Actively Secure Setup for SPDZ

Dragos Rotaru[1,2], Nigel P. Smart[2], Titouan Tanguy[2], Frederik Vercauteren[2], and Tim Wood[2]

[1] Cape Privacy, New York, U.S.A.
[2] imec-COSIC, KU Leuven, Leuven, Belgium.
dragos@capeprivacy.com, nigel.smart@kuleuven.be,
titouan.tanguy@kuleuven.be, frederik.vercauteren@esat.kuleuven.be,
twood1089@gmail.com

Abstract. We present the first actively secure, practical protocol to generate the distributed secret keys needed in the SPDZ offline protocol. As an added bonus our protocol results in the resulting distribution of the public and secret keys is such that the associated SHE 'noise' analysis is the same as if the distributed keys were generated by a trusted setup. We implemented the presented protocol for distributed BGV key generation within the `SCALE-MAMBA` framework. Our method makes use of a new method for creating doubly (or even more) authenticated bits in different MPC engines, which has applications in other areas of MPC-based secure computation. We were able to generate keys for two parties and a plaintext size of 64 bits in around five minutes, and a little more than eighteen minutes for a 128 bit prime.
This version contains differences from the published version due to an attack on the method in the published version discovered by Victor Treinsoutrot. In this revised version we describe a fix (due to Carsten Baum) which would result in a roughly two fold performance loss in our protocol. This revised version does not update the running times of the protocol precisely, but estimates the expected running time from extrapolating micro-benchmarks. The new expected run times are roughly two times slower than those in the previous version. This file contains, to aid the reader, first the new version and then a `diff` with the published version which appeard in Journal of Cryptology.

Keywords: MPC · Somewhat Homomorphic Encryption · Key Generation

1 Introduction

The SPDZ protocol for Multi-Party Computation (MPC) was introduced in 2012 [20]. This protocol is in the pre-processing family of protocols which are actively secure-with-abort for a dishonest majority of participants. Due to many improvements over the intervening years it provides a highly efficient mechanism to perform MPC for an arbitrary number of participants. However, the protocol comes with a major security issue: namely that it seems to require a trusted setup. This trusted setup is the creation of a public key for the Brakerski-Gentry-Vaikuntanathan [10] (BGV) homomorphic encryption scheme in which the private key is securely distributed amongst the n-parties.

In the original SPDZ paper [20] this was assumed to come from some trusted setup. In the follow up paper [18] a *covertly* secure protocol for generating a suitably distributed set of private keys, and the associated public key was introduced. However, this came with a number of disadvantages, as well as the reduction to just covert security. In particular the distributions of the underlying public keys were different from those one could attain via a

trusted setup, which led to a more complicated noise analysis, and (more importantly) larger parameters which results in a less efficient protocol overall.

In subsequent works the issue of the setup of the public key for the BGV encryption scheme has been dismissed as a setup assumption, which could either be performed in a live system using trusted hardware or via another MPC protocol. Given the complexity of the covertly secure key generation protocol from [18] it has always been assumed that actively secure key generation for SPDZ would require a *complex* MPC protocol to perform it.

The problem of producing distributed key generation for an SHE/FHE scheme arises in other contexts, and not just for the SPDZ setup operation. For example, inspired by Gentry's FHE based passively secure low round complexity MPC protocol from [25], various authors have looked at distributed decryption for FHE/SHE schemes. Each of these proposals needs a method to perform a distributed key generation phase. In [4] the authors present a key generation method for a distributed SHE scheme using various Σ-protocols. To our knowledge this has never been implemented, and the methodology produces a key generation which is different from what would be done via a trusted setup. In [35] a passively secure distributed key generation method is used for threshold SHE schemes, again producing a distribution different from that one would have in a purely trusted setup. Paper [14] introduces a variant of SPDZ, called SPDZ2k, for the case of MPC over the ring $\mathbb{Z}_{2^k}$. The offline phase in this latter work is based on MASCOT, [33]. However, in [37] the authors present an offline protocol based on distributed FHE for SPDZ2k. In all of these cases the contribution of this paper can aid in the creation of the necessary secret keys.

Our Contribution: In this paper we present the first practical *actively secure* distributed key generation method for SPDZ. As an added bonus our method results in virtually identical secret key distributions as in the trusted setup case. In particular the noise analysis for the resulting public key is identical to that one would have if using a trusted setup. Our protocol is also relatively simple, although it does make use of complex generic MPC technology. In particular, our protocol generates a public/private key with exactly the same underlying distribution as the ideal trusted setup does in the `SCALE-MAMBA` system[3], bar the fact we generate secret keys with *expected* Hamming weight h as opposed to *exact* Hamming weight h.

We are also able to generate secret keys from binomial distributions, which can be seen as approximate Gaussian error distributions. These do not suffer from the security concerns that low Hamming weight secret distributions have [16]. In addition, for our purposes, using such keys produces a faster distributed key generation procedure. The effect of using keys selected from a binomial distribution makes the ring parameters slightly bigger (compared to those from an (exact or expected) Hamming weight distribution), this decreases marginally the triple production throughput of the resulting offline phase. On the other hand such a choice has the beneficial effect (see later) of decreasing the runtime of our distributed key generation protocol by about a half. Thus one not only gets a faster key generation method, but the resulting keys do not suffer from the problems outlined in [16].

[3] We use `SCALE-MAMBA` as a reference work throughout this paper as it gives a fixed target (including key sizes) for the final distributed keys we are trying to produce.

Our protocol makes use of a generic MPC functionality for actively secure MPC-with-abort for dishonest majorities over a finite field. This might seem to imply that we require SPDZ to create SPDZ, however this circular dependency is removed by utilizing either the BDOZ protocol [8] or the SPDZ protocol executed with the MASCOT pre-processing phase [33]. The first of these, BDOZ, makes use of n public keys for a linear homomorphic encryption scheme where one private key is held by each player. The second option, MASCOT, is based on Oblivious Transfer. To avoid confusion we will refer to SPDZ with a MASCOT based pre-processing as MASCOT-SPDZ, whereas when we talk about SPDZ we mean the pure SPDZ protocol with a pre-processing based on homomorphic encryption.

Both of these *base* MPC protocols (BDOZ and MASCOT-SPDZ) are not as efficient as the SPDZ protocol based on homomorphic encryption, but we will only be using the base protocols for the one-time setup phase for SPDZ. In particular the underlying generic MPC protocol that we will use for key generation (MASCOT-SPDZ) is $O(n^2)$ in complexity; but we use this to create the distributed secret keys for an MPC protocol (SPDZ) which has complexity $O(n)$.

The overall construction of our protocol is based on four key observations; all of which are relatively simple. Firstly, the generation of the public key data given the secret key data and randomness for a BGV public key is essentially a linear operation and thus comes for free in LSSS based MPC protocols such as BDOZ and MASCOT-SPDZ. Secondly, the BGV public key for SPDZ is a two level BGV scheme thus the ciphertext modulus q needed to construct the BGV public key is a product of two values $q = p_0 \cdot p_1$. In particular the public key is simply the lift to modulo q of the public key modulo p_0 and p_1, performed via the Chinese Remainder Theorem (CRT). If we select p_0 and p_1 to be prime, as `SCALE-MAMBA` does, then we can use two MPC systems (one over p_0 and one over p_1) to perform the operations, and then obtain the final result via application of the CRT. We assume these two MPC systems come as ideal functionalities $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. In practice these will be implemented via MASCOT-SPDZ. Thirdly, all the random values required in BGV key generation can be boiled down to the generation of random bits, which are then processed in various ways. Thus a key issue is how to generate these random bits with the required distributions. Whilst BDOZ and MASCOT-SPDZ can be adapted to produce authenticated (uniformly distributed) random bits as part of their pre-processing, using much the same trick as proposed in [18], this will produce *different* random bits in $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Thus our fourth, and final, observation is that we can produce sharings of the same random bit in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ using an adaption of the daBit method from [3] and [39].

Indeed our new method for daBit is more general and more efficient than the method presented in [3,39]. We require a daBit method which works for two large primes, whereas [3,39] require a method for a large prime and a small prime (in particular the prime two). Our new method deals with any prime size for the two MPC engines, can be extended to more MPC engines than just two, and is built upon an abstraction which allows it to be used with any form of LSSS based MPC engine in the SPDZ family (e.g. BDOZ, MASCOT or SPDZ itself).

The most expensive part of the daBit generation procedure for producing daBits with active security in [3,39] was in verifying consistency of the daBits between the two instances of $\mathcal{F}_{\mathsf{MPC}}$. The idea in these works was to check the same random linear combinations of bits in both instances simultaneously, which was challenging because one field had characteristic 2 and the other some large prime p, which meant the XOR had to be emulated in the prime field, requiring multiplication in MPC. In addition, to generate a bit, one needed to perform XOR in both of the fields under consideration. Our observation in this work is that we use an auxiliary MPC engine for a large prime p to generate bits using the standard square-root trick for generating bits, these are then mapped into the target MPC engines. The auxiliary MPC engine is used to obtain a masked subset sum over the integers, which is then compared to the equivalent masked subset sum in the target MPC engines. Security now reduces to a standard application of the Smudging Lemma.

We decided to use MASCOT-SPDZ as the underlying MPC protocol for the BGV key generation, with our implementation building upon the already existing code-base for OT present in the `SCALE-MAMBA` framework. In addition we ran experiments with different values for the standard deviation of the centred binomial distribution, and experiments between Hamming weight restricted secret keys and secret keys generated from a centred binomial distribution. In the fastest case, of standard deviation $\sigma = \sqrt{1/2} = 0.707$ for the centred binomial distribution and FHE keys distributed following this same distribution, for two parties and a 128 bit plaintext modulus our results show that we can distributively generate BGV keys in around 40 minutes. Our original experiments were for a related (insecure) protocol; and achieved 18.5 minutes. Using the modified (secure) method in this version the time would be roughly twice that, i.e. 40 minutes. We ran experiments for 64 and 128 bit primes for the plaintext space for two and three parties for all settings; and for our fastest settings we also ran experiments for four and five parties.

For the parameters used in versions of `SCALE-MAMBA` prior to our work, which used $\sigma = \sqrt{10} = 3.16$ and Hamming weight limited secret keys, we find a key generation time of approximately five hours for two parties and a 128 bit modulus. We give a detailed report of our implementation in Section 7, in which the triple generation throughput and the shared bit throughput are given for the standard case and wall clock time for the whole protocol is given for all our test cases. We note that a variant of our protocol has now been incorporated into the MP-SPDZ software [30] as well as into `SCALE-MAMBA`.

We end this introduction by noting that in [9] a method to perform the SPDZ offline phase using no-communication is presented. However, this method is impractical as currently presented. The method still requires a distributed decryption capability of the underlying SHE scheme. Thus to use this work, even in theory, one needs to be able to generate such distributed keys in a secure manner, such as this work enables. We also note that using the silent-OT method of [9] one may be able to achieve better runtimes. The paper reports that they can achieve 600,000 correlated OT's per second. However, due to the increased computational costs of the silent-OT method this might not translate to the LAN setting in our experiments.

2 Historical Context of SPDZ

For those readers wishing to understand the historical context of SPDZ we now provide some background. For readers interested simply in the results of this paper, they can safely skip this section.

The first application of threshold-like cryptographic schemes to build MPC protocols appears to be the work of Franklin and Haber [22,23]. This utilized an ElGamal style encryption scheme over an RSA-style group to evaluate an MPC protocol over binary circuits. The scheme has, to our knowledge, never been implemented (it would be very inefficient by modern standards) and provides only passive security.

The Franklin-Haber protocol used a so-called 'joint encryption' scheme. True threshold encryption was introduced into MPC protocols by Cramer, Damgård and Nielsen in [15]. This protocol was based on threshold homomorphic encryption and provided a (robust) actively secure protocol in the honest majority setting. The underlying protocol made use of Pailliers encryption scheme, or a variant of the scheme of Franklin and Haber. As such it required a set up phase of generating a secret shared RSA style private key; at the time this was relatively complex but can now be done using the techniques in [13]. The protocol is interesting as it utilizes a trick of keeping an encryption of the data a, as well as $\alpha \cdot a$, for some user chosen key α. Thus this is reminiscent of the usage of the pair-wise MACs in the BDOZ protocol (see below). The overall multiplication protocol required use of public key machinery, and can thus also be considered inefficient by todays standards.

Also in this period the VIFF system [17] developed the pre-processing model for MPC protocols, in VIFF's case for honest-majority protocols. This enabled all the difficult public key machinery to be placed in the offline phase, making the online phase more efficient. The work of [29] formalised the pre-processing model and investigated communication complexity upper and lower bounds in this model.

The modern era of MPC protocols in the case of dishonest majority can be said to start with the work of [19] which showed how to achieve active security for dishonest majority protocols for a small constant additional amount of work. This then led to the information theoretic online phase in the BDOZ [8] and TinyOT [36] protocols. The BDOZ protocol is an n-party dishonest majority protocol for MPC over arithmetic circuits, whilst TinyOT is a two party protocol for binary circuits. Both achieve an essentially information theoretic online phase (the phase where the function is actually evaluated) using information theoretic MACs; the underlying MACs and commitments having been previously introduced ten years earlier in an unpublished manuscript of Rivest [38].

The BDOZ paper followed along the blue-print outlined by Cramer et al. in the case of honest majority, but made a number of crucial changes. Firstly, (as we just said) active security was *cheaply* obtained by using (pair-wise) information theoretic MACs (so called BDOZ-style MACs). Secondly, an efficient online phase was obtained by using the Beaver circuit randomization technique [6]. Thirdly, the Beaver-triples where produced using a linearly homomorphic encryption scheme. The overall protocol gave a complexity of $O(n^2)$, but could be made highly efficient using a linear homomorphic encryption scheme based on

Ring-LWE, which allowed Smart-Vercauteren style SIMD packing to enable a high triple production throughput.

Following closely on the heels of the BDOZ protocol was the SPDZ protocol [20]. This followed much the same methodology, but it replaced the pair-wise information theoretic MACs with global information theoretic MACs (so called SPDZ-style MACs). By using a star-like topology, for MAC-checking and data transfer during the online phase, a total complexity of $O(n)$ could be achieved. The other main advance of SPDZ was the use of a *threshold* Somewhat Homomorphic Encryption scheme of depth two to produce the Beaver triples, as opposed to the linear homomorphic scheme of BDOZ. The use of SHE enabled some of the zero-knowledge proofs required in BDOZ to be removed. Again, by using SIMD packing in a Ring-LWE system, high triple production throughput could be achieved. The Ring-LWE threshold SHE was performed utilizing the techniques developed in [7] for generic lattice-based threshold schemes. However, the use of threshold SHE required a trusted setup assumption (to set up the threshold SHE keys). This issue was side-stepped in the paper [20], and all subsequent papers on SPDZ, until the current one.

Over the years various improvements have been suggested for SPDZ. The most relevant for our discussion are the following ones. The paper [18] introduced various simplifications to the basic SPDZ protocol; the two most important (from a practical perspective) was making it able to compute reactive computations, and replace the depth two SHE scheme with a depth one variant. From the point of view of the current paper, the main contribution of [18] was to try to solve the problem of the trusted setup. It did this by providing only a *covertly* secure key generation protocol for the threshold SHE keys, and in addition the resulting distribution of the underlying noise (inherent in any SHE style key) was less than ideal. This lead to larger parameters for the SHE scheme, and thus a slower protocol.

The next variant of SPDZ developed was to replace the SHE based offline phase with one based on Oblivious Transfer [33]. This offline phase has complexity $O(n^2)$, due to the need for pairwise Oblivious Transfers. But for small n it was seen, for a short while, as being more efficient than the SHE based offline phase. However, in [34] it was shown that, by providing more efficient zero-knowledge proofs in the case of the SHE-based variants, the OT-based variants were no longer competitive. The paper [34] provided two forms of pre-processing for SPDZ. One based on linearly homomorphic encryption (essentially adapting BDOZ style pre-processing for the SPDZ situation) which was efficient for small n. This variant was called LowGear. A second variant, for large n, utilized a joint zero-knowledge proof, as opposed to pair-wise proofs in [18], to obtain a performance boost for SHE based pre-processing. This latter variant was denoted HighGear.

In [5] an improvement to HighGear was made, denoted TopGear, which improved the zero-knowledge proofs in the SHE based pre-processing. TopGear is more efficient than HighGear, but there is no experimental evidence as to whether the TopGear improvements mean LowGear is no longer competitive for small n. This is perhaps a direction for future investigation.

Thus the only outstanding issue regarding SPDZ is the fact that there is no methodology to generate the required threshold SHE keys. The problem is to ensure the underlying

distributions of the secret keys are correct and do not create performance problems later on. This is the main technical contribution of this paper; to obtain active security we utilize the SPDZ system as the MACs to obtain active security introduce very little additional cost over a passively secure variant. Given an efficiently passively secure protocol for this task, one can easily generate (theoretically) an actively secure one by utilizing zero-knowledge proofs. However, by utilizing a sub-protocol such as MASCOT-SPDZ we achieve the additional active security virtually 'for free' via the use of the SPDZ MACs.

3 Preliminaries

In this section we provide the necessary background on the type of BGV public key we need to produce, as well as the underlying distributions and the base MPC protocols we will be using.

3.1 Cyclotomic Rings and Distributions over such Rings

The BGV encryption scheme is defined over a cyclotomic ring $R = \mathbb{Z}[X]/(X^N + 1)$, where for our purposes we take N to be a power of two. Thus $X^N + 1$ is the $m = 2 \cdot N$-th cyclotomic polynomial, and $N = \phi(m)$. We let $\odot$ denote the multiplication operation in R.

Following [26][Full version in [27], Appendix A.4] the `SCALE-MAMBA` system utilizes the following distributions in the key generation procedure.

- $\mathsf{HWT}(h, N)$: This generates a vector of length N with elements chosen at random from $\{-1, 0, 1\}$ subject to the condition that the number of non-zero elements is equal to h.
- $\mathsf{dN}(\sigma^2, N)$: This generates a vector of length N with elements chosen according to an *approximation* to the discrete Gaussian distribution with variance σ^2, by sampling from a centred binomial distribution.
- $\mathsf{U}(q, N)$: This generates a vector of length N with elements generated uniformly modulo q.

In particular for the distribution $\mathsf{dN}(\sigma^2, N)$ `SCALE-MAMBA` approximates $\mathsf{dN}(\sigma^2, N)$ using the approximation from [1]. In particular $\mathsf{dN}(\sigma^2, N)$ is replaced by the centred binomial distribution where elements are returned using the formula

$$c_j = \sum_{i=0}^{k-1} b_{2 \cdot i} - b_{2 \cdot i + 1}$$

for uniformly random bits $b_j \in \{0, 1\}$ for $j = 0, \ldots, 2 \cdot k - 1$. The default settings of `SCALE-MAMBA` use $k = 20$, giving us $\sigma = \sqrt{k/2} = \sqrt{10} = 3.16$.

We make a small change to one of the above distributions in our work. The distribution $\mathsf{HWT}(h, N)$ is used to sample the secret key, where in [26] (and in `SCALE-MAMBA`) the value h is selected to be a power of two; in particular $h = 64$. In our work we replace $\mathsf{HWT}(h, N)$ with the distribution which picks each coefficient with respect to the Bernoulli distribution $B(h/N)$. Thus we use the approximation $\mathsf{HWT}(h, N) \approx B(h/N)^N$. The Hamming weight of the vectors output by this distribution follows a binomial distribution with mean h. We

still use $h = 64$ in our recommended construction though. The "noise analysis" behind the homomorphic operations used in the SPDZ protocol are easily checked not to be affected by this change, and in addition the security arguments for using low Hamming weight secret keys (as discussed in [26]) are also not affected. In particular the noise analysis used in [26], or `SCALE-MAMBA`, is an 'average case' analysis in the key generation, thus the standard deviation in the canonical norm of the secret key is $\sqrt{h}$ if an exact Hamming weight of h is used. It is this standard deviation which is the contributing term in the noise analysis. If one generates the secret key using only an expected Hamming weight then you obtain the same standard deviation; thus nothing changes in the analysis by using our slightly different secret key distribution.

3.2 The BGV Key Generation Procedure

For a modulus q we let R_q denote the above ring localised at the modulus q, i.e. $R_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N + 1)$. The SPDZ protocol requires a two-leveled scheme with ciphertext moduli $q_0 = p_0$ and $q_1 = p_0 \cdot p_1$. The plaintext modulus for the BGV scheme is defined by p. We require, for efficiency, that

$$\begin{aligned} p_1 &\equiv 1 \pmod{p}, \\ p_0 - 1 \equiv p_1 - 1 \equiv p - 1 &\equiv 0 \pmod{N}, \end{aligned}$$

where p is the plaintext modulus. The moduli p_0 and p_1 are selected to be distinct primes for efficiency. In which case, by the CRT, we have $R_q \cong R_{p_0} \times R_{p_1}$. In addition, due to the above restrictions on the primes p_0 and p_1, there is an efficient FFT algorithm on R_{p_i}, which requires no extension field arithmetic. Thus one can efficiently multiply in R_{p_i} by executing

$$\mathbf{a} \odot \mathbf{b} = \mathsf{FFT}^{-1}(\mathsf{FFT}(\mathbf{a}) \cdot \mathsf{FFT}(\mathbf{b}))$$

where $\cdot$ here is the component wise product. Note that the FFT operation is a *linear* operation and thus can be executed in an MPC engine for free. These facts we shall use in our distributed key generation protocol.

The two ciphertext moduli q_0 and q_1 are used to define ciphertexts at level zero and level one, see [26] for further details as to how this aids efficiency. Since SPDZ only requires a depth one SHE scheme we only need to define two such ciphertext moduli.

The BGV public key is of the form $(\mathbf{a}, \mathbf{b}) \in R_q$ where

$$\mathbf{a} \leftarrow \mathsf{U}(q, N) \quad \text{and} \quad \mathbf{b} = \mathbf{a} \odot \mathfrak{sk} + p \cdot \mathbf{e}$$

where $\mathbf{e} \leftarrow \mathsf{dN}(\sigma^2, N)$. The secret key $\mathfrak{sk}$ for our purposes will be selected from the distribution $B(h/N)^N$. We also require, for the SPDZ protocol, the switching key data $(\mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}, \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2})$ which is of the form

$$\mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{U}(q, N) \quad \text{and} \quad \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2} = \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2} \odot \mathfrak{sk} + p \cdot e_{\mathfrak{sk},\mathfrak{sk}^2} - p_1 \cdot \mathfrak{sk}^2$$

where $\mathbf{e}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{dN}(\sigma^2, N)$.

