RenVM Secure Multiparty Computation

Ross Pure, Zian-Loong Wang

May 1st, 2020 DRAFT

Ren

Contents

1 Introduction

A threshold ECDSA signature scheme allow groups of n players to jointly produce an ECDSA signature without exposing the corresponding ECDSA private key, and no subset of less than $t+1$ players is able to produce a signature. Generally, such schemes also allow n players to jointly generate an ECDSA public/private key pair, without the need for a dealer, and where the public key is known by everyone.

Interest in threshold ECDSA schemes has increased recently, with the popularisation of blockchains (such as Bitcoin and Ethereum) that make use of ECDSA signatures to custody and spend cryptocurrencies. Using threshold ECDSA schemes, groups of n players can take joint custody of cryptocurrencies, and only spend these funds when at least $t+1$ players agree that this should be done.

The most recent threshold DSA scheme, which is general enough to support ECDSA specifically, was introduced by Gennaro et al. [1]. This scheme allows $n \geq t+1$ players to generate public/private keys and produce signatures. This work is primarily an extension of [2], improving its efficiency and reducing its complexity. Threshold DSA schemes were previously introduced by [3], and [4]. The main advantage of [2] and [1] over these schemes is threshold-optimality (the scheme is secure against t adversaries, and only requires participation from $t+1$ players).

In the context of large-scale globally distributed Byzantine networks, where participants are distributed all around the world, can be anyone, are constantly changing, and latency is large and highly variant, robustness against offline participants is critical to the practical usability of any protocol. In [1], the use of additive secret sharing, inspired by SPDZ [5], introduces moments in the protocol where failure to participate by one player causes the protocol to halt (and it needs to be repeated from a prior round). This might be usable for small groups, but it is impractical for large globally distributed groups where honest players are likely to experience unexpected downtime or network outages. In [2], the need for trusted setup of an additively homomorphic encryption scheme (the removal of which was the core improvement made by [1]) prevents it from being efficient with a large number of players in a high-latency network.

In [3], a scheme is presented that is robust against malicious adversaries, but makes extensive use of secure broadcasts and rounds of complaining against misbehaving players. This introduces extra rounds of communication, and more constraints on network timing, which make it inappropriate for large-scale globally distributed networks. On the other hand [4] requires $O(t)$ rounds of communication using Paillier's encryption with a modulus $N = O(q^{3t-1})$, making it unsuitable for a large number of participants.

In this paper, we present a threshold ECDSA scheme, based on [3], that is practical for use in large-scale globally distributed networks. It is robustly secure against t malicious adversaries, assuming $n = 3t+1$ players, and requires fewer rounds than [3]. During both ECDSA key generation and signing, up to t players can go offline at the beginning, middle, or end of a round, and the protocols will complete successfully without the need to go back repeat from a prior round.

1.1 Contributions

Our threshold ECDSA scheme is based on [3] but replaces the protocols that generate the random sharings (random number generation, random zero generation, and random key-pair generation) with ones that don't require secure broadcasts and can be proven in Canetti's security framework [6]. Briefly, these new protocols are as follows.

- (Biased RNG $\pi^{n,k}_{\mathtt{BRNG}}$): This is similar to the RNG protocol in [7], but we call it biased because of the ability of a rushing adversary to influence the sharing (but not the underlying random number). The exact protocol is similar to the one in [3] but secure broadcasting is replaced with a Byzantine fault tolerant consensus protocol.
- (RNG $\pi^{n,k}_{\mathtt{RNG}}$): This is the unbiased version of $\pi^{n,k}_{\mathtt{BRNG}}$. It is built using $\pi^{n,k}_{\mathtt{BRNG}}$. Because the shares of honest players cannot be biased, we are able to prove its security in Canetti's framework.
- (Random zero generation $\pi^{n,k}_{\mathtt{RZG}}$): This is the same as $\pi^{n,k}_{\mathtt{RNG}}$ except that instead of generating shares of a random value, it generates shares of the additive identity (zero element) of the field. In [3] this is achieved using a variation of their robust RNG protocol. However, in this paper we will use a protocol very similar to our $\pi^{n,k}_{\mathtt{RNG}}$ to again avoid the biasing problem.
- (Random key-pair generation $\pi^{n,k}_{\mathtt{RKPG}}$): This is a distributed key-pair generation protocol built using $\pi^{n,k}_{\mathtt{RNG}}$. A distributed key-pair generation protocol is also presented in [8], but it requires extra rounds of secure broadcasting to remove biasing of the public key.

Additionally, we also present proofs of security for open protocols; these protocols are the familiar opening of a shared secret. The proofs are given for Canetti's security framework, as we want to prove that the ECDSA signature algorithm is secure in this framework, and opens are needed for multiplying shared secrets and also inverting a shared secret. An added benefit of having security proofs of our subprotocols in Canetti's framework is that from the composability property, they can be readily used for other applications as well. However, we will see that for producing ECDSA signatures, we get the (weaker) security property proven in [3] for free as the protocols are almost the same.

1.2 Organisation

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. After this, some cryptographic primitives that will be used are defined. The ECDSA signature algorithm will then be introduced, and how it will be achieved in

SMPC, making comparisons to previous work. 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 present the formal theorems for the security of our protocol for computing ECDSA signatures.

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

A *negligible* function is one that approaches zero very fast. 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 here. For more details on the underlying measure and probability theory, some key definitions and theorems are collected in Appendix A for convenience.

We first 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 5. *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 present 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. The proof can be found in Appendix B.

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.

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 [9]. 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 $c_1, \dots, c_{k-1} \in \mathbb{F}$ 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

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

where naturally the secret is c_0.

We now present 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. A similar result can be found in [10]. The proof of this result can be found in Appendix C.

Theorem 3. *Let $x_1, \dots, x_n$ 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 $c_1, \dots, c_{k-1}$ 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*

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

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.

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 6. *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* $(c_0,\dots,c_{k-1})\in\mathbb{F}^k$ *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* $t<k\le n$. *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 $y_1,\dots,y_k\in\mathbb{F}$ with corresponding indices $i_1,\dots,i_k\in\mathbb{F}$, 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 $c_0,\dots,c_{k-1}$, 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 present 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. The proof can be found in Appendix C.

