RenVM Secure Multiparty Computation

Ross Pure, Zian-Loong Wang

Contents

1 Introduction

The purpose of this paper is to specify the secure multiparty computation (SMPC) that is used in RenVM to perform distributed ECDSA signatures, and prove its security. To achieve this, we start by describing the fundamental SMPC primitives of secret sharing, opening shared secrets, random number generation, and random keypair generation. Using these we can construct more complicated primitives such as multiplication and field inversion, and finally use all of these to construct a distributed ECDSA signature protocol. Security will defined using a composable framework. This will allow us to prove security for the primitives, and then compose them together in a secure way.

The rest of the paper is structured as follows. First, background material including probability theory, Shamir secret sharing and computational assumptions will be presented. Next, the security framework to be used is described. The rest of the paper is dedicated to the SMPC protocols and their security. Initially, the more primitive protocols such as opening shared secrets and random number generation are described. Next, higher level protocols like multiplication and field inversion are constructed using these primitives. Finally, we will be in a position to specify how distributed ECDSA signatures are computed.

2 Background

In this section we outline some background material that will be used throughout the paper. This includes basic probability theory definitions and theorems, Shamir Secret Sharing, cryptographic assumptions and also some notation.

2.1 Notation

Here we outline some common notation that will be used in this paper. The natural numbers are denoted by the set $\mathbb{N}$, where $0 \notin \mathbb{N}$; i.e. $\mathbb{N} = \{1, 2, \ldots\}$. We will use $\mathbb{F}$ to denote an arbitrary field (in this paper we will only be concerned with finite fields). When talking about field elements, it is understood that 0 and 1 represent respectively the additive and multiplicative identity of the field. An anonymous probability measure is represented by $\mathbb{P}$; this will appear in cases where a probability measure is being used where one has not been explicitly defined, and the definition of it should be clear from the context. The number of elements in a finite set A will be denoted $|A|$. Generally, we will denote index sets as $\mathcal{I}$ or its non calligraphic form I. We also state the following definitions. For a given index set $\mathcal{I}$, we define the notation $(x_i)_{i\in\mathcal{I}}$ to be the $|\mathcal{I}|$-tuple of elements $(x_{i_1}, x_{i_2}, \ldots, x_{i_n})$, where $\{i_1, \ldots, i_n\} = \mathcal{I}$. In all cases in this paper, there will exist some order on $\mathcal{I}$ (almost always we will have $\mathcal{I} \subset \mathbb{F}$ for some finite field $\mathbb{F}$), and so in the above we will assume that $i_1 < i_2 < \cdots < i_n$. The empty set will be denoted $\emptyset$.

Definition 1. *Let a set A be given. Denote the power set of A, i.e. the set of all subsets of A, by $\wp(A)$.*

Definition 2. *Let the set of consecutive natural numbers from 1 to n inclusive, i.e. $1, 2, \ldots, n$, be denoted by $[n]$.*

Definition 3. *Let the set of consecutive natural numbers from 0 to n inclusive, i.e. $0, 1, \ldots, n$, be denoted by $[n]_0$.*

2.2 General Definitions

We will sometimes refer to a function as being *negligible*. This is made precise in the following definition.

Definition 4 (Negligible function)**.** *A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for all $c \in \mathbb{N}$ there exists some $N \in \mathbb{N}$ such that for all $x \geq N$*

$$|f(x)| < \frac{1}{x^c}.$$

In this case we say that f ***is a negligible function****, or that f* ***is negligible****.*

2.3 Probability

Many of the security definitions and consequently also the proofs of security are formulated in terms of probability theory. We therefore collect some key definitions and theorems that we will use in this paper.

We will use the measure theoretic formulation of probability theory. The key structure from measure theory is that of a *measure space*, which is defined as follows.

Definition 5. *A measure space is a triple (X, Σ, μ) where X is a set, Σ is a σ-algebra on X and μ is a function from Σ to $\mathbb{R}$ such that the following conditions hold:*

- *For all $E \in \Sigma$, $\mu(E) \geq 0$*
- $\mu(\emptyset) = 0$
- *For any countable collection of disjoint sets $E_1, E_2, \ldots \in \Sigma$ we have*

$$\mu\left(\bigcup_{i=1}^{\infty} E_i\right) = \sum_{i=1}^{\infty} \mu(E_i)$$

In this formulation of probability theory, a *probability space* is simply a special case of a measure space:

Definition 6. *A probability space is a measure space $(\Omega, \Sigma, \mathbb{P})$ such that the measure of the entire space is 1; that is,*

$$\mathbb{P}(\Omega) = 1.$$

We will often consider the uniform probability measure on a finite set. We define the standard notation that we will use and formalise the concept as follows.

Definition 7 (Uniform probability measure)**.** *Let E be a finite set. We denote the uniform probability measure on the measurable space $(E, \wp(E))$ by*

$$U_E : \wp(E) \to \mathbb{R}.$$

That is, for each $A \in \wp(E)$ we have $U_E(A) = \frac{|A|}{|E|}$ where the notation $|\cdot|$ denotes the number of elements in the given set.

The next key object from probability theory is the *random variable*. This is formalised as a measurable function.

Definition 8. *Let $(\Omega, \Sigma, \mathbb{P})$ be a probability space and $(E, \mathcal{E})$ be a measurable space. A random variable X is a measurable function $X : \Omega \to E$.*

Note that the random variable X induces a probability measure μ on the measurable space $(E, \mathcal{E})$ which we can define by

$$\begin{aligned} \mu : \mathcal{E} &\to \mathbb{R} \\ A &\mapsto \mathbb{P}(\{\omega \in \Omega | X(\omega) \in A\}), \end{aligned}$$

and so we also have the induced probability space $(E, \mathcal{E}, \mu)$. Also note that for a measurable space $(F, \mathcal{F})$ and measurable function $f : E \to F$, we can define a new random variable $Y = f \circ X$. Thus when we write $f(X)$ we understand this to be the random variable Y. It is also useful to define the following common short hand notations:

Definition 9. *Let $(\Omega, \Sigma, \mathbb{P})$ be a probability space, $(E, \mathcal{E})$ be a measurable space and $X : \Omega \to E$ be a random variable. Let the induced probability measure on $(E, \mathcal{E})$ be μ. Define the following short hand notations:*

- $\mathbb{P}(X = x) = \mu(X = x) = \mathbb{P}(\{\omega \in \Omega | X(\omega) = x\}) \quad \forall x \in E$
- $\mathbb{P}(X \in A) = \mu(X \in A) = \mu(A) \quad \forall A \in \mathcal{E}$

While the latter is actually not shorter than the technically correct version, it is conceptually clearer which is why we use it.

We now define what it means for two random variables to have equal distributions. The idea is that the possible outcomes for the random variables should have the same probabilities.

Definition 10. *Let $(\Omega_1, \Sigma_1, \mathbb{P}_1)$ and $(\Omega_2, \Sigma_2, \mathbb{P}_2)$ be probability spaces, $(E, \mathcal{E})$ a measurable space, and $X_1 : \Omega_1 \to E$ and $X_2 : \Omega_2 \to E$ be two random variables. We say that X_1 and X_2 are **identically distributed** if $\forall A \in \mathcal{E}$ we have*

$$\mathbb{P}_1\left(\{\omega \in \Omega_1 | X_1(\omega) \in A\}\right) = \mathbb{P}_2\left(\{\omega \in \Omega_2 | X_2(\omega) \in A\}\right).$$

In this case we write $X_1 \overset{d}{=} X_2$.

Here we prove a result that will be useful in the proofs of security. This formalises the notion that adding a uniformly random element to some other element results in an element that is itself uniformly random. This is presented abstractly in the following theorem, and then the result of interest follows as a corollary.

Theorem 1. *Let $(\Omega, \Sigma, \mathbb{P})$ be a probability space, E a finite set with n elements, and $f_i : \Omega \to E$ for $1 \leq i \leq n$ be functions that are measurable in the context of the measurable spaces (Ω, Σ) and $(E, \wp(E))$ and satisfy the property*

$$f_i(\omega) \neq f_j(\omega)\ \forall \omega \in \Omega\ \forall i \neq j \text{ where } i, j \in \{1, \ldots, n\}.$$

Let $F = \{f_1, \ldots, f_n\}$ and consider the probability space $(F, \wp(F), U_F)$. Construct the product probability space $(\Omega \times F, \Sigma \times \wp(F), \mu)$ where μ is the appropriate product measure. Define the random variable

$$\begin{aligned} X : \Omega \times F &\to E \\ (\omega, f) &\mapsto f(\omega) \end{aligned}$$

Then X is uniformly distributed on E.

Proof. We need only show that $\mu(X = e) = \frac{1}{n}$ for all $e \in E$. To do this, define

$$A_{i,e} = \{\omega \in \Omega | f_i(\omega) = e\}.$$

Since each f_i is measurable, it follows that $A_{i,e} \in \Sigma$ for all $i \in [n]$ and $e \in E$. Notice that these sets partition Ω; this is summarised in the following claim.

Claim 1. *The sets $A_{i,e}$ for $i \in [n]$ partition Ω.*

Proof. To show this, we need to show that each $A_{i,e}$ is disjoint, and that they cover Ω. The first property holds due to the assumption that $f_i(\omega) \neq f_j(\omega)$ for all $i \neq j$, $i, j \in [n]$ and for all $\omega \in \Omega$. This assumption also shows that for all $\omega \in \Omega$ there must exist some $i \in [n]$ for which $f_i(\omega) = e$, as E only has n elements. But this is precisely the statement that $\omega \in A_{i,e}$. This completes the proof. □

Also, these sets define a partition of our set of interest:

$$\bigcup_{i=1}^{n} A_{i,e} \times \{f_i\} = \{(\omega, f) \in \Omega \times F | X(\omega, f) = e\}.$$

Thus for any $e \in E$ we have

$$\mu(X = e) = \sum_{i=1}^{n} \mu\left(A_{i,e} \times \{f_i\}\right). \tag{1}$$

By the definition of the product measure we know that

$$\mu(A_{i,e} \times \{f_i\}) = \mathbb{P}(A_{i,e}) U_F(\{f_i\}),$$

and so using this and the fact that $A_{1,e}, \ldots, A_{n,e}$ partitions Ω we can reduce (1) to

$$\begin{aligned}\mu(X = e) &= \frac{1}{n} \sum_{i=1}^{n} \mathbb{P}(A_{i,e}) \\ &= \frac{1}{n},\end{aligned}$$

which is the desired result. □

Corollary 1. *Let $(\mathbb{F}^n, \wp(\mathbb{F}^n), \mathbb{P})$ be a probability space, where $\mathbb{F}$ is a finite field. Let $x, y \in \mathbb{F}^n$ be random such that x is distributed according to $\mathbb{P}$, and y is uniformly distributed and independent from x. Then the random variable $x + y$ is uniformly distributed, where when $n > 1$ the addition is element-wise.*

Proof. This follows from Theorem 1 by setting $(\Omega, \Sigma, \mathbb{P}) = (\mathbb{F}^n, \wp(\mathbb{F}^n), \mathbb{P})$, $E = \mathbb{F}^n$, and defining f_i by $f_i(\omega) = \omega + i$ for all $i \in \mathbb{F}^n$. □

Here we formalise another result that will be used in the proofs of security, to ease their exposition. The result is simple: if two random variables have equal distributions, then applying some deterministic function f to these random variables results in two new random variables that also have equal distributions.

Theorem 2. *Let $(\Omega_1, \Sigma_1, \mathbb{P}_1)$ and $(\Omega_2, \Sigma_2, \mathbb{P}_2)$ be probability spaces, $(E, \mathcal{E})$ be a measurable space, and $X_1 : \Omega_1 \to E$ and $X_2 : \Omega_2 \to E$ be random variables. Let $(F, \mathcal{F})$ be a measurable space and $f : E \to F$ be a measurable function. Then if $X_1 \overset{d}{=} X_2$, it follows that $f(X_1) \overset{d}{=} f(X_2)$.*

Proof. Define the random variables $Y_1 = f \circ X_1$ and $Y_2 = f \circ X_2$ and let $A \in \mathcal{F}$ be arbitrary. We need to show that $\mathbb{P}_1(Y_1^{-1}(A)) = \mathbb{P}_2(Y_2^{-1}(A))$. But by assumption $\mathbb{P}_1(X_1^{-1}(B)) = \mathbb{P}_2(X_2^{-1}(B))$ for any $B \in \mathcal{E}$, and in particular for $B = f^{-1}(A)$. □

2.4 Secret Sharing

We will consider the secret sharing technique of Shamir [1]. Shamir secret sharing is a technique that allows for the distribution of a secret to n players such that any subset of k or more players can reconstruct the secret, but any less than k can learn nothing about the secret. The parameters $n, k \in \mathbb{N}$ (where $n \geq k$) can be chosen as required.

