CS406 Project Report
An Efficient Secret Sharing Tool

Kartik Gokhale & Ananya Rao

Spring 2024

Contents

1 Introduction

Secret sharing is a cryptographic technique employed to safeguard sensitive information by dispersing it among a group of participants in such a way that only a predefined subset of these participants, which we refer to as the qualified set, possesses sufficient information to reconstruct the original secret. Secret sharing is usually employed to ensure confidentiality, integrity, and availability of critical data in scenarios where distributing trust or authority is essential. In applications such as distributing nuclear launch codes, securing access to sensitive systems, or ensuring fault tolerance in distributed systems, secret sharing offers a versatile solution to mitigate risks associated with single points of failure or unauthorized access while allowing various qualified sets with the necessary power to obtain the sensitive data. By breaking down the secret into multiple shares, each seemingly random and unintelligible on its own, secret-sharing schemes provide a robust mechanism for collaborative access to sensitive information while preventing any single entity, or a collaboration of entities not containing any qualified set from gaining unauthorized knowledge or control over the secret.

In this project, we build a tool capable of sharing a secret message amongst various participants. We employ an efficient version based on the Shamir secret sharing and the Ramp secret sharing algorithms.

2 Preliminaries

2.1 Steps of Secret Sharing

Let us first review the steps involved in a secret-sharing protocol.

- **Generation of Shares:** The process begins with a trusted party, often referred to as the dealer, who possesses the original secret. Using some algorithm, the dealer generates shares by dividing the secret into multiple components. These shares are distributed to the participants, usually one share per participant.
- **Distribution of Shares:** Each participant receives their share, which appears to be a random value. Importantly, none of the participants individually possess enough information to reconstruct the secret.
- **Reconstruction:** To retrieve the original secret, a minimum threshold of shares from the qualified set is required. This threshold is predetermined during the setup phase. When the requisite number of shares is combined, the original secret can be reconstructed using mathematical operations.

A secret-sharing scheme focuses on the generation and reconstruction steps

2.2 Notation

We use the following notation for secret sharing. The following notation suffices for secret sharing schemes which are anonymous i.e. qualified sets do not depend on the identity of the individual. Intuitively, this prevents asymmetric qualified sets.

- N: the number of shares that each secret is split into
- R: the minimum number of shares needed to reconstruct the secret
- T: the maximum number of shares that may be seen without learning anything about the secret, also known as the privacy threshold
- K: the number of secrets shared together

where, logically, we must have $R \leq N$ since otherwise reconstruction is never possible, and we must have $T < R$ since otherwise privacy makes little sense. But, after all, what even is privacy?

2.3 Privacy in Secret Sharing

Let's denote the secret as M, the share received as C, and the probability distributions as $P(M)$ and $P(M|C)$, representing the prior probability distribution of the secret and the posterior probability distribution of the

secret given the share, respectively.

Perfect privacy can be mathematically expressed as:

$$P(M|C) = P(M) \tag{1}$$

This equation indicates that the probability distribution of the secret given the share ($P(M|C)$) is equal to the probability distribution of the secret ($P(M)$), implying that observing the share provides no additional information about the secret.

In other words, perfect privacy ensures that the share does not reveal any information about the secret beyond what was already known from the prior probability distribution.

There is a small caveat, however. Here, C can comprise the views of not just one individual but any number of individuals up to the privacy threshold T.

2.4 Shamir Secret Sharing

We introduce the first secret sharing protocol, which is Shamir Secret Sharing.

2.4.1 Generation

1. **Choose Parameters:**
   - Threshold (T): The minimum number of shares required to reconstruct the secret.
   - Total Shares (N): The total number of shares generated.
   - Field Size: Shamir's Secret Sharing operates over a finite field, often denoted as $\mathbb{F}_p$, where p is a prime number greater than the largest possible value of the secret.
2. **Generate Polynomial:** Choose a random polynomial of degree $T-1$, where the constant term represents the secret:
   $$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{T-1}x^{T-1}$$
3. **Assign Shares:** For each participant, evaluate the polynomial at a distinct point x_i in the field $\mathbb{F}_p$:
   $$f(x_i) = y_i$$
   The pair (x_i, y_i) constitutes a share, which is distributed to the corresponding participant.

