Foundations in Secret Sharing

Sam de Alfaro and Bradley Cushing
Cryptography of Blockchains
New York University
Courant Institute of Mathematical Sciences
{sam.dealfaro,bsc6146}@nyu.edu

Abstract

This work presents a Systemization of Knowledge on foundational secret sharing schemes, providing a structured and comparative analysis of six fundamental schemes. We've selected secret sharing by Shamir, Feldman, Pedersen, Blakley, Asmuth-Bloom, and the related notion of function secret sharing (FSS) by Boyle, Gilboa, and Ishai. Implementations of the first five schemes can be found in our GitHub repository.[1]

1 Introduction

We start by introducing the notion of secret sharing and give a first insecure construction to better understand what properties a secure scheme should have. We then give a definition for k-out-of-n secret sharing and construct a basic example for $k = n$, which aims to give intuition and serve as a primer for the k-out-of-n schemes we cover next.

Section 2 covers the different preliminaries required to understand and build the various schemes. These include for reference, relevant definitions, constructions, and necessary facts. In some cases we give a well-known fact without proof, and in others we may sketch a proof in more detail.

In Section 3, we give an overview of six secret sharing schemes. For the first five schemes, denoted by their authors Shamir, Feldman, Pedersen, Blakley, and Asmuth-Bloom, we give a basic construction, sketch a general proof of correctness and security, and list any defining properties. We then cover the slightly different notion of function secret sharing (FSS) for which we give a similar analysis.

Section 4 compares the different properties in each secret sharing scheme. This includes but is not limited to the assumptions, verifiability, and trade-offs of each. Section 5 is an efficiency and cost comparison based on our implementations. This gives an in-depth performance analysis across implementations of all comparable secret sharing schemes.

[1] https://github.com/SamdeAlfaro/SharingSchemes

The last section, Section 6, concludes our systemization of knowledge and analysis.

1.1 Motivation

The notion of secret sharing can be loosely understood as the splitting of a secret s between some set of n parties such that s can be recovered by some number of these parties at a later time. Motivation for secret sharing comes from many places including some cases that are related to dealing with blockchains. One such case is where someone might want to backup a wallet seed phrase, split among a group of parties they trust in the hopes of recovering it later. Another case may be that as part of some protocol, we require a group to collectively sign messages with a shared secret key. The ability to sign using this secret key should be tied to the approval of some subset of the group.

Another very powerful application is worth mentioning: secret sharing schemes are of special interest when constructing generic, secure Multi-Party Computation (MPC) [4]. Secret sharing schemes with homomorphic properties can be used to build generic protocols like GMW and BGW. BGW states that in the semi-honest model, we can securely evaluate any function across n parties, if $t > n/2$, where t is the number of honest parties. We cover a separate but related notion called function secret sharing.

1.2 How NOT to Share a Secret

A naive method we could consider for an n-out-of-n scheme is simply breaking up the secret s into n equal shares. Considering s as a σ-bit string where $n|\sigma$, we have that share s_i is of length $|s_i| = \sigma/n$ and define s_i to be the i-th (σ/n)-bit string s.t. $s = s_1||\cdots||s_n$. If we have all n shares we can reconstruct s as a particular permutation of the n shares. But let's consider what happens when the party P_i wants to recover the s by themselves.

Since party P_i has σ/n bits, they only need $\sigma - \sigma/n$ bits more to recover the secret and if some larger set of parties collude, they have even more information about the secret. This particular attempt to construct an n-out-of-n secret sharing scheme fails in that each share leaks information about the secret s. The more shares that an attacker has, the more information they have about the secret. Another problem with this scheme is that shares might depend on one another. Put differently, seeing one share might allow us to infer what another share might be and what it might not be. Therefore, it's important for us that our n-out-of-n scheme has the property that any set of $n-1$ shares leaks no information about s.

1.3 Definition

A (k, n) threshold secret sharing scheme for a secret s, where n is the number of total shares, and k is the number of shares required to recover s, has the

following two properties:

1. The secret s can be recovered from any k or more shares, and
2. knowledge of any $k-1$ shares leaks no information about s.

We want the $Pr_{r \leftarrow R}[s_i = r|s] = 1/\lambda$ for any i where $|s_i| = \lambda$, and that for any combination of $k-1$ shares the $Pr_{r_1,...,r_{k-1} \leftarrow R_1,\cdots R_{k-1}}[s_1, ..., s_{k-1} = r_1, ..., r_{k-1}|s] = 1/\lambda^{k-1}$. We note that an n-out-of-n scheme is a special case of this scheme where $k = n$.

1.4 A simple (n, n) scheme

Consider the group $\mathbb{G} = \mathbb{Z}_n$ and a message $m \in \mathbb{G}$ where $n \gg m$. For all $1 \leq i \leq n-1$, we randomly sample $s_i \xleftarrow{R} \mathbb{G}$, and give s_i to party P_i. We then compute $s_n = m - \sum_{i=1}^{n-1} s_i$ and give s_n to party P_n. We can see that the protocol is correct since $\sum_{i=1}^{n} s_i = (\sum_{i=1}^{n-1} s_i) + s_n = \sum_{i=1}^{n-1} s_i + (m - \sum_{i=1}^{n-1} s_i) = m$.

Security. For security we want to show that all possible sets of $n-1$ shares leak no information about m. It's clear that all randomly chosen s_i values are independent of m so we can safely say those $n-1$ values leak nothing, but one of the values in our subset may also be $s_n = m - \sum_{i=1}^{n-1}$. We argue that given only a strict subset of shares from $[n]$, there is always at least one random $s_i \notin [n-1]$, and so it is unknown to the observer, and s_n computed from this value is indistinguishable from random. More concretely, denote this unknown random share s^*, then $s_n = m - \sum_{i \in [n-1]} s_i = m - (\sum_{i \in [n-2]} s_i) + s^*$ and so s_n appears uniformly random considering s^* is unknown.

2 Preliminaries

2.1 Polynomials

The schemes from Shamir, Feldman, and Pedersen rely on the fact that two degree $k-1$ polynomials which agree on k points are the same polynomial. We explain polynomial uniqueness and interpolation further below.

Uniqueness. Given k points of the form (x_i, y_i) with distinct x_i values, there is a unique degree $k-1$ polynomial q, s.t $q(x_i) = y_i$ for all i. In other words, any $k-1$ degree polynomial is uniquely determined by k points. This fact also applies when using modular arithmetic over a finite field $\mathbb{F}_p$ where p is prime. We note that Lagrange interpolation is possible in this field. We will use it for the reconstruction process in the polynomial-based schemes.