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* $|\mathcal{I}|=n\ge k$. *Let the associated coefficients for the sharing be* $c_0,\dots,c_{k-1}$. *Then for each* $\mathcal{R}\subset\mathcal{I}$ *with* $|\mathcal{R}|=k$ *and* $j\in\{0,\dots,k-1\}$, *there exist field elements* $(\lambda_i^{(j)})_{i\in\mathcal{I}}$ *such that*

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

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 probabilistic polynomial-time 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 Model

In this section we will outline the security model and definitions that we will use to prove that the presented protocols are secure. In general, we will consider a malicious, rushing adversary with static corruptions. We also define the adversary to be computationally bounded, i.e. so that it cannot compute discrete logarithms, as in Assumption 1. In this paper these properties should be assumed of the adversary unless specified otherwise. Further, we assume that each party is connected to each other by a private point-to-point channel, and that these channels allow for each party to verify that the message came from the specified sender, using e.g. a cryptographic signature scheme.

The notion of security we will use is that of Canetti [6], 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 [6]. 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*. We call an adversary t-limited if it controls at most t parties.

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 *malicious* 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.

The random variable corresponding to the execution of the protocol π in the presence of an adversary $\mathcal{A}$ is defined as follows. Denote by $\texttt{ADVR}_{\pi,\mathcal{A}}(x,z,r)$ the output of $\mathcal{A}$ on running π with auxiliary input z and input $x=(x_1,\ldots,x_n)$ with random input r. This output consists of its auxiliary input, random input, corrupted parties' inputs, corrupted parties' random inputs, and all messages sent and received by the corrupted parties during the execution of π. Denote by $\texttt{EXEC}_{\pi,\mathcal{A}}(x,z,r)_i$ the output of party P_i at after running π. Recall that this output will be $\perp$ if P_i is corrupted. Then we have the following definition.

Definition 7. *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 joint random variable*

$$\left(\texttt{ADVR}_{\pi,\mathcal{A}}(x,z,r), \texttt{EXEC}_{\pi,\mathcal{A}}(x,z,r)_1, \ldots, \texttt{EXEC}_{\pi,\mathcal{A}}(x,z,r)_n\right),$$

where r is chosen uniformly randomly.

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 $\mathcal{S}$. Denote by $\texttt{ADVR}_{\pi,\mathcal{S}}(x,z,r)$ the output of $\mathcal{S}$ on running π with auxiliary input z and input $x=(x_1,\ldots,x_n)$ with random input r.

Definition 8. *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*

$$\left(\texttt{ADVR}_{\pi,\mathcal{S}}(x,z,r), \texttt{EXEC}_{\pi,\mathcal{S}}(x,z,r)_1, \ldots, \texttt{EXEC}_{\pi,\mathcal{S}}(x,z,r)_n\right),$$

where r is chosen uniformly randomly.

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 9. *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.

The main difference of this definition of security when compared to earlier models [11, 12, 13] is the combination of the notions of *secrecy* and *correctness*. The former captures the idea that no information is gained by the adversary, which corresponds to the distribution of the messages in the view. The latter captures the fact that the protocol should correctly do what it is designed to do, and this is captured by having the outputs of the parties in the view. Thus, Canetti's model captures both of these properties in the single equality of Definition 9, whereas earlier models required two separate proofs for each of the properties. Note that this difference is not merely cosmetic; see Canetti's paper [6] for examples and discussion of how a protocol that is intuitively insecure can be proven to be secure in earlier models but is correctly identified as insecure by Canetti's security definition.

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. Specifically, we need non-interactivity, meaning that a prover can create a single proof object that can be verified by a verifier without any additional interaction. We borrow the definitions of such a system from [14].

Let R be a binary relation, such that we have elements $(\phi, w) \in R$ where ϕ is called the statement and w is called the witness. We have the following algorithms.

- $(\sigma, \tau) \leftarrow \texttt{setup}(R)$: This is the setup that produces the common reference string σ and simulation trapdoor τ for R.
- $\pi \leftarrow \texttt{prove}(R, \sigma, \phi, w)$: This is the proving algorithm that produces a proof π for the statement $(\phi, w) \in R$ using σ.
- $b \leftarrow \texttt{verify}(R, \sigma, \phi, \pi)$: This is the verification algorithm that returns a result $b \in \{0, 1\}$ such that $b = 0$ signifies that the proof π for ϕ using σ was rejected, and $b = 1$ signifies acceptance.
- $\pi \leftarrow \texttt{sim}(R, \tau, \phi)$: This is the simulation algorithm that takes the trapdoor τ and a statement ϕ and returns a proof π.

We require the common properties of *completeness*, *perfect zero-knowledge* and *computational soundness*. They are summarised in the following definitions, using the notation from [14].

Definition 10 (Completeness). *Completeness states that an honest prover should be able to convince an honest verifier of a true statement. Precisely, we require that*

$$\mathbb{P}\left[\begin{matrix}(\sigma, \tau) \leftarrow \texttt{setup}(R) \\ \pi \leftarrow \texttt{prove}(R, \sigma, \phi, w)\end{matrix}\middle| \texttt{verify}(R, \sigma, \phi, \pi) = 1\right] = 1$$

Definition 11 (Perfect zero-knowledge). *Perfect zero-knowledge states that a proof does not leak any knowledge other than the truth of the statement. Precisely, it must hold that for all $(\phi, w) \in R$, auxiliary input z and any adversary $\mathcal{A}$*

$$\mathbb{P}\left[\begin{matrix}\sigma \leftarrow \texttt{setup}(R) \\ \pi \leftarrow \texttt{prove}(R, \sigma, \phi, w)\end{matrix}\middle| \mathcal{A}(R, z, \sigma, \tau, \pi) = 1\right] = \mathbb{P}\left[\begin{matrix}\sigma \leftarrow \texttt{setup}(R) \\ \pi \leftarrow \texttt{sim}(R, \tau, \phi)\end{matrix}\middle| \mathcal{A}(R, z, \sigma, \tau, \pi) = 1\right]$$

Definition 12 (Computational soundness). *Computational soundness states that it is impossible to prove a false statement. Specifically, let L_R be the language of statements such that there exists a matching witness in R. Then for all adversaries $\mathcal{A}$ we require that*

$$\mathbb{P}\left[\begin{matrix}(\sigma,\tau)\leftarrow \texttt{setup}(R)\\ (\phi,\pi)\leftarrow \mathcal{A}(R,z,\sigma)\end{matrix}\middle|\begin{matrix}\phi\notin L_R\\ \texttt{verify}(R,\sigma,\phi,\pi)=1\end{matrix}\right]\approx 0$$

4.3 Consensus

To build our random number generation protocol, 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. [15].