The goal in a distributed key generation protocol for the SPDZ system is to output the public values $\mathfrak{pk} = (\mathbf{a}, \mathbf{b}, \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}, \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2})$ to all players, whilst player P_i obtains a value $\mathfrak{sk}_i \in R_q$ such that

$$\mathfrak{sk} = \mathfrak{sk}_1 + \ldots + \mathfrak{sk}_n \pmod{q}.$$

We also require that no party can influence the choice of secret key, and no proper subset of the n parties can deduce any information about the secret key, bar what can be deduced from the public key. Thus we aim to create a protocol which securely realizes the functionality given in Figure 1, where $\mathsf{ParamGen}(1^\kappa, \log_2 p, n)$ is a function which produces the system parameters (p, p_0, p_1). We allow all parties, bar P_1, to input their precise share of the secret key. We then compute P_1's share according to the shares of other parties and the internally sampled secret key $\mathfrak{sk}$. Therefore P_1 has no input, and is treated differently from other parties. This is mirrored in our protocol, the parties generate a secret key but the *shares* of the parties $P_2, \ldots, P_n$ are selected by those parties; with party P_1 receiving their share from the protocol.

Functionality $\mathcal{F}_{\mathsf{KeyGen}}$

1. When receiving the message *start* from all honest parties, run $P \leftarrow \mathsf{ParamGen}(1^\kappa, \log_2 p, n)$, and then, using the parameters generated, run $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}()$ (P, and hence 1^κ, is an implicit input to all functions we specify in this paper). Send $\mathfrak{pk} = (\mathbf{a}, \mathbf{b}, \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}, \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2})$ to the adversary.
2. Receive from the adversary a set of shares $\mathfrak{sk}_j \in R_q$ for each corrupted party P_j for $j \neq 1$.
3. Construct a complete set of shares $(\mathfrak{sk}_1, \ldots, \mathfrak{sk}_n)$ consistent with the adversary's choices and $\mathfrak{sk}$. This is done by selecting $\mathfrak{sk}_i$ uniformly at random for honest i, subject to the constraint that $\mathfrak{sk} = \sum \mathfrak{sk}_i$. Note that this is always possible since the corrupted players form an unqualified set.
4. The functionality waits for an input from the adversary.
5. If this input is $\mathsf{Deliver}$ then send $\mathfrak{pk}$ to all players and $\mathfrak{sk}_i$ to each honest player P_i, and send $\mathfrak{sk}_1$ to player P_1 if P_1 is dishonest.
6. If the adversarial input is not equal to $\mathsf{Deliver}$ then abort.

Figure 1. The ideal functionality for key generation (adapted from [20]).

We aim to realize this functionality while doing only black box calls to the underlying MPC modulo p_0 and p_1 functionalities. However, due to the concerns raised in [16] in relation to low Hamming weight keys, we also examine the case of secret keys generated by a centred binomial distribution; namely when we select $\mathfrak{sk}$ from $\mathsf{dN}(\sigma^2, N)$. These lead to slightly larger parameters for the underlying FHE systems, but the method to produce the keys is simpler.

3.3 Base MPC Protocols

In Figure 2 we present the MPC functionality for our base MPC protocols, either BDOZ or MASCOT-SPDZ in the case where we are generating keys or SPDZ when we are doing traditional daBit generation.

To simplify presentation of protocols using this functionality we shall represent a value held in the memory of such an MPC functionality by $\langle x \rangle_p$, and then addition and multiplication of such elements will be represented by

$$\langle x \rangle_p + \langle y \rangle_p, \qquad \langle x \rangle_p \cdot \langle y \rangle_p.$$

For inputing and outputting values to/from a player/all players we will write

$$\langle x \rangle_p \leftarrow \mathsf{Input}(P_i), \qquad P_i \leftarrow \mathsf{Output}(\langle x \rangle_p), \qquad x \leftarrow \mathsf{Open}(\langle x \rangle_p).$$

That the BDOZ, MASCOT-SPDZ and SPDZ protocols implement such a functionality securely can be found proved in the respective papers [8], [33] and [20].

Functionality $\mathcal{F}^p_{\mathsf{MPC}}$

The functionality runs with parties $P_1, \ldots, P_n$ and an ideal adversary. Let $\mathcal{A}$ be the set of corrupt parties. Given a set I of valid identifiers, all values are stored in the form $(varid, x)$, where $varid \in I$.

Initialize: On input $(Init, p)$ from all parties, with p a prime, the functionality stores p. The adversary is assumed to have statically corrupted a subset $\mathcal{A}$ of the parties.
Input: This takes input $(Input, P_i, varid, x)$ from P_i, with $x \in \mathbb{F}_p$, and $(input, P_i, varid, ?)$ from all other parties, with $varid$ a fresh identifier. If the $varid$'s are the same the functionality stores $(varid, x)$, otherwise it aborts.
Add: On command $(Add, varid_1, varid_2, varid_3)$ from all parties:
1. If $varid_1, varid_2$ are not present in memory or $varid_3$ is then the functionality aborts.
2. The functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x + y)$.

Multiply: On input $(Multiply, varid_1, varid_2, varid_3)$ from all parties:
1. If $varid_1, varid_2$ are not present in memory or $varid_3$ is then the functionality aborts.
2. The functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y)$.

Output: On input $(Output, varid, i)$ from all parties (if $varid$ is present in memory),
1. The functionality retrieves $(varid, y)$.
2. If $i = 0$ and $\mathcal{A} \neq \emptyset$ then the functionality outputs y to the adversary, otherwise it outputs $\perp$ to the adversary.
3. The functionality waits for an input from the adversary.
4. If this input is $\mathsf{Deliver}$ then y is output to all players if $i = 0$, or y is output to player i if $i \neq 0$.
5. If the adversarial input is not equal to $\mathsf{Deliver}$ then abort.

Figure 2. The ideal functionality for MPC with abort over $\mathbb{F}_p$.

The underlying authentication mechanism, in practice, can be assumed to be one of SPDZ style MACs, i.e. there is a globally held shared secret value $\alpha_p \in \mathbb{F}_p$, shared with respect to an additive sharing amongst the n-players $\alpha_p = \alpha_{1,p} + \cdots + \alpha_{n,p}$. For each value x the $\langle x \rangle_p$ will be an additive sharing of $x = x_1 + \cdots + x_n$, as well as an additive sharing of $\alpha_p \cdot x = \gamma_1 + \cdots + \gamma_n$. In other words in the real world when a party P_i holds the value $\langle x \rangle_p$, it actually holds the value (x_i, γ_i).

In such MPC protocols addition, and in fact any linear operation, is a 'free' operation, whereas multiplication will be assumed to take a single 'time' unit of operation. Another metric one often examines is the round complexity, in this case a multiplication takes one round, but multiplications which can be performed in parallel also only take one round of operation. Inputing, outputting or opening a data item also requires one round of communication, and such operations can be performed in parallel.

3.4 The $\mathcal{F}^B_{\mathsf{Rand}}(M)$ Functionality

We also require a functionality $\mathcal{F}^B_{\mathsf{Rand}}(M)$ which allows the parties to agree on M random values in the range $[0, \ldots, B)$. In practice this can be implemented by all parties committing

to a seed, then the parties open the seeds. The seeds are then XOR'd together to produce a single shared seed, which is passed as the key to a PRF to produce the shared random values. We present this as an ideal functionality in Figure 3.

Functionality $\mathcal{F}^{B}_{\mathsf{Rand}}(M)$

1. On input $(\mathsf{Rand}, \text{cnt})$ from all parties, if the counter value is the same for all parties and has not been used before, the functionality samples $r_i \leftarrow [0, \ldots, B)$ for $i = 1, \ldots, M$.
2. The values r_i are sent to the adversary, and the functionality waits for an input.
3. If the input is $\mathsf{Deliver}$ then the values r_i are sent to all parties, otherwise the functionality aborts.

Figure 3. The ideal $\mathcal{F}^{B}_{\mathsf{Rand}}(M)$ functionality.

4 maBits: Generating Multiply Authenticated Bits

The main problem in performing actively secure key generation for SPDZ is to produce secure randomly shared bits within the MPC functionalities; in which the bit is zero with probability 1/2, and one with probability 1/2. We will require that the MPC functionality $\mathcal{F}^{p}_{\mathsf{MPC}}$, for a specific prime p, is extended by a command which we model via the ideal functionality $\mathcal{F}^{p}_{\mathsf{MPC}}.\mathsf{GenBit}()$ given in Figure 4. The reader should note that, the adversary is not able to choose their shares for the shared random bit in this functionality. This is important for our security proof.

Functionality $\mathcal{F}^{p}_{\mathsf{MPC}}.\mathsf{GenBit}()$

1. The functionality waits for a message abort or ok from the adversary. If the message is ok then it continues.
2. The functionality then samples a bit $b \in \{0, 1\}$ and stores it in the functionality $\mathcal{F}^{p}_{\mathsf{MPC}}$.
3. A sharing (b_i, γ_i) of the bit b is generated, and the adversaries share values are returned to it.

Figure 4. The ideal functionality for single random bits.

A protocol for BDOZ and MASCOT-SPDZ for $\mathcal{F}^{p}_{\mathsf{MPC}}.\mathsf{GenBit}()$ is given in Figure 5, adapted from [18]. The basic idea is to generate a secret random value x modulo p, square it, open the value, take the square root (if it is not zero) so as to obtain $z = \pm x$. Then a value in $\{-1, 1\}$ is obtained by computing x/z, which can then be mapped onto the any desired two element set[4].

However, we have a problem; if we execute this procedure with respect to both $\mathcal{F}^{p_0}_{\mathsf{MPC}}$ and $\mathcal{F}^{p_1}_{\mathsf{MPC}}$ then we will obtain two shared random bits $\langle b_0 \rangle_{p_0}$ and $\langle b_1 \rangle_{p_1}$ but we will not necessarily have $b_0 = b_1$. To obtain shared random bits in the two MPC systems which are *identical* we need to adapt the daBit idea from [39]. In that paper it is shown how to obtain identical

[4] If the underlying MPC system is SPDZ based then a more efficient way to perform the method is using the FHE pre-processing instead of directly within the Offline phase as implied by the given protocol. But this assumes the pre-processing is FHE based, which it will not be in our application.

Entering a Random Bit $\Pi^p_{\mathsf{MPC}}.\mathsf{GenBit}()$

1. For $i = 1, \ldots, n$ execute $\langle x_i \rangle_p \leftarrow \mathsf{Input}(P_i)$, where x_i is a random element in $\mathbb{F}_p$.
2. $\langle x \rangle_p \leftarrow \sum_{i=1}^{n} \langle x_i \rangle_p$.
3. $\langle y \rangle_p \leftarrow \langle x \rangle_p \cdot \langle x \rangle_p$.
4. $y \leftarrow \mathsf{Open}(\langle y \rangle_p)$.
5. If $y = 0$ then restart the process.
6. $z \leftarrow \sqrt{y} \pmod{p}$, picking the value $z \in [0, \ldots, p/2)$.
7. $\langle a \rangle_p \leftarrow \langle x \rangle_p / z$.
8. $\langle b \rangle_p \leftarrow (\langle a \rangle_p + 1)/2$.
9. Return $\langle b \rangle_p$.

Figure 5. 'Standard' method to produce a shared random bit in Π^p_{MPC}ssuming p is odd.

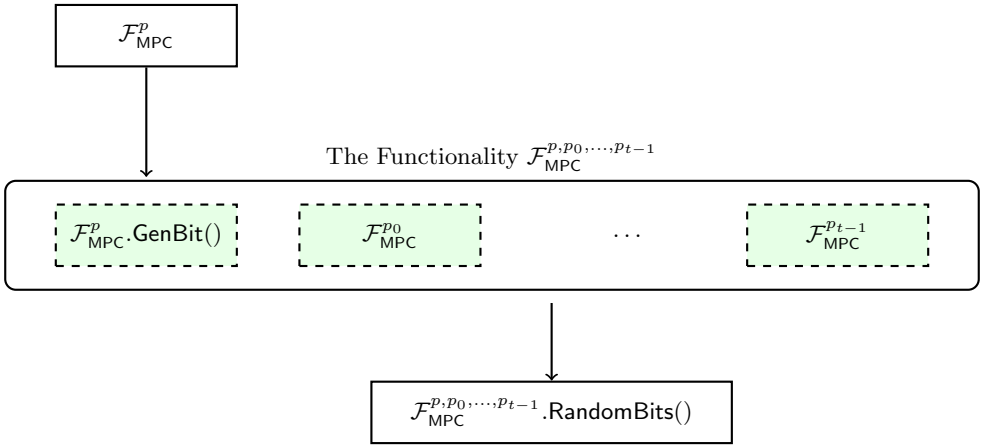

Fig. 6. Our functionalities.

shared random bits in two MPC systems; one being a SPDZ-like system modulo p, and one being a BDOZ-like system over $\mathbb{F}_2$ based on OT for garbled circuit style computations. In our key generation protocol we require shared random bits in two SDPZ-like systems for large moduli p_0 and p_1 (in practice MASCOT-SPDZ systems). This makes the protocol to generate the shared random bits a little easier to understand than the one considered in [39]. Indeed we present a more general protocol than that which is needed for our key generation method. Our new method includes the case considered in [39], and is more efficient than the improved method considered in [3] when at least one prime is 'large'.

For our generalisation we consider a set of t SPDZ-like MPC systems with moduli $p_0, \ldots, p_{t-1}$. Our goal is to generate shares $\langle b \rangle_{p_i}$ in all of these systems where $b \in \{0, 1\}$. Our method makes no restriction on the size of the primes p_i, nor the underlying SPDZ-like MPC engine, thus our method can be used as a replacement for the daBit methods in [3,39] as well.

Security of our protocol is defined by two security parameters sec and stat. The value sec is such that the probability of an adversary breaking the soundness of the protocol is $2^{-\mathsf{sec}}$, whereas stat is such that the statistical difference between the execution in the real and ideal

worlds is bounded by $2^{-\mathsf{stat}}$. The reader can think of $\mathsf{sec} = 128$ and $\mathsf{stat} = 80$ as is common in the literature.

We define p_{min} to be $\mathsf{min}(p_0, \ldots, p_{t-1})$ and we let γ be the smallest integer such that $p_{\mathsf{min}}^{\gamma} > 2^{\mathsf{sec}}$. In practical key generation for SPDZ we will have $\gamma = 1$, but we keep the discussion general to enable other applications of the technique in this section.

For efficiency we will generate the shared bits in batches of m at a time. We define a masking parameter W which satisfies

$$\lceil W/p_{\mathsf{min}} \rceil < W \cdot 2^{-\mathsf{sec}/\gamma}.$$

We set $T = \lceil \mathsf{stat} + \log_2 W + \log_2 m \rceil$, and define an auxiliary prime number p which satisfies

$$p > m \cdot W \cdot 2^{\mathsf{stat}} = 2^T.$$

The prime p can be the same as one of the primes p_i above if one of the p_i are large enough, but in this treatment for ease of exposition we treat it as a separate prime. It is with this prime p that we will generate our initial bits, using the protocol in Figure 5.

Thus we have $t+1$ independent MPC functionalities which we group into a single functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}$, as depicted in Figure 6. On their own, at first sight, these $t+1$ functionalities would not allow us to generate a shared random bit modulo $p_0, \ldots, p_{t-1}$. To obtain this extra ability we further extend the functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}$ to $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$; where the additional command is presented in Figure 7. The functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$ internally generates a set of m secure bits.

Functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$

1. The functionality waits for input of a positive integer m from all players. If the inputted values are not equal then the functionality aborts.
2. For $i = 1, \ldots, m$ the functionality executes the code of $\mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}()$ so as to generate a bit b_i.
3. The bits b_i are retrieved from $\mathcal{F}_{\mathsf{MPC}}^{p}$ and are enterred into the functionalities $\mathcal{F}_{\mathsf{MPC}}^{p_j}$ for $j = 0, \ldots, t-1$.
4. The functionality waits for a message abort or ok from the adversary. If the message is ok then it continues.

Figure 7. The ideal functionality for random bits.

Our protocol will make use of the following result

Lemma 4.1. *Let p be a prime, $n \in \mathbb{Z}_{\geq 2}$ and $x_i \in [0, \ldots, p)$ be uniformly chosen subject to the condition that*

$$x_1 + \ldots + x_n = \begin{cases} k \cdot p, & or \\ k \cdot p + 1. \end{cases}$$

Set $\Delta = \lceil p/n \rceil$ and write $x_i = l_i + \Delta \cdot h_i$ with $0 \leq l_i < \Delta$, then

$$k = \left\lceil \frac{\Delta \cdot \sum h_i}{p} \right\rceil.$$

with probability at least $1 - 3/p$.

Proof. We have, for $\epsilon \in \{0,1\}$,

$$k = \frac{\Delta \cdot \sum h_i}{p} + \frac{\sum l_i}{p} - \frac{\epsilon}{p}.$$

We have $0 \leq \sum l_i < p$ by construction, and so the equality on k will follow as long as $\sum l_i \geq \epsilon$. But this always happens unless $\epsilon = 1$ and $\sum l_i = 0$. If $l_i = 0$ then this means that Δ divides x_i. Since x_i is uniformly chosen in $[0, \ldots, p)$ for $i = 1, \ldots, n$ (subject to the condition that $\sum x_i \pmod p \in \{0,1\}$), and $\Delta = \lceil p/n \rceil$ we have $l_i = 0$ with probability $(n+1)/p$ for each $i \in [0, \ldots, n)$. Thus the probability that all the l_i are zero is less than $((n+1)/p)^{n-1}$, this is at most $3/p$. □

In Figure 8 we explain our protocol $\Pi_{\mathsf{RandomBits}}$ for producing shared random bits in the two MPC systems. Intuitively the protocol works as follows. The parties first generate, in step 2a, $m + \gamma \cdot T$ shared random bits in the MPC engine $\mathcal{F}^p_{\mathsf{MPC}}$ using the command GenBit. They then allow, in steps 2c to 2e, player P_1 to determine the associated k value for each shared bit using Lemma 4.1, this does not reveal any information about the hidden bit (and will be correct with probability $3/p$), but this clearly reveals some information about the sharing[5]. Thinking of the sharing now as over the integers, and then reducing modulo p_i, player P_1 can adjust his sharing so that the bit is correctly shared modulo p_i. These shares are then input into the MPC engines $\mathcal{F}^{p_i}_{\mathsf{MPC}}$. Assuming all parties are honest we now have a valid sharing, however due to Lemma 4.1 even honest parties have a probability of aborting of $3 \cdot (m + \gamma \cdot T)/p$, due to one, or more, of the k_i values being computed incorrectly.

To cope with dishonest parties we check the parties are honest by verifying random linear combinations. We verify γ random linear combinations in total. This is the purpose of steps 4 and 5. Here we note that the initial sharing in $\mathcal{F}^p_{\mathsf{MPC}}$ is guaranteed to be a sharing of a bit due to the active security of the operation GenBit in $\mathcal{F}^p_{\mathsf{MPC}}$. Opening a random linear combination S (in step 5d) of the shared bits in $\mathcal{F}^p_{\mathsf{MPC}}$ is then a subset-sum over the integers, due to the lower bound on p. We then compare this to the associated sum modulo p_i obtained from $\mathcal{F}^{p_i}_{\mathsf{MPC}}$ in step 5f. This has to be repeated γ times to cope with the smallest value of p_i. Due to the masking with a uniformly random value in $[0, \ldots, 2^T]$ the original subset sum on the first m bits (which is of size at most $2^{T-\mathsf{stat}}$) is statistically hidden.

The protocol $\mathcal{F}^p_{\mathsf{MPC}}.\mathsf{GenBit}()$ in Figure 5 requires one secure multiplication (per bit generated) and two rounds of communication (as a multiplication also requires a round of communication). To execute the rest of $\Pi_{\mathsf{RandomBits}}$ requires four rounds of communication (one for the initial opening to P_1, one for input into the MPC engines, one for executing $\mathcal{F}^W_{\mathsf{Rand}}$ and one for the final opening). If the $m + \gamma \cdot T$ bits required in $\Pi_{\mathsf{RandomBits}}$ are produced in parallel, as well as the various input/open operations etc, this means that protocol $\Pi_{\mathsf{RandomBits}}$ requires $m + \gamma \cdot T$ secure multiplications in $\mathcal{F}^p_{\mathsf{MPC}}$ and $2 + 4 = 6$ rounds of communication.

Security of $\Pi_{\mathsf{RandomBits}}$ then follows from the following theorem.

[5] In our security proof we show that this specific information can be perfectly simulated by the simulator, and leaks no information about the actual shared value.

Protocol $\Pi_{\mathsf{RandomBits}}$

This protocol works in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}$-hybrid model, and on input of m from all players executed the following steps:

1. Set $\Delta = \lceil p/n \rceil$ and $T = \lceil \mathsf{stat} + \log_2 W + \log_2 m \rceil$.
2. For $i = 1, \ldots, m + \gamma \cdot T$ do
   (a) $\langle b_i \rangle_p \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}()$.
   (b) Let $b_{i,\ell}$ denote Player ℓ's value such that $\sum b_{i,\ell} = b_i \pmod p$.
   (c) Player P_ℓ writes $b_{i,\ell} = l_{i,\ell} + \Delta \cdot h_{i,\ell}$ with $0 \le l_{i,\ell} < \Delta$.
   (d) Player P_ℓ sends $h_{i,\ell}$ to Player P_1.
   (e) Player P_1 sets
   $$k_i = \left\lceil \frac{\Delta \cdot \sum_\ell h_{i,\ell}}{p} \right\rceil.$$
   (f) For $v = 0, \ldots, t-1$ do
      - Party P_1 calls $\langle b_i^{(1)} \rangle_{p_v} \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p_v}.\mathsf{Input}(b_{i,1} - k_i \cdot p \pmod{p_v})$.
      - Party P_ℓ, for $\ell \neq 1$, calls $\langle b_i^{(\ell)} \rangle_{p_v} \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p_v}.\mathsf{Input}(b_{i,\ell} \pmod{p_j})$.
      - The parties compute $\langle b_i \rangle_{p_v} = \sum_{\ell=1}^{n} \langle b_i^{(\ell)} \rangle_{p_v}$.
3. Divide the shared bits $\langle b_i \rangle_{p_v}$ for $i \ge m$ into γ sets of size T; denoting the shared bits as $\langle b_{j,k} \rangle_{p_v}$ for $j = 1, \ldots, \gamma$ and $k = 1, \ldots, T$.
4. The parties initialize an instance of the functionality $\mathcal{F}_{\mathsf{Rand}}^{W}$. [Note, this needs to be done after all the previous steps, so the parties have no prior knowledge of the output].
5. For $j = 1, \ldots, \gamma$ do
   (a) For $i = 1, \ldots, m$ generate $r_{i,j}$ via $(r_{i,j})_i \leftarrow \mathcal{F}_{\mathsf{Rand}}^{W}(m)$.
   (b) Compute for $v = 0, \ldots, t-1$ the sharings $\langle S_{j,v} \rangle_{p_v} = \sum_{i=1}^{m} r_{i,j} \cdot \langle b_i \rangle_{p_v} + \sum_{k=1}^{T} 2^{k-1} \cdot \langle b_{j,k} \rangle_{p_v}$.
   (c) Compute $\langle S_j \rangle_p = \sum_{i=1}^{m} r_{i,j} \cdot \langle b_i \rangle_p + \sum_{k=1}^{T} 2^{k-1} \cdot \langle b_{j,k} \rangle_p$.
   (d) $S_j \leftarrow \mathsf{Open}(\langle S_j \rangle_p)$.
   (e) $S_{j,v} \leftarrow \mathsf{Open}(\langle S_{j,v} \rangle_{p_v})$ for $v = 0, \ldots, t-1$
   (f) Abort if $S_j \pmod{p_v} \neq S_{j,v}$ for any $v = 0, \ldots, t-1$.
6. Output $\langle b_i \rangle_{p_v}$ for $i = 1, \ldots, m$ and $v = 0, \ldots, t-1$.