Lagrange interpolation. Consider the polynomial $f \in \mathbb{Z}_q[x]$ of degree at most $k-1$ s.t each party P_i is given share $f(i)$. We can reconstruct the polynomial f from any k points $(i, f(i))$ using Lagrange interpolation [6]. Define $f(x) =$

$\sum_{j=1}^{k}(\prod_{l\neq j}(x-i_j)/(i_l-i_j))\cdot f(i_j)$, then we can compute the secret $s = f(0) = \sum_{j=1}^{k}(\prod_{l\neq j} i_j/(i_j-i_l))\cdot f(i_j)$.

2.2 Discrete log

Consider $g, h \in \mathbb{G}$, where g is a generator, and where $g^r \equiv h$. The group $\mathbb{G}$ is said to have hard discrete log (DLOG) if given both g, h, finding r is hard. This is believed to be the case in groups like $\mathbb{Z}_p^*$ and groups of unknown order like $\mathbb{Z}_N$, where $N = p \cdot q$ and p, q are large primes. We rely on this assumption in the verifiability of the Feldman and Pedersen secret sharing schemes to show commitments are binding and/or hiding.

2.3 Commitments

Commitment schemes allow a party to commit to some value which they can later open or reveal to prove that some specific value was in fact the one that was committed to. Feldman's scheme uses homomorphic commitments to the coefficients of a secret sharing polynomial, enabling public verifiability of each party's share using only the public commitments.

The scheme by Pedersen introduces a commitment scheme for verifying shares but with another added benefit. Unlike Feldman's scheme, which uses *non-hiding* commitments (based on discrete logarithms) to support public verification, Pedersen's scheme uses *hiding* commitments that also preserve the secrecy of the committed values even in the presence of public verification data.

Pedersen commitment. A commitment scheme is defined as a pair of algorithms $(commit, verify)$ where $commit(m, r) \rightarrow c$, $verify(m, c, r) = accept$ or $reject$, m is a message, r is some secret randomness, and c is a commitment. Commitment schemes are binding and optionally hiding. Binding states that the same commitment cannot be opened to two distinct messages. Hiding states that the commitment leaks no information about the underlying message.

The commitment scheme used by Pedersen allows for commitment to a single value. Consider the generators $g, h \in \mathbb{G}_q$, where $\mathbb{G}_q$ is a cyclic group of prime order q with hard DLOG. The committer has a value $s \in \mathbb{Z}_q$, chooses a value $t \xleftarrow{R} \mathbb{Z}_q$, and computes the commitment $E(s, t) = g^s h^t$. To open the commitment at a later time, the committer simply reveals s and t. We state without proof that this commitment scheme is both binding and hiding.

2.4 Chinese remainder theorem

The Chinese remainder theorem (CRT) is used to solve a system of equations, where each equation is of the form $x \equiv n_i \ mod \ m_i$ for any d number of equations. If each pair m_i and m_j where $i \neq j$ are coprime, and $0 \leq n_i < m_i$ for each i, then there is a unique solution $x \ mod \ M = \prod_{i=1}^{d} m_i$. This fact is used to build

the Asmuth-Bloom scheme and the CRT is used to reconstruct the secret from a threshold k number of shares, each being $n_i \ mod \ m_i$ for $i \in [k]$.

2.5 Point functions

We give the definition for point functions as given directly by Boyle, Gilboa, and Ishai [3]. For $a \in \{0,1\}^n$ and $b \in \{0,1\}^m$, the point function $P_{a,b} : \{0,1\}^n \to \{0,1\}^m$ is defined by $P_{a,b}(a) = b$ and $P_{a,b}(a') = 0^m$ for all $a \neq a'$. This definition is extended in the notion of a distributed point function (DPF), which is the secure computation of a point function between p parties. The FSS schemes we cover consider function families containing functions of this type.

3 Secret sharing schemes

3.1 How to Share a Secret

Shamir introduced a (k, n) threshold secret sharing scheme based on polynomials in 1979 [7]. The problem is stated as one where some data D is divided into n pieces, where any k pieces can recover D, but importantly $k-1$ pieces contain no information about D. We cover this scheme first as it's relatively simple and is used to build both of the verifiable schemes we present later on.

Protocol. The general idea is that if we have some polynomial q of degree $k-1$ where $q(0) = s$, we can evaluate $q(x_i) = y_i$ for $1 \leq i \leq n$ to create n pieces, and give the party P_i the share y_i. Later on when any k of the parties get together, using polynomial interpolation, they can reconstruct this polynomial q from their k points and evaluate $q(0)$ to recover s.

To help us define the protocol in more detail we introduce the notion of a trusted "dealer". We note that this addition is not a necessary requirement, but it simplifies the explanation for our purposes. Given the secret s, the dealer generates a random polynomial $q(x) = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1}$ where $a_0 = s$ and $a_i \xleftarrow{R} \mathbb{F}_p$ for $1 < i \leq k-1$. The dealer then evaluates $q(i) = y_i$ for each of the n parties, where $1 \leq i \leq n$, and securely sends share y_i to party P_i.

By the known fact for polynomials, including those over finite fields like $\mathbb{F}_p$, q is uniquely determined from any k points of the form (x_i, y_i), whose x_i values are distinct. We can construct this unique polynomial using Lagrange interpolation from any k shares where the i-th point is (i, y_i). Correctness of the scheme follows directly in that any k parties can reconstruct q and evaluate $q(0) = s$ to recover the secret s.

Security. We want that any $k-1$ points leak no information about the secret s. For any $k-1$ points and s, there is exactly one polynomial q' that can be constructed s.t. $q'(0) = s$ and $q'(x_i) = y_i$ for all $i \in [k-1]$ shares. Since these polynomials are all equally likely, the $k-1$ shares are said to leak nothing about the secret s. In other words, the probability of s conditioned on seeing any $k-1$

evaluations of q is the same as the probability of s by itself, and so any choice of $k-1$ points and s are independent.

3.2 Feldman Verifiable Secret Sharing

Feldman introduced a non-interactive verifiable secret sharing scheme in 1987 [5], which extends Shamir's secret sharing by enabling verification of shares. In secret sharing, verifiability allows each party to check whether the share they received is consistent with the secret being shared, without any further interaction with the dealer or knowledge of the secret.