2.4.2 Reconstruction

1. **Gather Shares:** Collect a minimum of T shares.
2. **Interpolate Polynomial:** Use Lagrange interpolation to construct the polynomial $f(x)$ passing through T of the provided points. The Lagrange interpolating polynomial, denoted as $L(x)$, is given by:
   $$L(x) = \sum_{i=1}^{n} y_i \cdot l_i(x) \tag{2}$$
   The Lagrange basis polynomials $l_i(x)$ are defined as:
   $$l_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{3}$$
   Note that all operations happen in the finite field.

3. **Retrieve Secret:** The constant term a_0 of the interpolated polynomial represents the secret:

$$\text{Secret} = f(0) = a_0 \tag{4}$$

2.5 Ramp Secret Sharing

While Shamir's scheme gets rid of the $R = N$ constraint and gives us flexibility in choosing T or R, it still has the limitation that $K = 1$. This means that each shareholder receives one share per secret, so a large number of secrets means a large number of shares for each shareholder. By using a generalized variant of Shamir's scheme known as packed or ramp sharing, we can remove this limitation and reduce the load on each individual shareholder.

To share a vector of K secrets $x = [x_1, x_2, ..., x_K]$, the shares are still computed as $f(1), f(2), \ldots, f(N)$ but the random polynomial is now sampled such that it satisfies $f(-1) = x_1$, $f(-2) = x_2$, $\ldots$, $f(-K) = x_K$.

Since it's less obvious how to sample such a polynomial in coefficient representation as we did before, to achieve the desired privacy threshold, we instead add T additional constraints $f(-K-1) = r_1$, $\ldots$, $f(-K-T) = r_T$ and simply use a point-value representation of the degree $T + K - 1$ polynomial.

This, however, means that we now have to perform interpolation instead of evaluation during sharing, which has an impact on efficiency. There's an additional caveat; since the degree of the polynomial increased, from T to $T + K - 1$, we also have to adjust either the privacy threshold or the number of shares needed to reconstruct.

To solve the efficiency problem, we use the Fast Fourier Transform (FFT)

2.6 Fast Fourier Transform

The Fast Fourier Transform (FFT) is a powerful algorithm used to efficiently compute the Discrete Fourier Transform (DFT) and its inverse. The Discrete Fourier Transform (DFT) is a mathematical operation that transforms a sequence of complex or real numbers into another sequence, representing the frequency components of the original data, defined as

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} nk} \tag{5}$$

While the DFT can be computed directly using the definition, it has a naive time complexity of $O(n^2)$. The Fast Fourier Transform (FFT) is an algorithmic technique that significantly reduces the computational complexity of computing the DFT to $O(nlogn)$. We review the Cooley-Tukey Algorithm here

2.6.1 Cooley-Tukey Algorithm

Input: A sequence of $N = 2^k$ complex numbers $x_0, x_1, ..., x_{N-1}$

Output: The DFT X_k of the input sequence

1. **Base case:** If $N = 1$, return the single point DFT $X_0 = x_0$.
2. **Split:** Split the input sequence into even-indexed and odd-indexed subsequences:

$$x_{\text{even}}[n] = x_{2n} \quad \text{and} \quad x_{\text{odd}}[n] = x_{2n+1}$$

3. **Recursion:** Compute the DFT of the even-indexed and odd-indexed subsequences recursively:

$$X_{\text{even}}[k] = \text{FFT}(x_{\text{even}}) \quad \text{and} \quad X_{\text{odd}}[k] = \text{FFT}(x_{\text{odd}})$$