Figure 8. Method to produce m shared random bits in $\mathcal{F}_{\mathsf{MPC}}^{p_0}, \ldots, \mathcal{F}_{\mathsf{MPC}}^{p_{t-1}}$.

Theorem 4.1. *Protocol $\Pi_{\mathsf{RandomBits}}$ securely implements the functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$ in the $\mathcal{F}_{\mathsf{Rand}}^{W}$, $\mathcal{F}_{\mathsf{MPC}}\mathsf{GenBit}$-hybrid model. In the presence of passive adversaries the protocol will abort with probability $3 \cdot (m + \gamma \cdot T)/p$, where*

$$\lceil W/p_{\mathsf{min}} \rceil < W \cdot 2^{-\mathsf{sec}/\gamma}$$

and

$$T = \lceil \mathsf{stat} + \log_2 W + \log_2 m \rceil.$$

In the presence of active adversaries, deviation from the protocol is caught with probability at least $1 - 2^{-\mathsf{sec}}$. The statistical distance between a transcript of the real execution and a simulator acting with ideal functionality is bounded by $2^{-\mathsf{stat}}$.

Proof. We divide the proof into two sections, one showing correctness the other giving the simulation.

Correctness: Assuming all parties are honest, the computation of k_i may be incorrect with probability $3/p$. So even an honest execution of the protocol will abort with probability $3 \cdot (m + \gamma \cdot T)/p$, which is negligible.

We now examine the case of a dishonest party sending an invalid value of $h_{i,\ell}$ in Step 2d or enterring invalid values in Step 2f. The values S_j are guaranteed to be correct, assuming the Open does not abort due to adversarial behaviour. In addition the values S_j are produced with no wrap-around modulo p, due to the lower bound on p of 2^T. Thus the γ values S_j define subset sums over the integers

$$S_j = \sum_{i=1}^{m} b_i \cdot r_{i,j} + \sum_{k=1}^{T} 2^{k-1} \cdot b_{j,k}$$

where we are guaranteed that $b_i, b_{j,k} \in \{0, 1\}$.

We then check these sums against the equivalent sums modulo p_v for $v = 0, \ldots, t-1$. We write these as

$$S_{j,v} = \sum_{i=1}^{m} b_i \cdot r_{i,j} + M_{j,v} \pmod{p_v}$$

where

$$M_j = \sum_{k=1}^{T} 2^{k-1} \cdot b_{j,k}$$

The fact that the values $S_{j,v}$ are equal to $S_j \pmod{p_v}$ implies with overwhelming probability (since $p_{\mathsf{min}}^{\gamma} > 2^{\mathsf{sec}}$) that the shared values entered modulo p_j are equal to the shared values in $\mathcal{F}_{\mathsf{MPC}}^{p}$.

To see this notice that the adversary is able, by entering invalid values in steps 2d or 2f to introduce an additive error $\delta_{i,v}$ into the sharing $\langle b_i \rangle_{p_v}$, or an additive error $\delta_{0,v}$ into

the sharing of $\langle M_j \rangle_{p_v}$. Modulo p_v these errors, to pass the test in line 5f, must satisfy the equation

$$\delta_{0,v} + \sum_{i=1}^{m} r_{i,j} \cdot \delta_{i,v} = 0 \pmod{p_v}.$$

Assuming there is an error introduced, then there is a $k \in [1, \ldots, m]$, with $\delta_{k,v} \neq 0 \pmod{p_v}$, and so we must have

$$r_{k,j} = -\left(\delta_{0,v} + \sum_{i \neq k} r_{i,j} \cdot \delta_{i,v}\right) / \delta_{k,v} \pmod{p_v}$$

Since the random coefficients $r_{i,j}$ are revealed only after the parties input their shares of the bits modulo p_v, this can only happen with probability less than $\lceil W/p_v \rceil / W \leq \lceil W/p_{\mathsf{min}} \rceil / W$.

As the sums are independent for each of the j iterations, the chance of not-detecting one of $\delta_{i,v}$ values being non-zero is bounded by $(\lceil W/p_{\mathsf{min}} \rceil / W)^{\gamma}$. By choice of W this is less than $2^{-\mathsf{sec}}$.

Since $\mathcal{F}^{p}_{\mathsf{MPC}}.\mathsf{GenBit}$ generates values which are guaranteed to be bits, this also implies that the values modulo p_j are also bits, and equal to each other.

Simulation: To complete the proof we must show that the protocol can be simulated. There are two sets of values which need simulating. The $h_{i,j}$ values which are sent to party P_1 by Player P_j and the subset sum values S_j and $S_{j,v}$ which are opened. We first deal with the $h_{i,\ell}$ values.

The simulator already knows the shares $h_{i,\ell}$ which the adversary should be sending to them (from $\mathcal{F}^{p}_{\mathsf{MPC}}.\mathsf{GenBit}$), and so knows whether to abort the functionality at this point. Thus we need to determine a method to simulate the values $h_{i,\ell}$ sent from the honest parties.

We let $\mathcal{H}$ denote the set of honest parties, and $\mathcal{A}$ denote the set of adversarial parties. The simulator knows the value of

$$b_{i,\mathcal{A}} = \sum_{\ell \in \mathcal{A}} b_{i,\ell}.$$

As we are sharing a bit we know that

$$b_{i,\mathcal{H}} = \sum_{\ell \in \mathcal{H}} b_{i,\ell} \in \{p - b_{i,\mathcal{A}}, p + 1 - b_{i,\mathcal{A}}\}.$$

When

$$b_{i,\mathcal{A}} \in S = \{\lambda \cdot (\Delta - 1) + 1 : 0 \leq \lambda < n\}$$

then the hidden bit b_i is revealed to the adversary by the values $h_{i,\ell}$. But this happens with probability n/p. In total the probability of this happening accross all bits is bounded by $n \cdot (m + \gamma \cdot T)/p$, which is negligible.

Assuming $b_{i,\mathcal{A}} \notin S$ then there is a set of shares for the honest parties consistent with either $b_i = 0$ or $b_i = 1$ such that $0 \leq h_{i,\ell} < n$ and

$$\sum_{\ell \in \mathcal{H}} h_{i,\ell} = \lfloor (p - b_{i,\mathcal{A}})/\Delta \rfloor.$$

Thus the simulator just picks a random assignment to the $h_{i,\ell}$ values for $\ell \in \mathcal{H}$ which satisfies requirement.

Similarly the simulator knows when to abort if the adversary enters in valid values in Step 2f as well. To simulate the values S_j, and hence $S_{j,v}$, we simply define the trivial simulator. The values $r_{i,j}$ are sampled using the ideal functionality $\mathcal{F}^W_{\mathsf{Rand}}$. Then the simulator picks a random value $S_j \in [0, \ldots, 2^T)$. and returns S_j and $S_{j,v} = S_j \pmod{p_v}$, for every v, to the adversary. By the Smudging Lemma the simulated values are within statistical distance $2^{-\mathsf{stat}}$ from the values in the real execution of the protocol.

By the above correctness argument the simulation will be correct except with probability bounded by $\max\{2^{-\mathsf{sec}}, \max\{3, n\} \cdot (m + \gamma \cdot T)/p\}$, which is negligible. □

5 Sampling Distributions In MPC

In this section we explain how we perform the various sampling operations we need in the key generation algorithm in Section 6. We assume that two actively-secure (with-abort) MPC functionalities, as in Figure 2, have already been initiated, one for the prime p_0 and one for the prime p_1. We call these two functionalities $\mathcal{F}^{p_0}_{\mathsf{MPC}}$ and $\mathcal{F}^{p_1}_{\mathsf{MPC}}$. As explained earlier these can be instantiated with either BDOZ or MASCOT-SPDZ. Using these functionalities we can instantiate the protocol $\Pi_{\mathsf{RandomBits}}$, from Section 4, to produce doubly-authenticated bits in both $\mathcal{F}^{p_0}_{\mathsf{MPC}}$ and $\mathcal{F}^{p_1}_{\mathsf{MPC}}$. Thus all the protocols in this section can be seen to work in the $\mathcal{F}^{p,p_0,p_1}_{\mathsf{MPC}}.\mathsf{RandomBits}()$ model, which we have already shown can be instantiated in the $\mathcal{F}^{p,p_0,p_1}_{\mathsf{MPC}}$-hybrid model.

The protocol Π_{Hamming} will only be used to generate secret keys which are equivalent to the traditional BGV style keys used in SCALE-MAMBA or HELib. For more "modern" implementations one would use Π_{Binomial} to generate the secret keys as these have less security concerns related to them. In both cases we need Π_{Binomial} and Π_{Uniform} to generate the error vectors needed to compute the public key. Additionally we see our Π_{Hamming} protocols family could potentially improve the cost of the HE-MPC conversions of Zhu et al. [41] if they would relax the assumptions of generating a secret key with fixed hamming weight h to one that has expected hamming weight h.

5.1 Protocol Π_{Hamming}

To sample our secret key we will need to sample a shared vector of length N, with elements in $\{-1, 0, 1\}$, with approximate Hamming weight h, i.e. we sample from $B(h/N)^N$. The values of N and h are selected such they are both powers of two, so we set $N = 2^\nu$ and $h = 2^\ell$. The reader should think of values of $N = 32768$ and $h = 64$, which were used in `SCALE-MAMBA`'s default configuration when we started this work.

To sample such vectors we use protocol $\Pi_{\mathsf{Hamming}}(h, N)$ given in Figure 9. The basic idea (for one such bit) is to sample $v - \ell$ random bits $b_1, \ldots, b_{v-\ell}$. Then the product $p = b_1 \cdots b_{v-\ell}$ will be one with probability $1/2^{v-\ell}$, and zero otherwise. Then a sign bit s is used to flip the

one to minus one, with probability 1/2. Performing this operation N times produces the required distribution.

Protocol $\Pi_{\mathsf{Hamming}}(h, N)$

1. Set $N = 2^\nu$ and $h = 2^\ell$.
2. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate $H = N \cdot (\nu - \ell)$ shared random bits $\langle b_i^j \rangle_{p_0}$ and $\langle b_i^j \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, \nu - \ell\}$.
3. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_{p_0} \leftarrow \langle b_i^1 \rangle_{p_0} \cdot \ldots \cdot \langle b_i^{\nu-\ell} \rangle_{p_0}$.
   (b) $\langle b_i \rangle_{p_1} \leftarrow \langle b_i^1 \rangle_{p_1} \cdot \ldots \cdot \langle b_i^{\nu-\ell} \rangle_{p_1}$.
4. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate N shared random bits $\langle s_i \rangle_{p_0}$ and $\langle s_i \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$.
5. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_{p_0} \leftarrow \langle b_i \rangle_{p_0} \cdot (2 \cdot \langle s_i \rangle_{p_0} - 1)$.
   (b) $\langle b_i \rangle_{p_1} \leftarrow \langle b_i \rangle_{p_1} \cdot (2 \cdot \langle s_i \rangle_{p_1} - 1)$.
6. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle b_i \rangle_{p_0}$ and $\langle b_i \rangle_{p_1}$.

Figure 9. Method to produce vectors of (expected) Hamming weight h with elements in $\{-1, 0, 1\}$.

If we assume that the two calls to $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ are implemented using our previous protocol $\Pi_{\mathsf{RandomBits}}$ using only one batch operation, then these operation requires 6 rounds of communication, $(H + N + \gamma \cdot T)$ multiplications in $\mathcal{F}_{\mathsf{MPC}}^{p}$. The products to produce the initial bits b_i require $\lceil \log_2(\nu - \ell) \rceil$ rounds of communication and $(\nu - \ell - 1) \cdot N$ multiplications in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Whereas the products to produce the final signed bits require one round of communication and N multiplications in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Hence in total we require

$$6 + \lceil \log_2(\nu - \ell) \rceil + 1 = 7 + \lceil \log_2(\nu - \ell) \rceil$$

rounds of communication and

$$\begin{aligned} N_{\mathsf{Hamming},0} &= N + H + \mathsf{sec} + (\nu - \ell + 1) \cdot N, \\ N_{\mathsf{Hamming},1} &= (\nu - \ell + 1) \cdot N \end{aligned}$$

total multiplications in $\Pi_{\mathsf{MPC}}^{p_0}$ and $\Pi_{\mathsf{MPC}}^{p_1}$ respectively.

5.2 Another Method to Implement Π_{Hamming}

In this section we describe a different protocol to generate a vector of size N with expected hamming weight h. The main advantage of this protocol is that it avoids the expensive multiplications modulo large prime numbers (modulo p_0 and modulo p_1) completely. This comes at the expense of requiring an extra preprocessing protocol which generates random shared bits modulo two from [32,40], as well as the need to implement a demultiplexer via a lookup table [31]. Thus this method requires us to base the SPDZ key generation methodology on two distinct MPC methodologies; TinyOT-style $\mathbb{F}_2$-based MPC and MASCOT style $\mathbb{F}_p$-base MPC. So as to keep the minimality in terms of base MPC protocols for our key generation process we include this section mearly as an additional method for the reader.

Note, to generate random bits shared modulo two is much cheaper than generating random shared bits modulo a large prime p. These binary sharings are usually done using a pairwise protocol TinyOT style found in Wang et al. or Keller et al. [32,40]. We call the functionality that generates binary sharings $\mathcal{F}_{\mathsf{bit}}[\mathbb{F}_2]$

The key idea of this protocol is to follow the same blueprint as $\Pi_{\mathsf{Hamming}}(h, N)$ defined in Section 5.1 but start with random binary sharings of bits $\langle b \rangle_2$ rather then arithmetic sharings $\langle b \rangle_{p_0}$ and $\langle b \rangle_{p_1}$. This will cause the bit multiplications to happen modulo two which is orders of magnitude faster than their counterpart modulo a large prime p.

However, the main challenge with this method is to be able to output the value in $\{0, \pm 1\}$ without the large field multiplications in Step 5 from Figure 5.1. This can be done by emulating a small demultiplexer on the values $b_i b_i^*$ ($\{00, 01, 10, 11\}$) and map them to $\{0, 0, 1, -1\}$. We define this public lookup table containing these values in binary form as $\mathsf{T} = \{00, 00, 10, 11\}$. One can think of the right hand bit in each entry of T as the sign of the secret key bit. In the last step we convert the table entries to integers modulo a large prime using maBits, i.e. we compute $2^0 \cdot T(i, 0) - 2^1 \cdot T(i, 1)$ where $T(i, j)$ represents the j'th bit of the i'th entry.

Protocol $\Pi'_{\mathsf{Hamming}}(h, N)$

1. Set $N = 2^\nu$ and $h = 2^\ell$.
2. Call $\mathcal{F}_{\mathsf{bit}}[\mathbb{F}_2]$ to generate $H = N \cdot (\nu - \ell)$ shared random bits $\langle b_i^j \rangle_2$ for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, \nu - \ell\}$.
3. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_2 \leftarrow \langle b_i^1 \rangle_2 \cdot \ldots \cdot \langle b_i^{\nu-\ell} \rangle_2$.
   (b) $\langle b_i^* \rangle_2 \leftarrow \mathcal{F}_{\mathsf{bit}}[\mathbb{F}_2]$
4. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate $2 \cdot N$ shared random bits $\langle s_i^j \rangle_2$, $\langle s_i^j \rangle_{p_0}$ and $\langle s_i^j \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$ and $j \in \{0, 1\}$.
5. For $i \in \{1, \ldots, N\}$
   (a) $(\langle d_0 \rangle_2, \cdots \langle d_3 \rangle_2) \leftarrow \mathsf{Demux}(\langle b_i \rangle_2, \langle b_i^* \rangle_2)$
   (b) $(\langle v_0 \rangle_2, \langle v_1 \rangle_2) \leftarrow \langle d_0 \rangle_2 \cdot \mathsf{T}(0) \oplus \langle d_1 \rangle_2 \cdot \mathsf{T}(1) \oplus \langle d_2 \rangle_2 \cdot \mathsf{T}(2) \oplus \langle d_3 \rangle_2 \cdot \mathsf{T}(3)$
   (c) $t_i^0 \leftarrow \mathsf{Open}(\langle s_i^0 \rangle_2 \oplus \langle v_0 \rangle_2)$ and $t_i^1 \leftarrow \mathsf{Open}(\langle s_i^1 \rangle_2 \oplus \langle v_1 \rangle_2)$
6. For $i \in \{1, \ldots, N\}$ [Now convert the lookup table entries to their arithmetic counterpart]
   (a) $\langle b_i \rangle_{p_0} \leftarrow 2^0 \cdot \mathsf{XOR}(t_i^0, \langle s_i^0 \rangle_{p_0}) - 2^1 \cdot \mathsf{XOR}(t_i^1, \langle s_i^1 \rangle_{p_0})$ (bit-compose of public-secret xors in $\mathbb{F}_{p_0}$)
   (b) $\langle b_i \rangle_{p_1} \leftarrow 2^0 \cdot \mathsf{XOR}(t_i^0, \langle s_i^0 \rangle_{p_1}) - 2^1 \cdot \mathsf{XOR}(t_i^1, \langle s_i^1 \rangle_{p_1})$ (bit-compose of public-secret xors in $\mathbb{F}_{p_1}$)
7. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle b_i \rangle_{p_0}$ and $\langle b_i \rangle_{p_1}$.

Figure 10. Second method to produce vectors of (expected) Hamming weight h with elements in $\{-1, 0, 1\}$.

5.3 Protocol Π_{Binomial}

The protocol for sampling shared values from the distribution $\mathsf{dN}(\sigma^2, N)$ is relatively straightforward, and is given in Figure 11. Apart from the generation of the random bits this protocol consists entirely of linear operations. Thus the round complexity is six and it requies $N_{\mathsf{Binomial}} = 2 \cdot k \cdot N + \gamma \cdot T$ multiplications in $\mathcal{F}_{\mathsf{MPC}}^p$, assuming $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ is implemented in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}$-hybrid model using our previous protocol $\Pi_{\mathsf{RandomBits}}$.

Protocol $\Pi_{\mathsf{Binomial}}(\sigma^2, N)$

1. Define k by $\sigma = \sqrt{k/2}$.
2. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate $2 \cdot k \cdot N$ shared random bits $\langle b_i^j \rangle_{p_0}$ and $\langle b_i^j \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$ and $j \in \{0, \ldots, 2 \cdot k - 1\}$.
3. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_{p_0} \leftarrow \sum_{j=0}^{k-1} \langle b_i^{2 \cdot j} \rangle_{p_0} - \langle b_i^{2 \cdot j+1} \rangle_{p_0}$.
   (b) $\langle b_i \rangle_{p_1} \leftarrow \sum_{j=0}^{k-1} \langle b_i^{2 \cdot j} \rangle_{p_1} - \langle b_i^{2 \cdot j+1} \rangle_{p_1}$.
4. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle b_i \rangle_{p_0}$ and $\langle b_i \rangle_{p_1}$.

Figure 11. Method to produce elements from $\mathsf{dN}(\sigma^2, N)$.

5.4 Protocol Π_{Uniform}

Our final protocol is a rather trivial one, it allows the parties to sample a uniform element from $\mathbb{Z}_q$ (recall $q = p_0 \cdot p_1$ with $\gcd(p_0, p_1) = 1$) in a secret shared form, we give it in Figure 12.

Protocol $\Pi_{\mathsf{Uniform}}(N)$

1. For $i \in \{1, \ldots, M\}$
   (a) For $j \in \{1, \ldots, n\}$
      i. Player P_i selects $u_i^j \in \mathbb{F}_{p_0}$, and $v_i^j \in \mathbb{F}_{p_1}$,
      ii. Execute $\langle u_i^j \rangle_{p_0} \leftarrow \mathsf{Input}(P_i)$.
      iii. Execute $\langle v_i^j \rangle_{p_1} \leftarrow \mathsf{Input}(P_i)$.
   (b) $\langle u_i \rangle_{p_0} \leftarrow \langle u_i^1 \rangle_{p_0} + \ldots + \langle u_i^n \rangle_{p_0}$.
   (c) $\langle v_i \rangle_{p_1} \leftarrow \langle v_i^1 \rangle_{p_1} + \ldots + \langle v_i^n \rangle_{p_1}$.
2. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle u_i \rangle_{p_0}$ and $\langle v_i \rangle_{p_1}$. We think of these as sharings of $(\langle a_i \rangle_{p_0}, \langle a_i \rangle_{p_1})$, of some value $a_i \in \mathbb{Z}_q$ with $a_i = u_i \pmod{p_0}$ and $a_i = v_i \pmod{p_1}$.

Figure 12. Protocol to sample N uniformly random elements from $\mathbb{Z}_q$.

6 SPDZ KeyGeneration

Given the previous algorithms to generate various distributions the computation of the actual key generation algorithm becomes relatively straight forward. We first sample the various distributions modulo p_0 and p_1, then we produce the square of the secret key (needed for the key switching matrices), and then we output the public key (modulo p_0 and p_1) and then recombine it using the CRT, finally we do the same to each players component of the secret key. The overall protocol is given in Figure 13; note that in line 3 one can select as to whether to choose the secret key from a restricted Hamming weight or from the centred binomial distribution. To make the protocol easier to follow we use the notation $\langle \mathbf{a} \rangle_{p_0}$ etc to denote a vector of N shares modulo p_0, i.e. $\langle \mathbf{a} \rangle_{p_0} = (\langle a_0 \rangle_{p_0}, \ldots, \langle a_{N-1} \rangle_{p_0})$.

We let $\langle \mathbf{a} \rangle_{p_0} \odot \langle \mathbf{b} \rangle_{p_0}$ denote the multiplication of two such vectors when considered as elements in the ring R_{p_0}. This requires one round of communication and N^2 secure multiplications (or $N \cdot (N-1)/2$ secure multiplications if $\mathbf{a} = \mathbf{b}$). If one vector is in the clear then we write $\mathbf{a} \odot \langle \mathbf{b} \rangle_{p_0}$, which is a linear operation and hence for "free". A more efficient

method to multiply is to use the FFT algorithm, which recall is a linear operation and thus 'free' when executed in the MPC engine. To multiply using FFT we utilize

$$\langle \mathbf{a} \rangle_{p_0} \odot \langle \mathbf{b} \rangle_{p_0} = \mathsf{FFT}^{-1}(\mathsf{FFT}(\langle \mathbf{a} \rangle_{p_0}) \cdot \mathsf{FFT}(\langle \mathbf{b} \rangle_{p_0})$$

which requires only N secure multiplications and one round of communication.

