

Definition 2.3. *A linear SSS has $\hat{r}$-product reconstruction if for every set $A \subseteq \mathcal{I}$, where $|A| \geq \hat{r}$, there exists some linear function $\rho_A : \mathbb{F}_q^{|A|} \to \mathbb{F}_q^\ell$ such that*

$$\rho_A\left(c_A * c'_A\right) = s * s',$$

where $s, s' \in \mathbb{S}$, $c \in X_s$, and $c' \in X_{s'}$.

In other words, given two share vectors in a linear SSS with $\hat{r}$-product reconstruction, the coordinatewise product of the two corresponding secrets can be computed using any $\hat{r}$ or more (corresponding) coordinates of the share vectors. Hence, by definition 2.3 it is possible to do coordinatewise multiplication of private data in a MPC using a linear SSS with $\hat{r}$-product reconstruction.

Note, that by definition, if a linear SSS has $\hat{r}$-product reconstruction it also has $\tilde{r}$-product reconstruction, for any $n \geq \tilde{r} > \hat{r}$.

Definition 2.4. *A multiplicative SSS is a linear SSS with n-product reconstruction.*

We introduce a bound on the privacy t of a multiplicative SSS.

Theorem 2.5. *If a SSS with t-privacy is multiplicative, then $2t < n$. [4]*

Proof. Let $2t \geq n$, and let $A_1, A_2 \in \mathbb{A}_0$, where $\mathbb{A}_0$ are the rejected sets of the SSS and $A_1 \cup A_2 = \mathcal{I}$. Hence, there exists $\tilde{c}_1 \in X_{s_1}$ and $\tilde{c}_2 \in X_{s_2}$, such that $(\tilde{c}_1)_{A_1} = 0$ and $(\tilde{c}_2)_{A_2} = 0$, and $s_1, s_2 \neq 0$. Therefore, $\tilde{c}_1 * \tilde{c}_2 = 0 \in X_0$, hence the theorem follows by contradiction. □

In general a multiplicative SSS is secure against passive corruption, however, for active corruption we need a stronger property than multiplicative, specifically strongly multiplicative. We discuss active security of MPC involving multiplications in chapter 5.

Definition 2.6. *[2] A strongly multiplicative SSS is a linear SSS with t-privacy and $(n-t)$-product reconstruction.*

We proof a bound of t for strongly multiplicative SSSs similar to theorem 2.5.

Theorem 2.7. *If a SSS with t-privacy is strongly multiplicative, then $3t < n$.*

Proof. Let $2t \geq n - t$ and let $A_1, A_2 \in \mathbb{A}_0$, where $\mathbb{A}_0$ are the rejected sets of the SSS and $|A_1 \cup A_2| = n - t$. Hence, there exists $\tilde{c}_1 \in X_{s_1}$ and $\tilde{c}_2 \in X_{s_2}$, such that $(\tilde{c}_1)_{A_1} = 0$ and $(\tilde{c}_2)_{A_2} = 0$, and $s_1, s_2 \neq 0$. Therefore, $\tilde{c}_1 * \tilde{c}_2$ will have at least $n-t$ zero coordinates, hence the SSS has $(< n-t)$-product reconstruction and the theorem follows by contradiction. □

From the proof of theorem 2.7 it is clear that we can generalize theorem 2.5 and theorem 2.7 to corollary 2.8.

Corollary 2.8. *If a SSS with t-privacy has $\hat{r}$-product reconstruction, then $2t < \hat{r}$.*

We now introduce an updated version of our protocol for MPC involving addition and scalar multiplication using linear SSSs, protocol 1, in order to include multiplication.

2.2.1 Passively secure MPC protocol for multiplication

In this section we show how the multiplicative property of a linear SSS used in protocol 1 permits a revised protocol that is passively secure and allows for multiplication of secrets. We do this by adding a multiplication step to the computation phase. As with the other steps during the computation phase, the multiplication step will outline the actions needed for a participant to generate a share of $s * s'$ from the participant's shares of the secrets s and s'.

Assume that each participant has received shares distributed by a multiplicative SSS with secret space $\mathbb{S} = \mathbb{F}_q^\ell$. Let $\rho_{\mathcal{I}} : \mathbb{F}_q^n \to \mathbb{F}_q^\ell$ be the linear function such that $\rho_{\mathcal{I}}(c * c') = s * s'$, where $s, s' \in \mathbb{F}_q^\ell$, $c \in X_s$, and $c' \in X_{s'}$. Since $\rho_{\mathcal{I}}$ is a linear function from $\mathbb{F}_q^n$ to $\mathbb{F}_q^\ell$, it can be represented as a matrix transformation, i.e. $\rho_{\mathcal{I}}(c * c') = (c * c')\, R$, where R is an $n \times \ell$ matrix called the recombination matrix. Let R_i be the ith row of the matrix R, for $1 \leq i \leq n$. By matrix multiplication we have that

$$\begin{aligned} s * s' &= \rho_{\mathcal{I}}\left(c * c'\right) = \left(c * c'\right) R \\ &= \left[\sum_{i=1}^{n} \left(c * c'\right)_i r_{i,1}, \ldots, \sum_{i=1}^{n} \left(c * c'\right)_i r_{i,\ell}\right] \\ &= \sum_{i=1}^{n} \left[c_i c_i' r_{i,1}, \ldots, c_i c_i' r_{i,\ell}\right] \\ &= \sum_{i=1}^{n} c_i c_i' R_i. \end{aligned}$$

Since the function $\rho_{\mathcal{I}}$ can be public without loss of security, $c_i c_i' R_i$ can be computed locally by participant P_i, for $1 \leq i \leq n$. However, due to privacy we cannot simply sum these publicly. Therefore, we use the linearity of the SSS, i.e. the fact that the sum of share vectors is a share vector for the sum of the secrets.

So P_i generates share vectors for the computed value $c_i c_i' R_i$, and distributes the shares among the participants, for $1 \leq i \leq n$. Such that P_j will have the jth share for each computed value $c_i c_i' R_i$, for $1 \leq i, j \leq n$. Then P_j simply sums these shares, and will now have a share for the secret $s * s'$, for $1 \leq j \leq n$. Below is a revised version of protocol 1 to include this computation step for coordinatewise product of secrets.

Input
Each participant generates shares of their private data according to the SSS, and distributes the shares among the participants.

Computation
The following steps are run for each computation in the function, until a share of f is stored for each participant.

- **Addition** If $s_m + s_h$ is the computation, where s_m and s_h are secrets, then P_j computes $d_{m,j} + d_{h,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ and $d_{h,j}$ are P_j's shares of s_m and s_h respectively. The result is stored privately as P_j's share of the secret $s_m + s_h$.
- **Scalar multiplication** If λs_m is the computation, where s_m is a secret and λ is a scalar, then P_j compute $\lambda d_{m,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ is P_j's share of the s_m. The result is stored privately as P_j's share of the secret λs_m.
- **Multiplication of secrets** If $s_m * s_h$ is the computation, where s_m and s_h are secrets, then each participant P_j takes the following steps, where $d_{m,j}$ and $d_{h,j}$ are P_j's shares of s_m and s_h, respectively, and R_j is the jth row of the SSS's recombination matrix, for $1 \leq j \leq n$.
  1. P_j computes $\delta_{(m,h),j} = d_{m,j} d_{h,j} R_j$.
  2. P_j generate shares of $\delta_{(m,h),j}$ according to the SSS, and distributes the shares among the participants.
  3. P_j sums the shares received in 2. The sum is stored privately as P_j's share of the secret $s_m * s_h$.

Output and reconstruction
Each participant outputs their share of f. The shares are then used for reconstruction using the methodology outlined in section 1.3 that corresponds to the linear SSS applied in this protocol.

Protocol 2: Passively secure MPC protocol including multiplications using a multiplicative SSS.

Note that protocol 2 is only secure against passive corruption, since an actively corrupt participant that distributes wrong shares during multiplication will corrupt each participant's following summation. The passive security of protocol 2 follows as for protocol 1 since the distribution of shares during multiplication uses the SSS. We present an actively secure MPC protocol that include multiplication in chapter 5.

2.3 Multiplicative $LSSS(C)$

In this project we will study multiplicative SSSs of the type $LSSS(C)$ exclusively. The advantage of using less space with SSSs of the type $LSSS(\hat{C}, C)$, rather than $LSSS(C)$, is heavily outweighed by the added complexity and computation necessary when constructing multiplicative $LSSS(\hat{C}, C)$ schemes, and performing multiplications.

Therefore, in the following we look at multiplicative SSSs of the type $LSSS(C)$. We present general results for multiplicative $LSSS(C)$, as well as some result for $\ell = 1$.

We introduce the following notations for the coordinatewise product of linear codes.

$$C * C' = \{c * c' : c \in C, c' \in C'\},$$
$$C^{*a} = \underbrace{C * \cdots * C}_{a}.$$

Theorem 2.9. *[2] $LSSS(C)$ has $\hat{r}$-product reconstruction if $LSSS(C^{*2})$ has r^*-reconstruction, where $r^* \leq \hat{r} \leq n$.*

$LSSS(C^{*2})$ has r^*-reconstruction if $LSSS(C)$ has $\hat{r}$-product reconstruction, for all $r^* \leq \hat{r} \leq n$.

Proof. Let C be an $[n + \ell, k, d]$ code over $\mathbb{F}_q$, and let $c = (s, \tilde{c}) \in C$ and $c' = (s', \tilde{c}') \in C$, then $\tilde{c} \in X_s$ and $\tilde{c} \in X_{s'}$ in $LSSS(C)$. Since the secret in $LSSS(C)$ is equal to the first ℓ coordinates of the codeword, the first ℓ coordinates of the coordinatewise product $c * c'$ will equal the coordinatewise product of the secrets s and s', i.e. $s * s'$.

For any two codewords $c, c' \in C$, we know that, by definition, $c * c' \in C^{*2}$, hence, if $LSSS(C^{*2})$ has r^*-reconstruction, then there exists a function $\rho_A : \mathbb{F}_q^{|A|} \to \mathbb{F}_q^{\ell}$, where $A \subseteq \mathcal{I}$ and $|A| \geq r^*$, such that

$$\rho_A(\tilde{c}_A * \tilde{c}'_A) = s * s'.$$

Indeed the function ρ_A is the reconstruction of the secret $s * s'$ in $LSSS(C^{*2})$, from the shares $(\tilde{c} * \tilde{c}')_A$, and since the SSS is linear ρ_A is linear. Thus, it follows that if $LSSS(C^{*2})$ has r^*-reconstruction, then $LSSS(C)$ has r^*-product reconstruction, and r^*-product reconstruction implies $\hat{r}$-product reconstruction for any $r^* \leq \hat{r} \leq n$.

Let $LSSS(C)$ have $\hat{r}$-product reconstruction for $r^* - 1 \leq \hat{r} \leq n$, and assume $LSSS(C^{*2})$ has r^*-reconstruction. Let $A \subset \mathcal{I}$, where $|A| = r^* - 1$ and $A \in \mathbb{A}_{\ell-1}$, i.e. the participants in A are able to recover exactly $\ell - 1$ q-bits of information about the secret. Hence, there is no function $\rho_A : \mathbb{F}_q^{|A|} \to \mathbb{F}_q^{\ell}$, such that $\rho_A(\tilde{c}_A * \tilde{c}'_A) = s * s'$, where $\tilde{c}$ and $\tilde{c}'$ are share vectors for the secrets s and s', respectively. Hence, the theorem follows by contradiction. □

The two parts of theorem 2.9 show that the reconstruction of $LSSS(C^{*2})$ is equal to the lowest value of product reconstruction of $LSSS(C)$.

Let $LSSS(C)$ have t-privacy and r-reconstruction, due to the bounds in theorem 1.7, $t \geq d^{\perp} - \ell - 1$ and $r \leq n + \ell - d + 1$, the following corollary follows from theorem 2.9.

Corollary 2.10. *$LSSS(C)$ has $\hat{r}$-product reconstruction, where $\hat{r} = n - d(C^{*2}) + \ell + 1$, and $d(C^{*2})$ is the minimum weight of C^{*2}.*

For non-ramp schemes, i.e. where $\ell = 1$, we will show that $LSSS(C)$ is multiplicative if C is self-orthogonal. First, we proof an equivalent expression for self-orthogonal codes, which is more suited for multiplicative codes.

Lemma 2.11. *Let C be an $[n, k]$ code, then C is self-orthogonal if and only if $C^{*2} \perp \mathbf{1}$.*

Proof. To proof that the two expression are equivalent, we need only proof that $c \cdot c' = (c * c') \cdot \mathbf{1}$, for all $c, c' \in C$, and the lemma follows. This is easily shown for all vectors.

$$c \cdot c' = \sum_{i=1}^{n} c_i c'_i = \sum_{i=1}^{n} c_i c'_i \cdot 1 = \left(c_1 c'_1, \ldots, c_n c'_n\right) \cdot \mathbf{1} = (c * c') \cdot \mathbf{1}.$$

□

Theorem 2.12. *Let C be an $[n + 1, k, d]$ code over $\mathbb{F}_q$. If $C^{*2} \perp \mathbf{1}$, then $LSSS(C)$ is multiplicative.*

Proof. Let C be an $[n + 1, k, d]$ code over $\mathbb{F}_q$, where $C^{*2} \perp \mathbf{1}$. Consider $LSSS(C)$, $c = (s, c_1, \ldots, c_n) \in C$, and $c' = (s', c'_1, \ldots, c'_n) \in C$. Note, that $(c_1, \ldots, c_n) \in X_s$ and $(c'_1, \ldots, c'_n) \in X_{s'}$. Since $C^{*2} \perp \mathbf{1}$, we know that $(c * c') \cdot \mathbf{1} = 0$, hence

$$(ss', c_1 c'_1, \ldots, c_n c'_n) \cdot \mathbf{1} = 0 \Leftrightarrow ss' + \sum_{i=1}^{n} c_i c'_i = 0 \Leftrightarrow ss' = -\sum_{i=1}^{n} c_i c'_i.$$

It is clear that $LSSS(C)$ satisfies the definition of multiplicative SSS, and that the linear function $\rho_{\mathcal{I}}(v) = -\sum_{i=1}^{n} v_i = x$, where $(x, v) \in C^{*2}$. □

In the following chapters we will examine $LSSS(C)$ for some algebraic families of linear codes in order to generate multiplicative SSSs based on these codes. We will also show how to solve MPCs involving coordinatewise multiplications of the private data using these SSS.

3

Reed-Solomon Codes

Reed-Solomon codes are linear codes, where the codewords are the evaluation of univariate polynomials in pairwise distinct points. More precisely an $[n, k]$ Reed-Solomon code C over $\mathbb{F}_q$ is

$$C = \{(f(p_1), \ldots, f(p_n)) : f \in \mathbb{F}_q[x], \deg(f) < k\},$$

where $p_1, \ldots, p_n \in \mathbb{F}_q$ are pairwise distinct. Due to the n pairwise distinct elements of $\mathbb{F}_q$, it is clear that $n \leq q$ for an $[n, k, d]$ Reed-Solomon code, in fact, in many applications $p_1, \ldots, p_n$ have to be non-zero, hence $n < q$ instead. Here we will work with non-zero points $p_1, \ldots, p_n$.

It is easy to show that Reed-Solomon codes are MDS. Let C be an $[n, k, d]$ Reed-Solomon code, and $c = (f(p_1), \ldots, f(p_n)) \in C$, since $\deg(f) < k$ the polynomial f has at most $k-1$ distinct roots. Therefore, any $c \in C$ has at most $k-1$ zeros, i.e. $d \geq n - (k-1)$, and by the Singleton bound, it follows that Reed-Solomon codes are MDS.