Any secret in a field can be shared. Let $(\mathbb{F}, +, \cdot)$ be a finite field and $n, k \in \mathbb{N}$ be given where $n \geq k$. Let the n players be $\{P_i\}_{i \in \mathcal{I}}$ where $\mathcal{I} \subset \mathbb{F} \setminus \{0\}$ and $|\mathcal{I}| = n$. We can "share" a secret $s \in \mathbb{F}$ by first choosing uniformly random $c_1, \ldots, c_{k-1} \in \mathbb{F}$ and constructing the polynomial

$$p(x) = s + \sum_{j=1}^{k-1} c_j x^j.$$

Player P_i is given a "share" $s_i = p(i)$. Note that $p(0) = s$ which is why we should ensure that $0 \notin \mathcal{I}$.

Any subset of players $\{P_i\}_{i\in\mathcal{R}}$ such that $\mathcal{R} \subset \mathcal{I}$ and $|\mathcal{R}| \geq k$ can reconstruct the secret s by interpolating to reconstruct the original polynomial as follows:

$$p(x) = \sum_{i\in\mathcal{R}} s_i \prod_{\substack{j\in\mathcal{R}\\ j\neq i}} \frac{x - x_j}{x_i - x_j}.$$

The secret is simply $p(0)$ and so the players can find s using the above;

$$s = \sum_{i\in\mathcal{R}} s_i \prod_{\substack{j\in\mathcal{R}\\ j\neq i}} \frac{x_j}{x_j - x_i}.$$

Define the function that takes the random coefficients to resulting shares by

$$\begin{aligned} \vartheta_k : \wp(\mathbb{F}) \times \mathbb{F}^k &\to \mathbb{F}^{|\mathcal{I}|} \\ \mathcal{I}, (c_0, \ldots, c_{k-1}) &\mapsto \left(\sum_{i=0}^{k-1} j^i c_i \right)_{j\in\mathcal{I}}, \end{aligned}$$

where naturally the secret is c_0.

We now prove a result that tells us how a subset of shares is distributed given that we know how the secret is distributed. This result will be useful later on in our security proofs.

Theorem 3. *Let $x_1, \ldots, x_n$ be a k-sharing of some $x \in \mathbb{F}$, where x has distribution determined by the probability space $(\mathbb{F}, \wp(\mathbb{F}), \mu)$. Suppose that the sharing is a uniformly random one; that the remaining coefficients $c_1, \ldots, c_{k-1}$ that define the sharing are all uniformly and independently distributed. Let $I \subset \mathcal{I}$ with $|I| = t$. Precisely, the set $(x_i)_{i\in I}$ is the random variable*

$$\begin{aligned} X : \mathbb{F}^k &\to \mathbb{F}^t \\ (x, c_1, \ldots, c_{k-1}) &\mapsto \left(x + \sum_{j=1}^{k-1} c_j i^j \right)_{i\in I} \end{aligned} \tag{2}$$

which has the associated product probability measure that we will denote by $\mathbb{P}_t$. Then this measure satisfies

$$\mathbb{P}_t(X = z) = \begin{cases} \mu(\{y\})|\mathbb{F}|^{-k+1} & t \geq k \\ |\mathbb{F}|^{-t} & t < k \end{cases}$$

for all $z \in \mathbb{F}^t$ that form part of some consistent k-sharing, where in the former case $y \in \mathbb{F}$ is determined uniquely from z.

Proof. We consider three separate cases.

- $t \geq k$: In this case, we can view equation (2) as a set of t linear equations with k unknowns, and hence has a unique solution for $(x, c_1, \ldots, c_{k-1})$ (and in the cases where $t > k$ and the set of equations is overconstrained, we know that it will not have no solutions because we have defined the sharing to be correct). Let this solution be $z = (y, z_1, \ldots, z_{k-1})$. Then from independence we have

$$\begin{aligned}\mathbb{P}_t(X = z) &= \mathbb{P}_t\left(x = y \cap \bigcap_{i=1}^{k-1} c_i = z_i\right) \\ &= \mathbb{P}_t(x = y)\mathbb{P}_t((c_1, \ldots, c_{k-1}) = (z_1, \ldots, z_{k-1})) \\ &= \mu(\{y\})|\mathbb{F}|^{-k+1},\end{aligned}$$

  which is the desired result for this case.

- $t = k - 1$: Suppose that we fix x in equation (2). We will again have a system of (in this case $k - 1$) linear equations with as many unknowns, and so we must have a unique solution d_y for $(c_1, \ldots, c_{k-1})$ for a given set of shares z. We may therefore write

$$\begin{aligned}\mathbb{P}_t(X = z) &= \sum_{y \in \mathbb{F}} \mathbb{P}_t(x = y \cap (c_1, \ldots, c_{k-1}) = d_y) \\ &= \sum_{y \in \mathbb{F}} \mu(\{y\})|\mathbb{F}|^{-k+1} \\ &= |\mathbb{F}|^{-k+1}.\end{aligned}$$

  Since $t = k - 1$ this is the desired result.

- $t < k - 1$: Here we take a similar approach to the previous case. If we fix $x = y$ but also $(c_{t+1}, \ldots, c_{k-1}) = c$ in equation (2), then we have t linear equations in the t unknowns $c_1, \ldots, c_t$, which has a unique solution for a given set of shares z that we will denote $d_{y,c}$. We may then write the probability $P(X = z)$ as

$$\sum_{y \in \mathbb{F}} \sum_{c \in \mathbb{F}^{k-1-t}} \mathbb{P}_t(x = y \cap (c_1, \ldots, c_t) = d_{y,c} \cap (c_{t+1}, \ldots, c_{k-1}) = c),$$

  which we can again use independence and our known distributions to simplify to

$$\sum_{y \in \mathbb{F}} \sum_{c \in \mathbb{F}^{k-1-t}} \mu(\{y\})|\mathbb{F}|^{-t}|\mathbb{F}|^{-k+1+t} = |\mathbb{F}|^{-t}.$$

  This final case completes the proof.

□

Often, we will need to consider what the distribution of the shares are, given some subset that has been fixed. The following definition introduces a notation for the set of possible sharings given the fixed subset.

Definition 11. *Let $t, k, n \in \mathbb{N}$ with $t < k \leq n$. Let $\mathcal{I} \in \mathbb{F}^n$ be an index set with $|\mathcal{I}| = n$ and let $I \subset \mathcal{I}$ be such that $|I| = t$. Let the shares $x = (x_i)_{i \in I} \in \mathbb{F}^t$ be fixed. Then we define the set $S_{x,\mathcal{I},I}$ to be the set of possible collections of $n - t$ shares that extend x to a consistent k-sharing that has n shares of some field element. Precisely, $S_{x,\mathcal{I},I}$ is the set of all values $(y_i)_{i \in \mathcal{I} \setminus I}$ in $\mathbb{F}^{n-t}$ such that there exists $(c_0, \ldots, c_{k-1}) \in \mathbb{F}^k$ for which the following hold:*

$$x_i = \sum_{j=0}^{k-1} c_j i^j \quad \forall i \in I,$$

$$y_i = \sum_{j=0}^{k-1} c_j i^j \quad \forall i \in \mathcal{I} \setminus I.$$

An easy property of $S_{x,\mathcal{I},I}$ is that it has $|\mathbb{F}|^{k-t}$ elements. This is summarised in the following theorem.

Theorem 4. *Let $t, k, n \in \mathbb{N}$ with $t < k \leq n$. Let $\mathcal{I} \in \mathbb{F}^n$ be an index set with $|\mathcal{I}| = n$ and let $I \subset \mathcal{I}$ be such that $|I| = t$. Let the shares $x = (x_i)_{i \in I} \in \mathbb{F}^t$ be fixed. Then*

$$|S_{x,\mathcal{I},I}| = |\mathbb{F}|^{k-t}.$$

Proof. We know that if we have k fixed shares $y_1, \ldots, y_k \in \mathbb{F}$ with corresponding indices $i_1, \ldots, i_k \in \mathbb{F}$, then they are related to the coefficients of the sharing by the k equations

$$y_j = \sum_{l=0}^{k-1} c_l i_j^l.$$

Since this is k linear equations in the k unknowns $c_0, \ldots, c_{k-1}$, we know that there is a unique solution. It follows that the same is true given our t fixed shares x if we fix a further $k - t$ of them. But these additional $k - t$ shares can also be any values, and so we have $|\mathbb{F}|^{k-t}$ choices for these, after which all of the shares will be fixed. The result follows. □

Next, we prove a theorem about the relationship of the shares of a secret to the coefficients of the associated polynomial. It is clear from the definition of the shares that they are a linear map of the coefficients, but the following theorem shows that the reverse is also true; the coefficients can be computed as a linear map applied to some set of k shares.

Theorem 5. *Let $x \in \mathbb{F}$ and $(x_i)_{i \in \mathcal{I}}$ be a k-sharing of x for some index set $\mathcal{I} \subset \mathbb{F}$ with $|\mathcal{I}| = n \geq k$. Let the associated coefficients for the sharing be $c_0, \ldots, c_{k-1}$.*

Then for each $\mathcal{R} \subset \mathcal{I}$ with $|\mathcal{R}| = k$ and $j \in \{0, \ldots, k-1\}$, there exist field elements $(\lambda_i^{(j)})_{i \in \mathcal{I}}$ such that

$$c_j = \sum_{i \in \mathcal{R}} \lambda_i^{(j)} x_i.$$

Proof. Consider the *Vandermonde* matrix for the elements $(x_i)_{i \in \mathcal{R}}$, which is defined as

$$V = \begin{pmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^t \\ 1 & i_2 & i_2^2 & \cdots & i_2^t \\ 1 & i_3 & i_3^2 & \cdots & i_3^t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^t \end{pmatrix},$$

where $\mathcal{R} = \{i_1, \ldots, i_k\}$. If we define s to be the k-vector of shares $(x_i)_{i \in \mathcal{R}}$, and c to be the k vector of coefficients, then by the way secret sharing is defined we have the relationship

$$Vc = s.$$

It is well known that the determinant of the Vandermonde matrix V above is

$$\det(V) = \prod_{1 \leq p < q \leq k} (i_p - i_q),$$

and is hence non-zero, if $i_p \neq i_q$ for all $p, q \in \mathcal{R}$, $p \neq q$, which in our case is true. This means that V is invertible, and so we may write

$$c = V^{-1} s.$$

We can thus complete the proof by defining $\lambda_i^{(j)}$ as the (j, i) element of V^{-1}. □

2.5 Discrete Logarithm Assumption

We define here a computational assumption that will be relevant to our random key pair generation protocol. The assumption is a standard and common one from the literature: that computing the discrete logarithm is hard. This is formalised for our specific context as follows.

Assumption 1 (Discrete Logarithm)**.** *Let p be a prime such that $p \geq 2^b$ where $b \in \mathbb{N}$. Let $\mathbb{G}$ be a group of prime order p with generator g and let $\mathbb{F}$ be the associated finite field of integers modulo p. Then for every Turing machine T and uniformly random field element $x \in \mathbb{F}$, $\mathbb{P}(T(\mathbb{G}, g, g^x) = x)$ is negligible.*

3 Security

In this section we will outline the security model and definitions that we will use to prove that the presented protocols are secure.