Protocol Π_{KeyGen}

- [*Generate the various values required*]
1. $(\langle \mathbf{a} \rangle_{p_0}, \langle \mathbf{a} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Uniform}}(N)$.
2. $(\langle \mathbf{a}' \rangle_{p_0}, \langle \mathbf{a}' \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Uniform}}(N)$.
3. $(\langle \mathbf{s} \rangle_{p_0}, \langle \mathbf{s} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Hamming}}(h, N)$ or $(\langle \mathbf{s} \rangle_{p_0}, \langle \mathbf{s} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Binomial}}(N)$.
4. $(\langle \mathbf{e} \rangle_{p_0}, \langle \mathbf{e} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Binomial}}(N)$.
5. $(\langle \mathbf{e}' \rangle_{p_0}, \langle \mathbf{e}' \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Binomial}}(N)$.
6. For $i \in \{2, \ldots, n\}$
   (a) $\langle \mathfrak{sk}_{0,i} \rangle_{p_0} \leftarrow \mathsf{Input}(P_i)$ for a random vector $\mathfrak{sk}_{0,i} \in R_{p_0}$ selected by player P_i.
   (b) $\langle \mathfrak{sk}_{1,i} \rangle_{p_1} \leftarrow \mathsf{Input}(P_i)$ for a random vector $\mathfrak{sk}_{1,i} \in R_{p_1}$ selected by player P_i.
- [*Compute the square of the secret key*]
7. $\langle \mathbf{f} \rangle_{p_0} \leftarrow \mathsf{FFT}(\langle \mathbf{s} \rangle_{p_0})$.
8. $\langle \mathbf{f}' \rangle_{p_0} \leftarrow \langle \mathbf{f} \rangle_{p_0} \cdot \langle \mathbf{f} \rangle_{p_0}$ [This is the component wise product].
9. $\langle \mathbf{s}' \rangle_{p_0} \leftarrow \mathsf{FFT}^{-1}(\langle \mathbf{f}' \rangle_{p_0})$.
- [*Open the values* $\mathbf{a}$ *and* $\mathbf{a}'$]
10. $\mathbf{a}_0 \leftarrow \mathsf{Open}(\langle \mathbf{a} \rangle_{p_0})$.
11. $\mathbf{a}_1 \leftarrow \mathsf{Open}(\langle \mathbf{a} \rangle_{p_1})$.
12. $\mathbf{a}'_0 \leftarrow \mathsf{Open}(\langle \mathbf{a}' \rangle_{p_0})$.
13. $\mathbf{a}'_1 \leftarrow \mathsf{Open}(\langle \mathbf{a}' \rangle_{p_1})$.
14. $\mathbf{a} \leftarrow \mathsf{CRT}([\mathbf{a}_0, \mathbf{a}_1], [p_0, p_1])$.
15. $\mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{CRT}([\mathbf{a}'_0, \mathbf{a}'_1], [p_0, p_1])$.
- [*Compute* $\mathbf{b}$ *and* $\mathbf{b}'$]
16. $\langle \mathbf{b} \rangle_{p_0} \leftarrow \mathbf{a}_0 \odot \langle \mathbf{s} \rangle_{p_0} + p \cdot \langle \mathbf{e} \rangle_{p_0}$.
17. $\langle \mathbf{b} \rangle_{p_1} \leftarrow \mathbf{a}_1 \odot \langle \mathbf{s} \rangle_{p_1} + p \cdot \langle \mathbf{e} \rangle_{p_1}$.
18. $\langle \mathbf{b}' \rangle_{p_0} \leftarrow \mathbf{a}'_0 \odot \langle \mathbf{s} \rangle_{p_0} + p \cdot \langle \mathbf{e}' \rangle_{p_0} - p_1 \cdot \langle \mathbf{s}' \rangle_{p_0}$.
19. $\langle \mathbf{b}' \rangle_{p_1} \leftarrow \mathbf{a}'_1 \odot \langle \mathbf{s} \rangle_{p_1} + p \cdot \langle \mathbf{e}' \rangle_{p_1}$.
- [*Fix the final key for the sharing of the secret key*]
20. $\langle \mathfrak{sk}_{0,1} \rangle_{p_0} \leftarrow \langle \mathbf{s} \rangle_{p_0} - \sum_{i=2}^{n} \langle \mathfrak{sk}_{0,i} \rangle_{p_0}$.
21. $\langle \mathfrak{sk}_{1,1} \rangle_{p_1} \leftarrow \langle \mathbf{s} \rangle_{p_1} - \sum_{i=2}^{n} \langle \mathfrak{sk}_{1,i} \rangle_{p_1}$.
22. For all $i \in \{1, \ldots, n\}$ assign $\mathfrak{sk}_i \leftarrow \mathsf{CRT}([\mathfrak{sk}_{0,i}, \mathfrak{sk}_{1,i}], [p_0, p_1])$.
23. $P_1 \leftarrow \mathsf{Output}(\langle \mathfrak{sk}_{0,1} \rangle_{p_0})$.
24. $P_1 \leftarrow \mathsf{Output}(\langle \mathfrak{sk}_{1,1} \rangle_{p_1})$.
- [*Open the values* $\mathbf{b}$ *and* $\mathbf{b}'$]
25. $\mathbf{b}_0 \leftarrow \mathsf{Open}(\langle \mathbf{b} \rangle_{p_0})$.
26. $\mathbf{b}_1 \leftarrow \mathsf{Open}(\langle \mathbf{b} \rangle_{p_1})$.
27. $\mathbf{b}'_0 \leftarrow \mathsf{Open}(\langle \mathbf{b}' \rangle_{p_0})$.
28. $\mathbf{b}'_1 \leftarrow \mathsf{Open}(\langle \mathbf{b}' \rangle_{p_1})$.
29. $\mathbf{b} \leftarrow \mathsf{CRT}([\mathbf{b}_0, \mathbf{b}_1], [p_0, p_1])$.
30. $\mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{CRT}([\mathbf{b}'_0, \mathbf{b}'_1], [p_0, p_1])$.
31. Output $\mathbf{a}, \mathbf{b}, \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}$ and $\mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2}$ to all players, and $\mathfrak{sk}_i$ to player P_i.

Figure 13. The distributed BGV key generation protocol.

We now examine each of the operations in this algorithm in turn. The lines 1-6 can all be executed in parallel and so require

$$\max\Big(6 + \lceil \log_2(\nu - \ell) \rceil + 1, 6, 1\Big) = 6 + \lceil \log_2(\nu - \ell) \rceil$$

rounds of communication [6] in the case where we select a secrey key with given expected Hamming weight and $\max\Big(6, 6, 1\Big) = 6$ rounds of communication in the case of the secret key generated from a centred binomial distribution. The number of secure multiplications is given by, in the two cases,

$$\begin{aligned} N_{\mathsf{Hamming},0} + N_{\mathsf{Hamming},1} + 2 \cdot N_{\mathsf{Binomial}} &= N + H + \gamma \cdot T + 2 \cdot (\nu - \ell + 2 \cdot k - 1) \cdot N, \\ 3 \cdot N_{\mathsf{Binomial}} &= 6 \cdot k \cdot N. \end{aligned}$$

Lines 7 and 9 are linear operations and thus can be executed as purely local operations. Line 8 requires one round of communication and N multiplications in $\mathcal{F}_{\mathsf{MPC}}^{p_0}$. Note, in lines 7-9 we only have to compute $\mathfrak{sk}^2$ modulo p_0 as it is multiplied by p_1 when added into $\mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2}$. Lines 10-13 can also be performed in parallel and hence require only one round of communication. The lines 16-21 are all local operations, and hence are for "free". Lines 25-24 are, again, able to be done in parallel and so require only one round of communication. The remaining lines are purely local organization of data into the correct format for outputing. Thus the total number of rounds of communication (assuming all shared random bits are produced in a single batch) is $12 + \lceil \log_2(\nu - \ell) \rceil$ or 12, depending on which variant one is using for the secret key.

Theorem 6.1. *The protocol Π_{KeyGen} UC-securely realises the functionality $\mathcal{F}_{\mathsf{KeyGen}}$ against a static, active adversary corrupting at most $n-1$ parties in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$-hybrid model.*

Proof. We define the simulator $\mathcal{S}$ as follows. The simulator emulates the behaviour of honest parties exactly, but additionally does the following:

- At the start of the execution, the simulator initialises a local copy of $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ and sends the message *start* to $\mathcal{F}_{\mathsf{KeyGen}}$ and awaits the public key $\mathfrak{pk} = (\bar{\mathbf{a}}, \bar{\mathbf{b}}, \bar{\mathbf{a}}_{\mathfrak{sk},\mathfrak{sk}^2}, \bar{\mathbf{b}}_{\mathfrak{sk},\mathfrak{sk}^2})$ in response.
- When the adversary and simulator execute Π_{Uniform}, the simulator replaces the values $\mathbf{a} \mod p_0$, $\mathbf{a} \mod p_1$, $\mathbf{a}' \mod p_0$ and $\mathbf{a}' \mod p_1$ stored in the instances of $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$, respectively, with $\bar{\mathbf{a}} \mod p_0$, $\bar{\mathbf{a}} \mod p_1$, $\bar{\mathbf{a}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_0$ and $\bar{\mathbf{a}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_1$.
- In Step 6, for each $j \in [n]$, if j is corrupt and $j > 2$ then the simulator awaits the input $\mathfrak{sk}_{j,0}$ and $\mathfrak{sk}_{j,1}$ for each corrupt party P_j and constructs $\mathfrak{sk}_j \leftarrow \mathsf{CRT}(\mathfrak{sk}_{j,0}, \mathfrak{sk}_{j,1})$, and sends these to $\mathcal{F}_{\mathsf{KeyGen}}$.
- Just before opening $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_0'$ and $\mathbf{b}_1'$ the simulator replaces these values stored in the instances of $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ with $\bar{\mathbf{b}} \mod p_0$, $\bar{\mathbf{b}} \mod p_1$, $\bar{\mathbf{b}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_0$ and $\bar{\mathbf{b}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_1$.

[6] Of course in practice we generate the secure bits in batches and hence this is just the minimal number of rounds required.

Since the only inputs to the protocol are randomly sampled by parties, the simulator can perfectly emulate the behaviour of honest parties throughout, as the environment does not observe the random tape of honest parties or the simulator. Indeed, since $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ are used as black boxes and the only communication between parties occurs via these functionalities, which are emulated locally honestly by the simulator, the replacements made in the simulation outlined are executed without being observed by the environment (at this point). Moreover, the inputs of corrupt parties can be extracted trivially and passed on to $\mathcal{F}_{\mathsf{KeyGen}}$ so that the final outputs have the correct distribution.

It only remains to show that the environment cannot observe a difference between the transcripts in a real execution and an execution in which the replacements described above are made. While the final outputs of the real and ideal worlds is the same, the distribution of the transcript differs since communication generated in the execution of the sampling subprotocols depends on the secret $\bar{\mathbf{s}}$ which is (implicitly) generated by the functionality $\mathcal{F}_{\mathsf{KeyGen}}$ when executing $\mathsf{KeyGen}()$ and cannot be computed by the simulator from the public key without breaking the LWE assumption for the security of the key. We must show that the amount by which the distributions differ is negligible.

The only time information stored in $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ or $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ is either revealed to the parties or is generated by parties is in the bits shared in $\Pi_{\mathsf{RandomBits}}$ and in Π_{Uniform}. In the protocols Π_{Binomial} and Π_{Hamming} the parties obtain bits from $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$, and in the remainder of the protocol Π_{KeyGen}, the correctness of the computations is guaranteed by the security of the black boxes $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. In Π_{Uniform}, every party contributes to every secret, and since there is at least one honest party (that samples uniformly), the output is always uniform. Thus no distinguishing environment can exist, by the choice of parameters for the subset sum, and therefore Π_{KeyGen} UC-securely realises $\mathcal{F}_{\mathsf{KeyGen}}$ in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$-hybrid model. □

7 Implementation

In our implementation we use MASCOT-SPDZ [33] as the base protocol used for our one-time setup phase for SPDZ. We could have selected here BDOZ [8] using LowGear [34] for the pre-processing. Both give $O(n^2)$ protocols with LowGear about a factor of four times faster for small values of n. Since our solution is built on top of the `SCALE-MAMBA` [2] framework, we re-used a lot of their already existing codebase and hence selected MASCOT-SPDZ as the underlying protocol. Our implementation of this key generation protocol is now included in the `SCALE-MAMBA` code base. As explained in the introduction our key generation protocol, is inherently $O(n^2)$ in nature. This seems to be unavoidable as the only practical $O(n)$ MPC protocol known is SPDZ, which is exactly the MPC protocol we are trying to instantiate with our key generation protocol.

Selection of FHE parameters We recall that the two-leveled BGV key generation procedure requires us to choose two prime moduli p_0 and p_1 and a polynomial degree N to define the ciphertext space $R_q \cong R_{p_0} \times R_{p_1}$ and a prime p which defines the size of our plaintext space. In addition we require that the relations presented in Section 3.2 hold.

The size of the plaintext space p defines the modulus of the underlying secret sharing scheme in the SPDZ protocol. Different values of p will be needed for different application scenarios. In this paper we focus on $p \approx 2^{64}$ and $p \approx 2^{128}$. The precise sizes of the other parameters are derived from a noise analysis of how the resulting encryption scheme is used, which takes into account the circuit being computed, the zero-knowledge proofs required, and the distributed decryption procedure, and the computational difficulty of the Ring-LWE problem. This analysis is quite involved and we referred to the `SCALE-MAMBA` documentation [2] to obtain the required parameters.

This gives us (for example), to guarantee a computational security of 128 bits with a polynomial degree $N = 32768$, that our ciphertext modulus q has to satisfy $q < 2^{883}$. For such parameters, the trusted setup of `SCALE-MAMBA`, as we have discussed earlier, produces a secret key with Hamming Weight exactly 64, and uses noise vectors distributed according to a centred binomial distribution with standard deviation of $3.16 = \sqrt{10}$. As a first set of experiments we use exactly the same methodology to select the secret key, but we pick a secret key with *expected* Hamming Weight 64. This does not change the noise analysis (which is done using an expected noise methodology for the secret key in any case), and thus we end up with the same system parameters as used in `SCALE-MAMBA`. In particular for a plaintext modulus of 128 bits this leads us to use a p_0 modulus of 345 bits and a p_1 modulus of 225 bits. So in order to run the key generation protocol presented above, we will need to run two instances of the MASCOT protocol; one for the 345 bits prime p_0 and another for the 225 bits prime p_1.

We also tried different sets of parameters in our experiments, which provide BGV keys for a SPDZ moduli of 64 and 128 bits, always taking the same parameters as the Setup phase for `SCALE-MAMBA`. In Table 1 we report the prime sizes, in bits, required for each set of parameters for two and three parties.

Number of Parties, n	2				3			
FHE Plaintext Size, $\log_2 p$	64		128		64		128	
Polynomial Degree	16384		32768		16384		32768	
$p_0 \vert p_1$ bit length	216	164	345	225	217	163	346	224

Table 1. Parameter Sizes.

With these size of primes, we can now set the value m for our batch size of shared bits which has to respect the bound stated in Section 4. In practice we want to have m a divisor of the total number of shared bits that we will need, and also small enough to avoid RAM or network overflow. Empirically we found that taking $50000 \leq m \leq 100000$ (depending on the setting) gives us good results. Although for smaller values of p_0 (if we wish to use p_0 as the auxillary prime p) we need to restrict m so that $m < p_0/(W \cdot 2^{\mathsf{stat}}) = p_0/2^{208}$.

Extended Random Oblivious Transfer: When starting this work, the `SCALE-MAMBA` framework did not have an implementation of the offline phase of MASCOT, and an implemen-

tation of the extended Random Oblivious Transfer for a prime field $\mathbb{F}_p$, thus we needed to implement these[7]. The triple generation method of MASCOT [33][Protocol 4] makes use of an extended Correlated Oblivious Transfer (COT) protocol. For this we used the passively secure protocol of Frederiksen et al. [24][Full Version, Figure 19] (which is essentially the protocol of Ishai et al [28]). Such a passively secure COT protocol is sufficient due to how it is used in the MASCOT offline phase. This COT protocol was already implemented in `SCALE-MAMBA`. For two parties P_i and P_j with the former acting as the sender and the later as the receiver, calls to the COT protocol outputs values $\{M_0^k, M_1^k = M_0^k + \Delta_i\}_{k\in[n]}$ to P_i and $\{M_{\mathbf{b}_k}^k\}_{k\in[n]}$ to P_j, where $\Delta_i \in \mathbb{F}_{2^{128}}$ is the input from P_i and $\mathbf{b} \in \{0,1\}^n$ is the choice vector from P_j, and $\forall k \in [n]\ M_0^k \leftarrow_\$ \mathbb{F}_{2^{128}}$

To obtain extended Random Oblivious Transfer (ROT), from these extended COTs, we used the decorrelation technique presented in [12][Figure 15] which consists in both parties hashing the output of the extended COT. This gives us an extended ROT in $\mathbb{F}_{2^{128}}$. However, we want to run the MASCOT protocol on prime fields $\mathbb{F}_{p_0}$ and $\mathbb{F}_{p_1}$, so to translate our extended ROT from $\mathbb{F}_{2^{128}}$ to $\mathbb{F}_p$, we take $\lceil \frac{\log_2(p)+128}{128} \rceil$ outputs in $\mathbb{F}_{2^{128}}$, concatenate them together and take the result mod p.

Results of our experiments Our implementation of the key generation protocol was tested in a LAN setting, with each party running on an Intel i7-7700K CPU with 32GB of RAM over a 10Gb/s network switch. In our experiments we found that executing more threads than the available number of cores (in our case eight) gave a performance improvement. This is because the computation between receiver and sender in the OT protocols is asymmetric, resulting in each party sometimes waiting for the other to perform some computation.

For each prime size, we report the throughput of our MASCOT implementation regarding triple generation in Table 2. We experimented with different number of threads, and for each case we give our results for 1, 5 and 10 threads. By comparison to [33][Figure 9], our implementation of the MASCOT protocol for triple generation seems to have room for improvement. In particular for a 128 bit field, we are about 7 times slower than MP-SPDZ [21]. Therefore, we believe that a better implementation of MASCOT using less CPU resources would give us a significant amelioration of our BGV key generation benchmark.

n	2				3			
$\log_2 p$	64		128		64		128	
1 Thread	126.77	177.1	70.01	133.29	58.78	79.01	28.7	57.22
5 Thread	587.77	784.69	292.18	567.62	276.38	376.65	134.62	279.27
10 Thread	759.1	1032.57	365.93	740.05	389.19	547.27	185.32	379.34

Table 2. Triple Throughput for $\mathbb{F}_{p_0}$ and $\mathbb{F}_{p_1}$ in Triples per Second.

Finally in Table 3 we give figures related to the throughput of the secure randomly shared bits, namely the throughput of the algorithm in Figure 8, in the four different scenarios we

[7] Our implementations are now included in the `SCALE-MAMBA` code-base.

experimented with. For each case, we give our result while also running the MASCOT triple generation, and also assuming that this offline phase has already been done beforehand. In the same table we eventually give the total runtime for the distributed BGV key generation procedure, which includes the time to perform the MASCOT offline phase.

Number of Parties	2		3	
FHE Plaintext size	64	128	64	128
Shared bit throughput including offline time (b/s)	416.67	263.16	194.55	124.69
Shared bit throughput excluding offline time (b/s)	1515.15	1449.28	1149.43	1041.67
Expected Expected BGV Key Generation Wall Clock	2 hours	5 hours	3 hours	9 hours

Table 3. Shared bit throughput and expected total time for the KeyGen protocol.

We estimated that for two parties it would take roughly five hours hours to complete the distributed key generation for two parties, and nine hours for three parties for 128-bit FHE plaintext size. Although it may seem inefficient, we argue that this protocol needs only to be done once in order to enable future computation of the offline phase of SPDZ. Moreover, we have shown that our protocol is highly parallelizable. So in practice, if such a protocol was to be run on high end servers owned by cloud service providers, the total runtime could be drastically reduced.

We pause to compare these run-times to the covertly secure distributed key generation protocol presented in [18]. This protocol did not produce public and secret keys with the same distribution as the non-distributed version. For a plaintext size of 64-bits the authors of [18] report 12 and 16 seconds key generation time for $n = 2$ and $n = 3$, for a covert security of 1/10, i.e. an adversary can cheat with probability 1/10. For a plaintext size of 128-bits the times are 33 and 44 seconds. The execution time of this covertly secure protocol is linear in c, where the covert security measure is $1/c$. Thus to obtain comparable security to our protocol, the protocol in [18] would be utterly inpractical.

Changing the Standard Deviation Our protocol can also be run when we select the standard deviation for the centred binomial distributions to be equal to $\sigma = \sqrt{1/2} = 0.707$. The analysis of the parameters, given in Table 1 is roughly the same. However, the associated run times for key generation become faster as we no longer need to generate as many doubly authenticated bits. This is reflected in Table 4. We see that by choosing $\sigma = 0.707$, we need only two bits instead of 40 for the sampling from the centred binomial distribution. We thus obtain a factor of at least 2.5 improvement overall, compared to the previous setting.

7.1 Secret Keys Generated According to a Centred Binomial Distribution

Finally we examine the case of using the centred binomial distribution for generating the secret keys, and not just the noise values, with standard deviation selected to be $\sigma = \sqrt{1/2} =$

Number of Parties	2		3	
FHE Plaintext size	64	128	64	128
Expected BGV Key Generation Wall Clock	30 mins	2 hours	1 hour	4 hours

Table 4. Shared bit throughput and expected total time for the KeyGen protocol for $\sigma = 0.707$.

0.707, as opposed to using a low Hamming weight based distribution for the secret keys. This pushes the parameter sizes for the underlying BGV scheme up a little, as we need to cope with more potential noise growth due to the 'heavier' secret key. Using the same analysis as before we find the parameter sizes given in Table 5, with the resulting run times for distributed key generation given in Table 6. This time we performed the experiments also for $n = 4$ and $n = 5$ so as to show how the times grow with n; recall the overall method is $O(n^2)$ as mentioned earlier.

Number of Parties, n	2				3				4				5			
FHE Plaintext Size, $\log_2 p$	64		128		64		128		64		128		64		128	
Polynomial Degree	16384		32768		16384		32768		16384		32768		16384		32768	
$p_0 \vert p_1$ bit length	222	159	352	219	223	158	353	218	223	158	353	228	223	158	353	228

Table 5. Parameter Sizes for secret keys distributed according to a centred binomial distribution, and Gaussian error distribution of $\sigma = 0.707$.