Protocol. Let the dealer choose a cyclic group $\mathbb{G}$ of prime order q with generator g, and work over a finite field $\mathbb{F}_p$. The dealer follows Shamir's scheme to generate a random polynomial $q(x) = a_0 + a_1 x + ... + a_{k-1}x^{k-1}$, where $a_0 = s$ is the secret and the other coefficients are sampled uniformly at random from $\mathbb{F}_p$.

The dealer then commits to each coefficient by computing $E_j = g^{a_j} \in \mathbb{G}$ for $j = 0, ..., k-1$ and broadcasts all E_j values to the parties. To verify their share $s_i = q(i)$, each party P_i check the equation $g^{s_i} \stackrel{?}{=} \prod_{j=0}^{k-1} E_j^{i^j}$. This works, since $\prod_{j=0}^{k-1} E_j^{i^j} = \prod_{j=0}^{k-1} g^{a_j i^j} = g^{q(i)} = g^{s_i}$. As a result, a party is convinced that their share is consistent with the committed to polynomial. Reconstructing the secret s is done using Lagrange interpolation using any k shares as before.

Security. While Feldman's scheme builds on the information-theoretic Shamir scheme, its secret sharing is only computationally secure. An attacker that can find discrete logs can recover each coefficient a_j of the polynomial q using only the commitments and use that to compute q. Commitments therefore reveal no information about the secret s under the discrete log assumption. We note that verifiability is secure against a malicious dealer with unbounded computation as they cannot generate an invalid share that still passes verification.

3.3 Pedersen Verifiable Secret Sharing

Pedersen's 1991 scheme [6] improves upon Feldman's by achieving information-theoretic hiding of coefficients in addition to verifiability. That is, even the commitments to the coefficients reveal no information about the secret, unlike Feldman's scheme where security relies on an assumption and the commitments (e.g. g^{a_0}) can leak information if the same secret is reused. To achieve this, Pedersen introduces a second randomly chosen polynomial and uses dual commitments based on two generators.

Protocol. In addition to the secret polynomial q defined as before, the dealer also generates a random hiding polynomial $r(x) = b_0 + b_1 x + ... + b_{k-1}x^{k-1}$, with random coefficients in $\mathbb{F}_p$. The dealer chooses generators $g, h \in G$ such that the discrete log $\log_g(h)$ is unknown. Then, for each j, they compute the dual commitments $E_j = g^{a_j} h^{b_j}$ and broadcasts all E_j values. Each party P_i

receives their share as the pair $(s_i, t_i) = (q(i), r(i))$, and verifies it by checking $g^{s_i} h^{t_i} \stackrel{?}{=} \prod_{j=0}^{k-1} E_j^{i^j}$.

We can notice that $\prod_{j=0}^{k-1} E_j^{i^j} = g^{q(i)+d \cdot r(i)} = g^{q(i)} h^{r(i)} = g^{s_i} h^{t_i}$ iff the commitment is correct or the dealer can find $log_g(h)$ which is thought to be hard in this group. Reconstructing the secret s is done using Lagrange interpolation using any k shares as before.

Security. As stated above, security of secret sharing in Pedersen is information theoretic. Security of the commitment scheme used in share verification relies on the intractable problem of finding $log_g(h)$ in G, which is thought to be hard. Security of the commitment scheme states that $E(s, t)$ reveals no information about s and that the dealer can't find a commitment s.t. $E(s, t) = E(s', t')$ unless they can find $log_g(h)$. Concretely, given $E(s, t) = E(s', t')$, the dealer can compute $log_g(h) = (s - s')/(t - t') \; mod \; q$.

3.4 Geometric Secret Sharing

Blakley proposed a method of secret sharing in 1979 [2] based on solving systems of linear equations in n-dimensional space. While this scheme was one of the first, published the same year as Shamir's scheme, it has the drawback that keys grow in size relative to the threshold t. There are also simple versions of this scheme which only succeed with high probability, meaning that they can fail to recover the secret s on rare occasion.

Protocol. At a high-level, this scheme encodes the secret s as a single point on a hyperplane of $n - 1$ dimensions in n-dimensional space. Each share is then a set of constants for a single equation defining some hyperplane. Given n equations, we can solve the system of equations in which there is a unique solution, and recover the secret s. In three dimensions we can think of this as a point that lies at the intersection of all planes. Solving for the intersection of the planes is just solving a system of linear equations with 3 unknowns.

Consider the example where we want a threshold $t = 3$, where we have a 2-dimensional plane in 3-dimensional space, on which some secret s lies. We can set some point $x = s$, randomly sample $y, z \stackrel{R}{\leftarrow} \mathbb{F}_p$, and randomly sample coefficients $a, b \stackrel{R}{\leftarrow} \mathbb{F}_p$, to define the linear equation $ax + by + z \equiv c \; mod \; p$ for some prime p, which specifies a 2-dimensional plane that contains this point. Solving for c in this equation gives us the constants (a, b, c) which we consider to be a single share. We repeat this process with the same initial x, y, z values to create any number of additional shares.

To recover the secret s, we can solve the system of equations. Given that we have a system of equations in three unknowns, the solution here is uniquely determined by three equations, or rather three shares in our case of secret sharing. We do this by solving $A\vec{x} \equiv \vec{c} \; mod \; p$ where A is an invertible matrix containing coefficients from the three equations, $\vec{x}$ is the vector of unknowns we're solving for, and $\vec{c}$ is a vector of individual c values. We note that this matrix will be

invertible with high probability over the random choices of coefficients in $\mathbb{F}_p$, but there are other methods that exist for choosing coefficients which guarantee an invertible matrix at this step.

Security. We're encoding the secret s on a hyperplane of $n-1$ dimensions in the example above, this being exactly a 2-dimensional plane, which one share represents. A single equation can be satisfied for any choice of $x = s$, and so one share leaks no information about the secret s. Any two equations specifies a line, a 1-dimensional subspace, which can also be satisfied for any secret s. More generally, we can extrapolate the theorem from this example intuitively, that any $n-1$ shares leak no information about s in this scheme.

3.5 Modular Secret Sharing

Asmuth and Bloom [1] proposed another secret sharing scheme in 1983 based on some well-known properties in modular arithmetic. This scheme also has some notion of inherent verifiability, where a shadow (i.e. share) can be checked for consistency. We give a high-level description of this scheme and it's properties in order to understand how it works.