The notion of security we will use is that of Canetti [2], which is based on the idea of comparing a designed protocol to an ideal case that is secure by definition. More precisely, we consider an n-party function f that takes as input the inputs of each of the parties, and gives as output the outputs for all of the parties; this defines what functionality we want our protocol to achieve. Then, we define a protocol π which we want to "securely" realise f, and prove that it is secure by comparing the "real-life model" in the presence of an adversary $\mathcal{A}$ and the "ideal case":

- *Real-life model*: The corrupted parties are controlled by the adversary $\mathcal{A}$, which learns their identities and inputs. The protocol π proceeds in rounds, where in each round uncorrupted parties first follow π correctly and send any messages they need to. Next, $\mathcal{A}$ receives any messages destined for corrupted parties, and then decides what messages the corrupted parties should send in that round. This process repeats each round until π has completed execution. The parties then produce their output; uncorrupted parties output their true output, while corrupted parties output a special symbol $\perp$ to indicate that they were corrupted.

- *Ideal case*: All parties hand their inputs for the protocol to an incorruptible trusted party T which computes the ideal functionality f and then hands each party their respective results. Each party produces their respective output as in the real-life model.

See the paper for more details [2]. An important property to note is that the adversary gets to see the honest parties' message in a given round before deciding its own; this property is called *rushing*. Note also that this model is synchronous because it proceeds in will defined rounds. However, we will discuss later that the synchronicity implied here is perhaps stronger than what we actually achieve with the presented protocols.

If for each $\mathcal{A}$ that operates in the real-life model, we can construct an adversary $\mathcal{S}$ in the ideal case such that for the random variable that represents the information gathered by $\mathcal{A}$ and the outputs of the parties, $\mathcal{S}$ can construct an output with the same distribution, then we will say that the protocol π is secure. Canetti also proves that this security definition enjoys a *composability* property, in that if a protocol π uses a secure subprotocol π' as part of its execution, then to prove π is secure one need only prove this when π' has been substituted by the associated n-party functionality f. Additionally, in this paper we will consider *active* adversaries, which means that they can deviate arbitrarily from the protocol and send arbitrary messages. Before the start of the protocol, the adversary (this applies to both $\mathcal{A}$ and $\mathcal{S}$) may also modify the inputs of the corrupted parties in any way. We also consider computationally bounded adversaries that cannot compute discrete logarithms, as in Assumption 1.

The random variable corresponding to the execution of the protocol π in the presence of an adversary $\mathcal{A}$ is defined as follows.

Definition 12. *Let π be an n-party protocol and let $\mathcal{A}$ be a t-limited adversary. Let x be some input for the parties, and z be some auxiliary input. Define $\texttt{EXEC}_{\pi,\mathcal{A}}(x,z)$ to be the random variable that is the messages that the corrupt parties send and receive during the execution of π, the auxiliary input and the outputs of each party P_i.*

The auxiliary input z is used to prove the composability of the security definition. It represents a possible state for $\mathcal{A}$ that might occur due to the protocol being a subprotocol of a larger protocol, and so can include information about previous executions up to that point. Similarly, we define the random variable that corresponds to the execution in the ideal case with and adversary $\mathcal{S}$.

Definition 13. *Let f be an n-party function and let $\mathcal{S}$ be an adversary. Let x be some input for the parties, and z be some auxiliary input. Define $\texttt{IDEAL}_{f,\mathcal{S}}(x,z)$ to be the random variable that is the output of $\mathcal{S}$ along with the outputs of each party P_i after an execution with the trusted party T evaluating f in the ideal case.*

We will often refer to $\texttt{EXEC}_{\pi,\mathcal{A}}(x,z)$ (or sometimes all parts of it excluding the outputs of the parties) as the *view* of $\mathcal{A}$ running π with inputs x and z, as this is what $\mathcal{A}$ "sees" during execution; particularly the messages. Similarly, we often call the output of $\mathcal{S}$ the *simulated view*.

Definition 14. *Let f be an n-party function and let π be an n-party protocol. We say that π t-securely evaluates f if for any nonadaptive and t-limited adversary $\mathcal{A}$ there exists a nonadaptive adversary S with running time polynomial in the running time of $\mathcal{A}$ such that for all x and z*

$$\texttt{EXEC}_{\pi,\mathcal{A}}(x,z) \stackrel{d}{=} \texttt{IDEAL}_{f,\mathcal{S}}(x,z).$$

Remark 1. While the auxiliary input z serves an important role in allowing for the security definition to be composable, its presence in the individual security proofs is not needed; since $\mathcal{S}$ is given z, it is trivial for it to include it in its output. For this reason, we do not mention the auxiliary input in our security proofs.

Remark 2. The definition presented is a simplified version of the one given in Canetti's paper. This is because the latter is made to be general over the different possible levels of security: *perfect* security, *statistical* security and *computational* security. It is also designed to be general over possibly infinite domains for the inputs. However, in our case we only consider finite domains and perfect security, so the definition is suitably simplified. Note that even though a protocol may use a subprotocol that has only statistical or computation security, we can still prove it is secure using perfect security, since perfect security implies statistical security, which in turn implies computational security. For the protocols in this paper we will prove perfect security since it is both the strongest result but also more convenient.

4 Cryptographic Primitives

In this section we briefly outline some cryptographic primitives that will be used in our SMPC protocols.

4.1 Public Key Encryption

For some protocols, we will assume the existence of a public key encryption system. The precise security definition of the encryption is not important for this context. We will denote the output of encryption of a message m using a public key κ by $E_\kappa(m)$.

4.2 Zero Knowledge Proofs

We will make use of general purpose zero knowledge (ZK) proofs in some of our protocols. In some cases, we will require that these proofs have the *perfect zero knowledge* property, which is summarised in the following definition by Goldreich [3].

Definition 15 (Perfect Zero Knowledge)**.** *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is **Perfect Zero Knowledge** if for every probabilistic polynomial time interactive Turing machine V^* there exists a probabilistic polynomial time algorithm M^* such that for every $x \in L$ the following two conditions hold:*

1. *M^* succeeds with probability at least $\frac{1}{2}$; i.e. if we denote a failure by the output of $\perp$, then*

$$\mathbb{P}(M^*(x) = \perp) \leq \frac{1}{2}.$$

2. *Let X be the random variable that is the output of V^* after interacting with the interactive machine P on common input x, and let $m^*(x)$ be the random variable that is exactly $M^*(x)$ but conditioned on the fact that $M^*(x) \neq \perp$. Then we require that these two random variables are equally distributed, i.e.*

$$m^*(x) \stackrel{d}{=} X.$$

4.3 Consensus

For some of the protocols described in this paper, we will make use of a consensus algorithm. A consensus algorithm allows a network of *processes*, some of which may be *faulty*, to arrive at a joint decision value. The key properties of a consensus algorithm that we will consider are those defined Buchman et al. [4].

1. *Termination*: Every nonfaulty process eventually decides on a value.

2. *Agreement*: No two nonfaulty processes decide on different values.

3. *Validity*: A decided value is valid, i.e., it satisfies the predefined predicate denoted `valid`().

Note that it is not explicitly stated, but it is necessary for the function `valid` to be global in the sense that every nonfaulty process computes the same result for `valid`(v). This is important to keep in mind for the following, in which some decision values will contain data encrypted for specific players, which of course means only those specific players can decrypt this data and hence the associated plaintext should not be used as a part of the checks in `valid`.

We define a specific protocol that uses consensus when generating global (secret shared) random numbers, which we denote $\rho^{n,k}_{\texttt{RNG}}$. It is defined as follows. Let the parties participating in the protocol be $\{P_i\}_{i\in\mathcal{I}}$, such that each party P_i has a public and private keypair where the public key is denoted κ_i. Each party P_i has input $c_0^{(i)},\ldots,c_{t-1}^{(i)}$, which are the coefficients of a polynomial that is to be used for a threshold t sharing of the number $c_0^{(i)} \in \mathbb{F}$. They then do the following to construct their input for a consensus protocol ρ:

1. Create n shares $r_1^{(i)},\ldots,r_n^{(i)}$ from the coefficients $c_0^{(i)},\ldots,c_{t-1}^{(i)}$.

2. Obtain the set of encrypted values $e_i = \{e_1^{(i)},\ldots,e_n^{(i)}\}$, where $e_j^{(i)} = E_{\kappa_j}\left(r_j^{(j)}\right)$.

3. Create a ZK proof ζ_i that asserts the following:

   - $e_j^{(i)} = E_{\kappa_j}\left(r_j^{(i)}\right)$ for all $j \in \mathcal{I}$.
   - The values $r_1^{(i)},\ldots,r_n^{(i)}$ constitute a valid and consistent sharing of some value in $\mathbb{F}$ (in this case, that value is $c_0^{(i)}$).

A possible decision value for ρ is a set $\{(e_i,\zeta_i)\}_{i\in I}$ where $I \subset \mathcal{I}$. The predicate is $\texttt{valid}_{\texttt{RNG}}$, which is true precisely when $|I| > t$ and $\texttt{verify}(\zeta_i) = 1$ for all $i \in I$. The output for each party is the decrypted set of shares which were encrypted for their public key; namely the output for party P_i is $(r_i^{(j)})_{j\in I}$.

Part of the use of the ZK proofs here is to achieve the goal of having *verifiable secret sharing*, for which the more common solution is to use more specialised (and hence usually more efficient) techniques such as that of Feldman [5] or Pedersen [6]. The reason that these solutions were not used is to weaken the synchronicity requirements but also reduce the number of rounds of communication. Using the standard solutions allows each party to identify when a dealer has not consistently shared their secret, but each party needs to be aware of this and hence they need to agree on dealers that are faulty. This usually requires additional rounds of broadcasting complaints (if there are any), and often doing so using a secure

broadcast channel[1]. Using the more powerful ZK proofs, in which it is proved that all of the encrypted shares are consistent, allows each party to check that every other parties' share is correct in the course of the consensus algorithm. This means that upon achieving consensus, no further coordinating or complaint broadcasting is needed as the only decision values that are selected are those for which the required threshold of parties agreed that `valid` returned true. Using $\rho^{n,k}_{\texttt{RNG}}$ thus only introduces the synchronicity required for the consensus algorithm, which for example in the case of Tendermint [4] is only partial synchronicity.

This consensus protocol will be used as a subprotocol for some of our SMPC primitives, and so we will provide some brief justification as to why they can be used securely. Recall that the two key properties that definition 14 captures are secrecy and correctness. The former requires that the messages do not leak any information, and we can see that this is the case given that the proposed values only contain encrypted data, ZK proofs and public keys, all of which do not leak information given appropriate assumptions. The latter requires that the adversary cannot influence the output. This is not strictly attained, for example when using Tendermint [4] consensus a party that is controlled by the adversary can propose a block and hence influence what is in it, but to counter this we will just combine the consensus protocol with other protocols to arrive at adequate security for the larger protocol that uses consensus. Regarding the computation model, we require that a subprotocol instance can behave like a "round" and have a definite completion time. This is ensured for protocols that want to use consensus by the termination property of the consensus protocol.

5 SMPC Primitives

In this section we will define the primitives which constitute our SMPC protocols and prove their security. It is clear that the security model trivially applies when a protocol is "local"; i.e. does not involve any exchange of messages. This means that we need only consider proofs of security for those protocols that involve sending messages. In our case, these protocols are open (Section 5.1), random number generation (Section 5.2), random zero generation (Section 5.3), and random keypair generation (Section 5.4).

5.1 Open

At some points during a computation on shared values, and almost certainly at the end of a larger protocol, we will want to reveal the secret corresponding to the underlying shares. We call this revealing operation "Opening". Probably the most obvious ideal functionality for this would be defined as follows: f takes as input the shares of the secret from each party and gives as output the

[1] In this context, *secure broadcast* means secure in the sense of our security definitions, and not in the intuitive encrypted messaging sense; a secure broadcast can be thought of as a protocol where each honest party outputs the same value. This is obviously not achieved by a naïve approach of simply sending the message to everyone, as the adversary may send different messages to different parties.

corresponding secret s to each party. However, if we were to try to achieve this with the simplest and most obvious protocol, everyone broadcasting their share and then reconstructing, we would not be able to prove security in the framework we are using because any simulator, knowing only the corrupted shares and the secret, would not be able to produce the shares of the honest parties which it would have to as these are messages that are sent during the protocol. The interpretation of this is clear: this protocol is not secure because it leaks private information (the input shares) from the honest parties, or alternatively, because the ideal functionality specifies that this private information is not revealed. The two solutions for these two perspectives leads to two different protocols which warrant use in their own contexts.