Theorem 3.1. *Let C be an $[n + \ell, k]$ Reed-Solomon code, then $LSSS(C)$ has $(2k-1)$-product reconstruction if $k \leq \frac{n+1}{2}$.*

Proof. Let C be an $[n + \ell, k]$ Reed-Solomon code over $\mathbb{F}_q$, by definition

$$C^{*2} = \{(f(p_1)g(p_1), \ldots, f(p_{n+\ell})g(p_{n+\ell})) : f, g \in \mathbb{F}_q[x], \deg(f) < k, \deg(g) < k\}.$$

Since the set of the monomials that generate C is $\{1, x, \ldots, x^{k-1}\}$ the set of all pairwise products of those monomials is a generating set for C^{*2}. Therefore, we can write C^{*2} as

$$C^{*2} = \{(h(p_1), \ldots, h(p_{n+\ell})) : h \in \mathbb{F}_q[x], \deg(h) < 2k-1\},$$

hence C^{*2} is an $[n+\ell, 2k-1]$ Reed-Solomon code, and by lemma 1.8 $LSSS(C^{*2})$ has $(2k-1)$-reconstruction. Thus, by theorem 2.9 $LSSS(C)$ has $(2k-1)$-product reconstruction if $2k - 1 \leq n \Leftrightarrow k \leq \frac{n+1}{2}$. □

If C is an MDS code, then $LSSS(C)$ has $(t = k - \ell)$-privacy, by lemma 1.8, hence the subsequent corollary follows from theorem 3.1.

Corollary 3.2. *Let C be an $[n+\ell, k]$ Reed-Solomon code, then $LSSS(C)$ has $(2t+2\ell-1)$-product reconstruction if $t \leq \frac{n+1-\ell}{2}$, where $LSSS(C)$ has t-privacy.*

3.1 Recombination Matrix for Reed-Solomon codes

In the following we will show how the linear functions ρ_A in definition 2.3 are given for a multiplicative $LSSS(C)$, where C is a Reed-Solomon code.

Consider $LSSS(C)$, where C is an $[n + \ell, k]$ Reed-Solomon code over $\mathbb{F}_q$, with $k \leq \frac{n+1}{2}$. It follows from theorem 3.1 that $LSSS(C)$ has $(2k - 1)$-product reconstruction.

Let $c = (s, \tilde{c}) = \big(f(p_1), \ldots, f(p_{n+\ell})\big) \in C$ and $c' = (s', \tilde{c}') = \big(g(p_1), \ldots, g(p_{n+\ell})\big) \in C$, note that $\deg(f) < k$, $\deg(g) < k$, $\tilde{c} \in X_s$, and $\tilde{c}' \in X_{s'}$.

From the definition of product reconstruction, definition 2.3, we have that there exists a linear function $\rho_A(\tilde{c}_A * \tilde{c}'_A) = s * s'$, for any $A \subseteq \mathcal{I}$, where $|A| \geq 2k - 1$.

First we note that $\tilde{c} * \tilde{c}' = \big(h(p_{\ell+1}), \ldots, h(p_{n+\ell})\big)$, where $h = (f \cdot g)$, hence $\deg h < 2k - 1$. By Lagrange interpolation it is possible to construct the polynomial h from $2k-1$ or more points. Let $\{\gamma_1, \ldots, \gamma_{2k-1}\} \subseteq \{p_1, \ldots, p_{n+\ell}\}$ be the $2k - 1$ points chosen to construct the polynomial h, by Lagrange interpolation we get

$$h(x) = \sum_{i=1}^{2k-1} f(\gamma_i) g(\gamma_i) \prod_{\substack{j=1 \\ j \neq i}}^{2k-1} \frac{x - \gamma_j}{\gamma_i - \gamma_j}.$$

Since $s = \big(f(p_1), \ldots, f(p_\ell)\big)$ and $s' = \big(g(p_1), \ldots, g(p_\ell)\big)$, we have that $s * s' = \big(h(p_1), \ldots, h(p_\ell)\big)$. Therefore, the mth coordinate of the secret $s * s'$ is

$$(s * s')_m = \sum_{i=1}^{2k-1} f(\gamma_i) g(\gamma_i) \prod_{\substack{j=1 \\ j \neq i}}^{2k-1} \frac{p_m - \gamma_j}{\gamma_i - \gamma_j}, \tag{3.1}$$

for $1 \leq m \leq \ell$.

Though the $2k - 1$ points in the Lagrange interpolation above are chosen from among all the points, in our application we cannot use any points that are equivalent to the secret, therefore, we use $\{\gamma_1, \ldots, \gamma_{2k-1}\} \subseteq \{p_{\ell+1}, \ldots, p_{n+\ell}\}$ from this point forward.

Since $f(\gamma_i)g(\gamma_i)$, where $\gamma_i = p_{a+\ell}$, is the product of participant P_a's shares of the secrets s and s', for any $1 \leq a \leq n$, we can express the function ρ_A, where $A \subseteq \mathcal{I}$ and $|A| \geq 2k-1$, from equation (3.1) as

$$\begin{aligned}\rho_A\left(c_A * c'_A\right) &= \left(\sum_{i\in A} c_i c'_i \prod_{\substack{j\in A\\ j\neq i}} \frac{p_1 - p_{\ell+j}}{p_{\ell+i} - p_{\ell+j}}, \ldots, \sum_{i\in A} c_i c'_i \prod_{\substack{j\in A\\ j\neq i}} \frac{p_\ell - p_{\ell+j}}{p_{\ell+i} - p_{\ell+j}}\right) \\ &= \sum_{i\in A} c_i c'_i \left(\prod_{\substack{j\in A\\ j\neq i}} \frac{p_1 - p_{\ell+j}}{p_{\ell+i} - p_{\ell+j}}, \ldots, \prod_{\substack{j\in A\\ j\neq i}} \frac{p_\ell - p_{\ell+j}}{p_{\ell+i} - p_{\ell+j}}\right). \end{aligned} \tag{3.2}$$

Now that we have the function ρ_A, we need a way to apply it for MPC.

Let $A = \{\alpha_1, \ldots, \alpha_{|A|}\} \subseteq \{1, \ldots, n\}$ and $R^A_{i,m} = \prod_{\substack{j\in A\\ j\neq\alpha_i}} \frac{p_m - p_{\ell+j}}{p_{\ell+\alpha_i} - p_{\ell+j}}$ for $1 \leq m \leq \ell$ and $1 \leq i \leq |A|$, where $|A| \geq 2k-1$. Note, that $R^A_{i,m}$ is only dependent on public information, hence it can be computed either locally or publicly without any loss of security. The matrix $R^A = \left(R^A_{i,m}\right) \in \mathbb{F}_q^{|A|\times\ell}$ is called the recombination matrix for the set of participants A, note that $R^{\mathcal{I}} = R$. By replacing the products in equation (3.2) with entries of R^A, we get the following expression

$$\rho_A\left(c_A * c'_A\right) = \sum_{i\in A} c_i c'_i \left(R^A_{i,1}, \ldots, R^A_{i,\ell}\right) = s * s'. \tag{3.3}$$

We already have that the entries of R^A can be computed from public information, and since $c_i c'_i$ is the product of participant P_i's shares for secrets s and s', respectively, for $i \in A$, they can be computed locally. Hence, each term of the sum in equation (3.3) can be computed by its corresponding participant without leaking any information.

Let $\delta_i = c_i c'_i \left(R^A_{i,1}, \ldots, R^A_{i,\ell}\right) \in \mathbb{F}_q^\ell$, for $i \in A$, then, by equation (3.3)

$$\sum_{i\in A} \delta_i = s * s'.$$

Due to the linearity of $LSSS(C)$ the sum of share vectors for δ_i for all $i \in A$, is a share vector for the secret $s * s'$. Therefore, each participant P_i, after locally computing δ_i, distributes shares of δ_i among the n participants, for every $i \in A$. Then all n participants simply sum their received shares in order to get a share for $s * s'$.

We have now shown how to compute shares for the coordinatewise product of two secrets, i.e. $s * s'$, in a multiplicative $LSSS(C)$, where C is an $[n + \ell, k]$ Reed-Solomon code. In order to clarify the methodology we give two examples of MPC with coordinatewise multiplication using $LSSS(C)$, where C is a multiplicative Reed-Solomon code.

The first example is a simple MPC with only a single coordinatewise multiplication of secrets as the public function, and will explain the process in detail.

The second example is an intricate MPC, where the public function includes both coordinatewise multiplication and addition. In this example some details are dropped.

Example 3.3

Let C be an $[n+\ell, k]$ Reed-Solomon code over $\mathbb{F}_7$, where $n = 4$, $\ell = 1$, and $k = 2$, with $p_i = i$, for $1 \leq i \leq 4+1$. Since $k \leq \frac{n+1}{2}$, it follows from theorem 3.1, that C has $(2k-1=3)$-product reconstruction. The matrices G and H are a generator matrix and a parity check matrix, respectively, for the code C.

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix}$$
$$, H = \begin{bmatrix} 1 & 5 & 1 & 0 & 0 \\ 2 & 4 & 0 & 1 & 0 \\ 3 & 3 & 0 & 0 & 1 \end{bmatrix}.$$

We want to compute the function $f(s_1, \ldots, s_4) = s_1 * s_2$, where s_i is the private data of participant P_i, for $1 \leq i \leq 4$. Note, that the function in this MPC problem only includes the secrets s_1 and s_2, hence only P_1 and P_2 need to generate shares for their secrets. However, all four participants are still needed during the computation, since the SSS will generate four shares for a secret.

Listed in table 3.1 are the private data of participants P_1 and P_2, as well as the shares generated for the secrets by the random vector $x_i \in \mathbb{F}_7$, for $1 \leq i \leq 2$.

P_i	s_i	x_i	$c_i = (s_i, x_i)G$	$\tilde{c}_i$
P_1	5	6	$(5, 4, 3, 2, 1)$	$(4, 3, 2, 1)$
P_2	2	3	$(2, 5, 1, 4, 0)$	$(5, 1, 4, 0)$

Table 3.1: The private data, random x_i, computed c_i, and share vector $\tilde{c}_i$ for participants P_1 and P_2.

Participants P_1 and P_2 distribute the share vectors $\tilde{c}_1$ and $\tilde{c}_2$, respectively, among the four participants. Table 3.2 shows the shares as distributed among the participants.

P_i	Share of s_1	Share of s_2
P_1	4	5
P_2	3	1
P_3	2	4
P_4	1	0

Table 3.2: The shares held by each participant.

In order to compute shares for the secret $s_1 * s_2$, we first need to choose a subset $A \subseteq \mathcal{I}$, where $|A| \geq 3$, since C has 3-product reconstruction. This choice of A will decide which recombination matrix, R^A, we need to use, and which participants P_i that have to compute $\delta_i = (\tilde{c}_1)_i \, (\tilde{c}_2)_i \, R^A_{i,1}$, for each $i \in A$. Here we choose $A = \{1, 2, 3\}$, and get the matrix

$$R^A = \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}.$$

Now each participant P_i, where $i \in A$, i.e. P_1, P_2, and P_3, locally computes $\delta_i = (\tilde{c}_1)_i \, (\tilde{c}_2)_i \, R^A_{i,1}$, the results can be seen in table 3.3.

P_i	$(\tilde{c}_1)_i$	$(\tilde{c}_2)_i$	$\delta_i = (\tilde{c}_1)_i \, (\tilde{c}_2)_i \, R^A_{i,1}$
P_1	4	5	$\delta_1 = 4 \cdot 5 \cdot 3 = 4$
P_2	3	1	$\delta_2 = 3 \cdot 1 \cdot 4 = 5$
P_3	2	4	$\delta_3 = 2 \cdot 4 \cdot 1 = 1$

Table 3.3: The δ_i's computed for each $i \in A$.

Each δ_i is treated as a secret, and each participant P_i generates a share vector for their corresponding δ_i, for each $i \in A$. The generated share vectors for each δ_i, noted $\tilde{c}_{\delta_i}$, can be seen in table 3.4.

P_i	δ_i	x_i	$(\delta_i, x_i) \, G$	$\tilde{c}_{\delta_i}$
P_1	4	6	$(4, 3, 2, 1, 0)$	$(3, 2, 1, 0)$
P_2	5	4	$(5, 2, 6, 3, 0)$	$(2, 6, 3, 0)$
P_3	1	5	$(1, 6, 4, 2, 0)$	$(6, 4, 2, 0)$

Table 3.4: The share vectors for each δ_i, where $i \in A$.

After the shares are generated, they are distributed among the four participants. In order to compute a share for the secret $s_1 * s_2$, each participant simply sums their shares of δ_1, δ_2, and δ_3. Each participant's shares of δ_1, δ_2, and δ_3 as well as the resulting sum, i.e. their share of $s_1 * s_2$ can be seen in table 3.5.

P_i	Share of δ_1	Share of δ_2	Share of δ_3	Share of $s_1 * s_2$
P_1	3	2	6	$3 + 2 + 6 = 4$
P_2	2	6	4	$2 + 6 + 4 = 5$
P_3	1	3	2	$1 + 3 + 2 = 6$
P_4	0	0	0	$0 + 0 + 0 = 0$

Table 3.5: Each participant's shares of δ_1, δ_2, and δ_3, and their computed share of $s_1 * s_2$.

We now simple use the methodology for reconstructing a secret in $LSSS(C)$. We assume that only the participants P_1, P_3, and P_4 attempts to reconstruct the secret. Therefore, we get the share vector $y = (4, y_2, 6, 0)$, where y_2 is the unknown share of participant P_2. Left is only to attempt reconstruction by solving $(s_1 * s_2, y)\, H^T = 0$, i.e.

$$(s_1 * s_2, 4, y_2, 6, 0) \begin{bmatrix} 1 & 2 & 3 \\ 5 & 4 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}.$$

From this we get the following linear equation system.

$$\begin{aligned} s_1 * s_2 + 4 \cdot 5 + y_2 \cdot 1 &= 0, \\ 2(s_1 * s_2) + 4 \cdot 4 + 6 \cdot 1 &= 0, \\ 3(s_1 * s_2) + 4 \cdot 3 + 0 \cdot 1 &= 0. \end{aligned}$$

Solving the equation system yields the results $y_2 = 5$ and $s_1 * s_2 = 3$. To show that our result is correct, we do a simple check and by direct computation $s_1 * s_2 = 5 \cdot 2 = 3$. Note, that this is only possible because we have full information, which would not normally be true. △

Example 3.4

Let C be an $[n + \ell, k]$ Reed-Solomon code over $\mathbb{F}_{11}$, where $n = 5$, $\ell = 2$, and $k = 3$, with $p_i = i$, for $1 \leq i \leq 5 + 2$. Since $k \leq \frac{n+1}{2}$ it follows that C has $(2k - 1 = 5)$-product reconstruction. The matrices G and H are a generator matrix and a parity check matrix, respectively, for the code C.

$$G = \begin{bmatrix} 1 & 0 & 10 & 9 & 8 & 7 & 6 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 3 & 6 & 10 & 4 \end{bmatrix},$$

$$H = \begin{bmatrix} 10 & 3 & 8 & 1 & 0 & 0 & 0 \\ 8 & 8 & 5 & 0 & 1 & 0 & 0 \\ 5 & 4 & 1 & 0 & 0 & 1 & 0 \\ 1 & 2 & 7 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

We want to solve the MPC problem $f(s_1, \ldots, s_5) = (s_1 + s_2) * s_3 + s_4 * s_5$, where s_i is the private data of participant P_i, for $1 \leq i \leq 5$. Each of the 5 participants generate shares for their private data according to $LSSS(C)$, and distributes the shares among the participants. The private data and corresponding share vectors can be seen in table 3.6.

P_i	s_i	x_i	$c_i = (s_i, x_i)G$	$\tilde{c}_i$
P_1	$(7, 2)$	2	$(7, 2, 10, 9, 10, 2, 7)$	$(10, 9, 10, 2, 7)$
P_2	$(1, 4)$	6	$(1, 4, 2, 6, 5, 10, 10)$	$(2, 6, 5, 10, 10)$
P_3	$(10, 5)$	0	$(10, 5, 0, 6, 1, 7, 2)$	$(0, 6, 1, 7, 2)$
P_4	$(2, 9)$	7	$(2, 9, 1, 0, 6, 8, 6)$	$(1, 0, 6, 8, 6)$
P_5	$(6, 3)$	9	$(6, 3, 9, 2, 4, 4, 2)$	$(9, 2, 4, 4, 2)$

Table 3.6: The private data, random x_i, computed c_i, and share vector $\tilde{c}_i$ for each participant P_i.

