Foundations and properties of Shamir's secret sharing scheme
Research Seminar in Cryptography

Dan Bogdanov
University of Tartu, Institute of Computer Science

May 1st, 2007

1 Introduction

Secret sharing is a technique for protecting sensitive data, such as cryptographic keys. It is used to distribute a secret value to a number of parts—*shares*—that have to be combined together to access the original value. These shares can then be given to separate parties that protect them using standard means, e.g., memorize, store in a computer or in a safe. Secret sharing is is used in modern cryptography to lower the risks associated with compromised data.

Sharing a secret spreads the risk of compromising the value across several parties. Standard security assumptions of secret sharing schemes state that if an adversary gains access to any number of shares lower than some defined threshold, it gains no information of the secret value. The first secret sharing schemes were proposed by Shamir [Sha79] and Blakley [Bla79].

This work gives the standard definition of a (k, n) threshold secret sharing scheme and its properties. We continue by exploring polynomial evaluations as the mathematical background for Shamir's scheme. After describing Shamir's scheme we prove its security and present algorithms for performing operations with shares.

2 Concept of secret sharing

Definition 2.1. *Let the secret data be a value s. An algorithm* $\mathbf{S}$ *defines a k*-out-of-*n* threshold secret sharing scheme*, if it computes* $\mathbf{S}(s) = [s_1, \ldots, s_n]$ *and the following conditions hold [Sha79, Dam02]:*

1. ***Correctness:*** *s is uniquely determined by any k shares from $\{s_1, \ldots, s_n\}$ and there exists an algorithm $\mathbf{S}'$ that efficiently computes s from these k shares.*

2. ***Privacy:*** *having access to any $k-1$ shares from $\{s_1, \ldots, s_n\}$ gives no information about the value of s, i.e., the probability distribution of $k-1$ shares is independent of s.*

A secret sharing scheme is *homomorphic* if it is possible to compute new valid shares from existing shares.

Definition 2.2. *Let s and t be two values and $[s] = [s_1, \ldots, s_n]$ and $[t] = [t_1, \ldots, t_n]$ be their shares. A secret sharing scheme is $(\oplus, \otimes)$-*homomorphic *if shares $[(s_1 \otimes t), \ldots, (s_n \otimes t)]$ uniquely determine the value $s \oplus t$.*

If individual shares are from a uniform distribution it can be shown that secret sharing is secure in a multiparty computation setting. Indeed, the protocol is simple—one party sends values from a uniform distribution to other parties in the system. In the ideal world this means the trusted third party F outputs nothing. The simulator is easy to build—it just generates a value from a uniform distribution and passes it to the adversary. Again, the values are from the same distribution and the adversary cannot distinguish between them.

To illustrate the concept of secret sharing, we use the following classical example [Sha79]. Assume that there is a corporation where the management needs to digitally sign cheques. The president can sign cheques on his or her own, the vice presidents need at least another member of the board to give a signature and other board members need at least two other managers to sign a cheque.

We can solve this problem by sharing the secret key needed for giving a signature with a 3-out-of-n threshold secret sharing scheme, where n is the required number of shares. We give the company president three shares, so he or she can sign cheques alone. Vice presidents get two shares each, so that they need the agreement of another manager to give a signature. Other members of the board get one share per member, so that three of them are needed for a signature.

The signing device is completely secure as it does not contain any secret information. It requires the members of the board to provide three shares to retrieve the signature key. This key is used for a single signature and then forgotten so the next signature will again require three shares of the key. If a malicious adversary coerces one manager to sign a cheque, then it has to be the president of the corporation. Otherwise the adversary will have to persuade more than one member of the board.