Protocol. In this scheme, each party P_i has a shadow $y_i \equiv y \ mod \ m_i$, where all m_i values are coprime to each other and the value y is a linear function of some shared secret x. Parameters are set s.t. given any r of these shadows, y can always be uniquely determined, which in turn is used to invert the linear function for x.

Consider a dealer who computes a set $\{p, m_1 < \cdots < m_n\}$ subject to the following conditions: 1) $gcd(m_i, m_j) = 1$ for $i \neq j$, 2) $gcd(p, m_i) = 1$ for all i, and 3) $M = \prod_{i=1}^{r} > p \cdot \prod_{i=1}^{r-1} m_{n-i+1}$, where n is the total number of shadows and r shadows can recover the key. We define the linear function $y = x + Ap$ under the assumption $0 \leq x < p$, where A is an arbitrarily chosen integer subject to the condition $0 \leq y < M$. We can then define each shadow as $y_i \equiv y \ mod \ m_i$ and distribute shadow y_i to party P_i for all $1 \leq i \leq n$.

Recovery of y can be done using the Chinese remainder theorem (CRT) by solving the system of equations $y \equiv y_i \ mod \ m_i$ for all i where $i \in [r]$ is any r shadows. This y is uniquely determined since $y \ mod \ N_1 = \prod_{i=1}^{r} m_i$, $N_1 \geq M$, and from before $0 \leq y < M$. Since $0 \leq y < M \leq N_1$ we are guaranteed the solution space includes the full range of possible values for y. After solving for y, since both p and A are publicly known, we can compute a linear function to retrieve the secret x.

Security. The scheme leaks no information about the secret x when only $r-1$ shadows are known. In this particular case we learn $y \ mod \ N_2 = \prod_{j=1}^{r-1} m_i$ but we have that $M/N_2 > p$ and $gcd(N_2, p) = 1$ due to the initial properties defined in the scheme. This means that any possible $y' \ mod \ N_2$ where $y' \leq M$, has more or less the same probability of being in any congruence class $mod \ p$. In

other words, the bound on choices that satisfy the equation $mod\ N_2$ is p, and so we learn nothing from $r-1$ shadows.

3.6 Function Secret Sharing

The notion of function secret sharing (FSS) allows shares of a function to be securely distributed and computed among any number of parties. Boyle, Gilboa, and Ishai [3] expanded on this idea by improving the computational complexity of existing 2-party schemes and by giving a first non-trivial construction of a p-party distributed point function (DPF) for $p \geq 3$. We introduce the definition of FSS in the multi-party setting, cover a trivial construction to build intuition, and then review the improved 2-party scheme from this work.

FSS. A p-party, T-secure function secret sharing (FSS) scheme with respect to share output decoder Dec and function class $\mathcal{F}$ is a pair of PPT algorithms $(Gen, Eval)$ with the following syntax:

- $Gen(1^\lambda, f)$: On the input security parameter 1^λ and function description $f \in \mathcal{F}$, the key generation algorithm outputs p keys $(k_1, ..., k_p)$.
- $Eval(i, k_i, x)$: On input party index i, key k_i, and input string x, the evaluation algorithm outputs y_i corresponding this this party's share of the evaluation of $f(x)$.

Correctness states that the decoding of all of the individual y_i share evaluations should always be equal to the output of $f(x)$. Formally, we have that the $Pr[(k_1, ..., k_p) \leftarrow Gen(1^\lambda, f) : Dec(..., Eval(i, k_i, x), ...) = f(x)] = 1$.

Security states that any strict subset of T adversaries should learn nothing about the function f. We model security formally using an indistinguishability game. In this game the adversary $\mathcal{A}$ gets to pick two functions $f_1, f_2 \in \mathcal{F}$. The challenger $\mathcal{C}$ computes a set of p keys for each corrupted party in $T \subset [p]$ using f_b where $b \xleftarrow{R} \{0,1\}$ is chosen uniformly at random. $\mathcal{A}$ is then given these keys and tries to guess which function they correspond to. We require the probability that $\mathcal{A}$ guesses correctly is $\leq 1/2 + negl(\lambda)$.

We note a few important assumptions that are made in this definition. A function family $\mathcal{F}$ is modeled as an infinite collection of strings f where we can easily know the input domain, output domain, and evaluation of each. To simplify things we consider that the function, input, and output domains are the same Abelian group $\mathbb{G}$ w.r.t. the $\oplus$ operation. We define Dec to be an additive output decoder, specifically the XOR of all individual evaluations in $\mathbb{G}$, the shared output domain.

Intuition. We can construct a trivial p-party FSS scheme for $f \in \mathcal{F}$ by secret sharing its truth table among any p parties. Consider some point function $f : \{0,1\}^n \to \{0,1\}^m$ that we want to share between 2 parties. Key generation starts by computing a string we call M of length $m \cdot 2^n$, where the i-the input value maps $x \mapsto f(x) = y$ for $\ell = x$, and 0^m for all other values $0 \leq \ell \neq$

$x \leq 2^n - 1$. We then generate a random string $k_1 \xleftarrow{R} \{0,1\}^{m \cdot 2^n}$, compute $k_2 \leftarrow k_1 \oplus M$, and distribute key k_i to party P_i. We can see that each party P_i can compute their evaluation share of $f(x)$ by simply outputting $k_i[\ell]$, the ℓ-th m-bit string in their key k_i.

Correctness follows from the fact that k_1 is a one-time pad, so the XOR of any m-bit strings at the same index ℓ yield $k_1[\ell] \oplus k_2[\ell] = M[\ell]$. Security follows from the one-time pad as well, since k_1 is sampled uniformly from $\{0,1\}^{m \cdot 2^n}$, k_2 also looks random to P_2. Any subset of $n-1$ parties, in this case only one party, learns nothing about the function f. We note that the above scheme can easily be extended to p parties where $p \geq 3$ using techniques similar to that shown in the n-out-of-n secret sharing scheme.

2-party point functions. We can make use of a pseudorandom generator (PRG) with seed length λ to construct a 2-party FSS scheme for point functions. The general idea is to construct key k_i for party P_i s.t. it defines a binary tree with some special properties. Each binary tree has depth n, where n is the bit length of the input x to the function f. Most importantly, these binary trees are the same on all paths except for the path defined by the input $a = a_1, ..., a_n$ given to $Gen(1^\lambda, a, b)$, where the values at each node along the path are chosen pseudorandomly and independently. We can build this tree in time $O(\lambda n)$ for PRG seeds of length λ, since we only need to focus on the "good" path.