We solve the function $f(s_1, \ldots, s_5) = (s_1 + s_2) * s_3 + s_4 * s_5$ in arithmetic order from left to right, hence we first do the computations corresponding to $(s_1 + s_2)$. Here, each participant simply sums their shares for s_1 and s_2 and store the result for future computations. The shares held by each participant after this operation can be seen in table 3.7.

P_i	Share of s_1	Share of s_2	Share of s_3	Share of s_4	Share of s_5	Share of $(s_1 + s_2)$
P_1	10	2	0	1	9	$10 + 2 = 1$
P_2	9	6	6	0	2	$9 + 6 = 4$
P_3	10	5	1	6	4	$10 + 5 = 4$
P_4	2	10	7	8	4	$2 + 10 = 1$
P_5	7	10	2	6	2	$7 + 10 = 6$

Table 3.7: The shares held by each participant P_i.

The next computation is $(s_1 + s_2) * s_3$, here we need to use our methodology for coordinatewise product of secrets. Firstly, we need to choose $2k - 1 = 5$ participants, however, here $n = 5$ so only one choice is possible, i.e. using every participant. The recombination matrix for the set of all participants, i.e. $R^{\mathcal{I}}$, is given as

$$R^{\mathcal{I}} = \begin{bmatrix} 4 & 5 \\ 4 & 1 \\ 1 & 10 \\ 9 & 6 \\ 5 & 1 \end{bmatrix}.$$

The 3 steps of the multiplication are as follows. In step 1, each participant P_i computes $f(i+2)g(i+2)R_i^{\mathcal{I}}$, where $f(i+2)$ and $g(i+2)$ are the participant's shares of $(s_1 + s_2)$ and s_3, respectively, and $R_i^{\mathcal{I}}$ is the ith row of $R^{\mathcal{I}}$, for $1 \leq i \leq n$.

In step 2, each participant generates a share vector for the vector computed in step 1, and distributes the shares among the participants. Lastly, as step 3, each participant sums the shares received during step 2. The result of this summation is now the participant's share of the secret $(s_1 + s_2) * s_3$.

The results of the three steps of the multiplication are summarized in table 3.8.

P_i	$\delta_i = f(i+2)g(i+2)R_i^{\mathcal{I}}$	$(\delta_i, x_i)G$	Share of $(s_1 + s_2) * s_3$
P_1	$1 \cdot 0 \cdot (4,5) = (0,0)$	$(0,0,7)G = (0,0,7,10,9,4,6)$	$7+8+2+9+10 = 3$
P_2	$4 \cdot 6 \cdot (4,1) = (8,2)$	$(8,2,1)G = (8,2,8,4,1,10,9)$	$10+4+0+8+10 = 10$
P_3	$4 \cdot 1 \cdot (1,10) = (4,7)$	$(4,7,3)G = (4,7,2,0,1,5,1)$	$9+1+1+6+1 = 7$
P_4	$1 \cdot 7 \cdot (9,6) = (8,9)$	$(8,9,10)G = (8,9,9,8,6,3,10)$	$4+10+5+3+5 = 5$
P_5	$6 \cdot 2 \cdot (5,1) = (5,1)$	$(5,1,2)G = (5,1,10,10,1,5,0)$	$6+9+1+10+0 = 4$

Table 3.8: For the operation $(s_1 + s_2) * s_3$, the locally computed values, noted δ_i, the corresponding share vector for some random chosen x_i, and each participant's resulting share of $(s_1 + s_2) * s_3$.

The next computation is again a coordinatewise multiplication, $(s_4 * s_5)$, the results are summarized in table 3.9.

P_i	$\delta_i = f(i+2)g(i+2)R_i^{\mathcal{I}}$	$(\delta_i, x_i)G$	Share of $s_4 * s_5$
P_1	$1 \cdot 9 \cdot (4,5) = (3,1)$	$(3,1,7)G = (3,1,6,7,4,8,8)$	$6+9+4+0+5 = 2$
P_2	$0 \cdot 2 \cdot (4,1) = (0,0)$	$(0,0,9)G = (0,0,9,5,10,2,3)$	$7+5+9+9+6 = 3$
P_3	$6 \cdot 4 \cdot (1,10) = (2,9)$	$(2,9,10)G = (2,9,4,9,2,5,7)$	$4+10+2+10+4 = 8$
P_4	$8 \cdot 4 \cdot (9,6) = (2,5)$	$(2,5,3)G = (2,5,0,9,10,3,10)$	$8+2+5+3+10 = 6$
P_5	$6 \cdot 2 \cdot (5,1) = (5,1)$	$(5,1,8)G = (5,1,5,6,4,10,2)$	$8+3+7+10+2 = 8$

Table 3.9: For the operation $s_4 * s_5$, the locally computed values, noted δ_i, the corresponding share vector for some random chosen x_i, and each participant's resulting share of $s_4 * s_5$.

The last computation is addition, for a better overview we present the results of the two previous computations as well as the final shares in table 3.10.

P_i	Share of $(s_1 + s_2) * s_3$	Share of $s_4 * s_5$	Share of $f(s_1, \ldots, s_5)$
P_1	3	2	$3 + 2 = 5$
P_2	10	3	$10 + 3 = 2$
P_3	7	8	$7 + 8 = 4$
P_4	5	6	$5 + 6 = 0$
P_5	4	8	$4 + 8 = 1$

Table 3.10: The resulting shares held by each participant P_i.

Lastly, we simply apply the methodology for reconstruction of a secret in $LSSS(C)$ to get the result of the function f, i.e. we need to solve $(s', y)H^T = 0$, where $y = (5, 2, 4, 0, 1)$ and $s' \in \mathbb{F}_{11}^2$.

$$(s'_1, s'_2, 5, 2, 4, 0, 1) \begin{bmatrix} 10 & 8 & 5 & 1 \\ 3 & 8 & 4 & 2 \\ 8 & 5 & 1 & 7 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = 0.$$

We get the solution $f(s_1, \ldots, s_5) = s' = (4, 2)$. △

4 Reed-Muller Codes

A generalization of Reed-Solomon codes, are Reed-Muller codes, where the polynomials that are used to construct the code have m variables. We define the degree of a multivariate monomial as $\deg(x_1^{i_1} \cdots x_m^{i_m}) = \sum_{j=1}^{m} i_j$.

A (ζ, m) Reed-Muller code C over $\mathbb{F}_q$ is

$$C = \{(f(p_1), \ldots, f(p_n)) : f \in \mathbb{F}_q[x_1, \ldots, x_m], \deg(f) \leq \zeta\},$$

where $\{p_1, \ldots, p_n\} = \mathbb{F}_q^m$.

We can see that the codelength of an (ζ, m) Reed-Muller code is $n = q^m$, however, the dimension and minimum weight of a Reed-Muller code are not clear from the definition.

Theorem 4.1. *The dimension of an (ζ, m) Reed-Muller code over $\mathbb{F}_q$ is*

$$k = \sum_{i=0}^{\zeta} \sum_{j=0}^{m} (-1)^j \binom{m}{j} \binom{i - jq + m + 1}{i - jq}.$$

Proof. Let C be an (ζ, m) Reed-Muller code over $\mathbb{F}_q$. Note, that the evaluation of the monomials in $\mathbb{F}_q[x_1, \ldots, x_m]$ of degree $\leq \zeta$ form a basis of C. Hence, the dimension of C is equal to the number of monomials in $\mathbb{F}_q[x_1, \ldots, x_m]$ of degree $\leq \zeta$.

For the theorem to follow, we only need to proof that the number of monomials of degree i, for $0 \leq i \leq \zeta$, is

$$\sum_{j=0}^{m} (-1)^j \binom{m}{j} \binom{i - jq + m + 1}{i - jq}. \tag{4.1}$$

The number of monomials of degree i is equal to the number of m-tuples $(j_1, \ldots, j_m)$, where $0 \leq j_a \leq q - 1$, for $1 \leq a \leq m$, and $j_1 + \cdots + j_m = i$.

This is clearly a combinatorial problem, where j_a is the number of objects in cell a, for $1 \leq a \leq m$, and i is the total number of objects. Firstly, we have number of ways i objects can be placed in m cells without any restrictions, which is

$$\binom{i+m-1}{i}.$$

This would be the number of monomials if there was no upper bound of j_a, for $1 \leq a \leq m$. However, since we are in $\mathbb{F}_q$, we need to exclude any of these combinations, where one or more cells contains at least q objects.

The number of ways i objects can be placed in m cells, such that j specified cells contain at least q objects is

$$\binom{i-jq+m+1}{i-jq}.$$

We need to subtract these combinations for $0 \leq j \leq m$, and thus by the inclusion-exclusion principle equation (4.1) follows, and thus the theorem. □

Theorem 4.2. *The minimum weight of an (ζ, m) Reed-Muller code over $\mathbb{F}_q$, where $\zeta = \lambda(q-1)+\alpha$ and $0 \leq \alpha < q-1$, is*

$$d = (q-\alpha)q^{m-\lambda-1}.$$

The proof of theorem 4.2 is omitted, since it would require deviation from the main focus of the project. A classic proof of theorem 4.2 is given in [10], and a proof based on Gröbner bases is given in [6].

From the minimum weight of a Reed-Muller code C and theorem 1.7, we can now give an upper bound for the reconstruction of $LSSS(C)$. Let C be an (ζ, m) Reed-Muller code over $\mathbb{F}_q$, where $\zeta = \lambda(q-1)+\alpha$ and $0 \leq \alpha < q-1$, then

$$r \leq (q^m - 1) - (q-\alpha)q^{m-\lambda-1} + 1 = ((q^\lambda - 1)q + \alpha)q^{m-\lambda-1},$$

where r is the reconstruction of $LSSS(C)$.

Now that we have the dimensions of Reed-Muller codes, we also want to know the dual code. The dual code can be used to find self-dual Reed-Muller codes, which we can use for multiplicative SSSs.

Theorem 4.3. *[1] The dual code $C^{\perp}$ of the (ζ, m) Reed-Muller code C over $\mathbb{F}_q$, is the $(m(q-1)-\zeta-1, m)$ Reed-Muller code over $\mathbb{F}_q$.*

Proof. Let C be an (ζ, m) Reed-Muller code over $\mathbb{F}_q$ and C' an (ζ', m) Reed-Muller code over $\mathbb{F}_q$. If $C^{\perp} = C'$, then $c \cdot c' = 0$ for all $c \in C$ and $c' \in C'$. Let $c = (f(p_1), \ldots, f(p_n)) \in C$ and $c' = (g(p_1), \ldots, g(p_n)) \in C'$, where $f = x_1^{f_1} \cdots x_m^{f_m}$ and $g = x_1^{g_1} \cdots x_m^{g_m}$, and let $(p_i)_j$ be the jth coordinate of the point p_i, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Then

$$\begin{aligned}
c \cdot c' &= f(p_1)g(p_1) + \cdots + f(p_n)g(p_n) \\
&= \sum_{i=1}^{n} f(p_i)g(p_i) = \sum_{i=1}^{n} \Big(((p_i)_1)^{f_1} \cdots ((p_i)_m)^{f_m}) \Big) \Big(((p_i)_1)^{g_1} \cdots ((p_i)_m)^{g_m}) \Big) \\
&= \sum_{i=1}^{n} \Big(((p_i)_1)^{f_1+g_1} \cdots ((p_i)_m)^{f_m+g_m} \Big) \\
&= \sum_{a_1 \in \mathbb{F}_q} a_1^{f_1+g_1} \left(\sum_{\substack{p_i \in \mathbb{F}_q^m \\ (p_i)_1 = a_1}} \Big(((p_i)_2)^{f_2+g_2} \cdots ((p_i)_m)^{f_m} \Big) \right) \qquad \text{(by the distributive law)} \\
&= \cdots = \sum_{a_1 \in \mathbb{F}_q} a_1^{f_1+g_1} \cdots \sum_{a_m \in \mathbb{F}_q} a_m^{f_m+g_m}.
\end{aligned}$$

It is clear that $c \cdot c' = 0$ if and only $\sum_{a_i \in \mathbb{F}_q} a_i^{f_i+g_i} = 0$, for one or more $1 \leq i \leq m$. Since each sum is a summation of all elements of $\mathbb{F}_q$ to some power, we need only consider the powers. Let b denote the power.

For $b = k(q-1)$, if $k = 0$ then

$$\sum_{a \in \mathbb{F}_q} a^b = \sum_{a \in \mathbb{F}_q} a^0 = q \equiv 0 \mod q.$$

If $k > 0$, from Fermat's Theorem we know that $a^q - a \equiv 0 \mod q$, for $a \in \mathbb{F}_q^*$, hence

$$\sum_{a \in \mathbb{F}_q} a^b = \sum_{a \in \mathbb{F}_q} a^{k(q-1)} = 0 + \sum_{a \in \mathbb{F}_q^*} a^{k(q-1)} = q - 1,$$

where $k > 0$.

For $b \neq k(q-1)$, since $\mathbb{F}_q^* = \{\alpha^0, \ldots, \alpha^{q-2}\} = \langle \alpha \rangle$, for some $\alpha \in \mathbb{F}_q^*$, we can write

$$\sum_{a \in \mathbb{F}_q} a^b = 0 + \sum_{i=0}^{q-2} \left(\alpha^i\right)^b = \sum_{i=0}^{q-2} \left(\alpha^b\right)^i = \frac{1 - \left(\alpha^b\right)^{q-1}}{1 - \alpha^b} = \frac{1-1}{1-\alpha^b} \equiv 0 \mod q.$$

Note that $\alpha^b = 1$, if and only $b = k(q-1)$, hence $1 - \alpha^b \neq 0$.

We see that $c \cdot c' = 0$ if and only if $f_i + g_i \neq k(q-1)$, where $k > 0$, for some $1 \leq i \leq m$. Hence, as long as $\deg f + \deg g < m(q-1)$, we know that $c \cdot c' = 0$. Since, f and g are monomials, it is clear that the result follows for all $c \in C$ and $c' \in C'$. Hence, $c \cdot c' = 0$, for all $c \in C$ and $c' \in C'$, if $\zeta + \zeta' < m(q-1)$, i.e. $\zeta' \leq m(q-1) - 1 - \zeta$.

Lastly, in order to proof that $C^{\perp} = C'$, where $\zeta' = m(q-1) - 1 - \zeta$, we need to show that $\dim C' = n - \dim C$. From theorem 4.1, we know that

$$\dim C = \sum_{i=0}^{\zeta} \sum_{j=0}^{m} (-1)^j \binom{m}{j} \binom{i - jq + m + 1}{i - jq}.$$

From the proof of theorem 4.1 we know that the number of monomials of degree i is

$$\sum_{j=0}^{m} (-1)^j \binom{m}{j} \binom{i - jq + m + 1}{i - jq},$$

and the dimension of the dual of C is equal to the number of monomials of degree $i \nleq \zeta$, i.e. $\zeta + 1 \leq i \leq m(q-1)$, which, by symmetry of binomial coefficients, is equal to the number of monomials of degree i, where $(\zeta + 1) - (\zeta + 1) = 0 \leq i \leq m(q-1) - (\zeta + 1)$. Hence,

$$\dim C^{\perp} = \sum_{i=0}^{m(q-1)-\zeta-1} \sum_{j=0}^{m} (-1)^j \binom{m}{j} \binom{i - jq + m + 1}{i - jq} = \dim C',$$

and it follows that the dual code of an (ζ, m) Reed-Muller code over $\mathbb{F}_q$ is an $(m(q-1) - 1 - \zeta, m)$ Reed-Muller code over $\mathbb{F}_q$. □

From theorem 2.12 we know that $LSSS(C)$ is multiplicative if the code C is self-orthogonal, and $\ell = 1$. Hence by theorem 4.3, we have the following result for multiplicative $LSSS(C)$, where C is a Reed-Muller code.