Number of Parties	2		3		4		5	
FHE Plaintext size	64	128	64	128	64	128	64	128
Expected BGV Key Generation	10 mins	40 mins	16 mins	50 mins	25 mins	2 hours	30 mins	4.5 hours

Table 6. Expected total time for the KeyGen protocol for $\sigma = 0.707$ and secret keys generated according to a centred binomial distribution.

Acknowledgements

The authors would like to thank Carsten Baum, Emmanuela Orsini and Victor Treinsoutrot for suggestions in relation to the work in this paper, and Claudio Orlandi in elaborating on the early history of the BDOZ and Tiny-OT work.

The work in this paper was carried out whilst Dragos Rotaru and Tim Wood were PhD students at the University of Bristol, and were employed by KU Leuven.

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare

Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070 and FA8750-19-C-0502, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via Contract No. 2019-1902070006, by the FWO under an Odysseus project GOH9718N, and by CyberSecurity Research Flanders with reference number VR20192203.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, ODNI, United States Air Force, IARPA, DARPA, the US Government or FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: Holz, T., Savage, S. (eds.) USENIX Security 2016: 25th USENIX Security Symposium. pp. 327–343. USENIX Association, Austin, TX, USA (Aug 10–12, 2016)
2. Aly, A., Cong, K., Cozzo, D., Keller, M., Orsini, E., Rotaru, D., Scherer, O., Scholl, P., Smart, N.P., Tanguy, T., Wood, T.: SCALE-MAMBA v1.14: Documentation (2021), https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf
3. Aly, A., Orsini, E., Rotaru, D., Smart, N.P., Wood, T.: Zaphod: Efficiently combining LSSS and garbled circuits in SCALE. In: Brenner et al. [11], pp. 33–44, https://doi.org/10.1145/3338469
4. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 483–501. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
5. Baum, C., Cozzo, D., Smart, N.P.: Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 11959, pp. 274–302. Springer, Heidelberg, Germany, Waterloo, ON, Canada (Aug 12–16, 2019)
6. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology – CRYPTO'91. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1992)
7. Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: Micciancio, D. (ed.) TCC 2010: 7th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 5978, pp. 201–218. Springer, Heidelberg, Germany, Zurich, Switzerland (Feb 9–11, 2010)
8. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 169–188. Springer, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011)
9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 489–518. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019)
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012: 3rd Innovations in Theoretical Computer Science. pp. 309–325. Association for Computing Machinery, Cambridge, MA, USA (Jan 8–10, 2012)
11. Brenner, M., Lepoint, T., Rohloff, K. (eds.): Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019. ACM (2019), https://doi.org/10.1145/3338469
12. Burra, S.S., Larraia, E., Nielsen, J.B., Nordholt, P.S., Orlandi, C., Orsini, E., Scholl, P., Smart, N.P.: High-performance multi-party computation for binary circuits based on oblivious transfer. J. Cryptol. 34(3), 34 (2021), https://doi.org/10.1007/s00145-021-09403-1

13. Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., Wang, R.: Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 590–607 (2021), https://doi.org/10.1109/SP40001.2021.00025
14. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 769–798. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
15. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 280–299. Springer, Heidelberg, Germany, Innsbruck, Austria (May 6–10, 2001)
16. Curtis, B.R., Player, R.: On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In: Brenner et al. [11], pp. 1–10, https://doi.org/10.1145/3338469
17. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 5443, pp. 160–179. Springer, Heidelberg, Germany, Irvine, CA, USA (Mar 18–20, 2009)
18. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013: 18th European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 8134, pp. 1–18. Springer, Heidelberg, Germany, Egham, UK (Sep 9–13, 2013)
19. Damgård, I., Orlandi, C.: Multiparty computation for dishonest majority: From passive to active security at low cost. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010. Lecture Notes in Computer Science, vol. 6223, pp. 558–576. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)
20. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
21. Data61: MP-SPDZ (2019), https://github.com/data61/MP-SPDZ
22. Franklin, M.K., Haber, S.: Joint encryption and message-efficient secure computation. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 266–277. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)
23. Franklin, M.K., Haber, S.: Joint encryption and message-efficient secure computation. Journal of Cryptology 9(4), 217–232 (Sep 1996)
24. Frederiksen, T.K., Keller, M., Orsini, E., Scholl, P.: A unified approach to MPC with preprocessing using OT. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9452, pp. 711–735. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015)
25. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)
26. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
27. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099 (2012), https://eprint.iacr.org/2012/099
28. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 145–161. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003)
29. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Sahai, A. (ed.) TCC 2013: 10th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 7785, pp. 600–620. Springer, Heidelberg, Germany, Tokyo, Japan (Mar 3–6, 2013)
30. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 20: 27th Conference on Computer and Communications Security. pp. 1575–1590. ACM Press, Virtual Event, USA (Nov 9–13, 2020)
31. Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster secure multi-party computation of AES and DES using lookup tables. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 17: 15th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 10355, pp. 229–249. Springer, Heidelberg, Germany, Kanazawa, Japan (Jul 10–12, 2017)

32. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 724–741. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
33. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security. pp. 830–842. ACM Press, Vienna, Austria (Oct 24–28, 2016)
34. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 158–189. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018)
35. Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Computing across trust boundaries using distributed homomorphic cryptography. Cryptology ePrint Archive, Report 2019/961 (2019), https://eprint.iacr.org/2019/961
36. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 681–700. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
37. Orsini, E., Smart, N.P., Vercauteren, F.: Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. Lecture Notes in Computer Science, vol. 12006, pp. 254–283. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 24–28, 2020)
38. Rivest, R.: Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer (1999), https://people.csail.mit.edu/rivest/Rivest-commitment.pdf
39. Rotaru, D., Wood, T.: MArBled circuits: Mixing arithmetic and Boolean circuits with active security. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) Progress in Cryptology - INDOCRYPT 2019: 20th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 11898, pp. 227–249. Springer, Heidelberg, Germany, Hyderabad, India (Dec 15–18, 2019)
40. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security. pp. 39–56. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017)
41. Zhu, R., Ding, C., Huang, Y.: Practical MPC+FHE with applications in secure multi-PartyNeural network evaluation. Cryptology ePrint Archive, Report 2020/550 (2020), https://eprint.iacr.org/2020/550

Actively Secure Setup for SPDZ

Dragos Rotaru[1,2], Nigel P. Smart[2], Titouan Tanguy[2], Frederik Vercauteren[2], and Tim Wood[2]

[1] Cape Privacy, New York, U.S.A.
[2] imec-COSIC, KU Leuven, Leuven, Belgium.
dragos@capeprivacy.com, nigel.smart@kuleuven.be,
titouan.tanguy@kuleuven.be, frederik.vercauteren@esat.kuleuven.be,
twood1089@gmail.com

Abstract. We present the first actively secure, practical protocol to generate the distributed secret keys needed in the SPDZ offline protocol. As an added bonus our protocol results in the resulting distribution of the public and secret keys is such that the associated SHE 'noise' analysis is the same as if the distributed keys were generated by a trusted setup. We implemented the presented protocol for distributed BGV key generation within the `SCALE-MAMBA` framework. Our method makes use of a new method for creating doubly (or even more) authenticated bits in different MPC engines, which has applications in other areas of MPC-based secure computation. We were able to generate keys for two parties and a plaintext size of 64 bits in around five minutes, and a little more than eighteen minutes for a 128 bit prime.

This version contains differences from the published version due to an attack on the method in the published version discovered by Victor Treinsoutrot. In this revised version we describe a fix (due to Carsten Baum) which would result in a roughly two fold performance loss in our protocol. This revised version does not update the running times of the protocol precisely, but estimates the expected running time from extrapolating micro-benchmarks. The new expected run times are roughly two times slower than those in the previous version. This file contains, to aid the reader, first the new version and then a `diff` with the published version which appeard in Journal of Cryptology.

Keywords: MPC · Somewhat Homomorphic Encryption · Key Generation

1 Introduction

The SPDZ protocol for Multi-Party Computation (MPC) was introduced in 2012 [20]. This protocol is in the pre-processing family of protocols which are actively secure-with-abort for a dishonest majority of participants. Due to many improvements over the intervening years it provides a highly efficient mechanism to perform MPC for an arbitrary number of participants. However, the protocol comes with a major security issue: namely that it seems to require a trusted setup. This trusted setup is the creation of a public key for the Brakerski-Gentry-Vaikuntanathan [10] (BGV) homomorphic encryption scheme in which the private key is securely distributed amongst the n-parties.

In the original SPDZ paper [20] this was assumed to come from some trusted setup. In the follow up paper [18] a *covertly* secure protocol for generating a suitably distributed set of private keys, and the associated public key was introduced. However, this came with a number of disadvantages, as well as the reduction to just covert security. In particular the distributions of the underlying public keys were different from those one could attain via a

trusted setup, which led to a more complicated noise analysis, and (more importantly) larger parameters which results in a less efficient protocol overall.

In subsequent works the issue of the setup of the public key for the BGV encryption scheme has been dismissed as a setup assumption, which could either be performed in a live system using trusted hardware or via another MPC protocol. Given the complexity of the covertly secure key generation protocol from [18] it has always been assumed that actively secure key generation for SPDZ would require a *complex* MPC protocol to perform it.

The problem of producing distributed key generation for an SHE/FHE scheme arises in other contexts, and not just for the SPDZ setup operation. For example, inspired by Gentry's FHE based passively secure low round complexity MPC protocol from [25], various authors have looked at distributed decryption for FHE/SHE schemes. Each of these proposals needs a method to perform a distributed key generation phase. In [4] the authors present a key generation method for a distributed SHE scheme using various Σ-protocols. To our knowledge this has never been implemented, and the methodology produces a key generation which is different from what would be done via a trusted setup. In [37] a passively secure distributed key generation method is used for threshold SHE schemes, again producing a distribution different from that one would have in a purely trusted setup. Paper [14] introduces a variant of SPDZ, called SPDZ2k, for the case of MPC over the ring $\mathbb{Z}_{2^k}$. The offline phase in this latter work is based on MASCOT, [34]. However, in [39] the authors present an offline protocol based on distributed FHE for SPDZ2k. In all of these cases the contribution of this paper can aid in the creation of the necessary secret keys.

Our Contribution: In this paper we present the first practical *actively secure* distributed key generation method for SPDZ. As an added bonus our method results in virtually identical secret key distributions as in the trusted setup case. In particular the noise analysis for the resulting public key is identical to that one would have if using a trusted setup. Our protocol is also relatively simple, although it does make use of complex generic MPC technology. In particular, our protocol generates a public/private key with exactly the same underlying distribution as the ideal trusted setup does in the `SCALE-MAMBA` system[3], bar the fact we generate secret keys with *expected* Hamming weight h as opposed to *exact* Hamming weight h.

We are also able to generate secret keys from binomial distributions, which can be seen as approximate Gaussian error distributions. These do not suffer from the security concerns that low Hamming weight secret distributions have [16]. In addition, for our purposes, using such keys produces a faster distributed key generation procedure. The effect of using keys selected from a binomial distribution makes the ring parameters slightly bigger (compared to those from an (exact or expected) Hamming weight distribution), this decreases marginally the triple production throughput of the resulting offline phase. On the other hand such a choice has the beneficial effect (see later) of decreasing the runtime of our distributed key generation protocol by about a half. Thus one not only gets a faster key generation method, but the resulting keys do not suffer from the problems outlined in [16].

[3] We use `SCALE-MAMBA` as a reference work throughout this paper as it gives a fixed target (including key sizes) for the final distributed keys we are trying to produce.

Our protocol makes use of a generic MPC functionality for actively secure MPC-with-abort for dishonest majorities over a finite field. This might seem to imply that we require SPDZ to create SPDZ, however this circular dependency is removed by utilizing either the BDOZ protocol [8] or the SPDZ protocol executed with the MASCOT pre-processing phase [34]. The first of these, BDOZ, makes use of n public keys for a linear homomorphic encryption scheme where one private key is held by each player. The second option, MASCOT, is based on Oblivious Transfer. To avoid confusion we will refer to SPDZ with a MASCOT based pre-processing as MASCOT-SPDZ, whereas when we talk about SPDZ we mean the pure SPDZ protocol with a pre-processing based on homomorphic encryption.

Both of these *base* MPC protocols (BDOZ and MASCOT-SPDZ) are not as efficient as the SPDZ protocol based on homomorphic encryption, but we will only be using the base protocols for the one-time setup phase for SPDZ. In particular the underlying generic MPC protocol that we will use for key generation (MASCOT-SPDZ) is $O(n^2)$ in complexity; but we use this to create the distributed secret keys for an MPC protocol (SPDZ) which has complexity $O(n)$.

The overall construction of our protocol is based on four key observations; all of which are relatively simple. Firstly, the generation of the public key data given the secret key data and randomness for a BGV public key is essentially a linear operation and thus comes for free in LSSS based MPC protocols such as BDOZ and MASCOT-SPDZ. Secondly, the BGV public key for SPDZ is a two level BGV scheme thus the ciphertext modulus q needed to construct the BGV public key is a product of two values $q = p_0 \cdot p_1$. In particular the public key is simply the lift to modulo q of the public key modulo p_0 and p_1, performed via the Chinese Remainder Theorem (CRT). If we select p_0 and p_1 to be prime, as `SCALE-MAMBA` does, then we can use two MPC systems (one over p_0 and one over p_1) to perform the operations, and then obtain the final result via application of the CRT. We assume these two MPC systems come as ideal functionalities $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. In practice these will be implemented via MASCOT-SPDZ. Thirdly, all the random values required in BGV key generation can be boiled down to the generation of random bits, which are then processed in various ways. Thus a key issue is how to generate these random bits with the required distributions. Whilst BDOZ and MASCOT-SPDZ can be adapted to produce authenticated (uniformly distributed) random bits as part of their pre-processing, using much the same trick as proposed in [18], this will produce *different* random bits in $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Thus our fourth, and final, observation is that we can produce sharings of the same random bit in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ using an adaption of the daBit method from [3] and [41].

Indeed our new method for daBit is more general and more efficient than the method presented in [3,41]. We require a daBit method which works for two large primes, whereas [3,41] require a method for a large prime and a small prime (in particular the prime two). Our new method deals with any prime size for the two MPC engines, can be extended to more MPC engines than just two, and is built upon an abstraction which allows it to be used with any form of LSSS based MPC engine in the SPDZ family (e.g. BDOZ, MASCOT or SPDZ itself).

The most expensive part of the daBit generation procedure for producing daBits with active security in [3,41] was in verifying consistency of the daBits between the two instances of $\mathcal{F}_{\mathsf{MPC}}$. The idea in these works was to check the same random linear combinations of bits in both instances simultaneously, which was challenging because one field had characteristic 2 and the other some large prime p, which meant the XOR had to be emulated in the prime field, requiring multiplication in MPC. In addition, to generate a bit, one needed to perform XOR in both of the fields under consideration. Our observation in this work is that we use an auxiliary MPC engine for a large prime p to generate bits using the standard square-root trick for generating bits, these are then mapped into the target MPC engines. The auxiliary MPC engine is used to obtain a masked subset sum over the integers, which is then compared to the equivalent masked subset sum in the target MPC engines. Security now reduces to a ~~variant of the Multiple Subset-Sum Problem~~[4] standard application of the Smudging Lemma.

We decided to use MASCOT-SPDZ as the underlying MPC protocol for the BGV key generation, with our implementation building upon the already existing code-base for OT present in the `SCALE-MAMBA` framework. In addition we ran experiments with different values for the standard deviation of the centred binomial distribution, and experiments between Hamming weight restricted secret keys and secret keys generated from a centred binomial distribution. In the fastest case, of standard deviation $\sigma = \sqrt{1/2} = 0.707$ for the centred binomial distribution and FHE keys distributed following this same distribution, for two parties and a ~~64~~ 128 bit plaintext modulus our results show that we can distributively generate BGV keys in around ~~five minutes.~~ 40 minutes. Our original experiments were for a related (insecure) protocol; and achieved 18.5 minutes. Using the modified (secure) method in this version the time would be roughly twice that, i.e. 40 minutes. We ran experiments for 64 and 128 bit primes for the plaintext space for two and three parties for all settings; and for our fastest settings we also ran experiments for four and five parties.

For the parameters used in versions of `SCALE-MAMBA` prior to our work, which used $\sigma = \sqrt{10} = 3.16$ and Hamming weight limited secret keys, we find a key generation time of ~~47 minutes~~ approximately five hours for two parties and a ~~64~~ 128 bit modulus. We give a detailed report of our implementation in Section 7, in which the triple generation throughput and the shared bit throughput are given for the standard case and wall clock time for the whole protocol is given for all our test cases. We note that a variant of our protocol has now been incorporated into the MP-SPDZ software [31] as well as into `SCALE-MAMBA`.

We end this introduction by noting that in [9] a method to perform the SPDZ offline phase using no-communication is presented. However, this method is impractical as currently presented. The method still requires a distributed decryption capability of the underlying SHE scheme. Thus to use this work, even in theory, one needs to be able to generate such distributed keys in a secure manner, such as this work enables. We also note that using the silent-OT method of [9] one may be able to achieve better runtimes. The paper reports that they can achieve 600,000 correlated OT's per second. However, due to the increased

[4] ~~Carsten Baum has pointed out that we can remove this reduction to the subset-sum by increasing, in some (important) cases, the number of bits we throw away. This however results in a less efficient protocol, thus we rely on the Multiple Subset-Sum Problem to obtain an efficient protocol. As our focus is primarily on trying to obtain as efficient a protocol as possible we prefer to keep the reliance on the MSSP problem.~~

computational costs of the silent-OT method this might not translate to the LAN setting in our experiments.

2 Historical Context of SPDZ

For those readers wishing to understand the historical context of SPDZ we now provide some background. For readers interested simply in the results of this paper, they can safely skip this section.

The first application of threshold-like cryptographic schemes to build MPC protocols appears to be the work of Franklin and Haber [22,23]. This utilized an ElGamal style encryption scheme over an RSA-style group to evaluate an MPC protocol over binary circuits. The scheme has, to our knowledge, never been implemented (it would be very inefficient by modern standards) and provides only passive security.

The Franklin-Haber protocol used a so-called 'joint encryption' scheme. True threshold encryption was introduced into MPC protocols by Cramer, Damgård and Nielsen in [15]. This protocol was based on threshold homomorphic encryption and provided a (robust) actively secure protocol in the honest majority setting. The underlying protocol made use of Pailliers encryption scheme, or a variant of the scheme of Franklin and Haber. As such it required a set up phase of generating a secret shared RSA style private key; at the time this was relatively complex but can now be done using the techniques in [13]. The protocol is interesting as it utilizes a trick of keeping an encryption of the data a, as well as $\alpha \cdot a$, for some user chosen key α. Thus this is reminiscent of the usage of the pair-wise MACs in the BDOZ protocol (see below). The overall multiplication protocol required use of public key machinery, and can thus also be considered inefficient by todays standards.

Also in this period the VIFF system [17] developed the pre-processing model for MPC protocols, in VIFF's case for honest-majority protocols. This enabled all the difficult public key machinery to be placed in the offline phase, making the online phase more efficient. The work of [30] formalised the pre-processing model and investigated communication complexity upper and lower bounds in this model.

The modern era of MPC protocols in the case of dishonest majority can be said to start with the work of [19] which showed how to achieve active security for dishonest majority protocols for a small constant additional amount of work. This then led to the information theoretic online phase in the BDOZ [8] and TinyOT [38] protocols. The BDOZ protocol is an n-party dishonest majority protocol for MPC over arithmetic circuits, whilst TinyOT is a two party protocol for binary circuits. Both achieve an essentially information theoretic online phase (the phase where the function is actually evaluated) using information theoretic MACs; the underlying MACs and commitments having been previously introduced ten years earlier in an unpublished manuscript of Rivest [40].

The BDOZ paper followed along the blue-print outlined by Cramer et al. in the case of honest majority, but made a number of crucial changes. Firstly, (as we just said) active security was *cheaply* obtained by using (pair-wise) information theoretic MACs (so called BDOZ-style MACs). Secondly, an efficient online phase was obtained by using the Beaver

circuit randomization technique [6]. Thirdly, the Beaver-triples where produced using a linearly homomorphic encryption scheme. The overall protocol gave a complexity of $O(n^2)$, but could be made highly efficient using a linear homomorphic encryption scheme based on Ring-LWE, which allowed Smart-Vercauteren style SIMD packing to enable a high triple production throughput.

Following closely on the heels of the BDOZ protocol was the SPDZ protocol [20]. This followed much the same methodology, but it replaced the pair-wise information theoretic MACs with global information theoretic MACs (so called SPDZ-style MACs). By using a star-like topology, for MAC-checking and data transfer during the online phase, a total complexity of $O(n)$ could be achieved. The other main advance of SPDZ was the use of a *threshold* Somewhat Homomorphic Encryption scheme of depth two to produce the Beaver triples, as opposed to the linear homomorphic scheme of BDOZ. The use of SHE enabled some of the zero-knowledge proofs required in BDOZ to be removed. Again, by using SIMD packing in a Ring-LWE system, high triple production throughput could be achieved. The Ring-LWE threshold SHE was performed utilizing the techniques developed in [7] for generic lattice-based threshold schemes. However, the use of threshold SHE required a trusted setup assumption (to set up the threshold SHE keys). This issue was side-stepped in the paper [20], and all subsequent papers on SPDZ, until the current one.

Over the years various improvements have been suggested for SPDZ. The most relevant for our discussion are the following ones. The paper [18] introduced various simplifications to the basic SPDZ protocol; the two most important (from a practical perspective) was making it able to compute reactive computations, and replace the depth two SHE scheme with a depth one variant. From the point of view of the current paper, the main contribution of [18] was to try to solve the problem of the trusted setup. It did this by providing only a *covertly* secure key generation protocol for the threshold SHE keys, and in addition the resulting distribution of the underlying noise (inherent in any SHE style key) was less than ideal. This lead to larger parameters for the SHE scheme, and thus a slower protocol.

The next variant of SPDZ developed was to replace the SHE based offline phase with one based on Oblivious Transfer [34]. This offline phase has complexity $O(n^2)$, due to the need for pairwise Oblivious Transfers. But for small n it was seen, for a short while, as being more efficient than the SHE based offline phase. However, in [35] it was shown that, by providing more efficient zero-knowledge proofs in the case of the SHE-based variants, the OT-based variants were no longer competitive. The paper [35] provided two forms of pre-processing for SPDZ. One based on linearly homomorphic encryption (essentially adapting BDOZ style pre-processing for the SPDZ situation) which was efficient for small n. This variant was called LowGear. A second variant, for large n, utilized a joint zero-knowledge proof, as opposed to pair-wise proofs in [18], to obtain a performance boost for SHE based pre-processing. This latter variant was denoted HighGear.