First, from the perspective that the simple broadcast protocol should not reveal the input shares, the solution is to improve the protocol so that this information isn't revealed. One way that this can be achieved is by first generating a random k-sharing of $0 \in \mathbb{F}$ and adding that to the shares before broadcasting them; now the messages sent during the protocol are for a *random* sharing of the secret, as opposed to the specific sharing that the parties started the protocol with, and a simulator can now generate these messages with the right distribution knowing only the secret. Second, from the perspective that the ideal functionality is too restrictive, we can modify it to also output the shares themselves along with the secret; a simulator for this ideal functionality can now easily produce an appropriate view for the simple broadcast protocol.

We can see that in either of the above cases the key difference is whether or not the original shares are considered safe to reveal. Often they are safe to reveal, but there are cases where they are not. One example in which they are not is having k-sharings of two private values $a, b \in \mathbb{F}$, and each party multiplies their shares locally to get a $2k$-sharing of ab and then opening this value. While it may be "safe" to reveal ab (e.g. if a is a secret but b is an unknown uniformly random value), it is well known that this $2k$ sharing is defined by a degree $2k-1$ polynomial that is not uniformly random (an easy reason that this is the case is that given it is the product of two polynomials, it can't possibly be irreducible), and in fact exhibits enough structure that knowing the shares, one can often easily determine both a and b individually.

With the above considerations in mind, we will define two protocols for open, one for each of the cases. In both cases, we will use a Reed-Solomon (RS) decoding technique, such as Berlekamp-Welch [7] or Gao's [8], to remain fault tolerant in the presence of corrupt parties broadcasting incorrect shares. For an (n, k) RS code (i.e. n, k Shamir secret sharing) these algorithms are able to detect when there are up to $d = n - k + 1$ errors and correct up to $\lfloor \frac{d}{2} \rfloor$ errors[2]. Notice that RS decoding allows us to recover the underlying polynomial, and so this means that we not only obtain the secret for our sharing but also all of the other shares of every other party.

The protocol implementing the basic ideal functionality (that outputs the

[2]Note that these algorithms are also able to locate the errors when they are corrected, which in our context means that we would be able to identify corrupt nodes.

secret as well as all of the input shares) $f^{n,k}_{\texttt{OPEN}}$ is denoted by $\pi^{n,k}_{\texttt{OPEN}}$ and is defined as follows.

1. Each party P_i broadcasts their input share x_i.
2. Each party performs the reconstruction of the secret using a suitable RS decoding algorithm and outputs the result, along with the other reconstructed shares.

Theorem 6. *Let $t \leq \frac{n-k+1}{2}$. Then the protocol $\pi^{n,k}_{\texttt{OPEN}}$ t-securely evaluates $f^{n,k}_{\texttt{OPEN}}$.*

Proof. Since the only messages sent are the input shares and these are included in the output of the trusted party, the proof is trivial. The only thing to note is that the output needs to include the input shares of all parties, including those corrupted parties that may broadcast arbitrary values. This is easily overcome by using an RS decoding technique and noting that the restriction on t ensures that it will always be possible to reconstruct their shares. □

The protocol implementing the stricter ideal functionality $f^{n,k}_{\texttt{SO}}$ is denoted by $\pi^{n,k}_{\texttt{SO}}$ and is the same as the simple version, except before revealing shares for reconstruction, the sharing is randomised by adding a random sharing of $0 \in \mathbb{F}$ first. The protocol for generating this randomising sharing, $\pi^{n,k}_{\texttt{RZG}}$, is defined in Section 5.3. The protocol $\pi^{n,k}_{\texttt{SO}}$ is defined as follows.

1. The parties participate in $\pi^{n,k}_{\texttt{RZG}}$ to get a share of zero z_i.
2. Each party P_i broadcasts their corresponding share $x_i + z_i$.
3. Each party performs the reconstruction of the secret using a suitable RS decoding algorithm and outputs the result.

Remark 3. We make a note here on synchronicity regarding the use of a RS decoding technique. The main feature of the security definition that relates to synchronicity is the fact that the protocol proceeds in rounds and each round has to be completed before moving on to the next. This would either mean waiting for *all* messages from other players, or using a timeout after which any messages that have not arrived are given some default value to be used instead. However, these alternatives need not be used for the final round of each open protocol in which each party sends its share to every other party. This is because as soon as a party has at least $n - \lfloor \frac{d}{2} \rfloor$ shares, the decoding algorithm will be able to successfully recover the secret, setting the yet to be received messages as any value. This means that if there are no more than $\lfloor \frac{d}{2} \rfloor$ adversaries (and this category also includes offline/unresponsive players), then the round is guaranteed to terminate according to any termination assumptions of *only the honest players*. This means that in practice we will not need to wait for all messages (which is not lively) and also will have no need for timeouts.

Concretely, the following is a way in which the honest parties can be sure to terminate without having to have synchronised clocks or use timeouts.

1. Wait until $n - \lfloor \frac{d}{2} \rfloor$ shares have been received. This is guaranteed to happen as there are at least this many honest parties.

2. Starting from the previous step and repeating on every new share until a reconstruction is possible: set the shares of the parties for which no shares have yet been received as 0, and run the reconstruction algorithm.

Repeating the last step will eventually terminate with the correct value. First, notice that it will never yield an incorrect value. This is because no reconstructions are attempted until there are at least $n - \lfloor \frac{d}{2} \rfloor$ values, which means at most $\lfloor \frac{d}{2} \rfloor$ of them will be from corrupt parties, and therefore in total there will be at most d incorrect shares (including the ones that have not yet been received). Thus the RS decoding algorithm will always be able to detect that there are errors. To see that it will always terminate, we need only realise that as soon as we have $n - \lfloor \frac{d}{2} \rfloor$ correct values the decoding will succeed, and this will eventually happen because there are at least this many honest parties.

Remark 4. When $k = \frac{2n}{3}$, which occurs during multiplication of two k-sharings when $k = \frac{n}{3}$, we see that this places the security requirement at $t < \frac{n}{6}$ (but the protocol will still be safe with abort for up to $t < \frac{n}{3}$). We could achieve a more relaxed $t < \frac{n}{3}$ requirement if we augment the opening process with ZK proofs that the shares a player broadcasts are correct. In this case, as soon as k shares with valid proofs have been received, it is safe to use this subset to reconstruct the secret and proceed.

Theorem 7. *Let t be such that both $t \leq \frac{n-k+1}{2}$ and $t < k-1$. Then the protocol $\pi_{\mathtt{SO}}^{n,k}$ t-securely evaluates $f_{\mathtt{SO}}^{n,k}$.*

Remark 5. Note that we require $t < k - 1$ instead of the usual k. This is because when we open, every party learns the secret, which we recall is the share corresponding to index $0 \in \mathbb{F}$. With this extra share, the adversary can actually reconstruct all of the original shares (if we have the worst case $t = k - 1$), which we want to avoid.

Proof. Let an adversary $\mathcal{A}$ be given. Let input x and auxiliary input z be given. We will outline briefly our proof strategy, as we will also use it in other proofs. By definition, our end goal is to construct $\mathcal{S}$ such that

$$\mathtt{EXEC}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{A}}(x,z) \stackrel{d}{=} \mathtt{IDEAL}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{S}}(x,z)$$

We could try to compare these random variables directly, but in general this is a little cumbersome and complicated, as many elements of these random variables will have dependencies on each other. In addition, the view for $\mathcal{S}$ will have to be constructed by reverse-engineering based on the output of the trusted party, which makes the equality of the distributions less clear. To try to make things easier, we will identify the key source of randomness that determines these views, and a deterministic function that maps this to the view. For example, if a view contains all of the shares of some k-sharing, instead of considering the shares

directly, we could use only the k coefficients that determine the sharing, and then we can map these deterministically to the shares. Specifically, we seek random variables X and Y (the simplified randomness) and a deterministic function h such that $h(X) \stackrel{d}{=} \texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$ and $h(Y) \stackrel{d}{=} \texttt{IDEAL}_{\pi_{\texttt{SD}}^{n,k},\mathcal{S}}(x,z)$. Then, we will show that $X \stackrel{d}{=} Y$, at which point we may apply Theorem 2 to arrive at our desired result.

In our current case, we begin by characterising $\texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$. The first messages that $\mathcal{A}$ receives are the t output messages $(z_i)_{i\in\mathcal{I}_c}$ from $\pi_{\texttt{RZG}}^{n,k}$. The next part of the view comes from the $n-t$ messages $(y_i)_{i\in\mathcal{I}_h}$ that the uncorrupted parties send in the broadcast round. Next, $\mathcal{A}$ sends its t messages $(m_i)_{i\in\mathcal{I}_c}$ in the broadcast round. The final part of $\texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$ is the outputs of the parties; this is t $\perp$s that are output by the corrupted parties and $n-t$ elements of $\mathbb{F}$ (that should all be the same and equal to the secret corresponding to the sharing x). We can thus write

$$\texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z) = \left((z_i)_{i\in\mathcal{I}_c}, (y_i)_{i\in\mathcal{I}_h}, (m_i)_{i\in\mathcal{I}_c}, s'\right),$$

where for simplicity we ignore the outputs of the corrupted parties and only include one field element s' to represent the output of the honest parties.

The "key randomness" for $\texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$ as discussed at the beginning of this proof is captured by the random variable

$$X = \left((c_i)_{i\in[k-1]_0}, (m_i)_{i\in\mathcal{I}_c}\right),$$

where $(c_i)_{i\in[k-1]_0}$ are independently and uniformly randomly distributed elements of $\mathbb{F}$, representing the coefficients for the sharing $(z_i)_{i\in\mathcal{I}}$ of zero, and $(m_i)_{i\in\mathcal{I}_c}$ are the messages that $\mathcal{A}$ would send in the broadcast round. We describe the latter more specifically. The messages that $\mathcal{A}$ will send in any given round depends on its random tape and all of the messages it has received up to that point. The random tape is picked uniformly randomly, so we need only consider the messages it has received up until the broadcast round; these are its shares of zero and the randomised shares sent by the honest parties. We will assume that the shares of zero come from a uniformly random sharing of zero, and that the shares received from the honest parties come from a uniformly random sharing of x, the secret corresponding to the input shares. If we let the coefficients that define the input sharing be $c_x \in \mathbb{F}^k$, then we can define our deterministic function h by the mapping

$$(c,m) \mapsto (\vartheta_k(\mathcal{I}_c, c), \vartheta_k(\mathcal{I}_h, c+c_x), m, s),$$

where we interpret $c + c_x$ to be element-wise addition.

We want to show that $h(X) \stackrel{d}{=} \texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$. As previously mentioned, the third element (the messages sent by $\mathcal{A}$) depend only on the previous two elements, as these are the messages received up until the broadcast round. This means that if we ensure that these first two elements have the correct distribution,

the third will too. Now, notice that the final element of $\texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$ will be fixed and equal to s. This is because by definition and the assumptions on t the honest parties will hold at least $\frac{n+k-1}{2}$ correct shares and so can always correctly reconstruct s. Since this final element is fixed for the given inputs, and in h is defined to be that same fixed value s, the final element will always be correct. Finally, we notice that in fact the first element is a deterministic function of the second element. This is because since $|\mathcal{I}_h| \geq k$ we know that $(y_i)_{i\in\mathcal{I}_h}$ completely determines the rest of the shares $(y_i)_{i\in\mathcal{I}_c}$ and the random shares of zero satisfy $z_i = y_i - x_i$ for all $i \in \mathcal{I}_c$, where $x = (x_i)_{i\in\mathcal{I}}$. Thus we consider only the second element. In $\texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$, this element is distributed as a part of a uniformly random sharing of s. This follows from the fact that the output shares of $\pi_{\texttt{RZG}}^{n,k}$ will be a uniformly random sharing of zero, and Corollary 1. In $h(X)$, this element will also have this distribution for exactly the same reason; c is distributed in the same way as the coefficients of a uniformly random sharing of zero. We therefore conclude that $h(X) \stackrel{d}{=} \texttt{EXEC}_{\pi_{\texttt{SD}}^{n,k},\mathcal{A}}(x,z)$.