Corollary 4.4. *Given an (ζ, m) Reed-Muller code C over $\mathbb{F}_q$, where $2\zeta = m(q-1) - 1$, then $LSSS(C)$ is multiplicative if $\ell = 1$.*

From theorem 4.3 we see that an (ζ, m) Reed-Muller code over $\mathbb{F}_q$ is self-dual, if and only if $2\zeta = m(q-1) - 1$. Note, that if q is odd then $m(q-1)$ is even for any m, and $2\zeta \neq m(q-1) - 1$ for all ζ. Since, the product of two odd numbers is an odd number, and $q = p^k$, for some prime number p, a Reed-Muller code over $\mathbb{F}_q$ can only be self-dual if $q = 2^k$.

Example 4.5

Let C be a $(1,3)$ Reed-Muller code over $\mathbb{F}_2$, by corollary 4.4 the scheme $LSSS(C)$, where $\ell = 1$, is multiplicative. The matrix G is a generator matrix for the code C.

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

We want to compute $s_1 * s_2$ using $LSSS(C)$. Note that the number of participants in $LSSS(C)$ is $q^m - \ell = 2^3 - 1 = 7$. Participants P_1 and P_2 generate shares for their secrets.

P_i	s_i	x_i	$(s_i, x_i)G$	Shares of s_i
P_1	1	$(0,1,0)$	$(1,1,0,1,0,1,0,0)$	$(1,0,1,0,1,0,0)$
P_2	0	$(1,1,1)$	$(0,1,1,1,0,0,0,1)$	$(1,1,1,0,0,0,1)$

Table 4.1: The secrets and generated shares of participants P_1 and P_2.

The shares are distributed among the participants, and each participant P_i computes $c_i c_i' R_i$, where R_i is the ith row of the recombination matrix, and c_i and c_i' are participant P_i's received shares of s_1 and s_2, respectively.

As shown in the proof of theorem 2.12, the recombination matrix is $R = [q-1, \ldots, q-1]^T = [1, \ldots, 1]$. Hence, the participants only have to compute the product of their received shares. Each participant's received shares and the product hereof are shown in table 4.2.

P_i	Share of s_1	Share of s_2	Product of shares, $\tilde{c}_i$
P_1	1	1	1
P_2	0	1	0
P_3	1	1	1
P_4	0	0	0
P_5	1	0	0
P_6	0	0	0
P_7	0	1	0

Table 4.2: The shares of s_1 and s_2 as distributed among the participants, and the computed products of those shares.

Each participant P_i now generates shares for the product $\tilde{c}_i$ computed above, for $1 \leq i \leq 7$. The shares generated by each participant are shown in table 4.3.

P_i	$\tilde{c}_i$	x_i	$(\tilde{c}_i, x_i)G$	Shares of $\tilde{c}_i$
P_1	1	$(1,0,0)$	$(1,0,1,1,0,0,1,0)$	$(0,1,1,0,0,1,0)$
P_2	0	$(0,0,0)$	$(0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$
P_3	1	$(1,1,0)$	$(1,0,0,1,1,0,0,1)$	$(0,0,1,1,0,0,1)$
P_4	0	$(0,1,1)$	$(0,0,1,1,1,1,0,0)$	$(0,1,1,1,1,0,0)$
P_5	0	$(0,1,1)$	$(0,0,1,1,1,1,0,0)$	$(0,1,1,1,1,0,0)$
P_6	0	$(0,1,0)$	$(0,0,1,0,1,0,1,1)$	$(0,1,0,1,0,1,1)$
P_7	0	$(1,1,1)$	$(0,1,1,1,0,0,0,1)$	$(1,1,1,0,0,0,1)$

Table 4.3: Shares generated by each participant for the secrets $\tilde{c}_i$.

The shares of the secrets $\tilde{c}_i$, for $1 \leq i \leq 7$, received by each participant are shown in table 4.4, as well as the sum of those shares.

P_i	Shares received	Sum of shares received
P_1	$(0,0,0,0,0,0,1)$	1
P_2	$(1,0,0,1,1,1,1)$	1
P_3	$(1,0,1,1,1,0,1)$	1
P_4	$(0,0,1,1,1,1,0)$	0
P_5	$(0,0,0,1,1,0,0)$	0
P_6	$(1,0,0,0,0,1,0)$	0
P_7	$(0,0,1,0,0,1,1)$	1

Table 4.4: Shares received by each participant for the secrets $\tilde{c}_i$, and their sums.

Each participant now have a share of the secret $s = s_1 * s_2$, and left is only the reconstruction of the secret from these shares. We get the equation $(s, x)G = (s, 1, 1, 1, 0, 0, 0, 1)$, which is equivalent to the equation system

$$\begin{aligned} s + x_1 &= 1, & s + x_2 &= 1, \\ s + x_3 &= 1, & s + x_1 + x_2 &= 0, \\ s + x_1 + x_3 &= 0, & s + x_2 + x_3 &= 0, \\ s + x_1 + x_2 + x_3 &= 1. \end{aligned}$$

Solving the equations we get $s = s_1 * s_2 = 0$. △

In order to examine the security of $LSSS(C)$ from example 4.5 we consider the bounds of privacy and reconstruction given in theorem 1.7. First, we need to compute the minimum weight, d, of the code C. From theorem 4.2 we know that $d = (q-\alpha)q^{m-\lambda-1}$, where $\zeta = \lambda(q-1)+\alpha$, and $0 \leq \alpha < q-1$. Note, that since $C \subset \mathbb{F}_2^8$ it is clear that $\lambda = \zeta = 1$ and $0 \leq \alpha < q-1 = 1$, i.e. $\alpha = 0$, hence

$$\begin{aligned} d &= (q-\alpha)q^{m-\lambda-1} \\ &= (2-0)2^{m-\zeta-1} \\ &= 2^{m-\zeta} = 2^{3-1} = 4. \end{aligned}$$

Now by theorem 1.7 we get following bounds for the privacy and reconstruction.

$$\begin{aligned} t &\geq d^{\perp} - \ell - 1 = d = 4, \\ r &\leq n + \ell - d + 1 = 8 - 4 + 1 = 5. \end{aligned}$$

Since we know that $t+\ell \leq r$ for any SSS, we have $4+1 \leq t+\ell \leq r \leq 5$, hence the bounds are sharp, i.e. $t = 4$ and $r = 5$.

5 Actively Secure MPC Protocol

In section 2.1 we argued that protocol 1 is an actively secure MPC protocol for functions consisting of addition and scalar multiplication, when we assume that the participants are honest during the initial distribution of shares. In this chapter we introduce a method for distributing shares, such that we are able to verify the received shares. By applying this method we also revise the multiplication step in protocol 2, in order to construct an actively secure MPC protocol for any function using a strongly multiplicative SSS with $\ell = 1$.

In the following we present a method for participants to verify the shares they receive without receiving additional information. The following methodology is heavily inspired by the 'Protocol Generalized Commit' from [4].

5.1 Distribution Method for Active Security

In this section we introduce a methodology for distributing and verifying participants' shares, using a strongly multiplicative $LSSS(C)$ with $\ell = 1$. The methodology is heavily inspired by the 'Protocol Generalized Commit' from page 154 of [4].

Throughout this section participant P_j will be the one distributing shares. Let C be an $[n+1, k, d]$ code over $\mathbb{F}_q$, let M be the generator matrix for the code C that is used for generating shares in the scheme $LSSS(C)$, and let t and $\hat{r}$ be the privacy and product reconstruction of $LSSS(C)$, respectively.

Participant P_j constructs a random symmetric matrix $Q_{a_j} \in \mathbb{F}_q^{k \times k}$, with a_j as the top left coordinate, where $a_j \in \mathbb{F}_q$ is the secret that P_j intends to share. Let $m_i \in \mathbb{F}_q^k$ be the ith column of the matrix M, then P_j sends $u_i = m_{i+1}^T Q_{a_j}$ to P_i, for $1 \leq i \leq n$. Note, that by the SSS the first coordinate of u_i is a share of a_j. Hence each participant P_i now has a vector u_i containing a share of a_j.

Note that since Q_{a_j} is symmetric then

$$u_i m_{h+1} = \left(m_{i+1}^T Q_{a_j}\right) m_{h+1} = m_{h+1}^T \left(m_{i+1}^T Q_{a_j}\right)^T = \left(m_{h+1}^T Q_{a_j}\right) m_{i+1} = u_h m_{i+1},$$

for all $1 \leq i, h \leq n$. Hence, if P_i computes $u_i m_{h+1}$ and sends it to P_h, then P_h can verify if the shares received by P_i and P_h are consistent. If $u_i m_{h+1} = u_h m_{i+1}$, then P_h will conclude that the shares are consistent. Furthermore, since Q_{a_j} is an unknown symmetric matrix for all participants except P_j, this verification process can be done without any loss of security.

During this distribution of shares, each P_i sends $u_i m_{h+1}$ to P_h, and each P_h checks that $u_i m_{h+1} = u_h m_{i+1}$, for $1 \leq i, h \leq n$. If the check is false, then P_h will accuse P_j of being corrupted. However, this only proves that at least one of P_i, P_j, and P_h is corrupted, since P_i could have sent the wrong value to P_h, or P_h could be corrupted and accuse P_j even though the check is true.

Therefore, we include a step such that an honest participant, P_h, will not accuse P_j if they receive the wrong $u_i m_{h+1}$ from P_i. We solve this by letting P_j respond to accusations, in this case by broadcasting publicly the value $u_i m_{h+1}$ corresponding to each accusation. Hence, if both P_i and P_h accuse P_j then both numbers will be broadcast publicly, if they are not equal then P_j is clearly corrupt. If only P_h accuses P_j then the number will be broadcast once, and P_h will be able to check if the number received from P_i is incorrect.

The problem here is that P_j can still be corrupted even if being honest when broadcasting, therefore, for each P_h, where $u_h m_{i+1}$ is broadcast, P_h checks that it matches with the u_h received. If this is not the case, then P_h accuses P_j for being corrupt. Hence, this step checks that P_j calculated the value $u_h m_{i+1}$ with the same u_h as was sent to P_h. For each P_h accusing P_j in this step, P_j broadcasts u_h.

Lastly, each P_h that accused P_j above, verifies that the u_h broadcast is equal to the u_h received privately, if not then P_h accuses P_j. If the broadcast $u_h m_{i+1}$ and u_h are inconsistent, or if more than t participants have accused P_j during the methodology, then the participants output fail. Otherwise participants who accused P_j due to the broadcast $u_h m_{i+1}$ and the privately received u_h not matching, replaces the privately received u_h with the broadcast u_h.

The reason for the methodology failing if more than t participants accuse P_j is that the SSS is strongly multiplicative, i.e. $\hat{r} = n - t$. Hence, if more than t participants accuse P_j, then either P_j is corrupt, or more than t participants are corrupt.

This expanded methodology for distribution and verifying shares can be summed up in the following 5 steps. Note that any accusation is a public broadcast.

1. P_j wants to distribute shares of a_j. P_j randomly generates a symmetrical matrix Q_{a_j}, where a_j is the top left coordinate, and sends $u_i = m_{i+1}^T Q_{a_j}$ to P_i for $1 \leq i \leq n$. P_i sends $u_i m_{h+1}$ to P_h.

2. If $u_i m_{h+1} \neq u_h m_{i+1}$ for some i, then P_h accuses P_j of being corrupted, and broadcasts publicly the corresponding is.
   For each accusation, P_j broadcasts publicly $u_i m_{h+1}$ for the corresponding i and h.

3. If the broadcast $u_h m_{i+1}$ does not match the u_h received, then P_h accuses P_j.
   For each accusation, P_j broadcasts publicly the distributed vector u_h for the corresponding h.

4. If the broadcast u_h does not match the privately received u_h, then P_h accuses P_j.

5. If the broadcast information is inconsistent, or if more than t participants have accused P_j, the participants output fail.
   Otherwise, participants who accused P_j in step 3 replaces the privately received vectors u_h with those broadcast by P_j, remaining participants keep the vectors received in step 1. The received share of a_j is equal to the first coordinate of the vectors.

In order to argue that more than t participants accusing P_j is enough for the methodology to fail, we give the following theorem.

Theorem 5.1. *In the methodology above, if the SSS is strongly multiplicative, and no more than t participants accuses P_j, then no matter how the corrupt participants behave, honest participants will either all output fail, or will all have a share of the same sharing of a_j. [4]*

Proof. Since the choice whether to return fail is based on public information it is clear that participants who remain honest will either all output fail, or will all have a share of a. We assume that the honest participants did not output fail. Let $\mathcal{I}$ be the set of all participants, A be the set of participants that accused P_j, and L be the set of corrupt participants at the end of the distribution.

We assume that $|L| \leq t$, since otherwise the methodology would not work, and that $|A| \leq t$. Since the SSS is strongly multiplicative we know from theorem 2.7 that $3t < n$, hence $S = (\mathcal{I} \setminus A) \setminus L$ is the set of honest participants, who did not accuse P_j, and $|S| > t$. Since S is not a rejected set, and $\ell = 1$, the set of participants S is able to reconstruct the secret.

Let H be the set of all honest participants. If $P_i \in H$ had a new vector broadcast in step 3, and $P_h \in S$ accuses P_j in step 4, then $P_h \notin S$. Conversely if, P_i did not have a new vector broadcast in step 3, then P_i keeps the originally received vector, as does P_h, hence neither have accused P_j at any time. Therefore, we know, that if $P_i \in H$ and $P_h \in S$, then the check $u_i m_{h+1} \neq u_h m_{i+1}$ in step 2 of the methodology will be false.

Consider the matrices U_H and U_S, with rows equal to the vectors $\{u_x : x \in H\}$ and $\{u_y : y \in S\}$ respectively. Let M be the generator matrix used for generating shares in the SSS, and let M_V be the matrix consisting of the columns $\{M_{i+1} : i \in V\}$. Then $U_S M_S$ is a matrix containing all check values $u_i m_{h+1}$, for $P_i, P_h \in S$. Therefore, from the above it follows that $U_H M_S = (U_S M_H)^T$.

Let v_S be the vector that reconstructs the secret a_j from the shares of participants in S. Let x_S be the vector, such that $x_S = v_S U_S$, i.e. $x_S = (a_j, x)M$, where $x \in \mathbb{F}_q^{k-1}$ is the random vector used to generate the distributed shares. Hence,

$$x_S M_H = v_S U_S M_H = v_S (U_H M_S)^T = v_S M_S^T U_H^T = \boldsymbol{e}_1^k U_H^T,$$

where $\boldsymbol{e}_1^k \in \mathbb{F}_q^k$ is the standard vector $(1, 0, \ldots, 0)$. Note that, $\boldsymbol{e}_1^k U_H^T$ is equal to the vector of shares of a_j distributed to participants in H, i.e. honest participants. Hence, it follows that if at most t participants accuses P_j, then the set S can reconstruct the secret, and the shares held by the honest participants are shares of a_j from the same sharing of a_j. □

5.2 Actively Secure Multiplication

In this section we will show how the methodology in section 5.1 is applied in order to create an actively secure multiplication in a MPC, where $LSSS(C)$ is strongly multiplicative, with $\ell = 1$. Firstly, we have to note that during multiplication in a MPC each participant will try distributing shares, hence the methodology is run for each participant. In the following any distribution of shares mentioned uses the methodology of section 5.1.

Assume that $s_1 s_2$ is the multiplication we wish to generate shares for, where s_1 and s_2 are secrets with distributed share vectors $(d_{1,1}, \ldots, d_{1,n})$ and $(d_{2,1}, \ldots, d_{2,n})$, respectively, and let R_i^A be the ith row of the recombination matrix for the set $A \subseteq \mathcal{I}$, for $1 \leq i \leq n$. Using protocol 2 each participant, P_i, would have distributed shares for $d_{1,i} d_{2,i} R_i^{\mathcal{I}}$, however, since the recombination matrix used is for all participants, i.e. $\mathcal{I}$, the distributed shares are only usable if all participants distributes shares correctly.