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 a 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 $\rho^{n,k}_{\texttt{RNG}}$. The purpose of consensus during the random number generation protocol is to pick a subset of players that contributed correctly to the protocol. Thus a decision value will contain shares from a subset of players along with ZK proofs that each of the sharings is valid (these are needed because the shares will be encrypted for the target players as they should be kept private).

The consensus protocol 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 key-pair where the public key is denoted κ_i. Each party P_i has input $c^{(i)}_0,\ldots,c^{(i)}_{t-1}$, which are the coefficients of a polynomial that is to be used for a threshold t sharing of the number $c^{(i)}_0\in\mathbb{F}$. They then do the following to construct their input for a consensus protocol ρ:

1. Create n shares $\left(r^{(i)}_j\right)_{j\in\mathcal{I}}$ from the coefficients $c^{(i)}_0,\ldots,c^{(i)}_{t-1}$, i.e.

$$r^{(i)}_j=\sum_{l=0}^{t-1}c^{(i)}_l j^l \quad \forall j\in\mathcal{I}. \tag{2}$$

2. Obtain the encrypted values $e_i = \left(e_j^{(i)}\right)_{j \in \mathcal{I}}$, where $e_j^{(i)} = E_{\kappa_j}\left(r_j^{(i)}\right)$.

3. Create a ZK proof π_i that attests to the validity of the sharing and encryptions. Specifically, let R be the relation with elements (ϕ, w) where the witness w consists of a private key for the encryption scheme and t elements of $\mathbb{F}$ that represent the coefficients of a sharing. The associated statement ϕ asserts that

   - $e_j^{(i)} = E_{\kappa_j}\left(r_j^{(i)}\right)$ for all $j \in \mathcal{I}$.
   - The values $\left(r_j^{(i)}\right)_{j \in \mathcal{I}}$ constitute a valid and consistent sharing of some value in $\mathbb{F}$ (in this case, that value is $c_0^{(i)}$), i.e. that Eq. (2) holds.

   Then $\pi_i \leftarrow \texttt{prove}(R, \sigma, \phi, w)$ where $(\sigma, \tau) \leftarrow \texttt{setup}(R)$.

A possible decision value for ρ is a set $\{(e_i, \pi_i)\}_{i \in I}$ where $I \subset \mathcal{I}$. The predicate is $\texttt{valid}_{\texttt{RNG}}$, which is true precisely when $|I| > t$ and $\texttt{verify}(R, \sigma, \phi, \pi_i) = 1$ 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 [16] or Pedersen [7]. 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. 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 $\texttt{valid}(R, \sigma, \phi, \pi_i)$ succeeded. Using $\rho_{\texttt{RNG}}^{n,k}$ thus only introduces the synchronicity required for the consensus algorithm, which for example in the case of Tendermint [15] is only partial synchronicity.

5 ECDSA Signature Algorithm

The ECDSA algorithm for an elliptic curve is defined as follows. Let an elliptic curve defined over the field $\mathbb{F}_p$ be given such that it has a generator G of prime order n. Let $d \in [n-1]$ be a private key with associated public key dG, and let m be the message to be signed. Let z be a hash of m such that the bit length of z is the same as the bit length of n. Then the following steps are used to generate the signature (r, s) of m under the private key d:

1. Pick k uniformly randomly from $[n-1]$.
2. Compute $(x, y) = kG$.
3. Let $r = x \mod n$.
4. Compute $s = k^{-1}(z + rd) \mod n$.
5. Output the signature (r, s).

Note that this is the same standard as in [3] except it is adapted for use with elliptic curves. To implement this signing algorithm in SMPC, the parties will need to have shares of the private key d such that no small enough subset knows d. This can be achieved by using a random key-pair generation algorithm. The parties will also need to generate k along with the associated public key kG, which can also be achieved using a random key-pair generation protocol. Next, the parties will need to perform some arithmetic operations on the shares of d and k and the hash of the message z. Arithmetic operations on shares can be performed locally when the operation is addition or when multiplying by a public (not shared) value. This means that computing r as well as $z + rd$ can be done locally. The two important operations are thus inverting k, and performing a multiplication of the shared values k^{-1} and $z + rd$.

5.1 Comparison to [3]

Here we will make an explicit comparison of our approach to [3]. The way that our threshold signature protocol is carried out is the same as [3] when viewed as the composition of subprotocols. That is, if we consider the subprotocols of random number generation (RNG), random zero generation (RZG), random key-pair generation (RKPG), multiplication of shares (including opening the result) and inversion of a shared value, then we use these subprotocols in the same way as in [3] to compute the signature, with the following small differences:

- We target the ECDSA algorithm adapted for elliptic curves. The difference is that [3] instead computes

$$r = \left(g^{k^{-1}} \mod p\right) \mod q$$
$$s = k(m + xr) \mod q,$$

  where p, q are primes such that $q \mid p - 1$ and g is an element of order q in the multiplicative group of integers modulo p, $\mathbb{Z}_p^*$. This just means that some values live in different sets, e.g. generating a random key-pair generates a random shared value $k \in [q]$ but the public value g^k is in the group generated by $g \in \mathbb{Z}_p^*$, whereas our public value will be a point on the elliptic curve.