$Gen(1^\lambda, a, b)$ computes the two trees corresponding to the keys k_1, k_2 s.t. when $Eval(0, k_0, x)$ and $Eval(1, k_1, x)$ are called, and they stray from the path set by a, the seeds to the PRG and by result the expanded string, will be exactly the same. Due to the previously stated properties of the key generation, once a string seed and computed string become the same at some step, they will remain the same through to the end of the computation. This also applies to strings that differ at the root since generation sets the initial seeds to be the same. We don't expand more on the details other than to say this is done internally by use of "correction" strings from each key for each level.

The scheme is correct since values that are the same XOR to 0^m on all inputs which stray from the path dictated by a. Values on the path will be different with high probability, which we can guarantee to be 1 with some slight modifications. This scheme is secure due to the fact that all information related to the function is encoded in the correction strings, which are masked by pseudorandom values in the other party's key. Therefore, each party P_i learns nothing about the function f, when only given their key k_i.

4 Properties

We now compare and contrast some of the different properties of the secret sharing schemes presented. We don't cover the FSS scheme in this section as it's a very different notion from traditional secret sharing. Our focus will be on shares size, threshold properties, security including any key assumptions or lack thereof, verifiability, and homomorphism. In the next section we focus on the

Property	**Shamir**	**Feldman**	**Pedersen**	**Asmuth-Bloom**	**Blakley**
Share size	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{Z}_{m_i}$	$\mathbb{F}_p^k$
Threshold	k-of-n	k-of-n	k-of-n	k-of-n	k-of-n
Share security	Info-theoretic	DLOG-based*	Info-theoretic	Info-theoretic	Info-theoretic
Verifiability	None	Yes	Yes (hiding DLOG commit-ments)**	Limited†	None
Homomorphism	Additive	Additive	Additive	None	None
PQ-safe	Yes	No*	Partially**	Yes	Yes

Table 1: Comparison of secret sharing schemes.
* Commitments in Feldman are binding only and relax security of secret sharing. Security of secret sharing is broken if the discrete log (DLOG) is broken.
**Pedersen commitments are hiding and so secret sharing is PQ-safe, but the verifiability itself relies on the discrete log (DLOG) assumption, which is not PQ-safe.
†Asmuth-Bloom has partial consistency checks, but lacks full verifiability.

practical efficiency and cost of these schemes.

Share size. The size of a share in the Shamir, Feldman, and Pedersen schemes is only a single field element in $\mathbb{F}_p$ where $p \gg s$, for the secret s. The size of a share in Asmuth-Bloom is also a single element, in this case $y \equiv y_i \ mod \ m_i$ is the i-th share, where $0 \leq y \leq M$, so y is guaranteed to be $\leq M$ but also $\leq m_i$ for the corresponding m_i. Blakley on the other hand has a share size of k field elements, where k is the threshold, and each field element is in $\mathbb{F}_p$.

Threshold. Shamir, Feldman, Pedersen, Asmuth-Bloom, and Blakley support k-out-of-n secret sharing, where k is the threshold, and n is the total number of parties participating in the scheme. Only the initial scheme given in the introduction is an n-out-of-n scheme, where the threshold $k = n$, and so all parties are required to recover the secret there.

Assumptions. Shamir secret sharing is information-theoretic, based on no assumptions or intractable problems. Feldman and Pedersen schemes, while the same as Shamir in terms or generating shares and secret reconstruction, have slightly different properties due to their verifiability. Commitments in Feldman are a binding function of each coefficient of the polynomial used to encode the secret s, specifically a generator g raised to each coefficient. The inverse of this function is the discrete log, therefore if the DLOG can be computed efficiently, then an attacker reconstruct the polynomial and break the scheme. We note

that verifiability in Feldman requires no assumptions and is secure against a cheating dealer with unbounded computation. Pedersen is similar except that it's commitments are not only binding but also hiding by introducing another polynomial for additional randomness. This makes it so that the security of the secret sharing in Pedersen remains information-theoretic like Shamir, but unlike Feldman, its verifiability can be broken if DLOG can be broken.

Both the Asmuth-Bloom and Blakley schemes have information-theoretic security as well. In Asmuth-Bloom, each value for the secret $s \leq p$ is equally likely, to an attacker with less than k shares. In Blakley, for an attacker with less than k shares, the set of equations can be satisfied for any values of $s \in \mathbb{F}_p$, and so an attacker can do no better than try every possible value. As these schemes aren't verifiable, no other information is distributed besides shares.

We note that a scheme, or separately the verifiability of a scheme, which is information-theoretic is also post-quantum (PQ) safe, meaning it is resistant to attack by quantum computers. Some problems are also thought to be intractable in the context of quantum computation, but DLOG is not one of those problems. Therefore, secret sharing in the Feldman scheme and the verifiability of Pedersen, are succeptible to efficient attack by quantum computers.

Verifiability. As mentioned or hinted at before, Shamir and Blakley shares are not verifiable but both Feldman and Pedersen shares are. Asmuth-Bloom shares are not verifiable in the traditional sense, but some consistency checking of shares is possible due to some properties of the scheme.

Homomorphism. Shamir, Feldman, and Pedersen also have the powerful property that they are additively homomorphic. This allows for secret shares to be added together or multiplied by a constant, to create a secret share for that addition or multiplication.

We briefly touched on generic, secure MPC protocols like BGW, which leverage this homomorphic property to securely compute any function. These protocols also extend the additive property to a multiplicative one, where shares can be multiplied by one another, and where degree reduction is used to reduce the threshold of these new, higher-degree polynomials. We note that there are also protocols which leverage the additive homomorphic property of commitments used to verify shares in both Feldman and Pedersen.

Both the Asmuth-Bloom and Blakley schemes don't have this homomorphism property and can't be used to build more generic protocols for secure computation in the same way.