Therefore, for actively secure multiplication of $s_1 s_2$, the participant P_i, distributes shares of $d_{1,i} d_{2,i}$. Let A be the set of participants for which this distribution did not fail. Then each participant should have received a share of $d_{1,j} d_{2,j}$ for which $j \in A$. From the definition of product reconstruction, definition 2.3, we know that $\sum_{j \in A} \Delta_{(1,2),j,i}$ is a share of $s_1 s_2$, where $\Delta_{(1,2),j,i}$ is the share of $d_{1,j} d_{2,j} R_j^A$ received by P_i. Due to linearity it follows that $\sum_{j \in A} \delta_{(1,2),j,i} R_j^A$ is a share of $s_1 s_2$, where $\delta_{(1,2),j,i}$ is the share of $d_{1,j} d_{2,i}$ received by participant P_i. Hence, each participant P_i preforms the following steps in order to generate shares for $s_1 s_2$.

1. P_i distributes shares for $d_{1,i}d_{2,i}$, where $d_{1,i}$ and $d_{2,i}$ is his or her shares of s_1 and s_2 respectively.

2. P_i computes $\sum_{j\in A} \delta_{(1,2),j,i} R_j^A$, where A is the set of participants for which the distribution of shares in step 1 did not fail, and $\delta_{(1,2),j,i}$ is the share of $d_{1,j}d_{2,j}$ received by P_i.

Since, the SSS has $(n-t)$-product reconstruction the MPC fails if the set of participants A is less than $n-t$. Protocol 3 is a revised version of perotocol 2 for active security.

Input
Each participant generates shares of their private data according to the SSS, and distributes the shares among the participants.

Computation
The following steps are run for each computation in the function, until a share of f is stored for each participant, or the protocol fails.

- **Addition** If $s_m + s_h$ is the computation, where s_m and s_h are secrets, then P_j computes $d_{m,j} + d_{h,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ and $d_{h,j}$ are P_j's shares of s_m and s_h respectively. The result is stored privately as P_j's share of the secret $s_m + s_h$.

- **Scalar multiplication** If λs_m is the computation, where s_m is a secret and λ is a scalar, then P_j compute $\lambda d_{m,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ is P_j's share of the s_m. The result is stored privately as P_j's share of the secret λs_m.

- **Multiplication of secrets** If $s_m s_h$ is the computation, where s_m and s_h are secrets, then each participant P_j takes the following steps, where $d_{m,j}$ and $d_{h,j}$ are P_j's shares of s_m and s_h, respectively, R_j^B is the jth row of the SSS's recombination matrix for the set of participants B, for $1 \leq j \leq n$, and $A = \mathcal{I}$

  1. P_j computes $\delta_{(m,h),j} = d_{m,j}d_{h,j}$.
  2. P_j generates shares of $\delta_{(m,h),j}$ according to the SSS, and distributes the shares among the participants. For each P_j, where the distribution of shares fail, remove P_j from A.
  3. If $|A| \geq n-t$, then P_j computes the sum $\sum_{P_j\in A} \Delta_{(m,h),i,j} R_j^A$, where $\Delta_{(m,h),i,j}$ is P_j's share of $d_{m,i}d_{h,i}$. The sum is stored privately as P_j's share of $s_m s_h$.
     If $|A| < n-t$, then the protocol fails.

Output and reconstruction
Each participant outputs their share of f. The shares are then used for reconstruction using the methodology outlined in section 1.3 that corresponds to the linear SSS applied in this protocol.

Protocol 3: Actively secure MPC protocol using a strongly multiplicative SSS, with $\ell = 1$.

6

Passively Secure MPC Protocol for Grouping of Multiplications

In previous chapters each participant had to generate and distribute shares among the participants each time two secret had to be multiplied. E.g. using protocol 2 with the function $f = s_1 * s_2 * s_3$, the participants would first have to compute shares for $s_1 * s_2$ by generating and distributing shares, and then by using that result they could compute shares for $s_1 * s_2 * s_3$, again by generating and distributing shares.

In this chapter we present a stronger definition of product reconstruction, which will allow us to group multiplication of secrets when using linear SSS with such property. This will allow us to revise our previous passively secure MPC protocol, protocol 2, as to reduce the number of times the participants communicate during the computation phase.

Definition 6.1. *A linear SSS has $(\mu, \hat{r})$-product reconstruction if for every set $A \subseteq \mathcal{I}$, where $|A| \geq \hat{r}$, there exists linear functions $\rho_A^\alpha : \mathbb{F}_q^{|A|} \to \mathbb{F}_q^\ell$, for $2 \leq \alpha \leq \mu$, such that*

$$\rho_A^\alpha\Big((c_1)_A * \cdots * (c_\alpha)_A\Big) = s_1 * \cdots * s_\alpha,$$

where $c_1, \ldots, c_\alpha$ are share vectors of the secrets $s_1, \ldots, s_\alpha$ respectively.

Note, that by definition 6.1 a SSS with $(\mu, \hat{r})$-product reconstruction allows the multiplication of μ secrets from their shares in one computation. Also note, that the previous definition of product reconstruction, definition 2.3, is equivalent to $(2, \hat{r})$-product reconstruction.

As with the previous definition of product reconstruction, since the functions ρ_A^α are linear they can be represented by a matrix transformation. We call, the matrix transformation that represents the function ρ_A^α, the recombination matrix for the set of participants A and the product of α secret, noted $R^{A,\alpha}$.

By changing the multiplication of secrets step of the computation phase in protocol 2 to allow for the grouping of the multiplication of μ secrets we get the passively secure MPC protocol for grouping of multiplications, protocol 4.

Input
Each participant generates shares of their private data according to the SSS, and distributes the shares among the participants.

Computation
The following steps are run for each computation in the function, until a share of f is stored for each participant.

- **Addition** If $s_m + s_h$ is the computation, where s_m and s_h are secrets, then P_j computes $d_{m,j} + d_{h,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ and $d_{h,j}$ are P_j's shares of s_m and s_h respectively. The result is stored privately as P_j's share of the secret $s_m + s_h$.
- **Scalar multiplication** If λs_m is the computation, where s_m is a secret and λ is a scalar, then P_j compute $\lambda d_{m,j}$, for $1 \leq j \leq n$, where $d_{m,j}$ is P_j's share of the s_m. The result is stored privately as P_j's share of the secret λs_m.
- **Multiplication of secrets** If $s_{m_1} * \cdots * s_{m_\alpha}$ is the computation, where $s_{m_1}, \ldots, s_{m_\alpha}$ are secrets, and $2 \leq \alpha \leq \mu$, then each participant P_j takes the following steps, where $d_{m_1,j}, \ldots, d_{m_\alpha,j}$ are P_j's shares of $s_{m_1}, \ldots, s_{m_\alpha}$, respectively, and $R_j^{\mathcal{I},\alpha}$ is the jth row of the SSS's recombination matrix for α multiplications, for $1 \leq j \leq n$.
  1. P_j computes $\delta_{(m,h),j} = \left(\prod_{1 \leq i \leq \alpha} d_{m_i,j}\right) R_j^{\mathcal{I},\alpha}$.
  2. P_j generate shares of $\delta_{(m,h),j}$ according to the SSS, and distributes the shares among the participants.
  3. P_j sums the shares received in 2. The sum is stored privately as P_j's share of the secret $s_{m_1} * \cdots * s_{m_\alpha}$.

Output and reconstruction
Each participant outputs their share of f. The shares are then used for reconstruction using the methodology outlined in section 1.3 that corresponds to the linear SSS applied in this protocol.

Protocol 4: Passively secure MPC protocol including grouping of up to μ multiplications using a linear SSS with (μ, n)-product reconstruction SSS.

Similar to theorem 2.12 for multiplicative SSS, we show that if $C^{*\alpha} \perp \mathbf{1}$, for all $2 \leq \alpha \leq \mu$, then C has (μ, n)-product reconstruction.

Theorem 6.2. *Let C be an $[n+1, k, d]$ code over $\mathbb{F}_q$. If $C^{*\alpha} \perp \mathbf{1}$, for all $2 \leq \alpha \leq \mu$, then $LSSS(C)$ has (μ, n)-product reconstruction.*

Proof. Let C be an $[n+1, k, d]$ code over $\mathbb{F}_q$, where $C^{*\alpha} \perp \mathbf{1}$, for all $2 \leq \alpha \leq \mu$. Consider $LSSS(C)$, and the codewords $c_i = (s_i, c_{i,1}, \ldots, c_{i,n}) \in C$, for $1 \leq i \leq \alpha$. Note, that $(c_{i,1}, \ldots, c_{i,n}) \in X_{s_i}$, for all $1 \leq i \leq \alpha$. Since $C^{*\alpha} \perp \mathbf{1}$, we know that $(c_1 * \cdots * c_\alpha) \cdot \mathbf{1} = 0$, hence

$$\left(\prod_{1 \leq i \leq \alpha} s_i, \prod_{1 \leq i \leq \alpha} c_{i,1}, \ldots, \prod_{1 \leq i \leq \alpha} c_{i,n} \right) \cdot \mathbf{1} = 0 \Leftrightarrow \prod_{1 \leq i \leq \alpha} s_i + \sum_{j=1}^{n} \left(\prod_{1 \leq i \leq \alpha} c_{i,j} \right) = 0$$

$$\Leftrightarrow \prod_{1 \leq i \leq \alpha} s_i = -\sum_{j=1}^{n} \left(\prod_{1 \leq i \leq \alpha} c_{i,j} \right).$$

It is clear that $LSSS(C)$ satisfies the definition of a linear SSS with (μ, n)-product reconstruction, and that the linear functions $\rho_{\mathcal{I}}^{\alpha}(v) = -\sum_{i=1}^{n} v_i = x$, where $(x, v) \in C^{*\alpha}$, for $2 \leq \alpha \leq \mu$. □

6.1 Spherically Punctured Reed-Muller Codes

In this section we will show how to construct linear codes C over $\mathbb{F}_2$ from Reed-Muller codes, such that $LSSS(C)$ has (μ, n)-product reconstruction.

The codes are similar to $(1, m)$ Reed-Muller codes over $\mathbb{F}_2$, except for two main differences. The polynomials that generate the code have no constant term, and rather than evaluating at all points of $\mathbb{F}_2^m$, the polynomials are only evaluated at points with certain weights.

Let $\mathbb{F}_2[x_1, \ldots, x_m]_{a=0}$ be the subspace of $\mathbb{F}_2[x_1, \ldots, x_m]$ consisting of polynomials with the constant term, a, equal to 0, and $V \subseteq \{1, \ldots, m\}$, we define the code

$$C_{(V,m)} = \{(f(p_1), \ldots, f(p_n)) : f \in \mathbb{F}_2[x_1, \ldots, x_m]_{a=0}, \deg f \leq 1\},$$

where $p_1, \ldots, p_n$ are the $n = \sum_{i \in V} \binom{m}{i}$ points in $\mathbb{F}_2^m$ with Hamming weight $\in V$.

For a $(1, m)$ Reed-Muller code over $\mathbb{F}_2$ the matrix G, where the first row is all 1's and the remaining m rows are a matrix with columns equal to all possible points in $\mathbb{F}_2^m$, is a generator matrix. E.g. let C be a $(1, 4)$ Reed-Muller code over $\mathbb{F}_2$, the following matrix is a generator matrix for C.

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

From this construction of G we can construct a generator matrix for $C_{(V,m)}$.

Let $A_{i,m}$ be the $m \times \binom{m}{i}$ matrix, where each column is a vector in $\mathbb{F}_2^m$ with weight i, for $0 \leq i \leq m$. Note, that the generator matrix G, shown above for a $(1,4)$ Reed-Muller code over $\mathbb{F}_2$, can be decomposed into the following matrices.

$$G = \left[\begin{array}{c|cccc|cccccc|cccc|c} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \underbrace{0}_{A_{0,4}} & \underbrace{0 & 0 & 0 & 1}_{A_{1,4}} & \underbrace{0 & 0 & 1 & 0 & 1 & 1}_{A_{2,4}} & \underbrace{0 & 1 & 1 & 1}_{A_{3,4}} & \underbrace{1}_{A_{4,4}} \end{array}\right].$$

Hence, if we delete the row of 1's from the generator matrix G, we will have the matrix $\tilde{G} = [A_{0,4}A_{1,4}A_{2,4}A_{3,4}A_{4,4}]$. Let $V = \{1,3,4\}$, then the code $C_{(V,4)}$ is generated by the matrix $[A_{1,4}A_{3,4}A_{4,4}]$. In other words, the matrix $[A_{i_1,m} \dots A_{i_{|V|},m}]$, where $V = \{i_1, \dots, i_{|V|}\}$, is a generator matrix for the code $C_{(V,m)}$. Note, that $A_{0,m}$ is never included in the generator matrix for any codes $C_{(V,m)}$, since $V \subseteq \{1, \dots, m\}$.

In order to construct $C_{(V,m)}$ such that $LSSS\left(C_{(V,m)}\right)$ has (μ, n)-product reconstruction, we show how to choose V and m such that $C^{*\mu}_{(V,m)} \perp \mathbf{1}$. Since we are in $\mathbb{F}_2$, we first show that it is enough that the codewords of $C^{*\mu}$ have even weight for $LSSS(C_{(V,m)})$ to have (μ, n)-product reconstruction.

Lemma 6.3. *Let C be an $[n+1, k, d]$ code over $\mathbb{F}_2$. If all codewords in $C^{*\mu}$ have even weight, then $LSSS(C)$ has (μ, n)-product reconstruction.*

Proof. Let C be an $[n+1, k, d]$ code over $\mathbb{F}_2$, where all codewords in $C^{*\mu}$ have even weight. Since $C \subseteq \mathbb{F}_2$, we have that $c * c = c$, for all $c \in C$, and it follows that $C^{*\alpha} \subseteq C^{*\mu}$, for all $1 \leq \alpha \leq \mu$, hence all codewords in $C^{*\alpha}$ for $1 \leq \alpha \leq \mu$ have even weight.

By theorem 6.2 it is enough to show that $C^{*\alpha} \perp \mathbf{1}$, for all $2 \leq \alpha \leq \mu$. Let $c_i = (c_{i,1}, \dots, c_{i,n+1}) \in C$, and $c_{i,j} = 1$, for m j's, where $1 \leq j \leq n+1$, for $1 \leq i \leq \alpha$. Hence $w_H(c_1 * \dots * c_\alpha) = m$, and thus $(c_1 * \dots * c_\alpha) \cdot \mathbf{1} = m$, but since $c_1 * \dots * c_\alpha \in C^{*\alpha}$ and all codeword in $C^{*\alpha}$ have even weight $m \equiv 0 \mod 2$. □

Hence, if we can construct an $[n+1, k, d]$ code C over $\mathbb{F}_2$, where the codewords of $C^{*\mu}$ all have even weight, then $LSSS(C)$ has (μ, n)-product reconstruction.

To construct a $C_{(V,m)}$ code where all codewords have even weight, we first look at the weight of the coordinatewise product of μ different rows of $A_{i,m}$. Recall that $C_{(V,m)}$ is a code over $\mathbb{F}_2$ and hence the matrices $A_{i,m}$ are also over $\mathbb{F}_2$.

Lemma 6.4. *The weight of the coordinatewise product of $\mu \geq 1$ different rows of $A_{i,m}$ for any $1 \leq i \leq m$ is $\binom{m-\mu}{i-\mu}$. [5]*

Proof. Let $v = r_1 * \cdots * r_\mu$, where $r_1, \ldots, r_\mu$ are pairwise distinct rows of $A_{i,m}$. If $v_j = 1$ then $r_1, \ldots, r_\mu$ all have 1 as their jth coordinate, else $v_j = 0$, for $1 \leq j \leq n$. Hence, the weight of v is equal to the number of coordinates where $r_1, \ldots, r_\mu$ all have 1.