2.1 Mathematical foundations of secret sharing

2.1.1 Polynomial evaluations

We start by describing some basic properties of polynomials. Let us consider a ring $\mathbf{R}$ and denote the set of all polynomials over $\mathbf{R}$ by $\mathbf{R}[x]$. Let $f(x) = f_0 + f_1 x + \cdots + f_k x^k$ be a polynomial in $\mathbf{R}[x]$. We also fix a vector $a = [a_0, \ldots, a_n] \in \mathbf{R}^n$ so that all values a_i are different and nonzero. Then we define the polynomial evaluation mapping $\mathbf{eval} : \mathbf{R}[x] \to \mathbf{R}^n$. as follows. We evaluate the polynomial $f(x)$ on the vector a and present the result as a vector.

$$\mathbf{eval}(f) := [f(a_0), \ldots, f(a_n)]^T \quad .$$

In the following theorem operations between vectors in $\mathbf{R}^n$ are defined elementwise. That is, if $u, v \in \mathbf{R}^n$ and $\oplus$ is a binary operator, then:

$$u \oplus v := [(u_1 \oplus v_1), \ldots, (u_n \oplus v_n)]^T \quad .$$

Theorem 2.1. *For any two polynomials f and g in $\mathbf{R}[x]$ and a scalar value $r \in \mathbf{R}$ the following conditions hold:*

(i) Additivity: $\mathbf{eval}(f + g) = \mathbf{eval}(f) + \mathbf{eval}(g)$.

(ii) Multiplicativity: $\mathbf{eval}(f \cdot g) = \mathbf{eval}(f) \cdot \mathbf{eval}(g)$.

(iii) Multiplicativity w.r.t. scalar values: $\mathbf{eval}(r \cdot f) = r \cdot \mathbf{eval}(f)$

The mapping **eval** *is a linear transformation.*

Proof. The conditions hold because of the duality with the respective polynomial operations:

(i) Additivity: $(f + g)(a) = f(a) + g(a)$

(ii) Multiplicativity: $(f \cdot g)(a) = f(a) \cdot g(a)$

(iii) Multiplicativity w.r.t. scalar values: $(r \cdot f)(a) = r \cdot f(a)$

The conclusion that the mapping is a linear transformation directly follows from the above conditions. Thus we have shown that **eval** is a linear mapping between the evaluation positions of the polynomial and the result vector. □

We will now give a further analysis of the properties of this mapping. Let $\vec{f} = [f_0, \ldots, f_k]$ be the array of coefficients of the polynomial f. Note that in further discussion we consider a polynomial f being equivalent to the vector of its coefficients.

We now compute the vector $\vec{y} = \mathbf{eval}(f) = [f(a_0), \dots, f(a_n)]^T$.

$$\begin{aligned}\vec{y} &= \sum_{i=0}^{k} f_i \mathbf{eval}(x^i) = \sum_{i=0}^{k} f_i \left[a_0^i, \dots, a_n^i\right]^T \\ &= f_0 \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + f_1 \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} + \dots + f_k \begin{bmatrix} a_0^k \\ a_1^k \\ \vdots \\ a_n^k \end{bmatrix} = \begin{bmatrix} f_0 + f_1 a_0 + \dots + f_k a_0^k \\ f_0 + f_1 a_1 + \dots + f_k a_1^k \\ \dots \\ f_0 + f_1 a_n + \dots + f_k a_n^k \end{bmatrix} .\end{aligned}$$

We notice that the vector $\vec{y}$ is actually the product of a matrix and another vector.

$$\begin{bmatrix} f_0 + f_1 a_0 + \dots + f_k a_0^k \\ f_0 + f_1 a_1 + \dots + f_k a_1^k \\ \dots \\ f_0 + f_1 a_n + \dots + f_k a_n^k \end{bmatrix} = \begin{bmatrix} 1 & a_0 & a_0^2 & \cdots & a_0^k \\ 1 & a_1 & a_1^2 & \cdots & a_1^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^k \end{bmatrix} \times \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_k \end{bmatrix} . \tag{1}$$

We denote the matrix by V and notice that the vector on the right side is the vector $\vec{f}$ of polynomial coefficients. This way we can rewrite equation (1) as follows:

$$\vec{y} = V\vec{f} \ .$$

This shows that the evaluation mapping between the coefficients $f_0, \dots, f_n$ and evaluations $f(a_0), \dots, f(a_n)$ of a polynomial is a linear tranformation determined by the matrix V.