5 Efficiency

To compare the efficiency of these various schemes, we implemented them in Python and benchmarked them against multiple metrics. The full code for these implementations, along with the full list of results can be found in our

GitHub repository.[2]

5.1 Methodology

We implemented five secret sharing schemes (attributed to Shamir, Pedersen, Feldman, Blakley, and Asmuth-Bloom) in Python. For schemes that shared similar functions (such as the Lagrange interpolation for reconstructing the secret in the Shamir, Pedersen, and Feldman schemes), we reused code across the different implementations to standardize our benchmarking as much as possible. We then ran a function to benchmark these schemes, which worked as follows:

```
secret_list <- generate 4 random bit
    sequences of lengths 32, 64, 128, 255

num_shares <- [10, 100]

thresholds_by_share = {
    10: [2, 5, 10],
    100: [5, 25, 50, 100]
}

for secret in secret_list:
    for share_count in num_shares:
        for threshold in thresholds_by_share[share]:
            - Time each phase of each scheme
            - Store the average of each phase
              over 1000 iterations
```

Each scheme is broken into the phases of split shares, verify shares (if applicable), and reconstruct secret given the correct amount of shares meeting the minimal threshold. In this way, we time each scheme in three parts (its generation time, verification time, and reconstruction time) along 3 axes: the size of the secret[3], the total number of shares, and the threshold needed to reconstruct the secret.

The choice to use 1000 iterations was to ensure a large enough sample so that our average would be meaningful, but also to maintain a reasonable runtime. For this same runtime reasoning, we were unable to include larger numbers of shares. Originally, we wanted to look at 1000 shares as well, but generating these shares for various thresholds proved to be too computationally intensive, particularly due to verifying the shares in the Pedersen and Feldman schemes.

[2] `https://github.com/SamdeAlfaro/SharingSchemes`

[3] We used 255 bits instead of 256 because for some implementations, we required the secret to be strictly smaller than our generated prime (for ease of implementation).

We then ran this loop and stored all the data in a CSV, so that we could use it for graphing without needing to benchmark again each time. In total, generating this CSV from the code took around 6 hours.

5.2 Results

We now present some results from this empirical analysis. Given the generated CSV, we created graphs to visualize the performance of each scheme based on different metrics, and the comparative performance of different schemes on the same metric. This yielded around 150 graphs, all of which can be found in the GitHub repository in the results folder.

Included in this write-up are some inline graphs showing the runtime for various schemes. Below are the central behaviours we observed:

1. As expected, the reconstruction time across all schemes is greater for larger thresholds, and the share generation time is greater for higher numbers of shares. This can be seen in Figure 1.
2. The Pedersen scheme is slightly less efficient than the Feldman scheme. This can be seen in all figures that don't use a log scale on the y-axis, including in Figure 3. We note three observations for these two schemes:
   (a) In our implementations, the runtimes for Feldman and Pedersen are consistently dominated by their verification step, as seen in Figure 2.
   (b) Pedersen's generation phase (or split phase) is consistently slower than Feldman's, as seen in Figure 3.
   (c) Pedersen's verification step is slower than Feldman's implementation, as seen in Figure 4.
3. Blakley's reconstruction time is consistently larger than other scheme's reconstruction times. This is seen in all figures, including Figure 5.
4. The secret size impacts the Asmuth-Bloom scheme's split time strongly, while other schemes are impacted much less, as can be seen in Figure 6 and Figure 7.
5. However, for varying thresholds, Asmuth-Bloom's split time remains very consistent, while other schemes tend to display a greater increase in split, verify, and reconstruction times. This can be seen in Figure 8.

We now discuss these phenomena and explain why they are occurring.

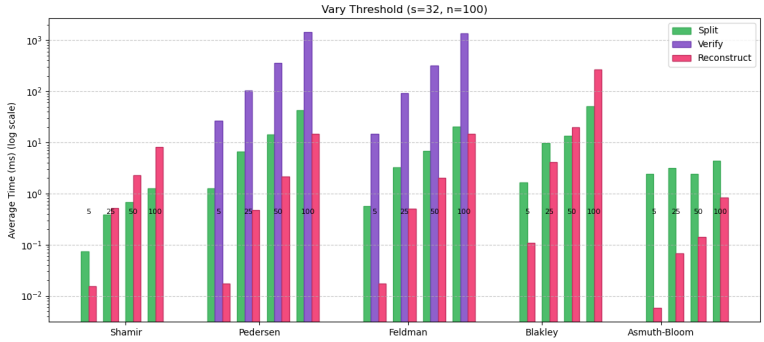

Figure 1: Average time (ms) (log scale) of each scheme's split, verification, and reconstruction phases for a secret of size 32 bits, 100 shares, and thresholds of 5, 25, 50, and 100 shares.

Observation 1: As expected, the reconstruction time across all schemes is greater for larger thresholds, and the share generation time is greater for higher numbers of shares. This can be seen in Figure 1.

This is our most straightforward observation. Reconstruction time is larger across all schemes for larger thresholds, because for each scheme, this means that more shares are used to reconstruct, thus taking more time. Likewise, the share generation (or split time) is greater when there are more shares, as we must generate a greater number of shares.

In Figure 1, note that for every scheme other than Asmuth-Bloom, the split time is also larger for larger thresholds. We discuss this phenomenon further in Observation 5.

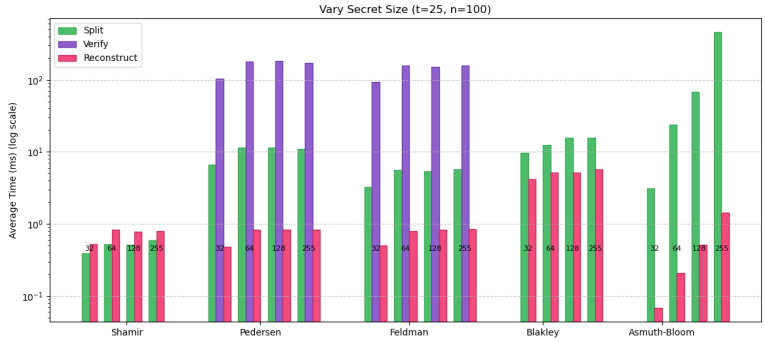

Figure 2: Average time (ms) (log scale) of each scheme's split, verification, and reconstruction phases for 100 shares and a threshold of 25, for secrets of size 32, 64, 128, and 255 bits.

Observation 2(a): In our implementations, the runtimes for Feldman and Pedersen are consistently dominated by their verification step, as seen in Figure 2.

It is important to note that the way the verification time is calculated in graphs with all five implementations is not fully accurate to how these schemes are deployed in practice. In fact, to compute the verification time, we ran the verification function on all distributed shares.

The main reason we see such a large spike in the verification steps is that they were timed sequentially. However, in real applications, shares are verified by the receiving parties. Therefore, the verification step happens once for each party, and can be modeled as parallel computations.