The columns of $A_{i,m}$ are by definition all the possible vectors of length m with weight i. Hence, μ distinct rows of the matrix $A_{i,m}$ all have 1 in the same number of coordinates as the number of vectors of length $m - \mu$ with weight $i - \mu$. It follows that the weight of v, i.e. the coordinatewise product of μ different rows of $A_{i,m}$, is $\binom{m-\mu}{i-\mu}$. □

We introduce the following notation. The matrix $M^{*\mu}$ is the matrix with rows equal to the vectors

$$\{m_1 * \cdots * m_\mu : m_i \text{ is row a of } M, \text{ and } m_i \neq m_j, \text{ for } 1 \leq i, j \leq \mu\}.$$

That is, the matrix $M^{*\mu}$ is the matrix with rows equal to the coordinatewise product of μ different rows of the matrix M. For the matrices $A_{i,m}^{*\mu}$ we proof the following properties.

Lemma 6.5. *For $1 \leq \mu \leq i$ the matrix $A_{i,m}^{*\mu}$ is an $\binom{m}{\mu} \times \binom{m}{i}$ matrix, with columns of weight $\binom{i}{\mu}$ and rows of weight $\binom{m-\mu}{i-\mu}$. Furthermore, the columns and rows of $A_{i,m}^{*\mu}$ are all distinct. [5]*

Proof. First we proof that the rows of $A_{i,m}^{*\mu}$ are distinct. Let $\{r_1, \ldots, r_\mu\} \neq \{r'_1, \ldots, r'_\mu\}$ be two different sets of $1 \leq \mu \leq i$ pairwise distinct rows of $A_{i,m}$, where $r_1 * \cdots * r_\mu = r'_1 * \cdots * r'_\mu$. Let $s_1, \ldots, s_{i-\mu}$ be $i - \mu$ different rows of $A_{i,m}$ not in the two previous sets, hence we have that

$$r_1 * \cdots * r_\mu * s_1 * \cdots * s_{i-\mu} = r'_1 * \cdots * r'_\mu * s_1 * \cdots * s_{i-\mu}.$$

However, by lemma 6.4, the coordinatewise product of i different rows of $A_{i,m}$ has weight $\binom{m-i}{i-i} = \binom{m-i}{0} = 1$ and length $\binom{m}{i}$. Furthermore, any i distinct rows of $A_{i,m}$ only have one unique coordinate, where all i rows contain 1. Hence, $r_1 * \cdots * r_\mu \neq r'_1 * \cdots * r'_\mu$, which is a contradiction. Therefore, it follows that the rows of $A_{i,m}^{*\mu}$ are all distinct. Furthermore, from lemma 6.4 the weight of each row is $\binom{m-\mu}{i-\mu}$.

For the dimensions of $A_{i,m}^{*\mu}$, since the length of the rows is preserved during coordinatewise product the number of columns is equal to that of $A_{i,m}$, i.e. $\binom{m}{i}$. The number of rows in $A_{i,m}^{*\mu}$ is equal to the number of possible permutations of μ different rows of $A_{i,m}$, i.e. $\binom{m}{\mu}$. Hence, $A_{i,m}^{*\mu}$ is an $\binom{m}{\mu} \times \binom{m}{i}$ matrix.

We now proof that the columns in $A_{i,m}^{*\mu}$ are all distinct. Since the rows in $A_{i,m}^{*\mu}$ are all the possible coordinatewise products of μ different rows of $A_{i,m}^{*\mu}$, for two columns in $A_{i,m}^{*\mu}$ to be the equal, the two corresponding columns in $A_{i,m}^{*\mu}$ must both be equal to 1 in the same coordinates. Hence, if two columns are equal in $A_{i,m}^{*\mu}$, then the corresponding columns in $A_{i,m}$ are equal, which is a contradiction, hence the columns in $A_{i,m}^{*\mu}$ are all distinct.

Lastly, we need to proof the weight of each column in $A_{i,m}^{*\mu}$ is $\binom{i}{\mu}$. Note again that the rows in $A_{i,m}^{*\mu}$ are all the possible coordinatewise products of μ different rows of $A_{i,m}$, and the columns of $A_{i,m}$ are equal to all vectors of length m with weight i. Hence, the weight of the columns in $A_{i,m}^{*\mu}$ is equal to the number of ways in which to choose μ elements from a set of i elements. Note, that these elements are equal to the 1's in the columns of $A_{i,m}$. It follows that the weight of the columns in $A_{i,m}^{*\mu}$ is $\binom{i}{\mu}$. □

By lemma 6.5, if M is a generator matrix for the code $C_{(V,m)}$ then $M^{*\mu}$ is a generator matrix for some code

$$\tilde{C}_{(V,m)}^{*\mu} = \Big\{ c_1 * \cdots * c_\mu : c_i \in C_{(V,m)}, \text{ and } c_i \neq c_j, \text{ for } 1 \leq i, j \leq \mu \Big\}.$$

Since the code $C_{(V,m)}^{*\mu} = \big\{ c_1 * \cdots c_\mu : c_i \in C_{(V,m)}, \text{ for } 1 \leq i \leq \mu \big\}$, we have that

$$C_{(V,m)}^{*\mu} = \bigoplus_{1 \leq i \leq \mu} \tilde{C}_{(V,m)}^{*i}.$$

Hence, if all codewords in $\tilde{C}_{(V,m)}^{*i}$ have even weight for all $1 \leq i \leq \mu$, then all codewords in $C_{(V,m)}^{*\mu}$ have even weight, and by lemma 6.3 $LSSS(C_{(V,m)})$ has (μ, n)-product reconstruction.

We introduce a vector representation for the choice of $V \subseteq \{1, \ldots, m\}$. Let $v \in \mathbb{F}_2^m$, where $v_i = 1$ if $i \in V$, and $v_i = 0$ if $i \notin V$, for $1 \leq i \leq m$. For notational ease we let $C_{(V,m)} = C_{(v,m)}$.

In the following we proof that the set of possible choices of $v \in \mathbb{F}_2^m$ such that all codewords in $\tilde{C}_{(v,m)}^{*\alpha}$ have even weight, for some $1 \leq \alpha \leq m$, is a subspace of $\mathbb{F}_2^m$.

Theorem 6.6. *The set of possible $v \in \mathbb{F}_2^m$ such that all codewords in $\tilde{C}_{(v,m)}^{*\alpha}$ have even weight, for $1 \leq \alpha \leq m$, is a subspace of $\mathbb{F}_2^m$ with dimension $m - 1$.*

Proof. First note that if the rows of the generator matrix $[A_{i_1,m}^{*\alpha} \ldots A_{i_{|V|},m}^{*\alpha}]$, where $V = \{i_1, \ldots, i_{|V|}\}$, have even weight, then any codeword of $\tilde{C}_{(v,m)}^{*\alpha}$ will also have even weight.

We divide the matrices $A_{i,m}^{*\alpha}$, for $1 \leq i \leq m$, into three categories based on the weight of their rows, zero, non-zero even, and odd.

Let $\boldsymbol{e}_i^m$ be the ith standard vector of length m, i.e. $\boldsymbol{e}_i^m = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is the ith coordinate. The weight of the rows of each $A_{i,m}^{*\alpha}$, for $1 \leq i \leq \alpha - 1$, all have weight zero, we define the set $Z_\alpha = \{\boldsymbol{e}_j^m : 1 \leq j \leq \alpha - 1\}$. The set Z_α denotes the choices of a matrix $A_{j,m}$, where the rows of $A_{j,m}^{*\alpha}$ have weight zero. Note, that if $\alpha = 1$, then $Z_\alpha = \emptyset$.

For the rows of even weight we define the set $E_\alpha = \left\{\boldsymbol{e}_j^m : \alpha \leq j \leq m, \binom{m-\alpha}{j-\alpha} \text{ is even } \right\}$. The set E_α denotes the choices of a matrix $A_{j,m}$, where the rows of $A_{j,m}^{*\alpha}$ have even weight.

For the matrices with rows of odd weight, we define a set of choices of pairs of matrices, such that the combined matrix of the two will have rows of even weight. Let $\boldsymbol{F}_{i,j}^m = (0, \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{F}_2^m$, where the two 1's occur at position i and j, and $i < j$. Define the set $O_\alpha = \left\{\boldsymbol{f}_{i,j}^m : \binom{m-\alpha}{i-\alpha} \text{ and } \binom{m-\alpha}{j-\alpha} \text{ are odd, } \binom{m-\alpha}{x-\alpha} \text{ is even for all } i < x < j\right\}$. The set O_α denotes the choices of a consecutive pair of matrices, $A_{i,m}$ and $A_{j,m}$, where the rows of $A_{i,m}^{*\alpha}$ and $A_{j,m}^{*\alpha}$ have odd weight, hence the matrix $[A_{i,m}^{*\alpha} A_{j,m}^{*\alpha}]$ has rows of even weight.

Let v be a linear combination of vectors from Z_α, E_α, and O_α, then the resulting generator matrix $M^{*\alpha}$ will have rows of even weight. Hence, $Z_\alpha \cup E_\alpha \cup O_\alpha \subset \mathbb{F}_2^m$ is a basis for the choices of v for which all codewords of $\tilde{C}_{v,m}^{*\alpha}$ have even weight. Also, since $|Z_\alpha|$ is equal to the number of matrices with weight zero, $|E_\alpha|$ is equal to the number of matrices with rows of non-zero even weight, and $|O_\alpha|$ is equal to the number of matrices with odd weight minus one, it is clear that $|Z_\alpha \cup E_\alpha \cup O_\alpha| = m - 1$. Hence, the possible set of choices of $v \in \mathbb{F}_2^m$, where all codewords in $\tilde{C}_{v,m}^{*\alpha}$ have even weight is a subspace of $\mathbb{F}_2^m$ with dimension $m - 1$. □

The proof of theorem 6.6 not only shows that for half of all possible $v \in \mathbb{F}_2^m$ the codewords of $\tilde{C}_{(v,m)}^{*\alpha}$ all have even weight, but also which v's.

Example 6.7

Here we give an example of how to find the possible choices of V such that all codewords in $C_{(V,m)}^{*2}$ have even weight. Let $m = 4$, and let $R(A_{(i,m)})$ be the weight of the rows in $A_{(i,m)}$, for $1 \leq i \leq m$, then

$$R(A_{(1,4)}) = \binom{4-1}{1-1} = 1, \qquad R(A_{(2,4)}) = \binom{4-1}{2-1} = 3,$$
$$R(A_{(3,4)}) = \binom{4-1}{3-1} = 3, \qquad R(A_{(4,4)}) = \binom{4-1}{4-1} = 1.$$

Let Z_1, E_1, and O_1 be as in the proof of theorem 6.6, then

$$Z_1 = \emptyset, \qquad E_1 = \emptyset, \quad \text{and} \quad O_1 = \{(1,1,0,0), (0,1,1,0), (0,0,1,1)\}.$$

Hence, the set of choices of V for which all codewords in $C_{(V,4)}$ have even weight is

$$V_1 = \Big\{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3,4\}\Big\}.$$

Note, that of the $2^4 = 16$ sets in total, we find 8 possible choices for which the codewords of $C_{(V,4)}$ will have even weight, which is equal to half of the sets, as shown in theorem 6.6.

We now do the same for $\tilde{C}^{*2}_{(V,4)}$.

$$R\left(A^{*2}_{(1,4)}\right) = \binom{4-2}{1-2} = 0, \qquad R\left(A^{*2}_{(2,4)}\right) = \binom{4-2}{2-2} = 1,$$
$$R\left(A^{*2}_{(3,4)}\right) = \binom{4-2}{3-2} = 2, \qquad R\left(A^{*2}_{(4,4)}\right) = \binom{4-2}{4-2} = 1.$$

Let Z_2, E_2, and O_2 be as in the proof of theorem 6.6, then

$$Z_2 = \{(1,0,0,0)\}, \qquad E_2 = \{(0,0,1,0)\}, \quad \text{and} \quad O_2 = \{(0,1,0,1)\}.$$

Hence, the set of choices of V for which all codewords in $\tilde{C}^{*2}_{(V,4)}$ have even weight is

$$V_2 = \Big\{\emptyset, \{1\}, \{3\}, \{1,3\}, \{2,4\}, \{1,2,4\}, \{2,3,4\}, \{1,2,3,4\}\Big\}.$$

It should be clear that the possible choices of sets V for which the codewords of both $C_{(V,m)}$ and $\tilde{C}^{*2}_{(V,m)}$ have even weight are equal to the sets in $V_1 \cap V_2$. Hence, $V_1 \cap V_2$ is the set of possible sets of V for which all codewords of $C^{*2}_{(V,m)}$ have even weight, i.e. the sets

$$V_1 \cap V_2 = \Big\{\emptyset, \{1,3\}, \{2,4\}, \{1,2,3,4\}\Big\}.$$

It should be clear that even though $\emptyset$ is a possible choice, the resulting code is empty, and hence is of no interest.

Let $V = \{1,3\}$, then a generator matrix for $C_{(V,4)}$ is

$$G = [A_{1,4}A_{3,4}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Hence, $LSSS(C_{(V,4)})$ has $(2,7)$-product reconstruction, i.e. it is multiplicative. $\triangle$

In the above example we applied lemma 6.6 twice in order to find the set of V's for which all codewords of $C^{*2}_{(V,4)}$ have even weight. In the following theorem we proof the number of possible sets of V for which all codewords of $C^{*\mu}_{(V,m)}$ have even weight.

Theorem 6.8 is closely related to theorem 6 in [5], however, the latter part of the proofs are severely different.

Theorem 6.8. *The set of possible $v \in \mathbb{F}_2^m$ such that all codewords of $C^{*\mu}_{(v,m)}$ have even weight is a subspace of $\mathbb{F}_2^m$ with dimension $m - \mu$, for $1 \leq \mu \leq m$.*

Proof. Let $V = \{i_1, \ldots, i_{|V|}\}$, then $M = \begin{bmatrix} A_{i_1,m} \ldots A_{i_{|V|},m} \end{bmatrix}$ is a generator matrix for the code $C_{(v,m)}$.

We note that if all codewords in $C^{*\mu}_{(v,m)}$ have even weight, then the codewords in the codes $\tilde{C}^{*\alpha}_{(v,m)}$ all have even weight, for $1 \leq \alpha \leq \mu$, where

$$\tilde{C}^{*\alpha}_{(v,m)} = \Big\{ c_1 * \cdots * c_\alpha : c_i \in C_{(v,m)}, \text{ and } c_i \neq c_j, \text{ for } 1 \leq i, j \leq \alpha \Big\}.$$

From lemma 6.5 we have that the weight of the rows of $A^{*\alpha}_{i,m}$ is $\binom{m-\alpha}{i-\alpha}$, for $1 \leq i \leq m$. Since, $M^{*\alpha}$ is the generator matrix for $\tilde{C}^{*\alpha}_{(v,m)}$, it follows that if $\sum_{i \in V} \binom{m-\alpha}{i-\alpha}$ is even, then all codewords of $\tilde{C}^{*\alpha}_{(v,m)}$ have even weight, for $1 \leq \alpha \leq \mu$.

From theorem 6.6 we know that the set of possible $v \in \mathbb{F}_2^m$ such that all codewords in $C_{(v,m)}$ have even weight is a subspace of $\mathbb{F}_2^m$ with dimension $m - 1$. Let this subspace be $\Lambda_1 \subset \mathbb{F}_2^m$, and fix an isomorphism $\Psi_1 : \mathbb{F}_q^{m-1} \to \Lambda_1$. Then let $\Lambda_2 \subset \Lambda_1 \subset \mathbb{F}_2^m$ be the subspace consisting of v's for which all codewords of both $C_{(v,m)}$ and $\tilde{C}^{*2}_{(v,m)}$ have even weight.

Since, $\Psi_1\left(\mathbb{F}_2^{m-1}\right) = \Lambda_1$, to find Λ_2 is equivalent to finding the set U of possible vectors $u \in \mathbb{F}_2^{m-1}$ for which all codewords in $C_{(u,m-1)}$ have even weight, i.e. $\Psi_1(U) = \Lambda_2$. Hence from theorem 6.6 it follows that the dimension of $\Lambda_2 = (m-1) - 1 = m - 2$. We can now fix an isomorphism $\Psi_2 : \mathbb{F}_q^{m-2} \to \Lambda_2$, and by induction we have that if Λ_μ is the set of possible $v \in \mathbb{F}_2^m$ for which every codeword in $\tilde{C}^{*\alpha}_{(v,m)}$, for all $1 \leq \alpha \leq \mu$, have even weight, then $\dim \Lambda_\mu = m - \mu$, and the theorem follows. □

From lemma 6.3 and theorem 6.8 we get the following corollary.