2.1.2 Reconstructing the polynomial

If $k = n$ then the matrix V is a $(k+1) \times (k+1)$ square matrix. A matrix in this form is known as the Vandermonde matrix. It's determinant is equal to [Kil05, page 147]

$$\Delta(V) = \prod_{\substack{i,j \\ i>j}} (a_i - a_j) \ .$$

We need the evaluation mapping to be reversible and for this we need to show that the matrix V is invertible. A matrix is invertible, if it is regular that is, its determinant is invertible [Kil05, page 143]. We have defined the values of $a_0, \dots, a_n$ to be distinct so the differences $(a_i - a_j)$ in the given sum are nonzero. To achieve that the product of the differences is nonzero it is enough to make sure that the ring has no zero divisors. For that reason we require from now on that $\mathbf{R}$ is a field, since fields have no zero divisors. With this assumption we ensure that $\Delta(V)$ is invertible and therefore V is invertible, if all values a_i are distinct. This in turn confirms that the transformation provided by V is also invertible and we can express f by using the inverse of V.

$$\vec{f} = V^{-1}\vec{y}$$

We will now show, how to reconstruct the polynomial f from its evaluations. We define vectors $\vec{e_i}$ as unit vectors in the form $\left[\begin{array}{ccc} e_0 & \dots & e_n \end{array}\right]$.

$$\begin{aligned} \vec{e_0} &= \left[\begin{array}{cccc} 1 & 0 & \dots & 0 \end{array}\right] \\ \vec{e_1} &= \left[\begin{array}{cccc} 0 & 1 & \dots & 0 \end{array}\right] \\ \dots \\ \vec{e_n} &= \left[\begin{array}{cccc} 0 & 0 & \dots & 1 \end{array}\right] \end{aligned}$$

Let $\vec{b_i}$ be such that

$$\vec{e_i} = V\vec{b_i}. \tag{2}$$

Because of the properties of V we showed earlier we can rewrite equation (2) and express $\vec{b_i}$ as follows.

$$\vec{b_i} = V^{-1}\vec{e_i} \ .$$

Noting that

$$\vec{y} = \sum_{i=0}^{n} y_i\vec{e_i}$$

we see that we can reconstruct $\vec{f}$ from evaluations as follows

$$\vec{f} = \sum_{i=0}^{n} V^{-1}y_i\vec{e_i} = \sum_{i=0}^{n} y_i\vec{b_i} \ .$$

It follows that we can reconstruct the coefficients of the polynomial f, if we have access to its evaluations at $n+1$ positions and the vectors $\vec{b_i}$ and therefore we have constructively proved the well-known Lagrange interpolation theorem.

Theorem 2.2 (Lagrange interpolation theorem)**.** *Let* $\mathbf{R}$ *be a field and* $a_0, \dots, a_n$, $y_0, \dots, y_n \in \mathbf{R}$ *so that all values* a_i *are distinct. Then there exists only one polynomial* f *over* $\mathbf{R}$ *so that* $\deg f \leq n$ *and* $f(a_i) = y_i$, $(i = 0, \dots, n)$. □

The Lagrange interpolation polynomial can be computed as the sum

$$f(x) = y_0b_0(x) + \dots + y_nb_n(x)$$

where the base polynomials $b_i(x)$ are defined as

$$b_i(x) = \prod_{\substack{j=0 \\ i \neq j}}^{n} \frac{(x - a_j)}{(a_i - a_j)} \quad .$$

As one could expect, the Lagrange interpolation polynomial has a useful property—its base polynomials correspond to our vectors $\vec{b_i}$:

$$\mathbf{eval}(b_i) = \left[\begin{array}{ccc} b_i(a_0) & \dots & b_i(a_n) \end{array} \right]^T$$

Since

$$b_i(a_j) = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases}$$

we see that

$$\mathbf{eval}(b_i) = \vec{e_i} \quad .$$

We also have to handle the cases where $k \neq n$. We will reduce these cases to the $(k+1) \times (k+1)$ case observed before. First we consider the case where $n > k$. If we choose $k+1$ different values $l_0, \dots, l_k \in \{0, \dots, n\}$, then we obtain a virtual matrix V' by choosing rows $l_0, \dots, l_k$ from the original matrix V:

$$V' = \begin{bmatrix} 1 & a_{l_0} & a_{l_0}^2 & \cdots & a_{l_0}^k \\ 1 & a_{l_1} & a_{l_1}^2 & \cdots & a_{l_1}^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_{l_k} & a_{l_k}^2 & \cdots & a_{l_k}^k \end{bmatrix} \quad .$$