- Order of subprotocols. When computing a signature, we will need to do two multiplications (one is needed during inversion, and then one for the final step in computing s) and for this we will need two random sharings of zero, obtained by calling RZG twice. In [3] these two calls to RZG are performed near the start of the protocol, to be used later. On the other hand, in our protocol we will include them as a part of the multiplication subprotocol. However this has no significant affect on the proof (this is true both in [3] and in this paper where we use Canetti's security framework). We make this difference in this paper so that we can prove the security of multiplication in Canetti's security framework which allows multiplication to be more easily reused in other contexts as a single securely composable protocol.

The only other important differences between the protocol presented in this paper and that of [3] is how the subprotocols for RNG, RZG, and RKPG are defined. These subprotocols are specified in Section 6.2, Section 6.3 and Section 6.4 respectively.

5.1.1 Security

One question that arises is the following: if the protocol in this paper has these differences to the protocol in [3], does the proof of security presented in [3] still apply here? We answer in the affirmative. To see why, we will consider the differences in turn. First, we turn our attention to the use in this paper of different realisations of the subprotocols RNG, RZG and RKPG. In this paper, we will prove that our realisation of these subprotocols are secure under Canetti's definition of security [6]. Notice that the definition of security used by [3] is based on older works [11, 12, 13] which provide a weaker definition. Specifically, Canetti's definition combines both the messages sent by the parties as well as the outputs of the honest parties as part of the view, whereas the older models only include the messages in the view (and so require separate proofs for correctness and secrecy as seen in [3]). This means that our security proofs will imply security in the weaker definition used by [3][1]. However, in this paper we prove that these subprotocols are secure *individually* and then make use of the composability property of Canetti's security definition to use them together, whereas in [3] the proof of security is given for the *entire* signature protocol, instead of its constituent parts. We argue that this still means that the proof in [3] is valid if we substitute our subprotocols, because we can take the simulators used in the proofs of security of $\pi^{n,k}_{\mathtt{RNG}}$, $\pi^{n,k}_{\mathtt{RZG}}$ and $\pi^{n,k}_{\mathtt{RKPG}}$ and use them to make the same arguments as their relevant parts in the proof of security in [3]. In those places in the proof where the simulator needs to cheat to cause the output of one of the subprotocols to be a specific value, we note that this can be achieved with the simulators in this paper due to the stronger notion of security under which they are proven.

[1]Note that in [3] there are multiple proofs for different adversary models. Here we are specifically interested in the proof corresponding to malicious adversaries, which is for protocol `DSS-Tresh-Sig-2` in their paper, as this corresponds to our adversary model.

6 SMPC Primitives

In this section we will define the primitives which constitute our SMPC protocols and prove their security. It is clear that security as defined in Definition 9 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 6.1), random number generation (Section 6.2), random zero generation (Section 6.3), and random key-pair generation (Section 6.4).

6.1 Open

When we reveal the secret corresponding to a sharing, we call this operation an *Open*. We will present two protocols for this operation, each warranting their use in different situations. The difference between them lies in how the ideal functionality is defined; one will provide security of the input shares, and the other will not.

Probably the most obvious ideal functionality for open would be defined as follows: f takes as input the shares of the secret from each party and gives as output the 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 against Definition 9 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 the two different protocols as follows.

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. This solution will be used in our multiplication protocol, as in [3]. The reason that this will be needed here is that after multiplying our shares, they will now be points on a degree $2k$ polynomial that is known to be the product of two k degree polynomials. This causes there to be enough structure in the shares that if they are revealed one can often efficiently find the original two secrets.

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 will not use this protocol in this paper, but it is included anyway for completeness as the security proof is trivial.

6.1.1 Reed-Solomon Decoding

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 [17] or Gao's [18], to remain fault tolerant in the presence of corrupt parties broadcasting incorrect shares. This idea was first introduced in [19]. 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. More precisely, we can define an RS decoding algorithm as having the following input/output relationship:

- Input: A codeword (i.e. shares) $(x_1, \ldots, x_n) \in \mathbb{F}^n$.
- Output: The message polynomial $c_0 + c_1 x + \cdots + c_{k-1} x^{k-1}$, i.e. a polynomial of degree $k - 1$ that satisfies

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

or "Decoding failure".

Note that the input for the RS decoding algorithm is *all* of the shares of the parties. If we were to naïvely achieve this in practice, we could have a timeout for receiving all of the shares, and then once the timeout has expired we could set the missing shares to arbitrary values and then run the protocol. However, we aim to avoid timeouts, and so we will present a simple algorithm that can be used to reconstruct a secret using RS decoding that avoids the need for a timeout. The idea is as follows. 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. This is achieved by setting the yet to be received messages as any value. 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 algorithm describes how the honest parties can reconstruct the secret during an open using RS decoding.

1. Wait until $n - \lfloor \frac{d}{2} \rfloor$ shares have been received.

[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 players.

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

We will explain why this will eventually allow the reconstruction of the secret value in the presence of no more than $\lfloor \frac{d}{2} \rfloor$ corrupted parties. First, since we assume that we will receive all messages from the honest parties, we are guaranteed to receive at least $n - \lfloor \frac{d}{2} \rfloor$ messages. Next, note that once we have all of the honest parties' shares, the RS decoding algorithm is guaranteed to succeed by definition. These two facts ensure that the reconstruction will eventually terminate successfully. If any corrupted parties shares have been modified and are received before all of the honest parties' shares, this will simply result in some initial decoding failures.

This technique for RS decoding without using timeouts is mainly a practical consideration, and is included as it aligns with the motivations for the design of our protocols. For the purposes of the security proofs for open, however, this technique is not needed. This is because in Canetti's security framework, the protocols proceed in rounds, and everyone receives the messages for each round before moving on to the next round. This means that honest parties will have all honest shares at the end of the round in which they all send them, and with all honest shares they need only run the RS decoding algorithm once to successfully reconstruct the secret (possibly setting missing malicious shares to an arbitrary value).