With this result established, we now seek to construct $\mathcal{S}$ and a random variable Y such that $h(Y) \stackrel{d}{=} \texttt{IDEAL}_{\pi_{\texttt{SD}}^{n,k},\mathcal{S}}(x,z)$ and $X \stackrel{d}{=} Y$. Define the simulator adversary $\mathcal{S}$ as follows.

1. Let the output from the trusted party be s (by definition it will always be the secret corresponding to the sharing x). $\mathcal{S}$ begins by constructing a random sharing $(y_i)_{i\in\mathcal{I}}$ of s.
2. $\mathcal{S}$ now defines
$$z_i = y_i - x_i \quad \forall i \in \mathcal{I}_c.$$
3. $\mathcal{S}$ now runs $\mathcal{A}$ after giving it $(z_i)_{i\in\mathcal{I}_h}$ and $(y_i)_{i\in\mathcal{I}_h}$ to get the messages $(m_i)_{i\in\mathcal{I}_c}$ that $\mathcal{A}$ sends in the broadcast round.
4. $\mathcal{S}$ then outputs
$$(z_i)_{i\in\mathcal{I}_c}, (m_i)_{i\in\mathcal{I}_c}, (y_i)_{i\in\mathcal{I}_h}$$
as the generated view.

We now aim to construct Y. Let $(y_i)_{i\in\mathcal{I}}$ and $(m_i)_{i\in\mathcal{I}_c}$ be distributed as in the description of $\mathcal{S}$. Let the coefficients corresponding to this sharing be $c_y = (c_i^{(y)})_{i\in[k-1]_0}$ and let $c_x = (c_i^{(x)})_{i\in[k-1]_0}$, and define the coefficients $c = (c_i)_{i\in[k-1]_0}$ by
$$c_i = c_i^{(y)} - c_i^{(x)} \quad \forall i \in [k-1]_0 \,. \tag{3}$$
We then define
$$Y = \left((c_i)_{i\in[k-1]_0}, (m_i)_{i\in\mathcal{I}_c}\right).$$

With Y defined, our next step is to show that $h(Y) \stackrel{d}{=} \texttt{IDEAL}_{\pi^{n,k}_{\texttt{SO}},\mathcal{S}}(x,z)$. The same argument as before shows that to do this we need only consider the second element of these distributions. For this element, the honest shares $(y_i)_{i\in\mathcal{I}_c}$, these clearly have the same distribution since $c + c_x = c_y$ by definition and $\vartheta_k(\mathcal{I}_h, c_y)$ has the same distribution as the corresponding element in $\texttt{IDEAL}_{\pi^{n,k}_{\texttt{SO}},\mathcal{S}}(x,z)$ also by definition. Thus $h(Y) \stackrel{d}{=} \texttt{IDEAL}_{\pi^{n,k}_{\texttt{SO}},\mathcal{S}}(x,z)$.

The final step for our proof is to show that $X \stackrel{d}{=} Y$. The first element in X is distributed as $k-1$ independently and uniformly randomly distributed elements of $\mathbb{F}$ (the coefficient c_0 is $0 \in \mathbb{F}$ as it is a sharing of zero). In Y the distribution is the same, which follows from Eq. (3), Corollary 1 and the fact that each $c_i^{(y)}$ is independently and uniformly distributed for each $i \in [k-1]$, and also the fact that $c_0^{(y)} = c_0^{(x)}$. Finally, the second elements have the same distribution due to their construction and the fact that they otherwise only depend on the first element. □

5.1.1 Directed Open

The protocols $\pi^{n,k}_{\texttt{OPEN}}$ and $\pi^{n,k}_{\texttt{SO}}$ reveal the shared secret to all players. However, sometimes we will only want to reveal the secret to a specific player. In this case we use a *directed open* protocol. The only difference in this case is that instead of sending the shares to everyone, each player sends their share to the specified player only. We denote the directed version of $\pi^{n,k}_{\texttt{OPEN}}$ as $\pi^{n,k,i}_{\texttt{DO}}$ and the directed version of $\pi^{n,k}_{\texttt{SO}}$ as $\pi^{n,k,i}_{\texttt{SDO}}$, where in each case $i \in \mathcal{I}$ is the index of the player which outputs the secret. The ideal functionalities $f^{n,k,i}_{\texttt{DO}}$ and $f^{n,k,i}_{\texttt{SDO}}$ are also defined correspondingly.

The security of these directed open protocols is summarised in the following two theorems; they enjoy the same security as their undirected counterparts. The proofs are almost identical and so are omitted here.

Theorem 8. *Let $t \leq \frac{n-k+1}{2}$. Then the protocol $\pi^{n,k}_{\texttt{DO}}$ t-securely evaluates $f^{n,k}_{\texttt{DO}}$.*

Theorem 9. *Let t be such that both $t \leq \frac{n-k+1}{2}$ and $t < k-1$. Then the protocol $\pi^{n,k}_{\texttt{SDO}}$ t-securely evaluates $f^{n,k}_{\texttt{SDO}}$.*

5.2 Random Number Generation

Generating shares of a uniformly distributed (but unknown to all parties) random number is key for many SMPC protocols. In this section we will describe how we achieve this and prove the security of our protocol. Security requires that the output cannot be influenced, and to construct a protocol that meets this requirement we will first create a protocol that does not meet all of the requirements for our security framework. We will then utilise this in creating a fully secure protocol.

5.2.1 Biased RNG

In biased RNG, the adversary has some influence over what shares it outputs after a run of the protocol. Importantly however, the secret will still be uniformly distributed; it's just the sharing that has been biased. This arises primarily because the adversary is rushing and so gets to see the shares that the other parties are going to contribute first before constructing its own. The common solution to this problem is to have a first round in which all players commit to their sharing, and then in the next round everyone decommits to their shares. The reason that we do not use this strategy comes back to synchronicity assumptions. After a party receives the commitment to a share, in the decommitment round it will either need to wait for the decommitment message, or have some sort of timeout. Further, there will need to be additional communication because the parties will need to need to agree on which parties published correct decommitments. As usual, we try to avoid this situation, and instead aim for a protocol that can move on to the next step before receiving (or deciding it didn't receive) a message from every party.

As previously stated, our first biased RNG protocol does not meet all of the security requirements for the desired RNG ideal functionality (outputting a random sharing of a random value). One way to overcome this could be to modify the ideal functionality so that it weakened to the point that our biased protocol securely evaluates it. However, in this case it is not very clear how to easily do this, so we take an alternative approach in which we will instead define the key properties that our biased RNG protocol needs to satisfy and prove that these hold for our designed protocol. This means that when we seek to use this protocol as a subprotocol, we will need to use it explicitly instead of being able to substitute it with a call to a trusted party. When analysing the protocol, we continue to use the execution model as described in the composable security framework; we operate in the presence of a t-limited, active and rushing adversary $\mathcal{A}$ and follow the protocol by proceeding in rounds. We are interested in the following properties.

- *Agreement*: All honest parties output consistent shares of a field element.
- *Global randomness*: The secret corresponding to the output shares is uniformly randomly distributed.
- *Secrecy*: No subset of less than k players knows anything about the shared secret other than the fact that it is uniformly randomly distributed. Further, the shares of the honest parties are uniformly random, constrained only by the fact that they are consistent with the shares of the corrupted parties.

The basic idea for biased RNG, denoted $\pi^{n,k}_{\texttt{BRNG}}$, is that each party generates its own random number r_i, some subset of which will be summed to give the global random number r. To get shares of r without revealing it, each party keeps r_i secret but shares it and gives the shares to the other players. Then, a subset of parties is chosen and the corresponding shares of their random numbers

can be summed locally. To choose this subset, we invoke the consensus protocol $\rho^{n,k}_{\texttt{RNG}}$. The security of the global random number comes from the fact that if we have t corrupted parties, we make sure to sum at least $t+1$ of the random numbers generated by the parties. This ensures that at least one uniformly random number was used in the sum, which means that the global random number itself will be uniformly randomly distributed as well. The protocol $\pi^{n,k}_{\texttt{BRNG}}$ proceeds as follows:

1. For all $i \in \mathcal{I}$, player P_i picks $r_i \in \mathbb{F}$ uniformly randomly and creates shares $(r_{i,j})_{j\in\mathcal{I}}$. Denote the coefficients of the random polynomial that determines this sharing by $c_0^{(i)}, \ldots, c_{t-1}^{(i)}$, so that $r_i = c_0^{(i)}$ and
$$r_{i,j} = \sum_{k=0}^{t-1} c_k^{(i)} j^k.$$
2. Each player P_i then participates in $\rho^{n,k}_{\texttt{RNG}}$ using the generated coefficients, obtaining the output $(r_{j,i})_{j\in I}$ where $I \subset \mathcal{I}$ is defined implicitly by $\rho^{n,k}_{\texttt{RNG}}$ as the set of players whose shares were agreed to be used by $\rho^{n,k}_{\texttt{RNG}}$.
3. The final output of player P_i is
$$y_i = \sum_{j\in I} r_{j,i}.$$

Remark 6. For $\pi^{n,k}_{\texttt{BRNG}}$ the synchronicity requirements are also weakened as discussed in Remark 3. The case is clearer here though; the only non-local part of the protocol is a call to $\rho^{n,k}_{\texttt{RNG}}$, which by definition has a desirable termination property.

Theorem 10. *Let $\mathcal{A}$ be a t-limited adversary where $t < k$. Then protocol $\pi^{n,k}_{\texttt{BRNG}}$ satisfies agreement, global randomness and secrecy.*

Proof. We will prove each property in turn.

- *Agreement*: The fact that all honest parties will agree on the same subset $I \subset \mathcal{I}$ of parties whose shares are to be included in the final sum is guaranteed by the termination and agreement properties of the consensus protocol. The consensus protocol also ensures (by dint of the validity property) that for each $i \in I$ all honest parties will receive consistent shares of a field element chosen by P_i. It follows that when each honest party obtains its output by locally summing each of the shares that came from parties in I, this sum will also be part of a consistent sharing of the sum of the secret random numbers.
- *Global randomness*: Since $t < k$ and k sharings are used in the final sum, at least one of the secrets used in the sum will have been generated by an honest party and hence be uniformly random. The property then follows from Corollary 1.

- *Secrecy*: We will begin by showing the second property required for secrecy: that the shares of the honest parties appear uniformly random, given the constraints. The first property will follow as a consequence. Let $y_c = (y_i)_{i \in \mathcal{I}_c}$ and $y_h = (y_i)_{i \in \mathcal{I}_h}$, and let $x = (x_i)_{i \in \mathcal{I}_h} \in S_{y_c, \mathcal{I}, \mathcal{I}_c}$ be arbitrary. Let $m \in I$ be such that P_m is honest. This is always possible because $|I| > t$. Let $K \subset \mathcal{I}_h$ be such that $|K| = k - t$. We will need to use the fact that $(r_{m,i})_{i \in K}$ is independent from $(r_{j,i})_{i \in K}$ for all $j \in I \setminus \{m\}$. To see this, realise that $(r_{m,i})_{i \in \mathcal{I}_c \cup K}$ is uniformly randomly distributed in $\mathbb{F}^k$, which follows from Theorem 3. Then the independence follows from this and the fact that the adversary chooses its shares knowing only the subset $(r_{m,i})_{i \in \mathcal{I}_c}$ of the shares from P_m.

  Now, from Theorem 4 we know that $|S_{y_c, \mathcal{I}, \mathcal{I}_c}| = |\mathbb{F}|^{k-t}$. Thus, we want to show that

  $$\mathbb{P}(x \mid y_c) = |\mathbb{F}|^{t-k}.$$

  But again, we know that x will be completely determined by y_c and any $k - t$ of the shares in x, so we can write

  $$\mathbb{P}(x \mid y_c) = \mathbb{P}((x_i)_{i \in K} \mid y_c). \tag{4}$$

  Since $(r_{m,i})_{i \in K}$ is independent from $(r_{j,i})_{i \in K}$ for all $j \in I \setminus \{m\}$, it follows that we can apply Corollary 1 to see that $(y_i)_{i \in K}$ is uniformly distributed in $\mathbb{F}^{k-t}$. This fact combined with Eq. (4) gives the desired result.