The square matrix V' is invertible as its determinant is nonzero, because it corresponds to the evaluation map at $[a_{l_0}, \dots, a_{l_k}]$ and by showing that we have reached the previously observed and proved case. In the third case when $n < k$ we generate $k - n$ values $a_{n+1}, \dots, a_k$ so that all values a_i are distinct. We use these new positions to add rows to the matrix V and get the virtual matrix V'

$$V' = \begin{bmatrix} 1 & a_0 & a_0^2 & \cdots & a_0^k \\ 1 & a_1 & a_1^2 & \cdots & a_1^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^k \\ 1 & a_{n+1} & a_{n+1}^2 & \cdots & a_{n+1}^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k & a_k^2 & \cdots & a_k^k \end{bmatrix} \quad .$$

This matrix V' is an invertible $(k+1) \times (k+1)$ square matrix that can replace V in the first observed case. Note that if $n < k$ the reconstruction of the polynomial is not unique and is determined by the choice of values $a_{n+1}, \dots, a_k$. This gives us a guarantee that it is not possible to uniquely reconstruct the polynomial if there are not enough pairs of positions and evaluations available. Later we will use this property to prove privacy of the following secret sharing scheme.

2.2 Shamir's secret sharing scheme

We now describe Shamir's secret sharing scheme that is based on polynomial evaluations [Sha79]. We start by explaining the infrastructure of secret sharing. The central party is the dealer that performs share computation operations on input secrets and distributes the resulting shares to other parties. When the secret has to be reconstructed, the parties give their shares to the dealer, that can then combine the shares and retrieve the secret.

In Shamir's scheme shares are evaluations of a randomly generated polynomial. The polynomial f is generated in such a way that the evaluation $f(0)$ reveals the secret value. If there are enough evaluations, the parties can reconstruct the polynomial and compute the secret. Algorithm 1 describes how shares are computated in Shamir's scheme.

Algorithm 1: Share computation algorithm for Shamir's scheme

```
**Data**: finite field $\mathbf{F}$, secret data $s \in \mathbf{F}$, threshold $k$, number of shares $n$
**Result**: shares $s_1, \dots, s_n$
Set $f_0 = s$
Uniformly generate coefficients $f_1, \dots, f_{k-1} \in \mathbf{F}$
Construct the polynomial $f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1}$
Evaluate the polynomial: $s_i = f(i)$, $(i = 1, \dots, n)$
```

Note that the indices of the shares start from one, as we cannot output $s_0 = f(0)$, because it is the secret value. The resulting shares $s_1, \dots, s_n$ can be distributed to their holders. If the original value needs to be retrieved, we need a subset of at least k shares. Note that it is important to store the index i together with the share s_i, because it is later needed for reconstruction.

The classical algorithm of Shamir's scheme reconstructs the whole polynomial, whereas we describe versions optimised for reconstructing only the secret $f(0) = s$. We only need to compute $f(0)$ so for our purposes we can simplify the base polynomials $b_i(x)$ as follows:

$$\beta_i = b_i(0) = \prod_{\substack{j=1 \\ i \neq j}}^{k} \frac{(-a_j)}{(a_i - a_j)}. \tag{3}$$

If the shares are computed using Shamir's scheme then algorithm 2 retrieves the secret value s.

```
Algorithm 2: Share reconstruction algorithm for Shamir's scheme
  Data: finite field $\mathbf{F}$, shares $s_{t_1}, \dots, s_{t_k} \in \mathbf{F}$ where $t_j \in \{1, \dots, n\}$ are
        distinct indices
  Result: secret data $s$
  compute the reconstruction coefficients $\beta_i$ according to equation (3)
  compute $f(0) = s_{t_1}\beta_{t_1} + \dots + s_{t_k}\beta_{t_k}$
  Return $s = f(0)$
```

Theorem 2.3. *Shamir's secret sharing scheme is correct and private in the sense of Definition 2.1.* □

The proof for this theorem is given in [Bog07]. The proof of privacy is based on the properties of the evaluation mapping to show that any number of shares less than k reveals nothing about the secret value s.