In [5] an improvement to HighGear was made, denoted TopGear, which improved the zero-knowledge proofs in the SHE based pre-processing. TopGear is more efficient than HighGear, but there is no experimental evidence as to whether the TopGear improvements mean

LowGear is no longer competitive for small n. This is perhaps a direction for future investigation.

Thus the only outstanding issue regarding SPDZ is the fact that there is no methodology to generate the required threshold SHE keys. The problem is to ensure the underlying *distributions* of the secret keys are correct and do not create performance problems later on. This is the main technical contribution of this paper; to obtain active security we utilize the SPDZ system as the MACs to obtain active security introduce very little additional cost over a passively secure variant. Given an efficiently passively secure protocol for this task, one can easily generate (theoretically) an actively secure one by utilizing zero-knowledge proofs. However, by utilizing a sub-protocol such as MASCOT-SPDZ we achieve the additional active security virtually 'for free' via the use of the SPDZ MACs.

3 Preliminaries

In this section we provide the necessary background on the type of BGV public key we need to produce, as well as the underlying distributions and the base MPC protocols we will be using.

3.1 Cyclotomic Rings and Distributions over such Rings

The BGV encryption scheme is defined over a cyclotomic ring $R = \mathbb{Z}[X]/(X^N + 1)$, where for our purposes we take N to be a power of two. Thus $X^N + 1$ is the $m = 2 \cdot N$-th cyclotomic polynomial, and $N = \phi(m)$. We let $\odot$ denote the multiplication operation in R.

Following [26][Full version in [27], Appendix A.4] the `SCALE-MAMBA` system utilizes the following distributions in the key generation procedure.

- $\mathsf{HWT}(h, N)$: This generates a vector of length N with elements chosen at random from $\{-1, 0, 1\}$ subject to the condition that the number of non-zero elements is equal to h.
- $\mathsf{dN}(\sigma^2, N)$: This generates a vector of length N with elements chosen according to an *approximation* to the discrete Gaussian distribution with variance σ^2, by sampling from a centred binomial distribution.
- $\mathsf{U}(q, N)$: This generates a vector of length N with elements generated uniformly modulo q.

In particular for the distribution $\mathsf{dN}(\sigma^2, N)$ `SCALE-MAMBA` approximates $\mathsf{dN}(\sigma^2, N)$ using the approximation from [1]. In particular $\mathsf{dN}(\sigma^2, N)$ is replaced by the centred binomial distribution where elements are returned using the formula

$$c_j = \sum_{i=0}^{k-1} b_{2 \cdot i} - b_{2 \cdot i + 1}$$

for uniformly random bits $b_j \in \{0, 1\}$ for $j = 0, \ldots, 2 \cdot k - 1$. The default settings of `SCALE-MAMBA` use $k = 20$, giving us $\sigma = \sqrt{k/2} = \sqrt{10} = 3.16$.

We make a small change to one of the above distributions in our work. The distribution $\mathsf{HWT}(h, N)$ is used to sample the secret key, where in [26] (and in `SCALE-MAMBA`) the value

h is selected to be a power of two; in particular $h = 64$. In our work we replace $\mathsf{HWT}(h, N)$ with the distribution which picks each coefficient with respect to the Bernoulli distribution $B(h/N)$. Thus we use the approximation $\mathsf{HWT}(h, N) \approx B(h/N)^N$. The Hamming weight of the vectors output by this distribution follows a binomial distribution with mean h. We still use $h = 64$ in our recommended construction though. The "noise analysis" behind the homomorphic operations used in the SPDZ protocol are easily checked not to be affected by this change, and in addition the security arguments for using low Hamming weight secret keys (as discussed in [26]) are also not affected. In particular the noise analysis used in [26], or `SCALE-MAMBA`, is an 'average case' analysis in the key generation, thus the standard deviation in the canonical norm of the secret key is $\sqrt{h}$ if an exact Hamming weight of h is used. It is this standard deviation which is the contributing term in the noise analysis. If one generates the secret key using only an expected Hamming weight then you obtain the same standard deviation; thus nothing changes in the analysis by using our slightly different secret key distribution.

3.2 The BGV Key Generation Procedure

For a modulus q we let R_q denote the above ring localised at the modulus q, i.e. $R_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N + 1)$. The SPDZ protocol requires a two-leveled scheme with ciphertext moduli $q_0 = p_0$ and $q_1 = p_0 \cdot p_1$. The plaintext modulus for the BGV scheme is defined by p. We require, for efficiency, that

$$\begin{aligned} p_1 &\equiv 1 \pmod{p}, \\ p_0 - 1 \equiv p_1 - 1 \equiv p - 1 &\equiv 0 \pmod{N}, \end{aligned}$$

where p is the plaintext modulus. The moduli p_0 and p_1 are selected to be distinct primes for efficiency. In which case, by the CRT, we have $R_q \cong R_{p_0} \times R_{p_1}$. In addition, due to the above restrictions on the primes p_0 and p_1, there is an efficient FFT algorithm on R_{p_i}, which requires no extension field arithmetic. Thus one can efficiently multiply in R_{p_i} by executing

$$\mathbf{a} \odot \mathbf{b} = \mathsf{FFT}^{-1}(\mathsf{FFT}(\mathbf{a}) \cdot \mathsf{FFT}(\mathbf{b}))$$

where $\cdot$ here is the component wise product. Note that the FFT operation is a *linear* operation and thus can be executed in an MPC engine for free. These facts we shall use in our distributed key generation protocol.

The two ciphertext moduli q_0 and q_1 are used to define ciphertexts at level zero and level one, see [26] for further details as to how this aids efficiency. Since SPDZ only requires a depth one SHE scheme we only need to define two such ciphertext moduli.

The BGV public key is of the form $(\mathbf{a}, \mathbf{b}) \in R_q$ where

$$\mathbf{a} \leftarrow \mathsf{U}(q, N) \quad \text{and} \quad \mathbf{b} = \mathbf{a} \odot \mathfrak{sk} + p \cdot \mathbf{e}$$

where $\mathbf{e} \leftarrow \mathsf{dN}(\sigma^2, N)$. The secret key $\mathfrak{sk}$ for our purposes will be selected from the distribution $B(h/N)^N$. We also require, for the SPDZ protocol, the switching key data $(\mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}, \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2})$ which is of the form

$$\mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{U}(q, N) \quad \text{and} \quad \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2} = \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2} \odot \mathfrak{sk} + p \cdot e_{\mathfrak{sk},\mathfrak{sk}^2} - p_1 \cdot \mathfrak{sk}^2$$

where $\mathbf{e}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{dN}(\sigma^2, N)$.

The goal in a distributed key generation protocol for the SPDZ system is to output the public values $\mathfrak{pk} = (\mathbf{a}, \mathbf{b}, \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}, \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2})$ to all players, whilst player P_i obtains a value $\mathfrak{sk}_i \in R_q$ such that

$$\mathfrak{sk} = \mathfrak{sk}_1 + \ldots + \mathfrak{sk}_n \pmod{q}.$$

We also require that no party can influence the choice of secret key, and no proper subset of the n parties can deduce any information about the secret key, bar what can be deduced from the public key. Thus we aim to create a protocol which securely realizes the functionality given in Figure 1, where $\mathsf{ParamGen}(1^\kappa, \log_2 p, n)$ is a function which produces the system parameters (p, p_0, p_1). We allow all parties, bar P_1, to input their precise share of the secret key. We then compute P_1's share according to the shares of other parties and the internally sampled secret key $\mathfrak{sk}$. Therefore P_1 has no input, and is treated differently from other parties. This is mirrored in our protocol, the parties generate a secret key but the *shares* of the parties $P_2, \ldots, P_n$ are selected by those parties; with party P_1 receiving their share from the protocol.

Functionality $\mathcal{F}_{\mathsf{KeyGen}}$

1. When receiving the message *start* from all honest parties, run $P \leftarrow \mathsf{ParamGen}(1^\kappa, \log_2 p, n)$, and then, using the parameters generated, run $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}()$ (P, and hence 1^κ, is an implicit input to all functions we specify in this paper). Send $\mathfrak{pk} = (\mathbf{a}, \mathbf{b}, \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}, \mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2})$ to the adversary.
2. Receive from the adversary a set of shares $\mathfrak{sk}_j \in R_q$ for each corrupted party P_j for $j \neq 1$.
3. Construct a complete set of shares $(\mathfrak{sk}_1, \ldots, \mathfrak{sk}_n)$ consistent with the adversary's choices and $\mathfrak{sk}$. This is done by selecting $\mathfrak{sk}_i$ uniformly at random for honest i, subject to the constraint that $\mathfrak{sk} = \sum \mathfrak{sk}_i$. Note that this is always possible since the corrupted players form an unqualified set.
4. The functionality waits for an input from the adversary.
5. If this input is Deliver then send $\mathfrak{pk}$ to all players and $\mathfrak{sk}_i$ to each honest player P_i, and send $\mathfrak{sk}_1$ to player P_1 if P_1 is dishonest.
6. If the adversarial input is not equal to Deliver then abort.

Figure 1. The ideal functionality for key generation (adapted from [20]).

We aim to realize this functionality while doing only black box calls to the underlying MPC modulo p_0 and p_1 functionalities. However, due to the concerns raised in [16] in relation to low Hamming weight keys, we also examine the case of secret keys generated by a centred binomial distribution; namely when we select $\mathfrak{sk}$ from $\mathsf{dN}(\sigma^2, N)$. These lead to slightly larger parameters for the underlying FHE systems, but the method to produce the keys is simpler.

3.3 Base MPC Protocols

In Figure 2 we present the MPC functionality for our base MPC protocols, either BDOZ or MASCOT-SPDZ in the case where we are generating keys or SPDZ when we are doing traditional daBit generation.

To simplify presentation of protocols using this functionality we shall represent a value held in the memory of such an MPC functionality by $\langle x \rangle_p$, and then addition and multipli-

cation of such elements will be represented by

$$\langle x \rangle_p + \langle y \rangle_p, \qquad \langle x \rangle_p \cdot \langle y \rangle_p.$$

For inputing and outputting values to/from a player/all players we will write

$$\langle x \rangle_p \leftarrow \mathsf{Input}(P_i), \qquad P_i \leftarrow \mathsf{Output}(\langle x \rangle_p), \qquad x \leftarrow \mathsf{Open}(\langle x \rangle_p).$$

That the BDOZ, MASCOT-SPDZ and SPDZ protocols implement such a functionality securely can be found proved in the respective papers [8], [34] and [20].

Functionality $\mathcal{F}^p_{\mathsf{MPC}}$

The functionality runs with parties $P_1, \ldots, P_n$ and an ideal adversary. Let $\mathcal{A}$ be the set of corrupt parties. Given a set I of valid identifiers, all values are stored in the form $(varid, x)$, where $varid \in I$.

Initialize: On input $(Init, p)$ from all parties, with p a prime, the functionality stores p. The adversary is assumed to have statically corrupted a subset $\mathcal{A}$ of the parties.
Input: This takes input $(Input, P_i, varid, x)$ from P_i, with $x \in \mathbb{F}_p$, and $(input, P_i, varid, ?)$ from all other parties, with *varid* a fresh identifier. If the *varid*'s are the same the functionality stores $(varid, x)$, otherwise it aborts.
Add: On command $(Add, varid_1, varid_2, varid_3)$ from all parties:
1. If $varid_1, varid_2$ are not present in memory or $varid_3$ is then the functionality aborts.
2. The functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x + y)$.

Multiply: On input $(Multiply, varid_1, varid_2, varid_3)$ from all parties:
1. If $varid_1, varid_2$ are not present in memory or $varid_3$ is then the functionality aborts.
2. The functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y)$.

Output: On input $(Output, varid, i)$ from all parties (if *varid* is present in memory),
1. The functionality retrieves $(varid, y)$.
2. If $i = 0$ and $\mathcal{A} \neq \emptyset$ then the functionality outputs y to the adversary, otherwise it outputs $\perp$ to the adversary.
3. The functionality waits for an input from the adversary.
4. If this input is Deliver then y is output to all players if $i = 0$, or y is output to player i if $i \neq 0$.
5. If the adversarial input is not equal to Deliver then abort.

Figure 2. The ideal functionality for MPC with abort over $\mathbb{F}_p$.

The underlying authentication mechanism, in practice, can be assumed to be one of SPDZ style MACs, i.e. there is a globally held shared secret value $\alpha_p \in \mathbb{F}_p$, shared with respect to an additive sharing amongst the n-players $\alpha_p = \alpha_{1,p} + \cdots + \alpha_{n,p}$. For each value x the $\langle x \rangle_p$ will be an additive sharing of $x = x_1 + \cdots + x_n$, as well as an additive sharing of $\alpha_p \cdot x = \gamma_1 + \cdots + \gamma_n$. In other words in the real world when a party P_i holds the value $\langle x \rangle_p$, it actually holds the value (x_i, γ_i).

In such MPC protocols addition, and in fact any linear operation, is a 'free' operation, whereas multiplication will be assumed to take a single 'time' unit of operation. Another metric one often examines is the round complexity, in this case a multiplication takes one round, but multiplications which can be performed in parallel also only take one round of operation. Inputing, outputting or opening a data item also requires one round of communication, and such operations can be performed in parallel.

3.4 The $\mathcal{F}^B_{\mathsf{Rand}}(M)$ Functionality

We also require a functionality $\mathcal{F}^B_{\mathsf{Rand}}(M)$ which allows the parties to agree on M random values in the range $[0, \ldots, B)$. In practice this can be implemented by all parties committing to a seed, then the parties open the seeds. The seeds are then XOR'd together to produce a single shared seed, which is passed as the key to a PRF to produce the shared random values. We present this as an ideal functionality in Figure 3.

Functionality $\mathcal{F}^B_{\mathsf{Rand}}(M)$

1. On input $(\mathsf{Rand}, \text{cnt})$ from all parties, if the counter value is the same for all parties and has not been used before, the functionality samples $r_i \leftarrow [0, \ldots, B)$ for $i = 1, \ldots, M$.
2. The values r_i are sent to the adversary, and the functionality waits for an input.
3. If the input is $\mathsf{Deliver}$ then the values r_i are sent to all parties, otherwise the functionality aborts.

Figure 3. The ideal $\mathcal{F}^B_{\mathsf{Rand}}(M)$ functionality.

4 maBits: Generating Multiply Authenticated Bits

The main problem in performing actively secure key generation for SPDZ is to produce secure randomly shared bits within the MPC functionalities; in which the bit is zero with probability 1/2, and one with probability 1/2. We will require that the MPC functionality $\mathcal{F}^p_{\mathsf{MPC}}$, for a specific prime p, is extended by a command which we model via the ideal functionality $\mathcal{F}^p_{\mathsf{MPC}}.\mathsf{GenBit}()$ given in Figure 4. The reader should note that, the adversary is not able to choose their shares for the shared random bit in this functionality. This is important for our security proof.

Functionality $\mathcal{F}^p_{\mathsf{MPC}}.\mathsf{GenBit}()$

1. The functionality waits for a message abort or ok from the adversary. If the message is ok then it continues.
2. The functionality then samples a bit $b \in \{0, 1\}$ and stores it in the functionality $\mathcal{F}^p_{\mathsf{MPC}}$.
3. A sharing (b_i, γ_i) of the bit b is ~~returned to the players~~generated, and the adversaries share values are returned to it.

Figure 4. The ideal functionality for single random bits.

A protocol for BDOZ and MASCOT-SPDZ for $\mathcal{F}^p_{\mathsf{MPC}}.\mathsf{GenBit}()$ is given in Figure 5, adapted from [18]. The basic idea is to generate a secret random value x modulo p, square it, open the value, take the square root (if it is not zero) so as to obtain $z = \pm x$. Then a value in $\{-1, 1\}$ is obtained by computing x/z, which can then be mapped onto the any desired two element set[4].

[4] If the underlying MPC system is SPDZ based then a more efficient way to perform the method is using the FHE pre-processing instead of directly within the Offline phase as implied by the given protocol. But this assumes the pre-processing is FHE based, which it will not be in our application.

Entering a Random Bit $\Pi^p_{\mathsf{MPC}}.\mathsf{GenBit}()$

1. For $i = 1, \ldots, n$ execute $\langle x_i \rangle_p \leftarrow \mathsf{Input}(P_i)$, where x_i is a random element in $\mathbb{F}_p$.
2. $\langle x \rangle_p \leftarrow \sum_{i=1}^{n} \langle x_i \rangle_p$.
3. $\langle y \rangle_p \leftarrow \langle x \rangle_p \cdot \langle x \rangle_p$.
4. $y \leftarrow \mathsf{Open}(\langle y \rangle_p)$.
5. If $y = 0$ then restart the process.
6. $z \leftarrow \sqrt{y} \pmod{p}$, picking the value $z \in [0, \ldots, p/2)$.
7. $\langle a \rangle_p \leftarrow \langle x \rangle_p / z$.
8. $\langle b \rangle_p \leftarrow (\langle a \rangle_p + 1)/2$.
9. Return $\langle b \rangle_p$.

Figure 5. 'Standard' method to produce a shared random bit in Π^p_{MPC}ssuming p is odd.

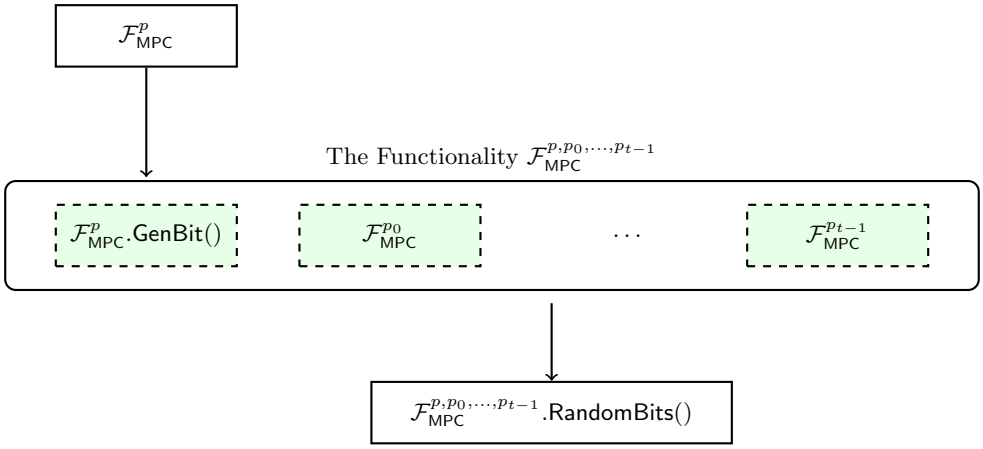

Fig. 6. Our functionalities.

However, we have a problem; if we execute this procedure with respect to both $\mathcal{F}^{p_0}_{\mathsf{MPC}}$ and $\mathcal{F}^{p_1}_{\mathsf{MPC}}$ then we will obtain two shared random bits $\langle b_0 \rangle_{p_0}$ and $\langle b_1 \rangle_{p_1}$ but we will not necessarily have $b_0 = b_1$. To obtain shared random bits in the two MPC systems which are *identical* we need to adapt the daBit idea from [41]. In that paper it is shown how to obtain identical shared random bits in two MPC systems; one being a SPDZ-like system modulo p, and one being a BDOZ-like system over $\mathbb{F}_2$ based on OT for garbled circuit style computations. In our key generation protocol we require shared random bits in two SDPZ-like systems for large moduli p_0 and p_1 (in practice MASCOT-SPDZ systems). This makes the protocol to generate the shared random bits a little easier to understand than the one considered in [41]. Indeed we present a more general protocol than that which is needed for our key generation method. Our new method includes the case considered in [41], and is more efficient than the improved method considered in [3] when at least one prime is 'large'.

For our generalisation we consider a set of t SPDZ-like MPC systems with moduli $p_0, \ldots, p_{t-1}$. Our goal is to generate shares $\langle b \rangle_{p_i}$ in all of these systems where $b \in \{0, 1\}$. Our method makes no restriction on the size of the primes p_i, nor the underlying SPDZ-like MPC engine, thus our method can be used as a replacement for the daBit methods in [3,41] as well.

Security of our protocol is defined by two security parameters sec and stat. The value sec is such that the probability of an adversary breaking the soundness of the protocol is $2^{-\mathsf{sec}}$, whereas stat is such that the statistical difference between the execution in the real and ideal worlds is bounded by $2^{-\mathsf{stat}}$. The reader can think of $\mathsf{sec} = 128$ and $\mathsf{stat} = 80$ as is common in the literature.

We define p_{min} to be $\mathsf{min}(p_0, \ldots, p_{t-1})$ and we let γ be the smallest integer such that $p_{\mathsf{min}}^{\gamma} > 2^{\mathsf{sec}}$~~, where sec is our security parameter.~~. In practical key generation for SPDZ we will have $\gamma = 1$, but we keep the discussion general to enable other applications of the technique in this section.

For efficiency we will generate ~~these~~ the shared bits in batches of m at a time. We define a masking parameter W which satisfies

$$\lceil W/p_{\mathsf{min}} \rceil < W \cdot 2^{-\mathsf{sec}/\gamma}.$$

We set $T = \lceil \mathsf{stat} + \log_2 W + \log_2 m \rceil$, and define an auxiliary prime number p which satisfies

$$p > ~~(~~m ~~+\gamma~~ \cdot \mathsf{sec} ~~)~~ W \cdot 2^{\mathsf{sec}\,\mathsf{stat}} = 2^T.$$

The prime p can be the same as one of the primes p_i above if one of the p_i are large enough, but in this treatment for ease of exposition we treat it as a separate prime. It is with this prime p that we will generate our initial bits, using the protocol in Figure 5.

Thus we have $t+1$ independent MPC functionalities which we group into a single functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}$, as depicted in Figure 6. On their own, at first sight, these $t+1$ functionalities would not allow us to generate a shared random bit modulo $p_0, \ldots, p_{t-1}$. To obtain this extra ability we further extend the functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}$ to $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$; where the additional command is presented in Figure 7. The functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$ internally generates a set of m secure bits.

Functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$

1. The functionality waits for input of a positive integer m from all players. If the inputted values are not equal then the functionality aborts.
2. For $i = 1, \ldots, m$ the functionality executes the code of $\mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}()$ so as to generate a bit b_i.
3. The bits b_i are retrieved from $\mathcal{F}_{\mathsf{MPC}}^{p}$ and are enterred into the functionalities $\mathcal{F}_{\mathsf{MPC}}^{p_j}$ for $j = 0, \ldots, t-1$.
4. The functionality waits for a message abort or ok from the adversary. If the message is ok then it continues.

Figure 7. The ideal functionality for random bits.