  The first property required for secrecy now follows easily: we saw above that $(y_i)_{i \in K}$ is uniformly distributed, and hence for any particular $i \in K$ it follows that y_i is uniformly distributed. However, we would theoretically have $0 \in K$, in which case y_0 would be the secret for the sharing, and hence this too is uniformly random conditioned on y_c.

□

5.2.2 Unbiased RNG

Unbiased random number generation is a protocol $\pi^{n,k}_{\texttt{RNG}}$ that generates a global random number that is unknown to all parties, but for which the parties hold shares. It can be described by the n-party function $f^{n,k}_{\texttt{RNG}}$ which takes no input from the parties, and gives output that is n shares of a random number r (unknown to all parties) with reconstruction threshold k, giving each share to the corresponding party.

To create an unbiased RNG protocol from a biased one, the idea is simple. Run the biased protocol k times, so that each of the random secrets represents a coefficient of a uniformly random polynomial. Then, compute each parties' share of this polynomial by operating on the shares, and open these resulting shares of shares to each corresponding party using a directed open. Note that creating the shares of shares is a linear combination of the coefficients, and so can be computed on the shares locally. The protocol is as follows.

1. The players invoke $\pi^{n,k}_{\texttt{BRNG}}$ k times. Let the k output sharings be denoted by $\left(r_i^{(1)}\right)_{i\in\mathcal{I}}, \ldots, \left(r_i^{(k)}\right)_{i\in\mathcal{I}}$, where player P_i receives the shares $r_i^{(j)}$ for all $1 \leq j \leq k$. These represent shares of the coefficients $c_0, \ldots, c_{k-1}$ of a random degree $k-1$ polynomial.

2. Each party P_i locally computes

$$r_{i,j} = \sum_{l=0}^{k-1} r_i^{(l)} j^l$$

for all $j \in \mathcal{I}$. This defines the sharings $(r_{i,j})_{i\in\mathcal{I}}$ for each $j \in \mathcal{I}$ which are shares of r_j where

$$r_j = \sum_{l=0}^{k-1} c_l j^l.$$

That is, r_j is the share for party P_j corresponding to the polynomial with coefficients $c_0, \ldots, c_{k-1}$.

3. The players invoke directed open n times: for each $j \in \mathcal{I}$, r_j is opened towards party P_j. Each party P_j then finishes by outputting r_j.

Theorem 11. *Let $k > t$. Then the protocol $\pi^{n,k}_{\texttt{RNG}}$ t-securely evaluates $f^{n,k}_{\texttt{RNG}}$.*

Proof. Let adversary $\mathcal{A}$, input x and auxiliary input z be given.

We will characterise $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z)$. Define m to be the messages that were sent and received during step 1 (except for the output shares of each invocation of $\pi^{n,k}_{\texttt{BRNG}}$, these will be labelled separately) and also define m_o to be the messages sent by $\mathcal{A}$ for each of the invocations of $\pi^{n,k,j}_{\texttt{DO}}$ in step 3. It follows that $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z)$ is equal to

$$m, \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_c}\right)_{j\in[k]}, m_o, \left(r_i, (r_{j,i})_{j\in\mathcal{I}}\right)_{i\in\mathcal{I}_c}, (y_i)_{i\in[n]},$$

where y_i is defined as in $\pi^{n,k}_{\texttt{RNG}}$ if $i \in \mathcal{I}_h$, and $\perp$ otherwise. Note that this is independent of the input x to $\pi^{n,k}_{\texttt{RNG}}$ as any such input is ignored by the protocol (and also by $f^{n,k}_{\texttt{RNG}}$). Denote the set in which this lives by F; i.e. $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z) \in F$.

We now seek to define a random variable X taking values in some set E and a function $h : E \to F$ in order to eventually apply Theorem 2. To do this, first define the random variable X as

$$X = \left(m, \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}}\right)_{j\in[k-1]_0}, m_o\right),$$

which is produced by $\mathcal{A}$ interacting with the trusted party for $\pi^{n,k}_{\texttt{BRNG}}$, and where the random tape for $\mathcal{A}$ and the trusted party is uniformly randomly chosen. Define the set E implicitly as the set that X takes values in. Now we can define our function h as

$$\begin{aligned} h : E &\to F \\ (m, r, m_o) &\mapsto \left(m, \nu(r), m_o, \left(\vartheta_k(\{i\}, \mu(r)), (\varphi(r, j, i))_{j\in\mathcal{I}}\right)_{i\in\mathcal{I}_c}, (\lambda(i, \mu(r)))_{i\in\mathcal{I}}\right), \end{aligned}$$

where we have the following definitions:

- ν simply projects r, which includes shares for all parties, to just those shares that the corrupted parties receive; i.e.

$$\nu\left(\left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}}\right)_{j\in[k-1]_0}\right) = \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_c}\right)_{j\in[k-1]_0}.$$

- μ maps the set of shares

$$\left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}}\right)_{j\in[k-1]_0}$$

  to their corresponding (uniquely defined) secrets for each $j \in [k-1]_0$; the output is $(c_i)_{i\in[k-1]_0}$. In the protocol these are the coefficients corresponding to the final random sharing.

- φ converts the shares of the coefficients into shares of the shares, as in step 2 of the protocol. It is defined as

$$\begin{aligned} \varphi : \mathbb{F}^{nk} \times \mathbb{F}^2 &\to \mathbb{F}^n \\ \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}}\right)_{j\in[k-1]_0}, (i, j) &\mapsto \sum_{l=0}^{k-1} r_i^{(l)} j^l \end{aligned}$$

- λ is defined by

$$\begin{aligned} \lambda : \mathbb{F} \times \mathbb{F}^k &\to \mathbb{F} \cup \{\perp\} \\ \left(i, (c_j)_{j\in[k-1]_0}\right) &\mapsto \begin{cases} \perp & i \in \mathcal{I}_c \\ \sum_{j=0}^{k-1} c_j i^j & i \notin \mathcal{I}_c \end{cases}, \end{aligned}$$

Now, we claim that $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x, z) \overset{d}{=} h(X)$. This is immediate for the first three elements, as they are generated in X exactly as they are in $\pi^{n,k}_{\texttt{RNG}}$ and using the same distributions. As for the second last element, which represents the messages during the invocations of $\pi^{n,k,j}_{\texttt{DO}}$, this is correct due to how it is defined from r in $h(X)$ and the correctness property that the output of $\pi^{n,k}_{\texttt{DO}}$ enjoys from

its proof of security. Similarly, the fact that the last element has the correct distribution also follows from its definition in $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z)$ and in $h(X)$.

Now that we have characterised $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z)$, we turn our attention to the simulated view. The idea behind the simulator is as follows. The simulator can run the protocol as usual up until step 3. At this point, $\mathcal{A}$ does not know what the secrets for the random sharings are, and the honest parties shares will be uniformly random conditioned on the fact that they are consistent with $\mathcal{A}$'s shares. This allows the simulator to simply extend the corrupted parties' shares (which are known by the simulator since it holds enough shares to reconstruct them) to valid random sharings of the target output shares, and this will have the correct distribution.

The simulator is defined as follows.

1. The trusted party is invoked first. Let the output shares of the trusted party be $(y_i)_{i\in\mathcal{I}}$.
2. $\mathcal{S}$ runs $\mathcal{A}$ and acts on behalf of the honest parties for Step 1. In doing so, $\mathcal{S}$ learns the biased shares $\left(r_i^{(j)}\right)_{j\in[k-1]_0}$ for each $i \in \mathcal{I}_c$, i.e. for each corrupted party. Label the messages sent and received by $\mathcal{A}$ during the invocations of $\pi^{n,k}_{\texttt{BRNG}}$ as m, and the messages that $\mathcal{A}$ would send to the invocations of $\pi^{n,k,j}_{\texttt{DO}}$ as m_o.
3. $\mathcal{S}$ constructs the honest parties' shares $\left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_h}\right)_{j\in[k-1]_0}$ as follows. First, $\mathcal{S}$ picks random coefficients $(c_i)_{i\in[k-1]_0}$ that are consistent with the target shares $(y_i)_{i\in\mathcal{I}_c}$, i.e. $\mathcal{S}$ picks uniformly randomly from the set
$$\left\{ (c_i)_{i\in[k-1]_0} \in \mathbb{F}^k \middle| y_i = \sum_{j=0}^{k-1} c_j i^j \ \forall i \in \mathcal{I}_c \right\}.$$
Next, the honest parties' shares are chosen at random under the condition that $\left(r_i^{(j)}\right)_{i\in\mathcal{I}}$ is a consistent sharing of c_j for each $j \in [k-1]_0$. Precisely, the shares are chosen uniformly randomly from the set
$$\left\{ \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_h}\right)_{j\in[k-1]_0} \in \mathbb{F}^{k(n-t)} \middle| \begin{array}{l} \forall j \in [k-1]_0 \\ \exists (c'_l)_{l\in[k-1]} \\ \forall i \in \mathcal{I} \end{array} : r_i^{(j)} = c_j + \sum_{l=1}^{k-1} c'_l i^l \right\}.$$
4. Let the random variable Y be defined as
$$Y = \left(m, \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}}\right)_{j\in[k-1]_0}, m_o \right).$$
$\mathcal{S}$ finishes by constructing its output by computing $h(Y)$ and discarding the last element, which corresponds to the output of the protocol; recall that this is not part of the simulator's output as instead this is defined by the output of the trusted party.

Now we need to consider $\texttt{IDEAL}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{S}}(x,z)$. We want to show that $h(Y) \stackrel{d}{=} \texttt{IDEAL}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{S}}(x,z)$. This is clearly true for all but the last element because of how $\texttt{IDEAL}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{S}}(x,z)$ was constructed. But the last element will also have the correct distribution with regard to the others due to its construction, as Y was reverse engineered to be consistent with the given output of the trusted party.

Finally, we want to show that $X \stackrel{d}{=} Y$, as then by Theorem 2 we will have $h(X) \stackrel{d}{=} h(Y)$, which gives the last equality needed to show that

$$\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z) \stackrel{d}{=} \texttt{IDEAL}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{S}}(x,z)$$

which will complete the proof. By construction, this is clearly true for all but the honest party shares

$$\left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_h}\right)_{j\in[k-1]_0},$$

so we need only focus our attention on these. In the case of X, we know from the secrecy property of the output shares of $\pi^{n,k}_{\texttt{BRNG}}$ that

$$r_h^{(j)} = \left(r_i^{(j)}\right)_{i\in\mathcal{I}_h}$$

is independently and uniformly distributed in $S_{r_c^{(j)},\mathcal{I},\mathcal{I}_c}$ for each $j \in [k-1]_0$, where

$$r_c^{(j)} = \left(r_i^{(j)}\right)_{i\in\mathcal{I}_c}.$$

The result for Y follows from the proceeding claims, the proofs of which will conclude the proof of security.