6.1.2 Share Revealing Open

The protocol implementing the basic ideal functionality (that outputs the secret as well as all of the input shares) $f^{n,k}_{\mathtt{OPEN}}$ is denoted by $\pi^{n,k}_{\mathtt{OPEN}}$ 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 $\pi^{n,k}_{\mathtt{OPEN}}$ t-securely evaluates $f^{n,k}_{\mathtt{OPEN}}$.*

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. □

6.1.3 Share Hiding Open

The protocol implementing the stricter ideal functionality $f^{n,k}_{\mathtt{SO}}$ is denoted by $\pi^{n,k}_{\mathtt{SO}}$ 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, $\pi^{n,k}_{\mathtt{RZG}}$, is defined in Section 6.3. The protocol $\pi^{n,k}_{\mathtt{SO}}$ is defined as follows.

1. The parties participate in $\pi^{n,k}_{\mathtt{RZG}}$ 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.

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

Remark 3. We will see later that if we invoke the multiplication protocol with two input sharings that have reconstruction threshold k, we will need to invoke $\pi^{n,2k}_{\mathtt{SO}}$. This means that if k is a large enough fraction of n, the restriction on t will be due to the RS decoding restriction, $t \leq \frac{n-k+1}{2}$, as opposed to the sharing restriction $t < k-1$. For example, when $k = \frac{n}{3}$, we will require that $t < \frac{n}{6}$. One way to overcome this limitation is to use verifiable secret sharing (e.g. Pedersen's [7]), which allows each party to determine if a given share is valid in isolation from the others. Thus if the parties have the Pedersen commitments for the shares they are opening, RS decoding is not needed and we will only have the sharing restriction $t < k-1$. To see how this can be applied in the case of multiplication, see [3] and [20]. We leave making this improvement to our protocols as future work.

Remark 4. 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. We will give a high level proof here. Full mathematical details can be found in Appendix D.

The idea of the proof is as follows. The simulator $\mathcal{S}$ needs to construct the view so that it is consistent with the output secret being s. Since $\pi^{n,k}_{\mathtt{SO}}$ randomises the sharing before reconstructing, these shares should look random and so $\mathcal{S}$ can simply construct a random sharing $(y_i)_{i \in \mathcal{I}}$ of s for that part of the view. Now, $\mathcal{S}$ needs to make sure that the rest of the view is consistent with the real world view: the corrupted parties inputs $(x_i)_{i \in \mathcal{I}_c}$ and random shares of zero $(z_i)_{i \in \mathcal{I}_c}$ need to satisfy the relationship $x_i + z_i = y_i$ for each $i \in \mathcal{I}_c$. Since the input shares are fixed, $\mathcal{S}$ simply defines $z_i = y_i - x_i$ for each $i \in \mathcal{I}_c$. This will have the right distribution because the sharing of s was chosen randomly and $t < k-1$. 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

$$z_i = y_i - x_i \quad \forall i \in \mathcal{I}_c. \tag{3}$$

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 will now justify why the view of $\mathcal{S}$, $\mathtt{IDEAL}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{S}}(x,z)$, has the correct distribution. First, in the real world view the shares of zero $(z_i)_{i\in\mathcal{I}_c}$ will look completely random as there are less than $k-1$ of them, whereas they are shared with threshold $k-1$. In the simulated view this is also true, based on their definition in Eq. (3) and the fact that $(y_i)_{i\in\mathcal{I}_c}$ are also completely random by construction. Next, the messages from the adversary $(m_i)_{i\in\mathcal{I}_c}$ will have the correct distribution because they were obtained by directly running $\mathcal{A}$ and all of the information that $\mathcal{A}$ has obtained up until this point, namely $(z_i)_{i\in\mathcal{I}_c}$, has the right distribution. Finally, the last element has the correct distribution because of its construction and the relationship Eq. (3).

As for correctness of the output, this is an easy consequence of the restrictions on t. These restrictions ensure that the honest parties can correctly reconstruct the secret using RS decoding. □

6.1.4 Directed Open

The protocols $\pi_{\mathtt{OPEN}}^{n,k}$ and $\pi_{\mathtt{SO}}^{n,k}$ reveal the shared secret to all players. However, sometimes we will only want to reveal the secret to a specific player. This occurs in the protocol $\pi_{\mathtt{RNG}}^{n,k}$. 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 $\pi_{\mathtt{OPEN}}^{n,k}$ as $\pi_{\mathtt{DO}}^{n,k,i}$ and the directed version of $\pi_{\mathtt{SO}}^{n,k}$ as $\pi_{\mathtt{SDO}}^{n,k,i}$, where in each case $i \in \mathcal{I}$ is the index of the player which outputs the secret. The ideal functionalities $f_{\mathtt{DO}}^{n,k,i}$ and $f_{\mathtt{SDO}}^{n,k,i}$ 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 $\pi_{\mathtt{DO}}^{n,k}$ t-securely evaluates $f_{\mathtt{DO}}^{n,k}$.*

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

6.2 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.

6.2.1 Biased RNG

The goal of random number generation is for each of the parties to end up with shares of a global random number that none of the parties know. This technique was first introduced by Pedersen [7] and works as follows. Each player begins by generating a random number and creating shares of it. These shares are then sent to all other players, after which each player should have a share of the random numbers of the other players, as well as one of their own. These shares are then summed to produce a single share of the sum of each players random number. If at least one of the players that participated picked their random number uniformly randomly, then the global random number will also be uniformly random.

The main challenge of using this protocol in the presence of malicious adversaries is that of having the honest parties select the same subset of shares to use in the sum; not doing so would lead to the honest parties holing an inconsistent sharing. The way that a corrupted party can cause this to happen is simply sending their shares to a subset of the parties, and not sending any to the others. One possible solution to this is to have parties also send acknowledgement messages upon receiving correct shares on a secure broadcast channel [3]. However, since we are not using a secure broadcast channel we use an alternative solution: a consensus protocol. The idea is that once consensus is achieved, all of the honest parties will agree on which subset of shares from other parties to include in the sum, and will therefore end up with consistent shares of the same global random number. The protocol is denoted $\pi^{n,k}_{\mathtt{BRNG}}$ 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 $c_0^{(i)}, \dots, c_{t-1}^{(i)}$, 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 $\rho^{n,k}_{\mathtt{RNG}}$ using the generated coefficients, obtaining the output $(r_{j,i})_{j\in I}$ where $I \subset \mathcal{I}$ is defined implicitly by $\rho^{n,k}_{\mathtt{RNG}}$ as the set of players whose shares were agreed to be used by $\rho^{n,k}_{\mathtt{RNG}}$.