Corollary 6.9. *The number of choices of $v \in \mathbb{F}_2^m$ such that $C_{(v,m)}$ has (μ, n)-product reconstruction is $2^{m-\mu}$.*

6.1.1 Security of $C_{(V,m)}$

In order to compute the bounds of the privacy and reconstruction of $LSSS(C_{(V,m)})$, we need the parameters of the code. It is clear from the matrices $A_{(i,m)}$ that $k = \dim C_{(V,m)} = m$, and that $n = \sum_{i \in V} \binom{m}{i}$, for the minimum weight d we present the following theorem.

Theorem 6.10. *The minimum weight of the code $C_{(V,m)}$ is*

$$d = \min_{1 \le u \le m} \sum_{i \in V} \left(\sum_{u_1 \le j \le u_2} \binom{u}{2j-1} \binom{m-u}{i-2j+1} \right),$$

where $u_1 = \max\left(1, \frac{u+1+i-m}{2}\right)$ and $u_2 = \min\left(\frac{i+1}{2}, \frac{u+1}{2}\right)$. [5]

Proof. Firstly, we consider the weight of a codeword obtained by the addition of u rows in $A_{i,m}$, i.e. the linear combination of rows, for any $1 \le i \le m$. Let $v = r_1 + \cdots + r_u = \left(v_1, \ldots, v_{\binom{m}{i}}\right)$, where $r_1, \ldots, r_u$ are u rows of $A_{i,m}$, and let r_{hj} be the jth coordinate of row r_h, for $1 \le h \le u$ and $1 \le j \le \binom{m}{i}$.

If $v_j = 1$, for some $1 \le j \le \binom{m}{i}$, then $\{r_{1j}, \ldots, r_{uj}\}$ contains an odd number of 1s. There are $\binom{u}{1}\binom{m-1}{i-1}$ possible choices of j, where $\{r_{1j}, \ldots, r_{uj}\}$ contains a single 1 and the rest are 0, hence, $\binom{u}{1}\binom{m-1}{i-1}$ 1s will appear in the vector v. Similarly, if $\{r_{1j}, \ldots, r_{uj}\}$ contains three 1s and the rest are 0, then there are $\binom{u}{3}\binom{m-u}{i-3}$ possible choices of j. By induction we get that the weight of the sum of m rows of $A_{i,m}$ is

$$\sum_j \binom{u}{2j-1} \binom{m-u}{i-2j+1}.$$

Note, that the sum only works if $0 \le 2j-1 \le u$ and $0 \le i-2j+1 \le m-u$, i.e. $\max(1, u+i+1-m) \le 2j \le \min(i+1, u+1)$, and the bounds u_1 and u_2 follow.

Now that we have the weight of the sum of u rows of $A_{i,m}$ we expand this to the weight of the sum of u rows of $\left[A_{i_1,m} \ldots A_{i_{|V|},m}\right]$, where $V = \left\{i_1, \ldots, i_{|V|}\right\}$. The vector resulting from the summation of u rows in this case, is simply equal to vectors as above appended together, each from a matrix $A_{i,m}$, where $i \in V$. Hence, the weight of the sum of u rows of the matrix $\left[A_{i_1,m} \ldots A_{i_{|V|},m}\right]$, where $V = \left\{i_1, \ldots, i_{|V|}\right\}$, is

$$\sum_{i \in V} \left(\sum_{u_1 \le j \le u_2} \binom{u}{2j-1} \binom{m-u}{i-2j+1} \right).$$

Lastly, since the matrix $\left[A_{i_1,m} \ldots A_{i_{|V|},m}\right]$, where $V = \left\{i_1, \ldots, i_{|V|}\right\}$, is a generator matrix for $C_{(V,m)}$, the possible sums of u rows of the matrix are the codewords of $C_{(V,m)}$ for $0 \le u \le m$, where the sum of 0 rows is the zero codeword. Hence, the minimum weight of the code $C_{(V,m)}$ is

$$\min_{1 \le u \le m} \sum_{i \in V} \left(\sum_{u_1 \le j \le u_2} \binom{u}{2j-1} \binom{m-u}{i-2j+1} \right),$$

where $u_1 = \max\left(1, \frac{u+1+i-m}{2}\right)$ and $u_2 = \min\left(\frac{i+1}{2}, \frac{u+1}{2}\right)$. □

We now give an example of MPC with multiplication using $LSSS(C_{(V,m)})$.

Example 6.11

Let $C_{(V,m)}$ be the $[n+\ell, k]$ code over $\mathbb{F}_2$, where $n = 7$, $\ell = 1$, and $k = 4$, constructed in example 6.7, i.e. $V = \{1, 3\}$ and $m = 4$.

In order to map the secret to the first coordinate, the first column of the generator matrix used in $LSSS(C_{(V,m)})$ has to be equal to $\boldsymbol{e}_1^k$. Here we use the matrix

$$G = [A_{1,4}A_{3,4}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Using $LSSS(C_{(V,m)})$ we want to solve the MPC $f = s_1 * s_2$. Since f is only dependent on s_1 and s_2, we only generate shares for s_1 and s_2. The secrets and shares for P_1 and P_2 are shown in table 6.1.

P_i	s_i	$x_i \in \mathbb{F}_2^3$	$(s_1, x_i)G$	Shares, c_i
P_1	1	$(0,1,1)$	$(1,0,1,1,0,0,1,0)$	$(0,1,1,0,0,1,0)$
P_2	0	$(0,1,0)$	$(0,0,1,0,1,0,1,1)$	$(0,1,0,1,0,1,1)$

Table 6.1: Secrets and shares for participants P_1 and P_2.

Each participant receives a share from both P_1 and P_2, then computes their product.

P_i	Share of s_1	Share of s_2	Product of shares, $\tilde{c}_i$
P_1	0	0	0
P_2	1	1	1
P_3	1	0	0
P_4	0	1	0
P_5	0	0	0
P_6	1	1	1
P_7	0	1	0

Table 6.2: The shares of s_1 and s_2 as distributed among the participants and the computed product of the received shares.

As shown in the proof of theorem 2.12, we know that

$$\rho_{\mathcal{I}}\left(c * c'\right) = -\sum_{i=1}^{n} c_i c_i' \equiv (q-1)\sum_{i=1}^{n} c_i c_i',$$

hence the recombination matrix is $R = [q-1, \ldots, q-1]^T$. Therefore, each participant P_i generates shares for $(q-1)\tilde{c}_i$, where $\tilde{c}_i$ is the product of the participant's received shares, for $1 \leq i \leq 7$. Note, that since $q = 2$, the participants generate shares for the product of the received shares.

P_i	$\tilde{c}_i$	$x_i \in \mathbb{F}_2^3$	$(\tilde{c}_i, x_i)G$	Shares of $\tilde{c}_i$
P_1	0	$(0,1,1)$	$(0,0,1,1,1,1,0,0)$	$(0,1,1,1,1,0,0)$
P_2	1	$(1,0,0)$	$(1,1,0,0,0,0,1,1)$	$(1,0,0,0,0,1,1)$
P_3	0	$(0,1,0)$	$(0,0,1,0,1,0,1,1)$	$(0,1,0,1,0,1,1)$
P_4	0	$(0,1,1)$	$(0,0,1,1,1,1,0,0)$	$(0,1,1,1,1,0,0)$
P_5	0	$(1,0,1)$	$(0,1,0,1,1,0,1,0)$	$(1,0,1,1,0,1,0)$
P_6	1	$(1,0,1)$	$(1,1,0,1,0,1,0,0)$	$(1,0,1,0,1,0,0)$
P_7	0	$(1,0,0)$	$(0,1,0,0,1,1,0,1)$	$(1,0,0,1,1,0,1)$

Table 6.3: Shares generated by each participant for the secrets $\tilde{c}_i$.

The shares generated for each participant's computed product of shares is listed in table 6.3. Each participant receives a share from each of the other participants and sums the shares to get a share of $s_1 * s_2$. The received shares for each participant and their sum are listed in table 6.4.

P_i	Shares received	Sum of shares
P_1	$(0,1,0,0,1,1,1)$	$4 \equiv 0$
P_2	$(1,0,1,1,0,0,0)$	$3 \equiv 1$
P_3	$(1,0,0,1,1,1,0)$	$4 \equiv 0$
P_4	$(1,0,1,1,1,0,1)$	$5 \equiv 1$
P_5	$(1,0,0,1,0,1,1)$	$4 \equiv 0$
P_6	$(0,1,1,0,1,0,0)$	$3 \equiv 1$
P_7	$(0,1,1,0,0,0,1)$	$3 \equiv 1$

Table 6.4: The shares received by each participant for the secrets $\tilde{c}_i$, and their sums.

Lastly, we reconstruct the secret from the share vector $(0,1,0,1,0,1,1)$, by solving the linear equation system $(s,x)G = (s,0,1,0,1,0,1,1)$, where $s = s_1 * s_2$. From the equation system we get

$$\begin{aligned} x_1 &= 0, & x_2 &= 1, \\ x_3 &= 0, & s + x_1 + x_2 &= 1, \\ s + x_1 + x_3 &= 0, & s + x_2 + x_3 &= 1, \\ x_1 + x_2 + x_3 &= 1. \end{aligned}$$

Solving the equations we get $s = s_1 * s_2 = 0$. $\triangle$

By lemma 6.3 the code $C_{(V,m)}$ used in example 6.7 and 6.11 is self dual, hence $d = d^{\perp}$. In order to compute the privacy and reconstruction bounds of the scheme, we first compute the minimum weight d.

$$\begin{aligned} d &= \min_{1\leq u\leq m} \sum_{i\in V} \left(\sum_{\max(1,\frac{u+1+i-m}{2})\leq j\leq \min(\frac{i+1}{2},\frac{u+1}{2})} \binom{u}{2j-1}\binom{m-u}{i-2j+1} \right) \\ &= \min_{1\leq u\leq 4} \sum_{i\in\{1,3\}} \left(\sum_{\max(1,\frac{u+1+i-4}{2})\leq j\leq \min(\frac{i+1}{2},\frac{u+1}{2})} \binom{u}{2j-1}\binom{4-u}{i-2j+1} \right) \\ &= \min\left(\binom{3}{0}+\binom{3}{2} = 4, \binom{2}{1}+\binom{2}{1} = 4, \binom{3}{1}+\binom{3}{3} = 4, \binom{4}{1}+\binom{4}{3} = 8 \right) = 4. \end{aligned}$$

Hence, the bounds of the privacy and reconstruction of $LSSS(C_{(V,m)})$, as given by theorem 1.7, are

$$\begin{aligned} t &\geq d^{\perp} - \ell - 1 = d^{\perp} = d = 4 \\ r &\leq n + \ell - d + 1 = 8 - 4 + 1 = 5. \end{aligned}$$

Since the security of the MPC follows from the security of the SSS used, optimal security requires a high t and a low r, as mentioned in section 1.3. In this case we see that the bounds are sharp, hence $t = 4$ and $r = 5$.

If we compare the security of $LSSS(C_{(V,m)})$ from example 6.11 and the Reed-Muller based SSS from example 4.5, since they have the same number of participants, 7, we can simply check the bounds of the of t and r. We see that in this case they are equal, and for both the bounds are sharp.

6.2 Example of Grouping Multiplications

In this section we give an example of protocol 4 using $LSSS(C_{(V,m)})$ with $(3, n)$-product reconstruction. However, we first give an example of the construction of $C_{(V,m)}$, where all codewords in $C^{*3}_{(V,m)}$ have even weight.

Example 6.12

In example 6.7 showed that the possible choices of V, such that $C_{(V,4)}$ has $(2, n)$-product reconstruction, are

$$V_1 \cap V_2 = \Big\{\emptyset, \{1,3\}, \{2,4\}, \{1,2,3,4\}\Big\}.$$

Hence, if V_3 is the set of possible V's where all codewords in $\tilde{C}^{*3}_{(V,4)}$ have even weight, then $(V_1 \cap V_2) \cap V_3$ is the set of possible V's for which $C_{(V,4)}$ has $(3, n)$-product reconstruction.

Let $R(A^{*\alpha}_{i,m})$ be the weight of rows in $A^{*\alpha}_{i,m}$, as in example 6.7, then

$$R\left(A^{*3}_{(1,4)}\right) = \binom{4-3}{1-3} = 0, \qquad R\left(A^{*3}_{(2,4)}\right) = \binom{4-3}{2-3} = 0,$$
$$R\left(A^{*3}_{(3,4)}\right) = \binom{4-3}{3-3} = 1, \qquad R\left(A^{*3}_{(4,4)}\right) = \binom{4-3}{4-3} = 1.$$

Let Z_3, E_3, and O_3 be as in the proof of theorem 6.6, then

$$Z_3 = \{(1,0,0,0),(0,1,0,0)\}, \qquad E_3 = \emptyset, \quad \text{and} \quad O_3 = \{(0,0,1,1)\}.$$

Hence, $V_3 = \Big\{\emptyset, \{1\}, \{2\}, \{1,2\}, \{3,4\}, \{1,3,4\}, \{2,3,4\}, \{1,2,3,4\}\Big\}$, and

$$(V_1 \cap V_2) \cap V_3 = \Big\{\emptyset, \{1,2,3,4\}\Big\}.$$

It follows that for $C_{(V,4)}$ to have $(3,n)$-product reconstruction $V = \{1,2,3,4\}$, and thus a generator matrix for $C_{(V,4)}$ is

$$G = [A_{1,4}A_{2,4}A_{3,4}A_{4,4}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

△

In the following example we will use the $C_{(V,m)}$ code constructed in example 6.12 to solve the function $f = s_1 * s_2 * s_3$ using protocol 4.

Example 6.13

Let $C_{(V,m)}$ be the $[n+\ell, k]$ code over $\mathbb{F}_2$, with $V = \{1,2,3,4\}$, $m = 4$, and $\ell = 1$, hence $k = 4$, $n = 14$, and $G = [A_{1,4}A_{2,4}A_{3,4}A_{4,4}]$ is a generator matrix for $C_{(V,m)}$. For the $LSSS(C_{(V,m)})$ the generator matrix used to generate shares must map the secret to the first coordinate, hence we choose to use G.

Using $LSSS(C_{(V,m)})$ we want to solve the MPC $f = s_1 * s_2 * s_3$. Since f is only dependent on s_1, s_2, and s_3, we only generate shares for these secrets. The secrets and shares for P_1, P_2, and P_3 are shown in table 6.5.

P_i	S_i	$x_i \in \mathbb{F}_2^3$	$(s_i, x_i)G$	Shares, c_i
P_1	1	$(1,1,0)$	$(1,1,1,0,0,0,1,0,1,1,1,0,0,0,1)$	$(1,1,0,0,0,1,0,1,1,1,0,0,0,1)$
P_2	1	$(0,1,0)$	$(1,0,1,0,1,0,1,1,0,1,0,1,0,1,0)$	$(0,1,0,1,0,1,1,0,1,0,1,0,1,0)$
P_3	1	$(1,0,1)$	$(1,1,0,1,0,1,0,1,0,1,0,1,0,0,1)$	$(1,0,1,0,1,0,1,0,1,0,1,0,0,1)$

Table 6.5: Secrets and shares for participants P_1, P_2, and P_3.

Each participant receives a share from P_1, P_2, and P_3, then computes their product. The shares received by participant P_i of each secret and the product of these shares are shown in table 6.6.

P_i	Share of s_1	Share of s_2	Share of s_3	Product of shares, $\tilde{c}_i$
P_1	1	0	1	0
P_2	1	1	0	0
P_3	0	0	1	0
P_4	0	1	0	0
P_5	0	0	1	0
P_6	1	1	0	0
P_7	0	1	1	0
P_8	1	0	0	0
P_9	1	1	1	1
P_{10}	1	0	0	0
P_{11}	0	1	1	0
P_{12}	0	0	0	0
P_{13}	0	1	0	0
P_{14}	1	0	1	0

Table 6.6: The shares of s_1, s_2, and s_3 as distributed among the participants.