Claim 2. *The coefficients $c = c_0, \ldots, c_{k-1}$ as defined in the simulation are independently and uniformly randomly distributed.*

Proof. Since the coefficients of a $k-1$ degree polynomial are completely determined by k points on that polynomial, we can see that for a given set of shares $y = (y_i)_{i\in\mathcal{I}_c}$ (which is t points), we could choose any set of $k-t$ points to determine the coefficients. Thus for a given y there are $|\mathbb{F}|^{k-t}$ possible choices for c, each one being picked with equal probability. Now, consider the probability of obtaining some given c, using the fact that y is chosen uniformly randomly. For this given c, there is a unique y that is consistent with c, and this y is chosen with probability $|\mathbb{F}|^{-t}$. But we not only require that the uniquely determined y was chosen, but also the unique c for the possibilities determined by y, for which we have just seen there exists $|\mathbb{F}|^{k-t}$ choices, each chosen with equal probability. Putting these two results together, we find that the probability for a given c is $|\mathbb{F}|^{-k}$. Since $c \in \mathbb{F}^k$, the result follows. □

Claim 3. *For each $j \in [k-1]_0$, the shares $r_h^{(j)}$ as defined in the simulation are independently and uniformly randomly distributed in $S_{r_c^{(j)},\mathcal{I},\mathcal{I}_c}$.*

Proof. The independence of each $r_h^{(j)}$ follows easily from the independence of each c_j. Next, notice that for a given c_j, $r_h^{(j)} \in A_{c_j}$ where for each $c \in \mathbb{F}$ we have

$$A_c = \left\{ r_h^{(j)} \in \mathbb{F}^{n-t} \middle| \exists (c'_i)_{i\in[k-1]} : \forall i \in \mathcal{I}_c, r_i = c + \sum_{l=1}^{k-1} c'_l i^l \right\}.$$

Since a polynomial with degree k is completely determined by k points, so too are its coefficients, and hence each distinct element (which is a set of $n - t > k$ shares) in A_c corresponds also to a distinct set of coefficients $c, c'_1, \ldots, c'_{k-1}$. From this it follows that for $c \neq d$ we have that A_c and A_d are disjoint sets. Now, since $r_h^{(j)}$ is chosen uniformly randomly from A_{c_j}, and c_j is uniformly random, it follows that $r_h^{(j)}$ is chosen uniformly randomly from the set

$$\bigcup_{c_j \in \mathbb{F}} A_{c_j}.$$

But it is easy to see that this set is equal to $S_{r_c^{(j)},\mathcal{I},\mathcal{I}_c}$. □

This completes the proof. □

5.3 Random Zero Generation

Random zero generation is almost the same as random number generation as in Section 5.2, except instead of obtaining shares of a global random number, the parties obtain a sharing of $0 \in \mathbb{F}$. Specifically, the ideal functionality $f_{\mathtt{RZG}}^{n,k}$ takes no inputs and outputs to each party P_i a share z_i of $0 \in \mathbb{F}$, such that this sharing is uniformly randomly distributed. The protocol is also almost the same as $\pi_{\mathtt{RNG}}^{n,k}$, except that instead of generating k random shared coefficients, we only generate $k-1$ because our secret (constant term in the polynomial) is fixed and not random. This is made precise in the following protocol $\pi_{\mathtt{RZG}}^{n,k}$:

1. The players invoke $\pi_{\mathtt{BRNG}}^{n,k}$ with no input $k-1$ times. Let the $k-1$ output sharings be denoted by $\left(r_i^{(1)}\right)_{i\in\mathcal{I}}, \ldots, \left(r_i^{(k-1)}\right)_{i\in\mathcal{I}}$, where player P_i receives the shares $r_i^{(j)}$ for all $1 \leq j \leq k-1$. These represent shares of the coefficients $c_1, \ldots, c_{k-1}$ of a random degree $k-1$ polynomial with a zero constant term.

2. Each party P_i locally computes

   $$r_{i,j} = \sum_{l=1}^{k-1} r_i^{(l)} j^l$$

   for all $j \in \mathcal{I}$. This defines the sharings $(r_{i,j})_{i\in\mathcal{I}}$ for each $j \in \mathcal{I}$ which are shares of r_j where

   $$r_j = \sum_{l=1}^{k-1} c_l j^l.$$

That is, r_j is the share for party P_j corresponding to the polynomial with coefficients $c_1, \ldots, c_{k-1}$.

3. The players invoke directed open n times: for each $j \in \mathcal{I}$, r_j is opened towards party P_j. Each party P_j then finishes by outputting r_j.

The security theorem and proof for $\pi^{n,k}_{\texttt{RZG}}$ is nearly identical to that for $\pi^{n,k}_{\texttt{RNG}}$, and so is omitted.

Theorem 12. *Let $t < k - 1$. Then the protocol $\pi^{n,k}_{\texttt{RZG}}$ t-securely evaluates $f^{n,k}_{\texttt{RZG}}$.*

Remark 7. Note that the requirement on t is that it is less than $k-1$, as opposed to k as was the case for $\pi^{n,k}_{\texttt{RNG}}$. This is because we are generating a sharing of a known, fixed field element which gives one extra share of information to the adversary (recall that the secret of a sharing is equal to the share corresponding to index $0 \in \mathbb{F}$).

5.4 Random Keypair Generation

Random key pair generation is again very similar to random number generation defined in Section 5.2, except instead of only obtaining shares of a global random number, the parties also obtain the public key that corresponds to the shared random number (private key). Specifically, the ideal functionality $f^{n,k}_{\texttt{RKPG}}$ takes no inputs and outputs to each party P_i a share x_i of some uniformly random $x \in \mathbb{F}$ and y such that $y = g^x \in \mathbb{G}$ where $\mathbb{G}$ is a group with generator g. Additionally, to enable the proof of security to work each party will also output g^{x_i} for each $i \in \mathcal{I}$. This modification will be discussed after presenting the passively secure protocol $\pi^{n,k}_{\texttt{RKPG}}{}'$ for $f^{n,k}_{\texttt{RKPG}}$. We begin with this passively secure protocol and then discuss how it can be augmented to achieve active security later. The protocol is defined as follows.

1. The parties invoke $\pi^{n,k}_{\texttt{RNG}}$, so that party P_i receives the share x_i.
2. Each party P_i sends g^{x_i} to every other party.
3. Each party then reconstructs g^x "in the exponent", along with g^{x_i} for each $i \in \mathcal{I}$. These, along with the share x_i constitute the output of the protocol for player P_i.

The reason that the public values g^{x_i} for each $i \in \mathcal{I}$ are included in the output for each party is to allow the simulator to construct a consistent view. If the simulator did not know these values, then under Assumption 1 it would have no hope of constructing $g_i \in \mathbb{G}$ such that $g_i = g^{x_i}$ for each $i \in \mathcal{I}_h$. However, for the same reason the discrete logarithm problem also implies that learning each g_i should not be a concern, and so we are happy to include them in the output of the protocol. We will formalise this reasoning somewhat after presenting the security theorem.

Theorem 13. *Let $k > t$. Then the protocol $\pi^{n,k\,\prime}_{\texttt{RKPG}}$ t-securely evaluates $f^{n,k}_{\texttt{RKPG}}$ in the presence of a passive adversary.*

Proof. The proof is straightforward; since everything in the view of the adversary is contained in the output of the trusted party, and since the adversary is passive, constructing a simulated view from this trusted output is trivial. □

We now argue that for a k-sharing of $x \in \mathbb{F}$, knowing a subset of less than k of these shares and g^{x_i} for all shares x_i, as well as knowing g^x, it is not possible to discover x if the discrete logarithm problem is hard. This is summarised in the following theorem.

Theorem 14. *Let $\mathbb{G}$ be a group of size at least 2^b with generator g and prime order, such that $\mathbb{F}$ is the finite field associated with this prime. Let $x \in \mathbb{F}$ be uniformly random, and $x_1, \ldots, x_n$ be a uniformly random k-sharing of x where $k, n \in \mathbb{N}$ and $k \leq n$. Let $t \in \mathbb{N}$ and $I \subset [n]$ be given such that $t < k$ and $|I| = t$. Suppose that a polynomial time Turing machine $\mathcal{D}$ takes as input $\mathbb{G}$, g, $(x_i)_{i \in I}$, $(g^{x_i})_{i \in [n]}$ and g^x, and outputs $z \in \mathbb{F}$. Then if Assumption 1 holds, it follows that for all such $\mathcal{D}$, $\mathbb{P}(z = x)$ is negligible in b.*

Proof. As is standard for these kinds of results, we will proceed by contradiction; assume that there exists $\mathcal{D}$ that outputs z such that $\mathbb{P}(z = x)$ is non-negligible in b. We will show how this can be used to solve a target instance of the discrete logarithm problem. To this end, let $y \in \mathbb{G}$ be arbitrary. Pick t independent and uniformly random elements of $\mathbb{F}$ $x_1, \ldots, x_t$, and $k - t - 1$ independent and uniformly random elements of $\mathbb{G}$, labelled $g_{t+1}, \ldots, g_{k-1}$. Similarly label $g_i = g^{x_i}$ for each $i \in [t]$. Also extend the labels of x_i so that for $i \in \{t+1, \ldots, k-1\}$ x_i is the unique value such that we have $g_i = g^{x_i}$ and also define x_0 to be the unique value that satisfies $y = g^{x_0}$. Note that these values are well defined but not known by any polynomial time Turing machine, they are merely defined for convenience of exposition. From Theorem 5 we know that there exist values $\lambda^{(i)}_0, \ldots, \lambda^{(i)}_{k-1}$ such that

$$c_i = \sum_{j=0}^{k-1} \lambda^{(i)}_j x_j \quad \forall i \in [k-1]_0 \,,$$

where $c_0, \ldots, c_{k-1}$ are the uniquely defined coefficients that correspond to a k-sharing that is consistent with the shares $x_0, \ldots, x_{k-1}$. Using these values we can compute

$$g^{c_i} = h_i = \prod_{j=0}^{k-1} {g_j}^{\lambda^{(i)}_j}$$

for all $i \in [k-1]_0$. This in turn allows us to compute the remaining shares in the exponents; we can compute

$$g_i = \prod_{j=0}^{k-1} {h_j}^{i^j} \quad \forall i \in \{k, \ldots, n\}.$$

By construction, the shares corresponding to $g_1, \ldots, g_n$ are a consistent sharing of x_0 and agree with the subset $x_1, \ldots, x_t$. We may now run $\mathcal{D}$ with input $\mathbb{G}$, g, $(x_i)_{i\in[t]}$, $(g_i)_{i\in[n]}$, and y. The output is z where $\mathbb{P}(z = x_0)$ with non-negligible probability. Since $y = g^{x_0}$, this completes the proof. □

We now discuss how to augment $\pi^{n,k\,\prime}_{\texttt{RKPG}}$ to be secure in the presence of an active adversary. This is not too difficult to achieve; the only part of the passively secure protocol in which the adversary can send (modified) messages is in step 2. This can hinder the reconstruction of the global public key and also the public keys corresponding to the shares if there is no way to detect that these values are incorrect. Thus, we need only include a way to ensure that honest parties can detect correct values, and after receiving k correct values they will be able to perform the reconstruction. To achieve this required detection, we will augment $\pi^{n,k}_{\texttt{RNG}}$ to output, along side the usual random shares, a perfectly hiding commitment to each of the shares of the parties. Then, $\pi^{n,k\,\prime}_{\texttt{RKPG}}$ is augmented by also submitting in step 2 a zero knowledge proof that has the perfect zero knowledge property that the value they sent corresponds to the commitment to their share. This will enable honest parties to detect which broadcasted values are correct. The associated simulator for the proof would then be able to work as previously, except to generate the ZK proof messages it can leverage the perfect zero knowledge property. The augmented protocol $\pi^{n,k}_{\texttt{RKPG}}$ is as follows.

1. The parties invoke the augmented RNG protocol $\pi^{n,k\prime}_{\texttt{RNG}}$, so that party P_i receives the share x_i. Every party also receives for each $i \in \mathcal{I}$ the commitment c_i to x_i.

2. Each party P_i sends g^{x_i} to every other party, along with a ZK proof z_i that attests to the fact that the discrete log of g^{x_i} is the same as the value committed to by c_i.

3. Each party then reconstructs g^x "in the exponent", along with g^{x_i} for each $i \in \mathcal{I}$. These, along with the share x_i constitute the output of the protocol for player P_i.

5.5 Local Arithmetic

We will now describe the fundamental computational primitives for the SMPC algorithm. This includes the standard operations that can be performed on elements of a field, i.e. the field operations and their inverses. For most of the operations, applying them locally to the shares (that is, each party applies them to their own shares, and on a collective level we can describe this as operating element-wise) results in the equivalent effect on the secret itself, with no effect to the threshold of the sharing. The one operation for which this is not the case is multiplication; here operating locally will indeed give a share of the product of the secrets, but the threshold will increase and the sharing will exhibit unwanted structure. This means some extra work will need to be done to achieve a desired multiplication protocol. For all of the other operations, the fact that they can

be carried out locally means that no messages are exchanged between parties and therefore a "protocol" that encapsulates one of these operations is trivially secure under the security definition, and hence the corresponding theorems will not be stated nor proven.