4. **Combine:** Combine the DFTs of the even and odd subsequences to compute the final DFT:

$$X_k = X_{\text{even}}[k] + e^{-\frac{2\pi i}{N}k} \cdot X_{\text{odd}}[k]$$

for $k = 0, 1, ..., N/2 - 1$

$$X_{k+N/2} = X_{\text{even}}[k] - e^{-\frac{2\pi i}{N}k} \cdot X_{\text{odd}}[k]$$

for $k = 0, 1, ..., N/2 - 1$

3 Methodology

3.1 DFT and Secret Sharing

In this section, we describe how the Fast Fourier Transform allows us to perform efficient secret-sharing.

Assume then that we have a polynomial $A(x) = 1 + 2x + 3x^2 + 4x^3$ over a prime field with $L = 4$ coefficients and degree $L - 1 = 3$. $A_{coeffs} = [1, 2, 3, 4]$ Our goal is to turn this list of coefficients into a list of values $[A(w_0), A(w_1), A(w_2), A(w_3)]$ of equal length, for points $w = [w_0, w_1, w_2, w_3]$. The standard way of evaluating polynomials is, of course, one way of doing this, which using Horner's rule, can be done in a total of $O(L^*L)$ operations.

But, FFT allows us to do so more efficiently when the length is sufficiently large, and the points are chosen with a certain structure; asymptotically, we can compute the values in $O(L^*logL)$ operations, using the Cooley-Tukey Algorithm. Let us see how this affects the various secret-sharing schemes. The key idea is that we choose points $w = [w_0, w_1, w_2, w_3]$ carefully such that w_0 is a generator of order equalling the number of shares to be generated.

3.1.1 Shamir Secret Sharing

In this scheme, we can easily sample our polynomial directly in coefficient representation, and hence the FFT is only relevant in the second step, where we generate the shares. Concretely, we can directly sample the polynomial with the desired number of coefficients to match our privacy threshold, and add extra zeros to get a number of coefficients matching the number of shares we want; below the former list is denoted as small, and the latter as large. We then apply the forward FFT to turn this into a list of values that we take as the shares.

3.1.2 Ramp Secret Sharing

Recall that for this scheme, it is less obvious how we can sample our polynomial directly in coefficient representation, and hence, we do so in point-value representation instead. Specifically, we first use the backward FFT for powers of 2 to turn such a polynomial into coefficient representation, and then as above, use the forward FFT for powers of 3 on this to generate the shares.

We are hence dealing with two sets of points: those used during sampling, and those used during share generation – and these cannot overlap! If they did, the privacy guarantee would no longer be satisfied, and some shares might equal some of the secrets.

Preventing this from happening is the reason we use the two different bases 2 and 3: by picking co-prime bases, i.e. $gcd(2, 3) = 1$, the subgroups will only have the point 1 in common (as the two generators raised to the zeroth power). As such, we are safe if we simply make sure to exclude the value at point 1 from being used. Recalling our walk-through example, this is the reason we used prime Q = 433 since its order $Q - 1 = 432 = 4 * 9 * k$ is divided by both a power of 2 and a power of 3.

So to do sharing, we first sample the values of the polynomial, fixing the value at point 1 to be a constant (in this case, zero). Using the backward FFT we then turn this into a small list of coefficients, which we then as in Shamir's scheme, extend with zero coefficients to get a large list of coefficients suitable for running through the

forward FFT. Finally, since the first value obtained from this corresponds to point 1, and hence is the same as the constant used before, we remove it before returning the values as shares.