From the proof of theorem 6.2 we have that $R_j^{\mathcal{I},3} \equiv 1 \mod 2$, for all $1 \leq j \leq 14$. Therefore, each participant P_i generates shares for $\tilde{c}_i$, which is the product of the participant's received shares, for $1 \leq i \leq 14$. The shares generated for each participant's computed product of shares is listed in table 6.7.

P_i	$\tilde{c}_i$	$x_i \in \mathbb{F}_2^3$	$(\tilde{c}_i, x_i)G$	Shares of $\tilde{c}_i$
P_1	0	$(1,1,1)$	$(0,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$	$(1,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$
P_2	0	$(0,1,1)$	$(0,0,1,1,0,1,1,1,1,0,1,1,0,0,0)$	$(0,1,1,0,1,1,1,1,1,0,1,1,0,0,0)$
P_3	0	$(1,1,1)$	$(0,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$	$(1,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$
P_4	0	$(0,0,1)$	$(0,0,0,1,0,0,1,0,1,1,0,1,1,1,1)$	$(0,0,1,0,0,1,0,1,1,0,1,1,1,1)$
P_5	0	$(1,1,1)$	$(0,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$	$(1,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$
P_6	0	$(1,0,0)$	$(0,1,0,0,1,0,0,1,1,0,1,1,0,1,1)$	$(1,0,0,1,0,0,1,1,0,1,1,0,1,1)$
P_7	0	$(0,1,1)$	$(0,0,1,1,0,1,1,1,1,0,1,1,0,0,0)$	$(0,1,1,0,1,1,1,1,1,0,1,1,0,0,0)$
P_8	0	$(1,0,0)$	$(0,1,0,0,1,0,0,1,1,0,1,1,0,1,1)$	$(1,0,0,1,0,0,1,1,0,1,1,0,1,1)$
P_9	1	$(0,1,0)$	$(1,0,1,0,1,0,1,1,0,1,0,1,0,1,0)$	$(0,1,0,1,0,1,1,0,1,0,1,0,1,0)$
P_{10}	0	$(0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
P_{11}	0	$(1,1,1)$	$(0,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$	$(1,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$
P_{12}	0	$(1,1,0)$	$(0,1,1,0,1,1,0,0,1,1,0,1,1,0,0)$	$(1,1,0,1,1,0,0,1,1,0,1,1,0,0)$
P_{13}	0	$(1,1,1)$	$(0,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$	$(1,1,1,1,1,1,1,0,0,0,0,0,0,1,1)$
P_{14}	0	$(1,0,0)$	$(0,1,0,0,1,0,0,1,1,0,1,1,0,1,1)$	$(1,0,0,1,0,0,1,1,0,1,1,0,1,1)$

Table 6.7: Shares generated by each participant for the secrets $\tilde{c}_i$.

Each participant receives a share from each participant and sums the shares to get a share of $s_1 * s_2 * s_3$. The received shares and their sum are listed in table 6.8.

P_i	Shares received	Sum of shares
P_1	$(1,0,1,0,1,1,0,1,0,0,1,1,1,1)$	$9 \equiv 1$
P_2	$(1,1,1,0,1,0,1,0,1,0,1,1,1,0)$	$9 \equiv 1$
P_3	$(1,1,1,1,1,0,1,0,0,0,1,0,1,0)$	$8 \equiv 0$
P_4	$(1,0,1,0,1,1,0,1,1,0,1,1,1,1)$	$10 \equiv 0$
P_5	$(1,1,1,0,1,0,1,0,0,0,1,1,1,0)$	$8 \equiv 0$
P_6	$(1,1,1,1,1,0,1,0,1,0,1,0,1,0)$	$9 \equiv 1$
P_7	$(0,1,0,0,0,1,1,1,1,0,0,0,0,1)$	$6 \equiv 0$
P_8	$(0,1,0,1,0,1,1,1,0,0,0,1,0,1)$	$7 \equiv 1$
P_9	$(0,0,0,1,0,0,0,0,1,0,0,1,0,0)$	$3 \equiv 1$
P_{10}	$(0,1,0,0,0,1,1,1,0,0,0,0,0,1)$	$5 \equiv 1$
P_{11}	$(0,1,0,1,0,1,1,1,1,0,0,1,0,1)$	$8 \equiv 0$
P_{12}	$(0,0,0,1,0,0,0,0,0,0,0,1,0,0)$	$2 \equiv 0$
P_{13}	$(1,0,1,1,1,1,0,1,1,0,1,0,1,1)$	$10 \equiv 0$
P_{14}	$(1,0,1,1,1,1,0,1,0,0,1,0,1,1)$	$9 \equiv 1$

Table 6.8: The shares received by each participant for the secrets $\tilde{c}_i$, and their sums.

Each participant now has a share of the secret $s_1 * s_2 * s_3$. Lastly, we reconstruct the secret from the share vector $(1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1)$, by solving the linear equation system $(s, x)G = (s, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1)$, where $s = s_1 * s_2 * s_3$. From the equation system we get

$$\begin{aligned}
x_1 &= 1, & x_2 &= 1,\\
x_3 &= 0, & s + x_1 &= 0,\\
s + x_2 &= 0, & s + x_3 &= 1,\\
x_1 + x_2 &= 0, & x_1 + x_3 &= 1,\\
x_2 + x_3 &= 1, & s + x_1 + x_2 &= 1,\\
s + x_1 + x_3 &= 0, & s + x_2 + x_3 &= 0,\\
x_1 + x_2 + x_3 &= 0, & s + x_1 + x_2 + x_3 &= 1.
\end{aligned}$$

Solving the equations we get $s = s_1 * s_2 * s_3 = 1$. △

7 Discussion

In this chapter we will examine the results and theory presented in the report, and look at the possible expansion for future work. We divide the chapter into three parts, protocols, secret sharing schemes, and future work.

7.1 Protocols

The project has presented four different protocols that uses linear SSSs to solve MPC problems. Each protocol starts with each participant generating shares for their private data and distributing the shares among the participants according to the SSS.

The actively secure MPC protocol for addition and scalar multiplication, protocol 1, simply applies the properties of a linear SSS. After having received the initial shares, the participants are each able to locally compute a share of the function's result from their received shares. However, the function has to consist solely of additions and scalar multiplications.

The passive and active security of protocol 1 follows directly from the linear SSS that is used. In this report we focus on linear SSS constructed with linear codes, hence, the reconstruction of the secret is equivalent to erasure and error correction, as shown in section 1.3.

The passively secure MPC protocol for any function, protocol 2, requires the SSS to be multiplicative. This is due to the fact that since we are working in finite fields, any function $f : \mathbb{F}_q^\ell \to \mathbb{F}_q^\ell$ is equivalent to a polynomial $p \in \mathbb{F}_q^\ell[x]$, where $\deg p < q^\ell$, hence only addition, scalar multiplication, and multiplication of secrets are necessary in order to solve any function in a finite field. And a multiplicative SSS allows for reconstruction of the coordinatewise product of secrets from the coordinatewise product of their shares, see definition 2.3 and definition 2.4.

Protocol 2 is not actively secure since each multiplication requires the participants to generate and distribute shares, and if one participant's shares are missing or incorrect, then the summation of the received shares will be incorrect for all participants. However, the protocol is still passively secure, which follows from the multiplicative SSS used.

For protocol 1 and protocol 2 it was initially assumed that all participants are honest during the initial distribution of shares, however, since this is not always the case we introduce a distribution methodology in section 5.1 that will fail if too many participants are accused as corrupt. To implement this for protocol 1 and protocol 2 we simply make the following observation.

For the initial distribution if only one participant's distribution fails, then any MPC involving that participant's private data would fail. For a participant to fail in a passively secure MPC, e.g. protocol 2, only one participant needs to accuse the participant, where for an actively secure MPC, using a SSS with t-privacy, more than t participants would have to accuse the participant for his or her initial distribution to fail.

By revising the multiplication step of protocol 2, and applying the improved distribution methodology from section 5.1, we construct an actively secure MPC protocol for any function in section 5.2, protocol 3, if $\ell = 1$, using a strongly multiplicative SSS, see definition 2.6. The active security of this protocol follows from the strongly multiplicative property, and the passive security follows from the SSS.

Lastly, in chapter 6 we show how to reduce the number of times the participants have to generate and distribute shares due to multiplications. In order to do this we first expand the definition of product reconstruction, see definition 6.1. From this property we revise protocol 2 to construct a passively secure MPC protocol for any function including grouping of up to μ multiplications, protocol 4, which uses a linear SSS with (μ, n)-product reconstruction. The security of protocol 4 follows as for protocol 2, with the passive security following from the (μ, n)-product reconstruction.

7.2 Secret Sharing Schemes

Throughout the project we solve MPC problems using SSSs, and every protocol we introduce throughout the project requires some form of SSS. Therefore, SSSs and their properties are central to the project.

Each protocol requires a different property of the SSSs. Protocol 1 only requires the SSS to be linear, protocol 2 requires that the SSS is multiplicative, protocol 3 requires that the SSS is strongly multiplicative, and lastly protocol 4 requires the SSS to have (μ, n)-product reconstruction.

In chapter 1 we introduce the basics of SSSs as well as two constructs of linear SSSs using linear codes. In section 1.1 we define the $LSSS(C)$ schemes, which are a ramp version of Massey's secret sharing scheme [11], see definition 1.5. And in section 1.2 we introduce $LSSS(\hat{C}, C)$, see definition 1.9, which hides the secret in a subcode, rather than coordinates of the codewords, as is done with $LSSS(C)$.

Therefore, $\hat{C} \subseteq \mathbb{F}_q^n$ in $LSSS(\hat{C}, C)$, hence all n coordinates of the codewords are used as shares, whereas in $LSSS(C)$ some of the codewords' length is reserved for hiding the secret, and hence cannot be used as shares. Furthermore, where $LSSS(C)$ has the restriction $\ell < d^{\perp}$, $LSSS(\hat{C}, C)$ only has the restriction $\ell < n$. And lastly, we proof in theorem 1.10 that any linear SSS is equivalent to some $LSSS(\hat{C}, C)$.

From section 1.1 and section 1.2 it would seem that $LSSS(\hat{C}, C)$ is the clear choice of SSSs to use for MPC. However, for the protocols where the SSS needs to be multiplicative, proving that $LSSS(\hat{C}, C)$ is multiplicative is difficult. Therefore, we study the codes C for which $LSSS(C)$ is multiplicative, strongly multiplicative, or has (μ, n)-product reconstruction.

In section 1.3 we introduce the concept of security of SSSs. Since the protocols in the project uses SSSs, most of the security described in section 1.3 applies to the protocols as well. We also show how to reconstruct secrets from shares for the two types of schemes $LSSS(C)$ and $LSSS(\hat{C}, C)$. Since both types of schemes are constructed by linear codes the methodologies for reconstruction implement the coding theoretical concepts of erasure and error correction.

Section 2.2 introduces multiplicative SSSs, and we proof a bound for the privacy of both multiplicative and strongly multiplicative SSSs, see theorem 2.5 and theorem 2.7. In section 2.3 we look at multiplicative $LSSS(C)$, and show that if C is self-orthogonal and $\ell = 1$, then $LSSS(C)$ is multiplicative, see theorem 2.12.

Chapter 3 and chapter 4 both consider different families of codes, Reed-Solomon and Reed-Muller codes respectively, and show how to find a code C, where $LSSS(C)$ is multiplicative. In chapter 3 the $LSSS(C)$ is shown to be multiplicative if C is an $[n + \ell, k]$ Reed-Solomon code, where $k \leq \frac{n+1}{2}$, see theorem 3.1, and in section 3.1 we show how to construct the SSS's recombination matrices.

In chapter 4 we show the dual code of a Reed-Muller code, see theorem 4.3, and by applying theorem 2.12 we found that $LSSS(C)$ is multiplicative, where C is an (ζ, m) Reed-Muller code over $\mathbb{F}_2$ if $2\zeta = m - 1$ and $\ell = 1$, see corollary 4.4.

Lastly, in section 6.1 we looked at constructing codes such that $LSSS(C)$ has (μ, n)-product reconstruction. The codes $C_{(V,m)}$ are constructed similarly to Reed-Muller codes, but rather than evaluating at all points only evaluates at points with Hamming weight i, where $i \in V$. By choosing V such that all codewords in $C_{(V,m)}^{*\mu}$ have even weight, as shown

in the proof of theorem 6.6, we have that $LSSS(C_{(V,m)})$ has (μ, n)-product reconstruction by lemma 6.3.

7.3 Future Work

This section will serve as a list of possible branches of work that can expand or improve on the results of the project.

Better bounds of privacy and reconstruction

The study of generalized Hamming weight (GHW) and relative generalized Hamming weight (RGHW) can be shown to find the exact values of the privacy and reconstruction of $LSSS(C)$ and $LSSS(\hat{C}, C)$, respectively, as shown in [8]. However, it can be difficult to find the exact values of GHW and RGHW, hence in the case of RGHW [8] employs the Feng-Rao bounds to estimate the RGHW.

Other families of codes

In this project we only show multiplicative $LSSS(C)$, where C is a Reed-Solomon code, a Reed-Muller code, or a code constructed from a Reed-Muller code. A clear way of expanding the results of the project would be to proof when $LSSS(C)$ is multiplicative, strongly multiplicative, or has (μ, n)-product reconstruction for other families of codes. E.g. toric codes, which was done in [9]. Similarly, the project could be further expanded by the study of when $LSSS(\hat{C}, C)$ is multiplicative, strongly multiplicative, or has (μ, n)-product reconstruction.

Improve protocols

In chapter 6 we showed how to group together up to μ multiplications in a MPC protocol, using a SSS with (μ, n)-product reconstruction, in order to reduce the distributions of secrets during multiplications. Though neither of our protocols clarifies this, the grouping of any number of additions and scalar multiplications is possible if the SSS is linear.

Protocol 4 is only passively secure, and no actively secure version of the protocol is presented in the project, hence it would be useful to attempt to construct an actively secure version of protocol 4, possibly as a revised version of protocol 3.

Bibliography

[1] Maria Bras-Amorós and Michael E. O'Sullivan. Duality of several families of evaluation codes. 2013. pages 41

[2] Ignacio Cascudo. Secret sharing schemes swith algebraic properties and applications. *Pursuit of the Universal: 12th Conference on Computability in Europe*, pages 68–77, 2016. pages 23, 26

[3] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert de Haan, and Vinod Vaikuntanathan. Secure computation from random error correcting codes. *Eurocrypt*, 2007. pages 4, 7, 8

[4] Ronald Cramer, Ivan Bjerre Damgaard, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. 2015. pages 23, 47, 49

[5] Iwan Duursma and Jiashun Shen. Multiplicative secret sharing schemes from reed-muller type codes. *IEEE International Symposium on Information Theory Proceedings*, 2012. pages 57, 61, 62

[6] Olav Geil. Evaluation codes from an affine variety code perspective. pages 40

[7] Olav Geil, Stefano Martin, Umberto Marínez-Peñas, Ryutaroh Matsumoto, and Diego Ruano. On asymptotically good ramp secret sharing schemes. 2016. pages 8

[8] Olav Geil, Stefano Martin, Ryutaroh Matsumoto, Diego Ruano, and Yuan Lou. Relative generalized hamming weights of one-point algebraic geommetric codes. *IEEE Transactions on Information Theory*, October 2014. pages 10, 74

[9] Johan P. Hansen. Secret sharing schemes with strong multiplication and a large number of players from toric varieties. 2014. pages 74

[10] Tadao Kasami, Shu Lin, and W. Wesley Peterson. New generalizations of the reed-muller codes - part i: Primitive codes. *IEEE Transactions on Information Theory*, 1968. pages 40

[11] J.L. Massey. Some apllications of coding theory in cryptography. *Codes and Ciphers: Cryptography and Coding IV*, 1995. pages 3, 73