3. The final output of player P_i is

$$y_i = \sum_{j \in I} r_{j,i}.$$

If we attempt to prove that this protocol (or similar ones such as [3]) Canetti's security model, we encounter a problem. This problem arises when we define the most obvious ideal functionality for $\pi^{n,k}_{\texttt{BRNG}}$. This obvious ideal functionality $f^{n,k}_{\texttt{RNG}}$ is: each party gives no input and the output is a uniformly random sharing of a uniformly random number, the output for each party being their respective share. The key here is that we have specified that not only is the global random number uniformly random, but also the *sharing* of this random number is random; each possible sharing of the random number is equally as likely. Unfortunately, this latter condition is easily thwarted by a rushing adversary, as they can observe the honest parties' shares first, and then pick their random numbers' associated sharings to be such that their final shares (the sum of the shares from the chosen subset of parties) are biased. For example, when using Tendermint for the consensus protocol, the proposer of a block may be a corrupted party. If this is the case, they are in fact able to realise the rushing property and can pick their own sharing to be such that they end up with any share of their choosing. This clearly violates the property that the sharing of the random number is itself random. In order to overcome this technical difficulty[3], we will not prove that $\pi^{n,k}_{\texttt{BRNG}}$ is secure in Canetti's security framework but rather just prove its key properties that will be relevant when we use $\pi^{n,k}_{\texttt{BRNG}}$ to construct an unbiased protocol $\pi^{n,k}_{\texttt{RNG}}$ that will capable of realising the ideal functionality $f^{n,k}_{\texttt{RNG}}$.

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.

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

[3] There may be a clever way to modify the ideal functionality so that it still faithfully represents the goal of random number generation but allows this biasing of the shares by the adversary. However, the authors are not aware of a way to do this, or whether it is safe in general to allow the biasing of the shares.

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*: The first property required for secrecy is straightforward to show. Since $t < k$, the corrupted parties never receive enough shares of any of the honest parties' contributions, of which there will be at least one since $|I| > t$. The result then follows from the basic properties of Shamir secret sharing.

  Now we will show the second property required for secrecy: that the shares of the honest parties appear uniformly random, given the constraints. 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.

□

6.2.2 Unbiased RNG

Unbiased random number generation is a protocol $\pi^{n,k}_{\mathtt{RNG}}$ 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 $f_{\mathtt{RNG}}^{n,k}$ which takes no input from the parties, and gives output that is n shares of a uniformly random sharing of a uniformly random number r (unknown to all parties) with reconstruction threshold k, giving each share to the corresponding party.

To create the unbiased protocol $\pi_{\mathtt{RNG}}^{n,k}$ from the biased protocol $\pi_{\mathtt{BRNG}}^{n,k}$ the idea is simple. Run $\pi_{\mathtt{BRNG}}^{n,k}$ k times, so that each of the random secrets represents a coefficient of a uniformly random polynomial p. Then, compute P_i's share of $p(i)$ for each $i \in \mathcal{I}$ by a linear combination of the shares of the k coefficients, and open these resulting shares of shares to each corresponding party using a directed open. Since creating the shares of $p(i)$ is a linear combination of the coefficients, this can be computed on the shares locally. The protocol is as follows.

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

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, $r_j = p(j)$, or in other words the share for party P_j corresponding to p.

3. The players invoke $\pi_{\mathtt{DO}}^{n,k,j}$ 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 $\pi_{\mathtt{RNG}}^{n,k}$ t-securely evaluates $f_{\mathtt{RNG}}^{n,k}$.*

Proof. We will present a high level proof here. Full mathematical details can be found in Appendix E.

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.
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

$$\left\{ \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_h}\right)_{j\in[k-1]_0} \in \mathbb{F}^{k(n-t)} \middle| \begin{array}{c} \forall j \in [k-1]_0 \\ \exists (c'_l)_{l\in[k-1]} \\ \forall i \in \mathcal{I} \end{array} : r_i^{(j)} = c_j + \sum_{l=1}^{k-1} c'_l i^l \right\}.$$

We will justify why $\mathcal{S}$ produces a view with the same distribution as in the real world case. The parts of the view that consist of messages that the corrupted party sends should have the correct distribution as they are generated by running a copy of $\mathcal{A}$ directly, and by ensuring that the preceding information that $\mathcal{A}$ gains has the right distribution. Also, the messages that the corrupted parties receive from the invocations of $\pi^{n,k}_{\texttt{BRNG}}$ will clearly have the right distribution as they are generated by $\mathcal{S}$ invoking $\pi^{n,k}_{\texttt{BRNG}}$ directly, acting on behalf of the honest players. Thus the two important parts of the view that we need to make sure have the right distribution are the messages the corrupted parties receive during the directed opens, and the output shares. Since the latter is determined by the former, we need only consider the former (messages received during the directed opens). In the real world view $\texttt{EXEC}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{A}}(x,z)$, these are just uniformly random shares of uniformly random numbers conditioned on the fact that they are consistent with the corrupted parties shares. This follows from the fact that this is true for the honest parties' shares from $\pi^{n,k}_{\texttt{BRNG}}$ (from the secrecy property), and that the shares in the directed opens are just linear combinations of these shares. In the ideal case view $\texttt{IDEAL}_{\pi^{n,k}_{\texttt{RNG}},\mathcal{S}}(x,z)$, these shares are constructed by working backwards from the output shares. Since the output sharing is uniformly random, it follows that the coefficients $(c_i)_{i\in[k-1]_0}$ that $\mathcal{S}$ constructs (which correspond the random numbers for the k invocations of $\pi^{n,k}_{\texttt{BRNG}}$) will also be uniformly random. When $\mathcal{S}$ extends the corrupted parties' shares to open to these values, it does this by picking uniformly randomly from the possible sharings, and this ensures that the distribution will be the same as in the real world view. □

6.3 Random Zero Generation

Random zero generation is almost the same as random number generation as in Section 6.2, except instead of obtaining shares of a global random number, the parties obtain a sharing of $0 \in \mathbb{F}$. Specifically, the ideal functionality $f^{n,k}_{\mathtt{RZG}}$ 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 $\pi^{n,k}_{\mathtt{RNG}}$, 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 $\pi^{n,k}_{\mathtt{RZG}}$:

1. The players invoke $\pi^{n,k}_{\mathtt{BRNG}}$ 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 $\pi^{n,k}_{\mathtt{RZG}}$ is nearly identical to that for $\pi^{n,k}_{\mathtt{RNG}}$, and so is omitted.