2.3 Secure computation with shares

We will now show what can be done with the shares once they have been distributed. We will investigate the possibility of using the homomorphic property of the secret sharing scheme to perform operations with the shares. In the following assume that a k-out-of-n threshold scheme is used. Assume that we have n parties $P_1, \dots, P_n$ and the dealer gives each one of them a share according to its index.
Addition. Assume that we have shared values $[u] = [u_1, \dots, u_n]$ and $[v] = [v_1, \dots, v_n]$. Because the evaluation mapping is a linear transformation, we can add the shares of $[u]$ and $[v]$ to create a shared value $[w]$ so that $u + v = w$. Each party k has to run the protocol given in Algorithm 3 to add two shared values.

```
Algorithm 3: Protocol for adding two Shamir shares for node $k$
  Data: shares $u_k$ and $v_k$
  Result: share $w_k$ that represents the sum of $[u]$ and $[v]$
  Round 1
    $w_k = u_k + v_k$
```

Multiplication with a scalar. Assume that we have a shared value $[u] = [u_1, \dots, u_n]$ and a public value t. Again, thanks to the linear transformation property of the evaluation mapping we can multiply the shares u_i with t so that the resulting shares represent the value $[w] = t[u]$. Algorithm 4 shows the protocol for multiplication a share value by a scalar.

Algorithm 4: Protocol for multiplying Shamir shares by a scalar value for node k

```
Data: shares $u_k$ and a public value $t$
Result: share $w_k$ that represents the value of $t[u]$
Round 1
    $w_k = tu_k$
```

Multiplication. Assume that we have shared values $[u] = [u_1, \ldots, u_n]$ and $[v] = [v_1, \ldots, v_n]$. Share multiplication, unfortunately, cannot be solved with the linear property of the transformation, as multiplying two polynomials with the same degree gives a polynomial with double the degree of the source polynomials. This means that we must use a k-out-of-n threshold scheme where $2k \leq n$ and the polynomials must have a degree of at most $2k$. By multiplying the respective shares, the miners actually compute a share that represents the polynomial storing the the product of the secrets. However, we must reconstruct the secret stored in the product polynomial and reshare it to make further multiplications possible. Otherwise, the multiplication of the product polynomial with another one will give us a polynomial with a degree larger than n and we cannot reconstruct the secret from such polynomials anymore.

We note that we can precompute the values of the optimised base polynomials β_i needed in the protocol by using equation (3) on page 7. This requires each node to know its number and also how many other nodes there are, but that is a reasonable assumption. Algorithm 5 gives the complete protocol for multiplying Shamir shares.

Algorithm 5: Protocol for multiplying two Shamir shares for node i

```
Data: shares $u_i$ and $v_i$, precomputed value $\beta_i$
Result: share $w_i$ that represents the value of $[u][v]$
Round 1
    $z_i = u_i v_i \beta_i$
    Share $z_i$ to $z_{i_1}, \ldots, z_{i_n}$ using the same scheme as the dealer uses
    Send to each other node $P_l, i \neq l$ the share $z_{i_l}$
Round 2
    Receive shares $z_{ji}, j \neq i$ from other nodes
    $w_i = z_{i_i} + \sum_{\substack{j=1 \\ j \neq i}}^{n} z_{ji}$
```

3 Conclusion

The Shamir's secret sharing scheme has a good abstract foundation which provides an excellent framework for proofs and applications. We presented algorithms for performing addition, standard and scalar multiplication with shares. We are currently developing a secure computation platform based on a simple secret sharing scheme than Shamir's. However, we have found, that the properties of our scheme are similar to the ones of Shamir's scheme.

References

[Bla79] George R Blakley. Safeguarding cryptographic keys. In *Proceedings of AFIPS 1979 National Computer Conference*, volume 48, pages 313–317, 1979.

[Bog07] Dan Bogdanov. How to securely perform computations on secret-shared data. Master's thesis, University of Tartu, 2007.

[Dam02] Ivan Damgård. Secret sharing. Course notes, 2002.

[Kil05] Mati Kilp. *Algebra I.* Estonian Mathematical Society, 2005.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.