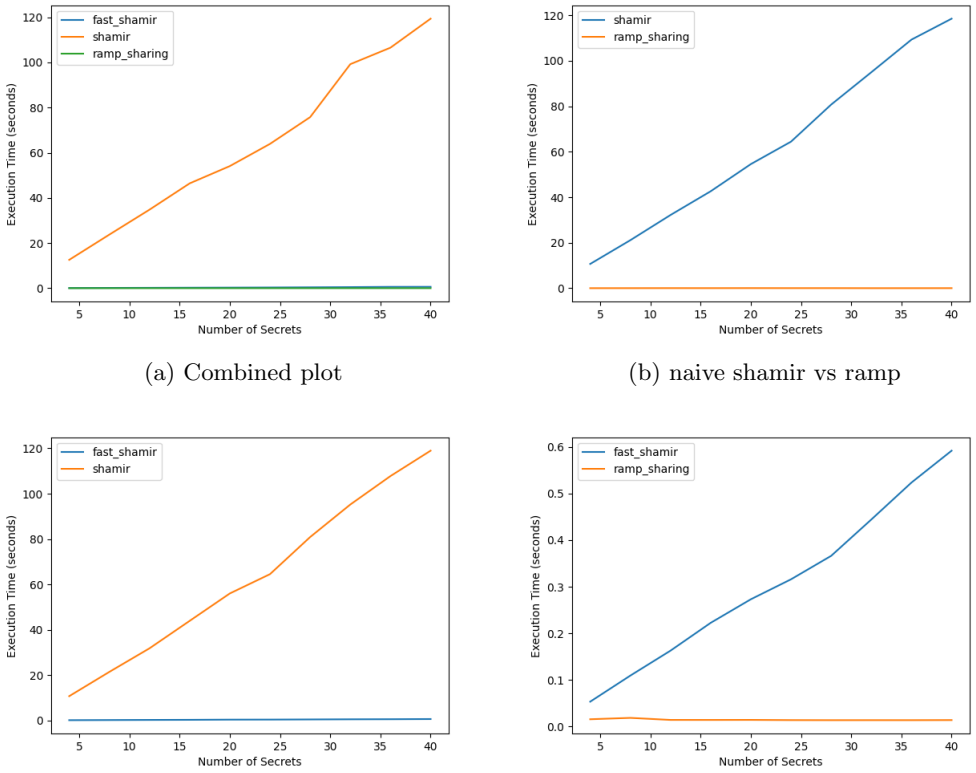
For this scheme, besides T, N, and the number K of secrets packed together, the parameters for this scheme are hence the prime Q and the two generators `OMEGA_SMALL` and `OMEGA_LARGE` of order respectively `ORDER_SMALL` $= T + K + 1$ and `ORDER_LARGE` $= N + 1$.

3.2 Generating Parameters

We borrowed the code from [1] for the `generate_parameters` function, which takes a desired minimum field size in bits, as well as the number of secrets k which to pack together, the privacy threshold T we want, and the number n of shares to generate. Accounting for the value at point 1 that we are throwing away (see earlier), to be suitable for the two FFTs, we obtain the parameters such that $k + t + 1$ is a power of 2 and that $n + 1$ is a power of 3. This constraint is relevant when we wish to cast a general secret-sharing problem in this form.

3.3 Experimental Results

We perform experimental analysis comparing the traditional Shamir secret sharing, efficient Shamir secret sharing which uses FFT for the evaluation but shares only one secret at a time and ramp secret sharing, which uses both FFT and incorporates multiple secrets. In all cases, for simplicity of evaluation, we stick to $R = N$, which means that the number of shares required to reconstruct the secret equals the number of servers. Please refer to Figure 1 for the results of our experimental analysis.



(a) Combined plot

(b) naive shamir vs ramp

(c) efficient shamir vs naive shamir

(d) efficient shamir vs ramp

Figure 1: Experimental Analysis

4 Secret Sharing Tool

4.1 Tool Overview

With the necessary improved algorithms in place, we seek to construct a tool based on the same. Given a secret message and the number of participants, we break the message into constituent characters. Each character is mapped to a unique element in the field. We use our custom padding of dummy characters at the end of the message to pad the message. Similarly, we appropriately pad the number of participants so that the constraints (Section 3.2) are met. These constraints are such that $k + t + 1$ is a power of 2 and that $n + 1$ is a power of 3.