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

Remark 5. Note that the requirement on t is that it is less than $k-1$, as opposed to k as was the case for $\pi^{n,k}_{\mathtt{RNG}}$. 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 $0 \in \mathbb{F}$).

6.4 Random Key-pair Generation

Random key pair generation is again very similar to random number generation defined in Section 6.2, 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 $y = xG \in \mathbb{G}$ where $\mathbb{G}$ is a group with generator G, written with additive notation. Additionally, to enable the proof of security to work each party will also output x_iG 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 security against malicious adversaries 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 x_iG to every other party.
3. Each party then reconstructs xG "in the exponent", along with x_iG 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 x_iG 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 $g_i = x_iG$ 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 ${\pi^{n,k}_{\texttt{RKPG}}}'$ 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 x_iG for all shares x_i, as well as knowing xG, 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 $k \le n$. 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, $(x_i)_{i \in I}$, $(x_iG)_{i \in [n]}$ and xG, 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 $g_i = x_i G$ 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 $g_i = x_i G$ and also define x_0 to be the unique value that satisfies $y = x_0 G$. 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 $\lambda_0^{(i)}, \ldots, \lambda_{k-1}^{(i)}$ such that

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

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

$$c_i G = h_i = \sum_{j=0}^{k-1} \lambda_j^{(i)} g_j$$

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

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

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 $y = x_0 G$, this completes the proof. $\square$

We now discuss how to augment ${\pi_{\mathtt{RKPG}}^{n,k}}'$ to be secure in the presence of a malicious 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 $\pi_{\mathtt{RNG}}^{n,k}$ to output, along side the usual random shares, a perfectly hiding commitment to each of the shares of the parties. Then, ${\pi_{\mathtt{RKPG}}^{n,k}}'$ is augmented by also submitting in step 2 a zero knowledge proof from a ZK scheme with completeness, perfect zero knowledge and computational soundness that the value they sent corresponds to the commitment to their share. This will enable honest parties to detect which sent 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 $\pi_{\texttt{RKPG}}^{n,k}$ is as follows.

1. The parties invoke the augmented RNG protocol $\pi_{\texttt{RNG}}^{n,k'}$, 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.

To achieve this, one could use Pedersen's verifiable secret sharing [7] to obtain the perfectly hiding commitments of the shares[4], and a ZKSNARK construction, such as in [14], for the ZK proofs. We leave the full details of this augmentation to future work.

6.5 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

[4] Pedersen's scheme actually only involves constructing and sending commitments to the coefficients of the sharing polynomial, but it is easy to obtain commitments to the shares from these.

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.

7 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.

7.1 Multiply and Open

The ideal functionality $f_{\mathtt{MO}}^{n,k}$ 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. The technique is the same as the multiplication protocol in [3]. There, and in our case, we don't need to multiply shares more than once in a row, which allows us to make use of this simpler protocol. If we wanted to achieve multiplication that could be performed as many times as we liked, there exist techniques such as in [21] and [22].

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

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

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 $\pi_{\mathtt{MO}}^{n,k}$ t-securely evaluates $f_{\mathtt{MO}}^{n,k}$.*

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 $\pi_{\mathtt{SO}}^{n,2k}$, and the second requirement on t is the usual one to disallow the adversary to reconstruct secrets.

7.2 Inversion

The ideal functionality $f^{n,k}_{\mathtt{INV}}$ 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. This is the same as the inversion protocol in [3] and was initially introduced in [23].

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

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

Remark 6. Because the shares of the product are randomised by $\pi^{n,k}_{\mathtt{MO}}$, it is conjectured that it is safe to define $r_{\mathcal{I}}$ using the less expensive biased RNG protocol $\pi^{n,k}_{\mathtt{BRNG}}$.

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 $t < k$. Let $x_{\mathcal{I}}$ be a k-sharing of some $x \in \mathbb{F} \setminus \{0\}$. Then the protocol $\pi^{n,k}_{\mathtt{INV}}(x_{\mathcal{I}})$ t-securely evaluates $f^{n,k}_{\mathtt{INV}}(x_{\mathcal{I}})$.*

The restrictions on k and t are due to $\pi^{n,k}_{\mathtt{MO}}$. 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.

8 Signature Algorithm

The ideal functionality $f^{n,k}_{\mathtt{SIGN}}$ 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 $\pi^{n,k}_{\mathtt{SIGN}}(d_{\mathcal{I}}, z)$ 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 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 $d_{\mathcal{I}}$ 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 7. 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] Rosario Gennaro and Steven Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". English. In: ACM, 2018, pp. 1179–1194. ISBN: 9781450356930;1450356931;

[2] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. "Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security". In: *Applied Cryptography and Network Security.* Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Cham: Springer International Publishing, 2016, pp. 156–174. ISBN: 978-3-319-39555-5.

[3] Rosario Gennaro et al. "Robust Threshold DSS Signatures". In: *Advances in Cryptology — EUROCRYPT '96.* Ed. by Ueli Maurer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 354–371. ISBN: 978-3-540-68339-1.

[4] Philip MacKenzie and Michael K. Reiter. "Two-party generation of DSA signatures". In: *International Journal of Information Security* 2.3 (Aug. 2004), pp. 218–239. DOI: 10.1007/s10207-004-0041-0. URL: https://doi.org/10.1007/s10207-004-0041-0.