To simplify notation, write the sharing $(x_i)_{i\in\mathcal{I}}$ as $x_{\mathcal{I}}$. We will denote local operations on shares as follows, where on the left we have the notation and the right we have what it means:

$$\begin{aligned} c_{\mathcal{I}} &= a_{\mathcal{I}} + b_{\mathcal{I}} & c_{\mathcal{I}} &= (a_i + b_i)_{i\in\mathcal{I}} \\ c_{\mathcal{I}} &= a_{\mathcal{I}} - b_{\mathcal{I}} & c_{\mathcal{I}} &= (a_i - b_i)_{i\in\mathcal{I}} \\ c_{\mathcal{I}} &= a_{\mathcal{I}} b_{\mathcal{I}} & c_{\mathcal{I}} &= (a_i b_i)_{i\in\mathcal{I}} \\ c_{\mathcal{I}} &= -a_{\mathcal{I}} & c_{\mathcal{I}} &= (-a_i)_{i\in\mathcal{I}} \\ c_{\mathcal{I}} &= r a_{\mathcal{I}} & c_{\mathcal{I}} &= (r a_i)_{i\in\mathcal{I}} \end{aligned}$$

We will consider an example to make this clear. Consider addition of two shared values where each party P_i holds the shares a_i and b_i. Then P_i simply defines the share c_i to be $a_i + b_i$, and this will be its output of the addition protocol.

6 SMPC Protocols

In this section we will outline the protocols that are built from the primitives outlined in the preceding section. The main goal that we are working towards is to be able to perform an ECDSA signature where the private key is distributed among the parties as a shared secret. To do this, we will construct the SMPC protocols that will be sufficient to perform the signature. We will then be able to leverage the composability property of the security framework to be able to compose our building blocks together in a secure way. However, this security definition does not capture all of the security properties that we seek in general. For instance, it does not capture the security of our secret (in the context of Shamir Secret Sharing). We can see this by considering the following obviously insecure protocol for multiplying two secrets and then immediately opening the result: simply open the two sharings and then multiply them together locally. The reason that this continues to satisfy security as defined in Definition 14, despite being intuitively insecure, is that it is evaluating an ideal functionality that we do not actually want (the ideal functionality that we want simply takes the input shares and gives the product of the secrets). Rather, it evaluates precisely the secure functionality defined by this example protocol: take the input shares and give the product of the secrets *but also the input secrets themselves* (the latter because of the opens that are performed in the protocol description). Now, it becomes clear that the reason this example protocol is secure under Definition 14, but not intuitively, is because the actual protocols explicitly describes itself as revealing information. We can see that the problem in this case arises because of opening values that should not be revealed. The solution to this is to make sure that whenever we perform an open during our protocol, the revealed value should be uniformly randomly distributed, so that nothing

is "learned" from it. Because open is the only protocol that reveals information about secrets, as long as we make sure that we do this when constructing a protocol out of our primitive protocols, we can be sure that we are securely evaluating what we intend without worrying about exactly how we got there.

6.1 Multiply and Open

The ideal functionality $f_{\texttt{MO}}^{n,k}$ that represents the multiply and then open protocol takes as input two sets of shares for two field elements and gives as output to each party the product of these two secrets.

Let $a_{\mathcal{I}}$ and $b_{\mathcal{I}}$ be two k-sharings of respective field elements $a, b \in \mathbb{F}$. The protocol $\pi_{\texttt{MO}}^{n,k}(a_{\mathcal{I}}, b_{\mathcal{I}})$ is defined by

$$\begin{aligned} c_{\mathcal{I}} &= a_{\mathcal{I}} b_{\mathcal{I}} \\ c &\leftarrow \pi_{\texttt{SO}}^{n,2k}(c_{\mathcal{I}}) \end{aligned}$$

Theorem 15. *Let $n \in \mathbb{N}$ and $k \leq \frac{n}{2}$. Let t be such that $t \leq \frac{n-2k+1}{2}$ and $t < k$. Then the protocol $\pi_{\texttt{MO}}^{n,k}$ t-securely evaluates $f_{\texttt{MO}}^{n,k}$.*

We require that $k \leq \frac{n}{2}$ otherwise after the multiplication the threshold of the sharing would be too large for it to be possible to reconstruct the secret even with n parties. The first requirement on t comes from $\pi_{\texttt{SO}}^{n,2k}$, and the second requirement on t is the usual one to disallow the adversary to reconstruct secrets.

6.2 Multiplication

The ideal functionality $f_{\texttt{MUL}}^{n,k}$ that represents the multiplication protocol takes as input two sets of shares for two field elements and gives as output to each party a share of the product of these two secrets. This protocol, along with addition, allows for in theory arbitrary arithmetic circuits and hence arbitrary computations. However, for complicated computations it is usually more efficient to use other more tailored protocols and primitives, as we will see is the case for ECDSA. The multiplication here protocol is included mainly for completeness.

Let $a_{\mathcal{I}}$ and $b_{\mathcal{I}}$ be two k-sharings of respective field elements $a, b \in \mathbb{F}$. The protocol $\pi_{\texttt{MUL}}^{n,k}(a_{\mathcal{I}}, b_{\mathcal{I}})$ is defined by

$$\begin{aligned} r_{\mathcal{I}} &\leftarrow \pi_{\texttt{RNG}}^{n,k} \\ c_{\mathcal{I}} &= a_{\mathcal{I}} b_{\mathcal{I}} \\ c'_{\mathcal{I}} &= c_{\mathcal{I}} + r_{\mathcal{I}} \\ c' &\leftarrow \pi_{\texttt{SO}}^{n,2k}(c'_{\mathcal{I}}) \\ c_{\mathcal{I}} &= c' - r_{\mathcal{I}} \end{aligned}$$

Theorem 16. *Let $n \in \mathbb{N}$ and $k \leq \frac{n}{2}$. Let t be such that $t \leq \frac{n-2k+1}{2}$ and $t < k$. Then the protocol $\pi_{\texttt{MUL}}^{n,k}$ t-securely evaluates $f_{\texttt{MUL}}^{n,k}$.*

The restrictions on k and t are as for $\pi^{n,k}_{\texttt{MO}}$. The only value that we open is $c_{\mathcal{I}} + r_{\mathcal{I}}$, which is uniformly random and so preserves our desired security. Note that $r_{\mathcal{I}}$ has threshold k whereas $c_{\mathcal{I}}$ has threshold $2k$. This is not a problem as when we add two sharings of different thresholds, the resulting sharing has a threshold equal to the larger of the two. Additionally, we do not need to worry about any structure this may introduce in the shares, because the sharing is randomised by $\pi^{n,2k}_{\texttt{SO}}$.

6.3 Inversion

The ideal functionality $f^{n,k}_{\texttt{INV}}$ that represents the field inversion protocol takes as input a sets of shares of a field element and gives as output to each party shares of the multiplicative inverse of this field element.

Let $a_{\mathcal{I}}$ be a k-sharing of a field element $a \in \mathbb{F}$. The protocol $\pi^{n,k}_{\texttt{INV}}(a_{\mathcal{I}})$ is defined by

$$\begin{aligned} r_{\mathcal{I}} &\leftarrow \pi^{n,k}_{\texttt{RNG}} \\ t &\leftarrow \pi^{n,k}_{\texttt{MO}}(a_{\mathcal{I}}, r_{\mathcal{I}}) \\ b_{\mathcal{I}} &= t^{-1} r_{\mathcal{I}} \end{aligned}$$

Remark 8. Because the shares of the product are randomised by $\pi^{n,k}_{\texttt{MO}}$, it is conjectured that it is safe to define $r_{\mathcal{I}}$ using the less expensive biased RNG protocol $\pi^{n,k}_{\texttt{BRNG}}$.

Theorem 17. *Let $n \in \mathbb{N}$ and $k \leq \frac{n}{2}$. Let t be such that $t \leq \frac{n-2k+1}{2}$ and $t < k$. Let $x_{\mathcal{I}}$ be a k-sharing of some $x \in \mathbb{F} \setminus \{0\}$. Then the protocol $\pi^{n,k}_{\texttt{INV}}(x_{\mathcal{I}})$ t-securely evaluates $f^{n,k}_{\texttt{INV}}(x_{\mathcal{I}})$.*

The restrictions on k and t are due to $\pi^{n,k}_{\texttt{MO}}$. The only value that is opened is the product of the secret and a random value, and so is uniformly random. Note however that we need to require that the input shares are of a non-zero field element, otherwise the open will reveal this fact. Also note that it is possible that the random number is itself zero, and so if we were to be careful we would check to see if the value we open is zero, and if it is, abort this attempt and retry the protocol with a different random number. However, in our case we ignore this because we will use a field with approximately 2^{256} elements and so the probability that the random number is zero is negligible.

7 Signature Algorithm

The ideal functionality $f^{n,k}_{\texttt{SIGN}}$ that represents the ECDSA protocol takes as input a sets of shares of a field element (the private key) and a public field element (the message digest) and gives as output to each party the values r, s which constitute a valid ECDSA signature for the given private key and message.

Let $d_{\mathcal{I}}$ be a k-sharing of a field element $d \in \mathbb{F}$ that represents an ECDSA private key, and let $z \in \mathbb{F}$ be a message digest. Let the associated group in which the public keys live, $\mathbb{G}$, have prime order q. The protocol $\pi^{n,k}_{\texttt{SIGN}}(d_{\mathcal{I}}, z)$ is defined by

$$\begin{aligned}
(k_{\mathcal{I}}, p_b) &\leftarrow \pi^{n,k}_{\texttt{RKPG}} \\
(x, y) &= p_b \\
r &= x \mod q \\
k'_{\mathcal{I}} &\leftarrow \pi^{n,k}_{\texttt{INV}}(k_{\mathcal{I}}) \\
t_{\mathcal{I}} &= r d_{\mathcal{I}} \\
t_{\mathcal{I}} &= t_{\mathcal{I}} + z_{\mathcal{I}} \\
s &\leftarrow \pi^{n,k}_{\texttt{MO}}(t_{\mathcal{I}}, k'_{\mathcal{I}})
\end{aligned}$$

Theorem 18. *Let $n \in \mathbb{N}$ and $k \leq \frac{n}{2}$. Let t be such that $t \leq \frac{n-2k+1}{2}$ and $t < k$. Let $d_{\mathcal{I}}$ be a k-sharing of a ECDSA private key, and let $z \in \mathbb{F}$ be a message digest. Then the protocol $\pi^{n,k}_{\texttt{SIGN}}(d_{\mathcal{I}}, z)$ t-securely evaluates $f^{n,k}_{\texttt{SIGN}}(d_{\mathcal{I}}, z)$.*

The restrictions on k and t are inherited from the constituent protocols.

Remark 9. For the system of interest, we set $k = \frac{n}{3}$. In this case, the requirement on t is that $t \leq \frac{n}{6} + \frac{1}{2}$. Note however that from the discussion of $\pi^{n,k}_{\texttt{SO}}$ that we still maintain *safety with abort* for $t \leq \frac{n}{3}$.

References

[1] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[2] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *Journal of Cryptology* 13.1 (Jan. 2000), pp. 143–202. DOI: 10.1007/s001459910006.

[3] Oded Goldreich. *Foundations of Cryptography*. Vol. 1. Cambridge University Press, 2001. DOI: 10.1017/CBO9780511546891.

[4] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The latest gossip on BFT consensus*. 2018. arXiv: 1807.04938v3 [cs.DC].

[5] P. Feldman. "A practical scheme for non-interactive verifiable secret sharing". In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. Oct. 1987, pp. 427–438. DOI: 10.1109/SFCS.1987.4.

[6] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '91. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 129–140. ISBN: 3540551883.

[7] Lloyd R. Welch and Elwyn R. Berlekamp. *Error Correction for Algebraic Block Codes*. U.S. Patent 4,633,470. Dec. 1986.

[8] Shuhong Gao. "A New Algorithm for Decoding Reed-Solomon Codes". In: *Communications, Information and Network Security*. Ed. by Vijay K. Bhargava et al. Boston, MA: Springer US, 2003, pp. 55–68. ISBN: 978-1-4757-3789-9. DOI: 10.1007/978-1-4757-3789-9_5.