The reason we see a growth in cumulative verification time as the threshold grows, even when the number of shares remains the same (as in Figure 5), is because with higher thresholds, the computed polynomial in both schemes grows in complexity. So, verifying even a single share becomes a more complex computation. When all shares are being verified (as in the cumulative model), we therefore see a significant increase in total verification time, even though the number of shares being verified is not modified—only the threshold is.

To explore how the verifications perform when we model them as parallel computations, we ran some additional tests on verifying a single share, to see which parameters affect the runtime of these operations. All data discussed in observations 2(b) and 2(c), as well as in Figure 3 and 4, is calculated in the following way: we ran the verification on all shares 1000 times, then divided this total by 1000 and by the number of shares.

$$T[\text{one verification}] = \frac{T[\text{all verifications}] \cdot 1000}{1000 \cdot \text{num\_shares}}$$

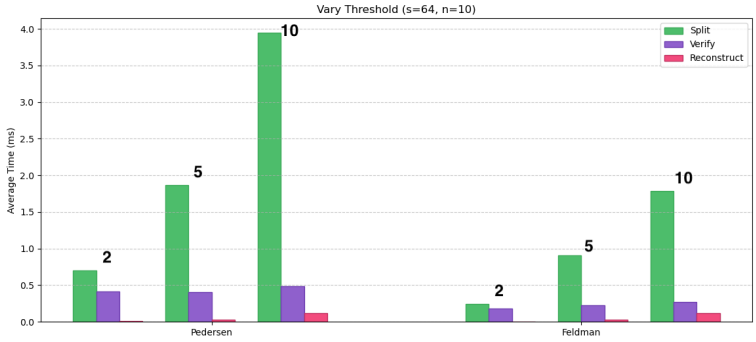

Figure 3: Average time (ms) of Pedersen and Feldman schemes' split, verification of a single share, and reconstruction phases for a secret of size 64 bits, 10 shares, and thresholds of 2, 5, and 10 shares.

Observation 2(b): Pedersen's generation phase (or split phase) is consistently slower than Feldman's, as seen in Figure 3.

We attribute this to the fact that Feldman's protocol requires one commitment per coefficient using only a single public generator g, while Pedersen's protocol adds an extra level of security by hiding the coefficients using two public generators g and h. Therefore, Feldman's scheme requires one exponentiation per coefficient, while Pedersen's requires two.

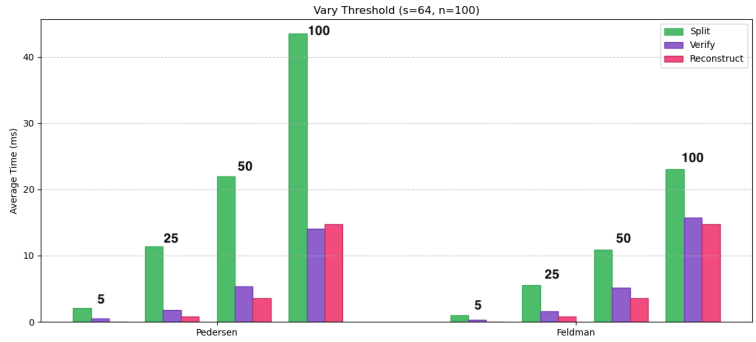

Figure 4: Average time (ms) of Pedersen and Feldman schemes' split, verification of a single share, and reconstruction phases for a secret of size 64 bits, 100 shares, and thresholds of 5, 25, 50, and 100 shares.

Observation 2(c): Pedersen's verification step is slower than Feldman's implementation, as seen in Figure 4.

First, note that when we only have 10 shares (Fig 3), variations in the threshold do not strongly affect the runtime of the verification step. However, when we have 100 shares (Fig 4), the increases in threshold have a more visible impact on verification time. We attribute this to the fact that with a greater number of shares, the increase in threshold is much larger, and thus the polynomials in both cases must be of much greater degree, thus causing a greater runtime.

Furthermore, the reason the Pedersen verification takes longer than the Feldman verification is due to the additional exponentiation per coefficient. Therefore, as the threshold grows, both schemes' verifications become more computationally expensive due to the fact that the polynomials involved are of higher degree. This is especially true for the Pedersen scheme, which has an extra layer of security in the commitment phase.

We note that the reconstruction phase grows with the threshold (in agreement with our Observation 1), but does not differ between Pedersen and Feldman's schemes. This is because both schemes run the exact same Lagrange interpolation on polynomials of the same degree, so it is expected that we would get the same average runtime.

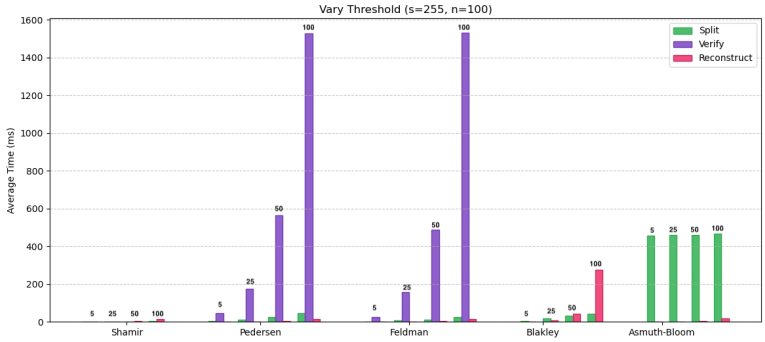

Figure 5: Average time (ms) of Shamir, Pedersen, Feldman, Blakley, and Asmuth-Bloom schemes' split, verification of all shares, and reconstruction phases for a secret of size 255 bits, 100 shares, and thresholds of 5, 25, 50, and 100 shares.

Observation 3: Blakley's reconstruction time is consistently larger than other scheme's reconstruction times. This is seen (in pink) in all figures, including Figure 5.

We attribute this to the nature of the linear algebra in Blakley's scheme. This scheme uses hyperplanes in vector spaces and reconstructs secrets through Gaussian elimination. This involves finding a pivot in column i, and computing the modular inverse of the pivot. Then, we normalise the pivot row and subtract a multiple of the pivot row from all rows below. Overall, this results in $O(k^3)$ time.

This process is significantly more computationally complex than the Lagrange interpolation in Shamir, Pedersen, and Feldman schemes which are $O(k^2)$. It is also quicker than using the Chinese remainder theorem ($O(k \log M)$, where M is the product of moduli).