Our protocol will make use of the following result

Lemma 4.1. *Let p be a prime, $n \in \mathbb{Z}_{\geq 2}$ and $x_i \in [0, \ldots, p)$ be uniformly chosen subject to the condition that*

$$x_1 + \ldots + x_n = \begin{cases} k \cdot p, & or \\ k \cdot p + 1. \end{cases}$$

Set $\Delta = \lceil p/n \rceil$ *and write* $x_i = l_i + \Delta \cdot h_i$ *with* $0 \leq l_i < \Delta$, *then*

$$k = \left\lceil \frac{\Delta \cdot \sum h_i}{p} \right\rceil.$$

with probability at least $1 - 3/p$.

Proof. We have, for $\epsilon \in \{0, 1\}$,

$$k = \frac{\Delta \cdot \sum h_i}{p} + \frac{\sum l_i}{p} - \frac{\epsilon}{p}.$$

We have $0 \leq \sum l_i < p$ by construction, and so the equality on k will follow as long as $\sum l_i \geq \epsilon$. But this always happens unless $\epsilon = 1$ and $\sum l_i = 0$. If $l_i = 0$ then this means that Δ divides x_i. Since x_i is uniformly chosen in $[0, \ldots, p)$ for $i = 1, \ldots, n$ (subject to the condition that $\sum x_i \pmod p \in \{0, 1\}$), and $\Delta = \lceil p/n \rceil$ we have $l_i = 0$ with probability $(n+1)/p$ for each $i \in [0, \ldots, n)$. Thus the probability that all the l_i are zero is less than $((n+1)/p)^{n-1}$, this is at most $3/p$. □

In Figure 8 we explain our protocol $\Pi_{\mathsf{RandomBits}}$ for producing shared random bits in the two MPC systems. Intuitively the protocol works as follows. The parties first generate, in step 2a, ~~$m + \gamma \cdot \mathsf{sec}$~~ $m + \gamma \cdot T$ shared random bits in the MPC engine $\mathcal{F}^p_{\mathsf{MPC}}$ using the command GenBit. They then allow, in steps 2c to 2e, player P_1 to determine the associated k value for each shared bit using Lemma 4.1, this does not reveal any information about the hidden bit (and will be correct with probability $3/p$), but this clearly reveals some information about the sharing[5]. Thinking of the sharing now as over the integers, and then reducing modulo p_i, player P_1 can adjust his sharing so that the bit is correctly shared modulo p_i. These shares are then input into the MPC engines $\mathcal{F}^{p_i}_{\mathsf{MPC}}$. Assuming all parties are honest we now have a valid sharing, however due to Lemma 4.1 even honest parties have a probability of aborting of ~~$3 \cdot (m + \gamma \cdot \mathsf{sec})/p$~~ $3 \cdot (m + \gamma \cdot T)/p$, due to one, or more, of the k_i values being computed incorrectly.

To cope with dishonest parties we check the parties are honest by verifying random linear combinations. We verify γ random linear combinations in total. This is the purpose of steps 4 and 5. Here we note that the initial sharing in $\mathcal{F}^p_{\mathsf{MPC}}$ is guaranteed to be a sharing of a bit due to the active security of the operation GenBit in $\mathcal{F}^p_{\mathsf{MPC}}$. Opening a random linear combination S (in step 5d) of the shared bits in $\mathcal{F}^p_{\mathsf{MPC}}$ is then a subset-sum over the integers, due to the lower bound on p. We then compare this to the associated sum modulo p_i obtained from $\mathcal{F}^{p_i}_{\mathsf{MPC}}$ in step 5f. This has to be repeated γ times to cope with the smallest value of p_i. ~~We thus obtain an instance of the Multiple-Subset-Sum-Problem (MSSP) considered in [?], which we shall discuss below~~ Due to the masking with a uniformly random value in $[0, \ldots, 2^T]$ the original subset sum on the first m bits (which is of size at most $2^{T-\mathsf{stat}}$) is statistically hidden.

[5] In our security proof we show that this specific information can be perfectly simulated by the simulator, and leaks no information about the actual shared value.

Protocol $\Pi_{\mathsf{RandomBits}}$

This protocol works in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}$-hybrid model, and on input of m from all players executed the following steps:

1. Set $\Delta = \lceil p/n \rceil$ and $T = \lceil \mathsf{stat} + \log_2 W + \log_2 m \rceil$.
2. For ~~$i = 1, \ldots, m + \gamma \cdot \mathsf{sec}$~~ $i = 1, \ldots, m + \gamma \cdot T$ do
   (a) $\langle b_i \rangle_p \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}()$.
   (b) Let ~~$b_{i,j}$ denote Player j~~ $b_{i,\ell}$ denote Player ℓ's value such that ~~$\sum b_{i,j} = b_i \pmod p$~~ $\sum b_{i,\ell} = b_i \pmod p$.
   (c) Player ~~P_j writes $b_{i,j} = l_{i,j} + \Delta \cdot h_{i,j}$ with $0 \le l_{i,j} < \Delta$~~ P_ℓ writes $b_{i,\ell} = l_{i,\ell} + \Delta \cdot h_{i,\ell}$ with $0 \le l_{i,\ell} < \Delta$.
   (d) Player ~~P_j sends $h_{i,j}$~~ P_ℓ sends $h_{i,\ell}$ to Player P_1.
   (e) Player P_1 sets

$$k_i = \left\lceil \frac{\Delta \cdot \sum_j h_{i,j}}{p} \; \frac{\Delta \cdot \sum_\ell h_{i,\ell}}{p} \right\rceil.$$

   (f) For ~~$j = 0, \ldots, t-1$~~ $v = 0, \ldots, t-1$ do
   - Party P_1 calls ~~$\langle b_i^{(1)} \rangle_{p_j} \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p_j}.\mathsf{Input}(b_{i,1} - k_i \cdot p \pmod{p_j})$~~ $\langle b_i^{(1)} \rangle_{p_v} \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p_v}.\mathsf{Input}(b_{i,1} - k_i \cdot p \pmod{p_v})$.
   - Party P_ℓ, for $\ell \neq 1$, calls ~~$\langle b_i^{(\ell)} \rangle_{p_j} \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p_j}.\mathsf{Input}(b_{i,\ell} \pmod{p_j})$~~ $\langle b_i^{(\ell)} \rangle_{p_v} \leftarrow \mathcal{F}_{\mathsf{MPC}}^{p_v}.\mathsf{Input}(b_{i,\ell} \pmod{p_j})$.
   - The parties compute ~~$\langle b_i \rangle_{p_j} = \sum_{\ell=1}^{n} \langle b_i^{(\ell)} \rangle_{p_j}$~~ $\langle b_i \rangle_{p_v} = \sum_{\ell=1}^{n} \langle b_i^{(\ell)} \rangle_{p_v}$.
3. Divide the shared bits $\langle b_i \rangle_{p_v}$ for $i \ge m$ into γ sets of size T; denoting the shared bits as $\langle b_{j,k} \rangle_{p_v}$ for $j = 1, \ldots, \gamma$ and $k = 1, \ldots, T$.
4. The parties initialize an instance of the functionality ~~$\mathcal{F}_{\mathsf{Rand}}^{2^{\mathsf{sec}}}$~~ $\mathcal{F}_{\mathsf{Rand}}^{W}$. [Note, this needs to be done after all the previous steps, so the parties have no prior knowledge of the output].
5. For $j = 1, \ldots, \gamma$ do
   (a) For ~~$i = 1, \ldots, m + \gamma \cdot \mathsf{sec}$ generate $r_{i,j} \leftarrow \mathcal{F}_{\mathsf{Rand}}^{2^{\mathsf{sec}}}(m + \gamma \cdot \mathsf{sec})$~~ $i = 1, \ldots, m$ generate $r_{i,j}$ via $(r_{i,j})_i \leftarrow \mathcal{F}_{\mathsf{Rand}}^{W}(m)$.
   (b) Compute for $v = 0, \ldots, t-1$ the sharings ~~$\langle S_{j,v} \rangle_{p_v} = \sum_i r_{i,j} \cdot \langle b_i \rangle_{p_v}$~~ $\langle S_{j,v} \rangle_{p_v} = \sum_{i=1}^{m} r_{i,j} \cdot \langle b_i \rangle_{p_v} + \sum_{k=1}^{T} 2^{k-1} \cdot \langle b_{j,k} \rangle_{p_v}$.
   (c) Compute ~~$\langle S_j \rangle_p = \sum_i r_{i,j} \cdot \langle b_i \rangle_p$~~ $\langle S_j \rangle_p = \sum_{i=1}^{m} r_{i,j} \cdot \langle b_i \rangle_p + \sum_{k=1}^{T} 2^{k-1} \cdot \langle b_{j,k} \rangle_p$.
   (d) $S_j \leftarrow \mathsf{Open}(\langle S_j \rangle_p)$.
   (e) $S_{j,v} \leftarrow \mathsf{Open}(\langle S_{j,v} \rangle_{p_v})$ for ~~$j = 0, \ldots, t-1$~~ $v = 0, \ldots, t-1$
   (f) Abort if $S_j \pmod{p_v} \neq S_{j,v}$ for any $v = 0, \ldots, t-1$.
6. Output ~~$\langle b_i \rangle_{p_j}$~~ $\langle b_i \rangle_{p_v}$ for $i = 1, \ldots, m$ and ~~$j = 0, \ldots, t-1$~~ $v = 0, \ldots, t-1$.

Figure 8. Method to produce m shared random bits in $\mathcal{F}_{\mathsf{MPC}}^{p_0}, \ldots, \mathcal{F}_{\mathsf{MPC}}^{p_{t-1}}$.

The protocol $\mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}()$ in Figure 5 requires one secure multiplication (per bit generated) and two rounds of communication (as a multiplication also requires a round of communication). To execute the rest of $\Pi_{\mathsf{RandomBits}}$ requires four rounds of communication (one for the initial opening to P_1, one for input into the MPC engines, one for executing ~~$\mathcal{F}_{\mathsf{Rand}}^{2^{\mathsf{sec}}}$~~ $\mathcal{F}_{\mathsf{Rand}}^{W}$ and one for the final opening). If the ~~$m+\gamma\cdot\mathsf{sec}$~~ $m+\gamma\cdot T$ bits required in $\Pi_{\mathsf{RandomBits}}$ are produced in parallel, as well as the various input/open operations etc, this means that protocol $\Pi_{\mathsf{RandomBits}}$ requires ~~$m+\gamma\cdot\mathsf{sec}$~~ $m+\gamma\cdot T$ secure multiplications in $\mathcal{F}_{\mathsf{MPC}}^{p}$ and $2+4=6$ rounds of communication.

4.1 ~~Multiple Subset Sum Problem~~

~~Given weights $w_{i,j}\in\mathbb{Z}_{>0}$ for $i=1,\ldots,\gamma$ and $j=1,\ldots,\nu$ and target values $s_1,\ldots,s_\gamma\in\mathbb{Z}$, the MSSP problem is to find values $b_i\in\{0,1\}$ such that~~

$$\sum_{j=1}^{\nu} w_{i,j}\cdot b_j = s_i \quad \text{for } i=1,\ldots,\gamma.$$

~~In matrix notation we can write this as given $W\in\mathbb{Z}_{>0}^{\gamma\times\nu}$ and a vector $\mathbf{s}\in\mathbb{Z}^{\gamma}$ to find $\mathbf{b}\in\{0,1\}^{\nu}$ such that $W\cdot\mathbf{b}=\mathbf{s}$. This generalises the standard Subset Sum Problem of Given weights $w_j\in\mathbb{Z}_{>0}$ for $j=1,\ldots,\nu$ and a target value $s\in\mathbb{Z}$, the SSP is to find to values $b_i\in\{0,1\}$ such that~~

$$\sum_{j=1}^{\nu} w_j\cdot b_j = s.$$

~~Just as the single subset-sum problem has a notion of density, for which a low value imples one can easily find solutions, the MSSP also has a notion of density. We define the density of an MSSP~~[6]

$$d=\frac{\nu}{\gamma\cdot\max_{i,j}\log a_{i,j}}.$$

~~The following result on the MSSP generalizes the classical result for the standard SSP, that low-density subset sums are 'easy', If $d<0.9408$ then the MSSP problem can 'almost always' be solved with a single call to a lattice oracle. In this work we restrict to MSSP problems with high density, i.e. $d\geq 1$. In our protocol the density of the subset sums S over the integers, which are revealed, is given by~~

$$d=\frac{m+\gamma\cdot\mathsf{sec}}{\gamma\cdot\mathsf{sec}}>1,$$

~~although the subset-sum component which actually contributes to the security~~[6] ~~of the m-bits is of size $\gamma\cdot\mathsf{sec}$. Informally, we note that the security of the protocol follows from the security of the underlying MPC engines, except for the leaked information. Thus we need to argue that the leaked information reveals no information about the underlying honest players' data values, and that even in the presence of malicious players the output is correct (i.e. m shared bits are the same in all $\mathcal{F}_{\mathsf{MPC}}^{p_j}$). The main potential leakage of information comes from the opened subset sum values over the integers, i.e. S. To deal with this leaked information we consider the following (decision) variant of the subset sum problem.~~

[6] ~~The density of a standard subset sum problem is given by $d=\frac{\nu}{\max_i\log a_i}$.~~

[6] ~~See the proof of the theorem below.~~

The MSSG problem is parametrized by three positive integers ν, γ and sec. The parameters γ defines the number of subset sums, sec is a security parameter controlling the sizes of the weights and ν controls the number of terms. The problem is only defined when $\nu \geq \gamma \cdot \mathsf{sec}$.

Given a set of random weights $w_{i,j} \in [0, \ldots, 2^{\mathsf{sec}})$ for $i = 1, \ldots, \nu$ and $j = 1, \ldots, \gamma$, define $\mathsf{mx}_j = \sum_i w_{i,j}$. The problem is to distinguish between the two different distributions:

1. In the first distribution the challenger picks random bits $b_i \in \{0, 1\}$ and sets $T_j = \sum_i b_i \cdot w_{i,j}$. The values $(T_1, \ldots, T_\gamma)$ are returned to the adversary. We write this as $\{T_j\} \leftarrow D_1$.
2. In the second distribution the challenger samples values $T_j \in [0, \ldots, \mathsf{mx}_j]$ uniformly at random and returns it to the adversary. We write this as $\{T_j\} \leftarrow D_2$.

If $\mathcal{A}$ is an adversary then we define the advantage of $\mathcal{A}$ in solving this problem by

$$\mathrm{Adv}^{\mathsf{MSSG}}_{\mathcal{A}}(\nu, \gamma, \mathsf{sec}) = 2 \cdot \Big| \Pr\Big[\mathcal{A}(\ \{w_{i,j}\},\ \{T_j\}\ ) = b \mid b \leftarrow \{1, 2\},\ w_{i,j} \leftarrow w_{i,j} \in [0, \ldots, 2^{\mathsf{sec}}),\ \{T_j\} \leftarrow D_b \Big] - 1/2 \Big|.$$

We say the problem is hard if $\mathrm{Adv}_{\mathcal{A}}$ is a negligible function of sec for all polynomial time adversaries $\mathcal{A}$. We first discuss this problem. The condition $\nu \geq \gamma \cdot \mathsf{sec}$ implies that the MSSP in the first distribution is nota low density multiple subset sum, and thus (if sec is chosen large enough) the underlying subset sum problem is hard. The best algorithms for the subset-sum problem, on elements of size $V = 2^{\mathsf{sec}}$, have time either $O(2^{\mathsf{sec}/2})$ [28] or $O(\mathsf{sec} \cdot 2^{\mathsf{sec}})$ [**?**], or $O(V \cdot \sqrt{\mathsf{sec}})$ [36]. With our parameters, these all have exponential time (in sec).

There are approximately $\nu \cdot 2^{\mathsf{sec}}$ elements in the range $[0, \ldots, \mathsf{mx}_j]$, but 'only' at most 2^ν of these correspond to valid subset sums. Thus we see that, when $2^\nu < \nu \cdot 2^{\mathsf{sec}}$ there couldbe choices of $T_j \in [0, \ldots, \mathsf{mx}_j]$ which do not correspond to a subset sum. In the important case (for us) of $\nu = \gamma \cdot \mathsf{sec}$ and $\gamma = 1$ we indeed have $2^{\mathsf{sec}} < \mathsf{sec} \cdot 2^{\mathsf{sec}}$ always. However, when $\gamma > 2$ the number of subsets mapping to each value in $T_j \in [0, \ldots, \mathsf{mx}_j]$ is bigger than 2^{sec}. Thus in this case, i.e. when $\gamma > 2$ and $\nu = \gamma \cdot \mathsf{sec}$, we see that the MSSG problem is to distinguish between two distributions which are already statistically close.

Thus the MSSG problem, when $\nu = \gamma \cdot \mathsf{sec}$, is only an issue in the (relatively important) cases of $\gamma = 1$ and $\gamma = 2$. However, in these cases, if we replaced $m + \gamma \cdot \mathsf{sec}$ in our protocol by $m + 2 \cdot \gamma \cdot \mathsf{sec}$, then the number of possible subset sums is exponentially larger than the total number of elements in $[0, \ldots, \mathsf{mx}_j]$. This forms the basis of an observation by Baum that, by increasing $m + \gamma \cdot \mathsf{sec}$ in our protocol to $m + 2 \cdot \gamma \cdot \mathsf{sec}$, one can remove the need for the MSSG problem; at the expense of performing more operations. This obviously reduces the efficiency of the protocol. If one is willing to pay this price then security becomes a purely statistical argument, since the MSSG problem is in this case statistically secure, as the two distributions are indistinguishable, with a statisical distance of $2^{-\mathsf{sec}}$. Such an argument is similar to arguments used in prior works in various situations, see for example [33,42]. We prefer to keep with assuming the MSSG problem as it leads to more efficient protocols, and the only case where this is a computational problem is when $\gamma = 1$ and $\gamma = 2$; which are the most important cases from an application point of view.

4.1 ~~Security of $\Pi_{\mathsf{RandomBits}}$~~

Security of $\Pi_{\mathsf{RandomBits}}$ then follows from the following theorem.

Theorem 4.1. ~~*Assume the problem Multiple Subset Sum Guessing(MSSG Problem) is hard for the parameter $\nu = \gamma \cdot \mathsf{sec}$, then protocol*~~ *Protocol $\Pi_{\mathsf{RandomBits}}$* ~~*securely implements the functionality*~~ *securely implements the functionality $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,\ldots,p_{t-1}}.\mathsf{RandomBits}()$ in the* ~~$\mathcal{F}_{\mathsf{Rand}}^{2^{\mathsf{sec}}}$,~~ $\mathcal{F}_{\mathsf{Rand}}^{W}$, $\mathcal{F}_{\mathsf{MPC}}\mathsf{GenBit}$-*hybrid* ~~*model. However, in the presence of passive adversaries the protocol will abort with probability $3 \cdot (m + \gamma \cdot \mathsf{sec})/p$.*~~ *model. In the presence of passive adversaries the protocol will abort with probability $3 \cdot (m + \gamma \cdot T)/p$, where*

$$\lceil W/p_{\mathsf{min}} \rceil < W \cdot 2^{-\mathsf{sec}/\gamma}$$

and

$$T = \lceil \mathsf{stat} + \log_2 W + \log_2 m \rceil.$$

In the presence of active adversaries, deviation from the protocol is caught with probability at least $1 - 2^{-\mathsf{sec}}$. The statistical distance between a transcript of the real execution and a simulator acting with ideal functionality is bounded by $2^{-\mathsf{stat}}$.

~~We divide the proof into two sections, one showing correctness the other giving the simulation.~~

Proof. We divide the proof into two sections, one showing correctness the other giving the simulation.

Correctness: ~~The~~ Assuming all parties are honest, the computation of k_i may be incorrect with probability $3/p$. So even an honest execution of the protocol will abort with probability $3 \cdot (m + \gamma \cdot T)/p$, which is negligible.

We now examine the case of a dishonest party sending an invalid value of $h_{i,\ell}$ in Step 2d or entering invalid values in Step 2f. The values S_j are guaranteed to be correct, assuming the Open does not abort due to adversarial behaviour. In addition the values S_j are produced with no wrap-around modulo p, due to the ~~bound on m~~lower bound on p of 2^T. Thus the γ values S_j define subset sums over the integers

$$S_j = \sum_{i=1}^{\overset{m+\gamma\cdot\mathsf{sec}}{m}} b_i \cdot r_{i,j} + \sum_{k=1}^{T} 2^{k-1} \cdot b_{j,k}$$

where we are guaranteed that ~~$b_i \in \{0,1\}$~~$b_i, b_{j,k} \in \{0,1\}$.

We then check these sums against the equivalent sums modulo p_v for $v = 0, \ldots, t-1$. ~~Since the random coefficients $r_{i,j}$ are revealed only after the parties input their shares $b_i^{(p)}$ of the bits modulo p_v, the~~ We write these as

$$S_{j,v} = \sum_{i=1}^{m} b_i \cdot r_{i,j} + M_{j,v} \pmod{p_v}$$

where

$$M_j = \sum_{k=1}^{T} 2^{k-1} \cdot b_{j,k}$$

The fact that the values $S_{j,v}$ are equal to $S_j \pmod{p_v}$ implies with overwhelming probability (since $p_{\mathsf{min}}^{\gamma} > 2^{\mathsf{sec}}$) that the shared values entered modulo p_j are equal to the shared values in $\mathcal{F}_{\mathsf{MPC}}^{p}$.

To see this notice that the adversary is able, by entering invalid values in steps 2d or 2f to introduce an additive error $\delta_{i,v}$ into the sharing $\langle b_i \rangle_{p_v}$, or an additive error $\delta_{0,v}$ into the sharing of $\langle M_j \rangle_{p_v}$. Modulo p_v these errors, to pass the test in line 5f, must satisfy the equation

$$\delta_{0,v} + \sum_{i=1}^{m} r_{i,j} \cdot \delta_{i,v} = 0 \pmod{p_v}.$$

Assuming there is an error introduced, then there is a $k \in [1, \ldots, m]$, with $\delta_{k,v} \neq 0 \pmod{p_v}$, and so we must have

$$r_{k,j} = -\left(\delta_{0,v} + \sum_{i \neq k} r_{i,j} \cdot \delta_{i,v}\right) / \delta_{k,v} \pmod{p_v}$$

Since the random coefficients $r_{i,j}$ are revealed only after the parties input their shares of the bits modulo p_v, this can only happen with probability less than $\lceil W/p_v \rceil / W \leq \lceil W/p_{\mathsf{min}} \rceil / W$.

As the sums are independent for each of the j iterations, the chance of not-detecting one of $\delta_{i,v}$ values being non-zero is bounded by $(\lceil W/p_{\mathsf{min}} \rceil / W)^{\gamma}$. By choice of W this is less than $2^{-\mathsf{sec}}$.