An overview of the padding algorithm is as follows

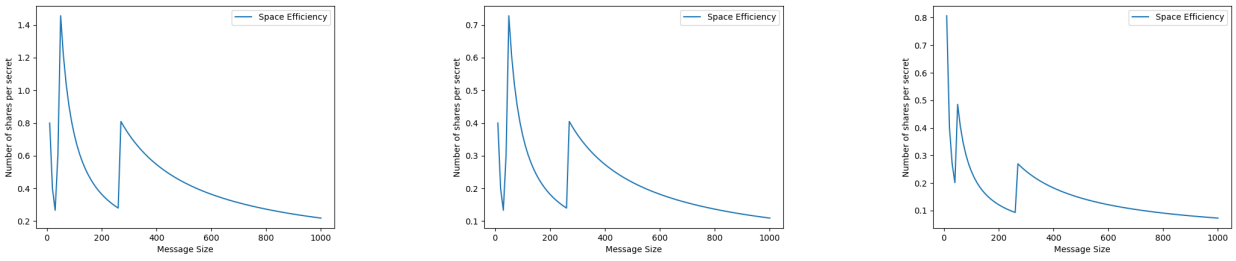
```
def padded_params(num_server, num_secret):
    num_serv = power_of_3_greater_than_x(num_server) - 1
    y = power_of_2_greater_than_x(num_serv + num_secret) - 1

    while y > 3 * num_serv:
        num_serv = 3 * num_serv + 2
        y = power_of_2_greater_than_x(num_serv + num_secret) - 1

    num_secr = y - num_serv
    return num_serv, num_secr
```

Algorithm 1: Padding the Parameters

Now, we generate shares using ramp sharing as described in Section 3.1.2. Distribution of shares is done using a Round-Robin approach where the first share goes to the first server, the second share to the second server and so on till it wraps around and the $k + 1st$ share goes to the first server. Further details related to share distribution can be found in Section 5. Reconstruction is done using the Ramp sharing algorithm. The tool guarantees perfect secrecy from the perfect secrecy guarantee of ramp sharing for any combination of shares less than T

4.2 Performance Evaluation

Time efficiency Analysis is performed in Section 3.3. For Space efficiency, we used the metric of shares/character received by each individual for various message sizes, and performed linear interpolation between data points. For comparison, Shamirs secret sharing has one share per secret for each individual. Refer to Figure 2 for the analysis.



(a) 10 Servers (b) 20 Servers (c) 30 servers

Figure 2: Space Efficiency Evaluation

As can be seen, the algorithm performs poorly for small messages and fewer servers as the padding is comparatively much larger than the message size. However, the effects become negligible as the message size increases.

Additionally, the peaks are observed when the padding size suddenly increases to go to the next power of 2 or 3.

4.3 Instructions to Use the Tool

The instructions to use the tool are on the GitHub repository along with all the required code. Credit to [1] for some of the key algorithms in the code that form the basis of the tool as described above.

5 Conclusion & Further Directions

The tool is able to divide a message of arbitrary length attaining better space and time efficiency, as compared to traditional Shamir secret sharing schemes. As seen above, we are able to perform evaluations faster due to the FFT algorithm as well as achieve an average of below one share per character, as would be attained by Shamir secret sharing.

The next steps would be to analyse various techniques to extend the tool to handle asymmetric qualified sets. The current tool only deals with symmetric qualified sets, thus secrecy is a function of the number of shares available ONLY. This needs to be extended to cases where the qualified sets can have different cardinality i.e. the sharing mechanism is no longer anonymous.

One such direction to achieve the above is to tweak the share distribution step, rather than the generation and reconstruction. The shares generated can be distributed asymmetrically, or one share can be given to multiple entities such that any qualified set ends up having all the necessary shares, but there is at least one share missing from any group not containing a qualified set.

References

[1] Dahl, M. (2017). Secret sharing blog.