[5] Ivan Damgård et al. "Implementing AES via an Actively/Covertly Secure Dishonest-Majority MPC Protocol". In: *Security and Cryptography for Networks.* Ed. by Ivan Visconti and Roberto De Prisco. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 241–263. ISBN: 978-3-642-32928-9.

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

[7] 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.

[8] Rosario Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". English. In: *Journal of Cryptology* 20.1 (2007), pp. 51–83.

[9] 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.

[10] Gilad Asharov et al. "A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation". English. In: *Journal of Cryptology* 30.1 (2017), pp. 58–151.

[11] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". English. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208.

[12] Silvio Micali and Phillip Rogaway. "Secure Computation". In: *Advances in Cryptology — CRYPTO '91.* Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 392–404. ISBN: 978-3-540-46766-3.

[13] Donald Beaver. "Foundations of Secure Interactive Computing". In: *Advances in Cryptology — CRYPTO '91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 377–391. ISBN: 978-3-540-46766-3.

[14] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: May 2016, pp. 305–326. ISBN: 978-3-662-49895-8. DOI: `10.1007/978-3-662-49896-5_11`.

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

[16] 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`.

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

[18] 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`.

[19] R. McEliece and D. Sarwate. *On sharing secrets and Reed-Solomon codes*. English. 1981.

[20] Manuel Cerecedo, Tsutomu Matsumoto, and Hideki Imai. "Efficient and secure multiparty generation of digital signatures based on discrete logarithms". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 76 (Apr. 1993).

[21] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 1–10. ISBN: 0897912640. DOI: `10.1145/62212.62213`. URL: `https://doi.org/10.1145/62212.62213`.

[22] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '91. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 420–432. ISBN: 3540551883.

[23] J. Bar-Ilan and D. Beaver. "Non-cryptographic fault-tolerant computing in constant number of rounds of interaction". English. In: ACM, 1989, pp. 201–209. ISBN: 0897913264;9780897913263;

A Measure and Probability Theory

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 13. *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 14. *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 15 (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 16. *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 17. *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.

B Proof of Theorem 1

The following is the proof of Theorem 1

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. $\square$

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

$$\mu(X = e) = \sum_{i=1}^{n} \mu\left(A_{i,e} \times \{f_i\}\right). \tag{5}$$

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 (5) 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. □

C Shamir Secret Sharing Theorem Proofs

First we present the proof of Theorem 3.

Proof. We consider three separate cases.

- $t \geq k$: In this case, we can view equation (1) 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 (1). 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 (1), 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.

$\square$

Now we present a proof of Theorem 5.

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

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

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}. $\square$

D Proof of security of $\pi_{\mathtt{SO}}^{n,k}$

Here we provide full mathematical details for the proof of Theorem 7.

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 $h(X) \stackrel{d}{=} \mathtt{EXEC}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{A}}(x,z)$ and $h(Y) \stackrel{d}{=} \mathtt{IDEAL}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{S}}(x,z)$. 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 $\mathtt{EXEC}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{A}}(x,z)$. The first messages that $\mathcal{A}$ receives are the t output messages $(z_i)_{i\in\mathcal{I}_c}$ from $\pi_{\mathtt{RZG}}^{n,k}$. 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 $\mathtt{EXEC}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{A}}(x,z)$ 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

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

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 $\mathtt{EXEC}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{A}}(x,z)$ 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 $h(X) \overset{d}{=} \texttt{EXEC}_{\pi_{\texttt{SO}}^{n,k},\mathcal{A}}(x, z)$. 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 $\texttt{EXEC}_{\pi_{\texttt{SO}}^{n,k},\mathcal{A}}(x, z)$ 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 $\texttt{EXEC}_{\pi_{\texttt{SO}}^{n,k},\mathcal{A}}(x, z)$, 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 $h(X) \overset{d}{=} \texttt{EXEC}_{\pi_{\texttt{SO}}^{n,k},\mathcal{A}}(x, z)$.

With this result established, we now seek to construct $\mathcal{S}$ and a random variable Y such that $h(Y) \overset{d}{=} \texttt{IDEAL}_{\pi_{\texttt{SO}}^{n,k},\mathcal{S}}(x, z)$ and $X \overset{d}{=} Y$. 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

$$c_i = c_i^{(y)} - c_i^{(x)} \quad \forall i \in [k-1]_0 \,. \tag{6}$$

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 $h(Y) \stackrel{d}{=} \mathtt{IDEAL}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{S}}(x,z)$. 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 $\mathtt{IDEAL}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{S}}(x,z)$ also by definition. Thus $h(Y) \stackrel{d}{=} \mathtt{IDEAL}_{\pi_{\mathtt{SO}}^{n,k},\mathcal{S}}(x,z)$.

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. (6), 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. □

E Proof of Security of $\pi_{\mathtt{RNG}}^{n,k}$

Here we provide full mathematical details for the proof of Theorem 11.

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

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

$$\left\{ \left(\left(r_i^{(j)}\right)_{i\in\mathcal{I}_h}\right)_{j\in[k-1]_0} \in \mathbb{F}^{k(n-t)} \,\middle|\, \begin{array}{c} \forall j \in [k-1]_0 \\ \exists (c'_l)_{l\in[k-1]} \\ \forall i \in \mathcal{I} \end{array} : r_i^{(j)} = c_j + \sum_{l=1}^{k-1} c'_l i^l \right\}.$$

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 $\texttt{IDEAL}_{\pi_{\texttt{RNG}}^{n,k},\mathcal{S}}(x, z)$. We want to show that $h(Y) \stackrel{d}{=} \texttt{IDEAL}_{\pi_{\texttt{RNG}}^{n,k},\mathcal{S}}(x, z)$. This is clearly true for all but the last element because of how $\texttt{IDEAL}_{\pi_{\texttt{RNG}}^{n,k},\mathcal{S}}(x, z)$ 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

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

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 $\pi_{\texttt{BRNG}}^{n,k}$ 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. $\square$

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 $S_{r_c^{(j)},\mathcal{I},\mathcal{I}_c}$. $\square$

This completes the proof. $\square$