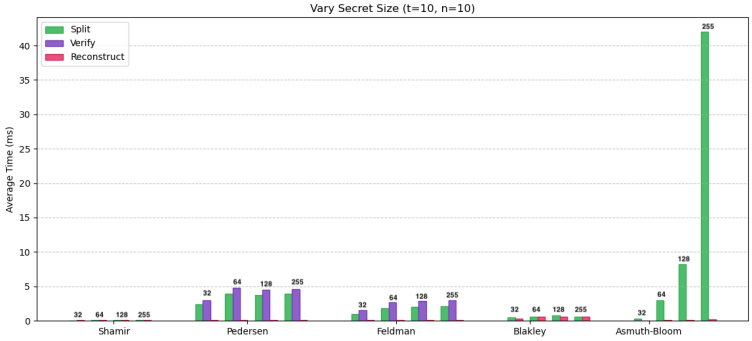

Figure 6: Average time (ms) of Shamir, Pedersen, Feldman, Blakley, and Asmuth-Bloom scheme's split, verification of all shares, and reconstruction phases for 10 shares and a threshold of 10, for secrets of sizes 32, 64, 128, and 255 bits.

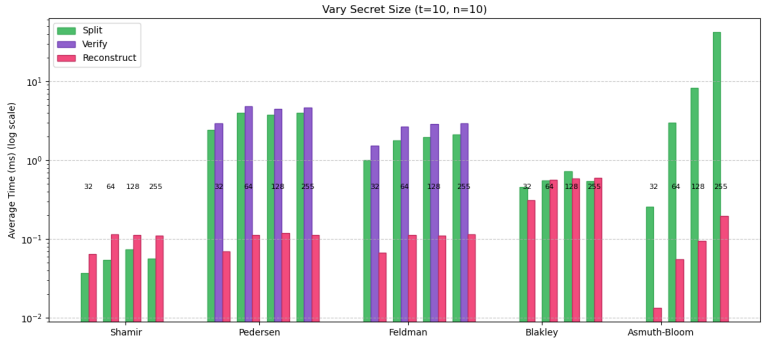

Figure 7: Average time (ms) (log scale) of each scheme's split, verification, and reconstruction phases for 10 shares and a threshold of 10, for secrets of sizes 32, 64, 128, and 255 bits.

Observation 4: The secret size impacts the Asmuth-Bloom scheme's split time strongly, while other schemes are not as impacted, as can be seen in Figure 7 and Figure 6.

We observe this because the Asmuth-Bloom scheme requires finding a prime $p >$ secret, then generating a sequence of primes $m_1, m_2, ..., m_n$ such that:

$$\prod_{i=1}^{k} m_i > p \cdot \prod_{i=n-k+2}^{n} m_i$$

We then compute $y = \text{secret} + r \cdot p$ for a random r such that $y < M = \prod_{i=1}^{k} m_i$. Shares are created by computing:

$$s_i = y \mod m_i$$

So when the secret size increases, then the initial prime becomes larger, and all other moduli in the sequence need to be much larger to satisfy the uniqueness condition $\prod_{i=1}^{k} m_i > p \cdot \prod_{i=n-k+2}^{n} m_i$.

This results in larger primes (which are slower to compute and work with), and more tries of different moduli if the condition isn't met right away. Therefore, the splitting time grows with the size of the secret.

For Shamir, Pedersen, and Feldman schemes, the secret must be less than the chosen prime, but the size of the secret doesn't impact how the polynomial is created. For Blakley, the secret is embedded as part of a vector, so the runtime also doesn't depend on secret size.

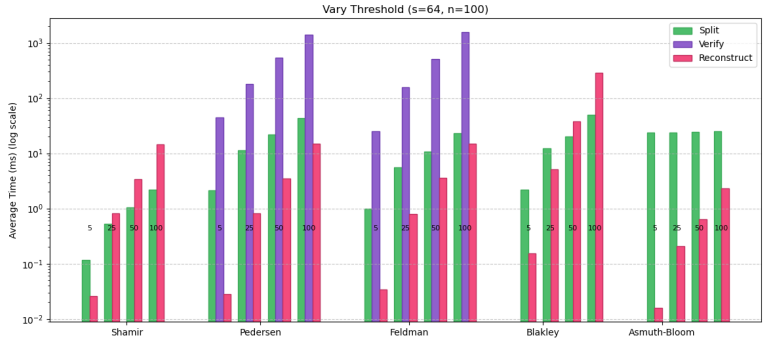

Figure 8: Average time (ms) (log scale) of each scheme's split, verification, and reconstruction phases for a secret of size 64 bits, 100 shares, and thresholds of 5, 25, 50, and 100 shares.

Observation 5: However, for varying thresholds, Asmuth-Bloom's split time remains very consistent, while other schemes tend to display a greater increase in split, verify, and reconstruction times. This can be seen in Figure 8.

The main reason for this is that the threshold k affects how many moduli are used to compute the product $M = \prod_{i=1}^{k} m_i$, but not the overall complexity of each share's computation.

During splitting, a sequence of n primes is generated, but only the first k primes are used to compute the total modulus M. The shares themselves are just $y_i = y \mod m_i$ for each $i \in [1, n]$.

Therefore, as k increases, we use more primes in the product for M, but the cost of computing this product is trivial compared to polynomial evaluation or Gaussian elimination. Furthermore, the share generation time is constant per share, regardless of how many moduli were used in computing M.

In other schemes, there is extra work in generating shares when k increases. For Shamir, Feldman, and Pedersen, splitting requires evaluating a degree $k-1$ polynomial at n points. For Blakley's scheme, each share is a hyperplane in $\mathbb{F}_p^k$. The time needed to generate the shares for these schemes increases as the threshold increases.

6 Conclusion

Secret sharing is a foundational primitive in cryptography, serving as a critical enabler for secure multiparty computation, threshold computation, and distributed trust systems. In this paper, we surveyed five classic secret sharing schemes, highlighting their security requirements and the algebraic structures that support them. We also covered the separate but related notion of function secret sharing, a powerful primitive for sharing functions.

Our discussion emphasizes that the design of a secret sharing scheme is not only a question of security but also one of structure and usability. By studying both the foundational theory and practical considerations of these schemes, we gain a clearer view of their capabilities and a roadmap for adapting them to the evolving needs of secure, distributed systems.

References

[1] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29(2):208–210, 1983.

[2] G. R. BLAKLEY. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318, 1979.

[3] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 337–367, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[4] David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*, pages 26–63. NOW Publishers, 2018.

[5] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

[6] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[7] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.