Since $\mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}$ generates values which are guaranteed to be bits, this ~~implies~~ also implies that the values modulo p_j are also bits, and equal to each other. ~~However, the computation of k_i may be incorrect with probability $3/p$. So even an honest execution of the protocol will abort with probability $3 \cdot (m + \gamma \cdot \mathsf{sec})/p$.~~

Simulation: To complete the proof we must show that the protocol can be simulated. There are two sets of values which need simulating. The $h_{i,j}$ values which are sent to party P_1 by Player P_j and the subset sum values S_j and $S_{j,v}$ which are opened. We first deal with the ~~$h_{i,j}$~~ $h_{i,\ell}$ values.

The simulator already knows the shares ~~$h_{i,j}$~~ $h_{i,\ell}$ which the adversary should be sending to them (from $\mathcal{F}_{\mathsf{MPC}}^{p}.\mathsf{GenBit}$), and so knows whether to abort the functionality at this point. ~~To generate the $h_{i,j}$ values for~~ Thus we need to determine a method to simulate the values $h_{i,\ell}$ sent from the honest parties~~the simulator simply picks random values for the honest partiesshares $b_{i,j}$ so that they sum to zero modulo p. The honest $h_{i,j}$ are then derived from~~

these values . These top bits of the honest shares will be a valid simulation even if the shared bit is actually one

We let $\mathcal{H}$ denote the set of honest parties, and $\mathcal{A}$ denote the set of adversarial parties. The simulator knows the value of

$$b_{i,\mathcal{A}} = \sum_{\ell \in \mathcal{A}} b_{i,\ell}.$$

As we are sharing a bit we know that

$$b_{i,\mathcal{H}} = \sum_{\ell \in \mathcal{H}} b_{i,\ell} \in \{p - b_{i,\mathcal{A}}, p + 1 - b_{i,\mathcal{A}}\}.$$

When

$$b_{i,\mathcal{A}} \in S = \{\lambda \cdot (\Delta - 1) + 1 : 0 \leq \lambda < n\}$$

then the hidden bit b_i is revealed to the adversary by the values $h_{i,\ell}$. But this happens with probability n/p. In total the probability of this happening accross all bits is bounded by $n \cdot (m + \gamma \cdot T)/p$, which is negligible.

Assuming $b_{i,\mathcal{A}} \notin S$ then there is a set of shares for the honest parties consistent with either $b_i = 0$ or $b_i = 1$ such that $0 \leq h_{i,\ell} < n$ and

$$\sum_{\ell \in \mathcal{H}} h_{i,\ell} = \lfloor (p - b_{i,\mathcal{A}})/\Delta \rfloor.$$

Thus the simulator just picks a random assignment to the $h_{i,\ell}$ values for $\ell \in \mathcal{H}$ which satisfies requirement.

Similarly the simulator knows when to abort if the adversary enters in valid values in Step 2f as well. To simulate the values S_j, and hence $S_{j,v}$, we simply define the trivial simulator. The values $r_{i,j}$ are sampled using the ideal functionality ~~$\mathcal{F}_{\mathsf{Rand}}^{2\mathsf{sec}}$, then the values $\mathsf{mx}_j = \sum_{i=1}^{m+\gamma\cdot\mathsf{sec}} r_{i,j}$ are computed. The simulator then~~ $\mathcal{F}_{\mathsf{Rand}}^{W}$. Then the simulator picks a random value ~~$S_j \in [0, \ldots, \mathsf{mx}_j]$~~ $S_j \in [0, \ldots, 2^T)$. and returns S_j and $S_{j,v} = S_j \pmod{p_v}$, for every v, to the adversary.

~~We now show that an environment $\mathcal{E}$, that can distinguish between the real execution and this ideal world simulated execution, can be used to solve an arbitrary instance of the MSSG problem with $\nu = \gamma \cdot \mathsf{sec}$.~~

~~We let $\mathcal{A}$ denote our adversary against the MSSG problem. This takes as input values $w_{1,j}, \ldots, w_{\nu,j}$, for $j = 1, \ldots, \gamma$, and target sums T_j for $j = 1, \ldots, \gamma$. The adversary $\mathcal{A}$ runs the environment $\mathcal{E}$ twice, in both instances setting $r_{i+m,j} = w_{i,j}$ for $i = 1, \ldots, \nu$ for all j and choosing the other $r_{i,j}$ at random (the same in both executions). The values S_j passed to the environment in the first execution are selected from $[0, \ldots, \mathsf{mx}_j]$, i.e. in the first execution the environment is interacting with an adversary which is itself interacting with the simulation~~

~~in the ideal world. At the end of the first execution the environment will output the bits b_i for $i \in [1, \ldots, m]$.~~

~~In the second execution the environment, adversary and honest players etc are run with the same random tapes. Note, that in the protocol these bits are selected before~~By the Smudging Lemma the simulated values are within statistical distance $2^{-\mathsf{stat}}$ from the ~~S_j values are computed, thus the bits b_i chosen will be the same. In the second execution our $\mathcal{A}$ against the MSSG problem forms the subset sum values returned to the environment as $S_j = T_j + \sum_{i=1}^{m} b_i \cdot r_{i,j}$.~~

~~The adversary on input of a MSSG distribution of type one, is presenting to the environment in the second execution a system which is identical to the~~ values in the real execution of the protocol. ~~When the input to $\mathcal{A}$ is from MSSG distribution of type two, then the environment $\mathcal{E}$ which is presented in the second execution which is identical to the simulated execution in the ideal world. Thus any $\mathcal{E}$ which can distinguish between the real and simulated world will be able to be used by $\mathcal{A}$ to solve the MSSG problem for $\nu = \gamma \cdot \mathsf{sec}$~~

By the above correctness argument the simulation will be correct except with probability bounded by $\max\{2^{-\mathsf{sec}}, \max\{3, n\} \cdot (m + \gamma \cdot T)/p\}$, which is negligible. □

5 Sampling Distributions In MPC

In this section we explain how we perform the various sampling operations we need in the key generation algorithm in Section 6. We assume that two actively-secure (with-abort) MPC functionalities, as in Figure 2, have already been initiated, one for the prime p_0 and one for the prime p_1. We call these two functionalities $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. As explained earlier these can be instantiated with either BDOZ or MASCOT-SPDZ. Using these functionalities we can instantiate the protocol $\Pi_{\mathsf{RandomBits}}$, from Section 4, to produce doubly-authenticated bits in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Thus all the protocols in this section can be seen to work in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ model, which we have already shown can be instantiated in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}$-hybrid model.

The protocol Π_{Hamming} will only be used to generate secret keys which are equivalent to the traditional BGV style keys used in SCALE-MAMBA or HELib. For more "modern" implementations one would use Π_{Binomial} to generate the secret keys as these have less security concerns related to them. In both cases we need Π_{Binomial} and Π_{Uniform} to generate the error vectors needed to compute the public key. Additionally we see our Π_{Hamming} protocols family could potentially improve the cost of the HE-MPC conversions of Zhu et al. [43] if they would relax the assumptions of generating a secret key with fixed hamming weight h to one that has expected hamming weight h.

5.1 Protocol Π_{Hamming}

To sample our secret key we will need to sample a shared vector of length N, with elements in $\{-1, 0, 1\}$, with approximate Hamming weight h, i.e. we sample from $B(h/N)^N$. The values of N and h are selected such they are both powers of two, so we set $N = 2^\nu$ and $h = 2^\ell$. The

reader should think of values of $N = 32768$ and $h = 64$, which were used in SCALE-MAMBA's default configuration when we started this work.

To sample such vectors we use protocol $\Pi_{\mathsf{Hamming}}(h, N)$ given in Figure 9. The basic idea (for one such bit) is to sample $v - \ell$ random bits $b_1, \ldots, b_{v-\ell}$. Then the product $p = b_1 \cdots b_{v-\ell}$ will be one with probability $1/2^{v-\ell}$, and zero otherwise. Then a sign bit s is used to flip the one to minus one, with probability $1/2$. Performing this operation N times produces the required distribution.

Protocol $\Pi_{\mathsf{Hamming}}(h, N)$

1. Set $N = 2^\nu$ and $h = 2^\ell$.
2. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate $H = N \cdot (\nu - \ell)$ shared random bits $\langle b_i^j \rangle_{p_0}$ and $\langle b_i^j \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, \nu - \ell\}$.
3. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_{p_0} \leftarrow \langle b_i^1 \rangle_{p_0} \cdot \ldots \cdot \langle b_i^{\nu-\ell} \rangle_{p_0}$.
   (b) $\langle b_i \rangle_{p_1} \leftarrow \langle b_i^1 \rangle_{p_1} \cdot \ldots \cdot \langle b_i^{\nu-\ell} \rangle_{p_1}$.
4. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate N shared random bits $\langle s_i \rangle_{p_0}$ and $\langle s_i \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$.
5. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_{p_0} \leftarrow \langle b_i \rangle_{p_0} \cdot (2 \cdot \langle s_i \rangle_{p_0} - 1)$.
   (b) $\langle b_i \rangle_{p_1} \leftarrow \langle b_i \rangle_{p_1} \cdot (2 \cdot \langle s_i \rangle_{p_1} - 1)$.
6. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle b_i \rangle_{p_0}$ and $\langle b_i \rangle_{p_1}$.

Figure 9. Method to produce vectors of (expected) Hamming weight h with elements in $\{-1, 0, 1\}$.

If we assume that the two calls to $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ are implemented using our previous protocol $\Pi_{\mathsf{RandomBits}}$ using only one batch operation, then these operation requires 6 rounds of communication, ~~$(H + N + \mathsf{sec})$~~ $(H + N + \gamma \cdot T)$ multiplications in $\mathcal{F}_{\mathsf{MPC}}^{p}$[6]. The products to produce the initial bits b_i require $\lceil \log_2(\nu - \ell) \rceil$ rounds of communication and $(\nu - \ell - 1) \cdot N$ multiplications in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Whereas the products to produce the final signed bits require one round of communication and N multiplications in both $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. Hence in total we require

$$6 + \lceil \log_2(\nu - \ell) \rceil + 1 = 7 + \lceil \log_2(\nu - \ell) \rceil$$

rounds of communication and

$$N_{\mathsf{Hamming},0} = N + H + \mathsf{sec} + (\nu - \ell + 1) \cdot N,$$
$$N_{\mathsf{Hamming},1} = (\nu - \ell + 1) \cdot N$$

total multiplications in $\Pi_{\mathsf{MPC}}^{p_0}$ and $\Pi_{\mathsf{MPC}}^{p_1}$ respectively.

5.2 Another Method to Implement Π_{Hamming}

In this section we describe a different protocol to generate a vector of size N with expected hamming weight h. The main advantage of this protocol is that it avoids the expensive multiplications modulo large prime numbers (modulo p_0 and modulo p_1) completely. This comes

[6] ~~Note that $\gamma = 1$ since p_0, p_1 are both big.~~

at the expense of requiring an extra preprocessing protocol which generates random shared bits modulo two from [33,42], as well as the need to implement a demultiplexer via a lookup table [32]. Thus this method requires us to base the SPDZ key generation methodology on two distinct MPC methodologies; TinyOT-style $\mathbb{F}_2$-based MPC and MASCOT style $\mathbb{F}_p$-base MPC. So as to keep the minimality in terms of base MPC protocols for our key generation process we include this section mearly as an additional method for the reader.

Note, to generate random bits shared modulo two is much cheaper than generating random shared bits modulo a large prime p. These binary sharings are usually done using a pairwise protocol TinyOT style found in Wang et al. or Keller et al. [33,42]. We call the functionality that generates binary sharings $\mathcal{F}_{\mathsf{bit}}[\mathbb{F}_2]$

The key idea of this protocol is to follow the same blueprint as $\Pi_{\mathsf{Hamming}}(h, N)$ defined in Section 5.1 but start with random binary sharings of bits $\langle b \rangle_2$ rather then arithmetic sharings $\langle b \rangle_{p_0}$ and $\langle b \rangle_{p_1}$. This will cause the bit multiplications to happen modulo two which is orders of magnitude faster than their counterpart modulo a large prime p.

However, the main challenge with this method is to be able to output the value in $\{0, \pm 1\}$ without the large field multiplications in Step 5 from Figure 5.1. This can be done by emulating a small demultiplexer on the values $b_i b_i^*$ ($\{00, 01, 10, 11\}$) and map them to $\{0, 0, 1, -1\}$. We define this public lookup table containing these values in binary form as $\mathsf{T} = \{00, 00, 10, 11\}$. One can think of the right hand bit in each entry of T as the sign of the secret key bit. In the last step we convert the table entries to integers modulo a large prime using maBits, i.e. we compute $2^0 \cdot T(i, 0) - 2^1 \cdot T(i, 1)$ where $T(i, j)$ represents the j'th bit of the i'th entry.

Protocol $\Pi'_{\mathsf{Hamming}}(h, N)$

1. Set $N = 2^\nu$ and $h = 2^\ell$.
2. Call $\mathcal{F}_{\mathsf{bit}}[\mathbb{F}_2]$ to generate $H = N \cdot (\nu - \ell)$ shared random bits $\langle b_i^j \rangle_2$ for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, \nu - \ell\}$.
3. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_2 \leftarrow \langle b_i^1 \rangle_2 \cdot \ldots \cdot \langle b_i^{\nu-\ell} \rangle_2$.
   (b) $\langle b_i^* \rangle_2 \leftarrow \mathcal{F}_{\mathsf{bit}}[\mathbb{F}_2]$
4. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate $2 \cdot N$ shared random bits $\langle s_i^j \rangle_2$, $\langle s_i^j \rangle_{p_0}$ and $\langle s_i^j \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$ and $j \in \{0, 1\}$.
5. For $i \in \{1, \ldots, N\}$
   (a) $(\langle d_0 \rangle_2, \cdots \langle d_3 \rangle_2) \leftarrow \mathsf{Demux}(\langle b_i \rangle_2, \langle b_i^* \rangle_2)$
   (b) $(\langle v_0 \rangle_2, \langle v_1 \rangle_2) \leftarrow \langle d_0 \rangle_2 \cdot \mathsf{T}(0) \oplus \langle d_1 \rangle_2 \cdot \mathsf{T}(1) \oplus \langle d_2 \rangle_2 \cdot \mathsf{T}(2) \oplus \langle d_3 \rangle_2 \cdot \mathsf{T}(3)$
   (c) $t_i^0 \leftarrow \mathsf{Open}(\langle s_i^0 \rangle_2 \oplus \langle v_0 \rangle_2)$ and $t_i^1 \leftarrow \mathsf{Open}(\langle s_i^1 \rangle_2 \oplus \langle v_1 \rangle_2)$
6. For $i \in \{1, \ldots, N\}$ [Now convert the lookup table entries to their arithmetic counterpart]
   (a) $\langle b_i \rangle_{p_0} \leftarrow 2^0 \cdot \mathsf{XOR}(t_i^0, \langle s_i^0 \rangle_{p_0}) - 2^1 \cdot \mathsf{XOR}(t_i^1, \langle s_i^1 \rangle_{p_0})$ (bit-compose of public-secret xors in $\mathbb{F}_{p_0}$)
   (b) $\langle b_i \rangle_{p_1} \leftarrow 2^0 \cdot \mathsf{XOR}(t_i^0, \langle s_i^0 \rangle_{p_1}) - 2^1 \cdot \mathsf{XOR}(t_i^1, \langle s_i^1 \rangle_{p_1})$ (bit-compose of public-secret xors in $\mathbb{F}_{p_1}$)
7. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle b_i \rangle_{p_0}$ and $\langle b_i \rangle_{p_1}$.

Figure 10. Second method to produce vectors of (expected) Hamming weight h with elements in $\{-1, 0, 1\}$.

5.3 Protocol Π_{Binomial}

The protocol for sampling shared values from the distribution $\mathsf{dN}(\sigma^2, N)$ is relatively straightforward, and is given in Figure 11. Apart from the generation of the random bits this protocol consists entirely of linear operations. Thus the round complexity is six and it requies ~~$N_{\mathsf{Binomial}} = 2 \cdot k \cdot N + \mathsf{sec}$~~ $N_{\mathsf{Binomial}} = 2 \cdot k \cdot N + \gamma \cdot T$ multiplications in $\mathcal{F}_{\mathsf{MPC}}^{p}$, assuming $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ is implemented in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}$-hybrid model using our previous protocol $\Pi_{\mathsf{RandomBits}}$.

Protocol $\Pi_{\mathsf{Binomial}}(\sigma^2, N)$

1. Define k by $\sigma = \sqrt{k/2}$.
2. Using $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$ generate $2 \cdot k \cdot N$ shared random bits $\langle b_i^j \rangle_{p_0}$ and $\langle b_i^j \rangle_{p_1}$, for $i \in \{1, \ldots, N\}$ and $j \in \{0, \ldots, 2 \cdot k - 1\}$.
3. For $i \in \{1, \ldots, N\}$
   (a) $\langle b_i \rangle_{p_0} \leftarrow \sum_{j=0}^{k-1} \langle b_i^{2 \cdot j} \rangle_{p_0} - \langle b_i^{2 \cdot j+1} \rangle_{p_0}$.
   (b) $\langle b_i \rangle_{p_1} \leftarrow \sum_{j=0}^{k-1} \langle b_i^{2 \cdot j} \rangle_{p_1} - \langle b_i^{2 \cdot j+1} \rangle_{p_1}$.
4. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle b_i \rangle_{p_0}$ and $\langle b_i \rangle_{p_1}$.

Figure 11. Method to produce elements from $\mathsf{dN}(\sigma^2, N)$.

5.4 Protocol Π_{Uniform}

Our final protocol is a rather trivial one, it allows the parties to sample a uniform element from $\mathbb{Z}_q$ (recall $q = p_0 \cdot p_1$ with $\gcd(p_0, p_1) = 1$) in a secret shared form, we give it in Figure 12.

Protocol $\Pi_{\mathsf{Uniform}}(N)$

1. For $i \in \{1, \ldots, M\}$
   (a) For $j \in \{1, \ldots, n\}$
      i. Player P_i selects $u_i^j \in \mathbb{F}_{p_0}$, and $v_i^j \in \mathbb{F}_{p_1}$,
      ii. Execute $\langle u_i^j \rangle_{p_0} \leftarrow \mathsf{Input}(P_i)$.
      iii. Execute $\langle v_i^j \rangle_{p_1} \leftarrow \mathsf{Input}(P_i)$.
   (b) $\langle u_i \rangle_{p_0} \leftarrow \langle u_i^1 \rangle_{p_0} + \ldots + \langle u_i^n \rangle_{p_0}$.
   (c) $\langle v_i \rangle_{p_1} \leftarrow \langle v_i^1 \rangle_{p_1} + \ldots + \langle v_i^n \rangle_{p_1}$.
2. Return, for $i \in \{1, \ldots, N\}$, the shares $\langle u_i \rangle_{p_0}$ and $\langle v_i \rangle_{p_1}$. We think of these as sharings of $(\langle a_i \rangle_{p_0}, \langle a_i \rangle_{p_1})$, of some value $a_i \in \mathbb{Z}_q$ with $a_i = u_i \pmod{p_0}$ and $a_i = v_i \pmod{p_1}$.

Figure 12. Protocol to sample N uniformly random elements from $\mathbb{Z}_q$.

6 SPDZ KeyGeneration

Given the previous algorithms to generate various distributions the computation of the actual key generation algorithm becomes relatively straight forward. We first sample the various distributions modulo p_0 and p_1, then we produce the square of the secret key (needed for

the key switching matrices), and then we output the public key (modulo p_0 and p_1) and then recombine it using the CRT, finally we do the same to each players component of the secret key. The overall protocol is given in Figure 13; note that in line 3 one can select as to whether to choose the secret key from a restricted Hamming weight or from the centred binomial distribution. To make the protocol easier to follow we use the notation $\langle \mathbf{a} \rangle_{p_0}$ etc to denote a vector of N shares modulo p_0, i.e. $\langle \mathbf{a} \rangle_{p_0} = (\langle a_0 \rangle_{p_0}, \ldots, \langle a_{N-1} \rangle_{p_0})$.

We let $\langle \mathbf{a} \rangle_{p_0} \odot \langle \mathbf{b} \rangle_{p_0}$ denote the multiplication of two such vectors when considered as elements in the ring R_{p_0}. This requires one round of communication and N^2 secure multiplications (or $N \cdot (N-1)/2$ secure multiplications if $\mathbf{a} = \mathbf{b}$). If one vector is in the clear then we write $\mathbf{a} \odot \langle \mathbf{b} \rangle_{p_0}$, which is a linear operation and hence for "free". A more efficient method to multiply is to use the FFT algorithm, which recall is a linear operation and thus 'free' when executed in the MPC engine. To multiply using FFT we utilize

$$\langle \mathbf{a} \rangle_{p_0} \odot \langle \mathbf{b} \rangle_{p_0} = \mathsf{FFT}^{-1}(\mathsf{FFT}(\langle \mathbf{a} \rangle_{p_0}) \cdot \mathsf{FFT}(\langle \mathbf{b} \rangle_{p_0})$$

which requires only N secure multiplications and one round of communication.

We now examine each of the operations in this algorithm in turn. The lines 1-6 can all be executed in parallel and so require

$$\max\Big(6 + \lceil \log_2(\nu - \ell) \rceil + 1, 6, 1\Big) = 6 + \lceil \log_2(\nu - \ell) \rceil$$

rounds of communication [6] in the case where we select a secrey key with given expected Hamming weight and $\max\Big(6, 6, 1\Big) = 6$ rounds of communication in the case of the secret key generated from a centred binomial distribution. The number of secure multiplications is given by, in the two cases,

$$N_{\mathsf{Hamming},0} + N_{\mathsf{Hamming},1} + 2 \cdot N_{\mathsf{Binomial}} = N + H + \mathsf{sec}\gamma \cdot T + 2 \cdot (\nu - \ell + 2 \cdot k - 1) \cdot N,$$

$$3 \cdot N_{\mathsf{Binomial}} = 6 \cdot k \cdot N.$$

Lines 7 and 9 are linear operations and thus can be executed as purely local operations. Line 8 requires one round of communication and N multiplications in $\mathcal{F}_{\mathsf{MPC}}^{p_0}$. Note, in lines 7-9 we only have to compute $\mathfrak{st}^2$ modulo p_0 as it is multiplied by p_1 when added into $\mathbf{b}_{\mathfrak{st},\mathfrak{st}^2}$. Lines 10-13 can also be performed in parallel and hence require only one round of communication. The lines 16-21 are all local operations, and hence are for "free". Lines 25-24 are, again, able to be done in parallel and so require only one round of communication. The remaining lines are purely local organization of data into the correct format for outputing. Thus the total number of rounds of communication (assuming all shared random bits are produced in a single batch) is $12 + \lceil \log_2(\nu - \ell) \rceil$ or 12, depending on which variant one is using for the secret key.

~~If we take typical values of $n = 2$, $N = 32768$, $h = 64$, and $\mathsf{sec} = 128$ then these work out to be 3277056 mults in $\Pi_{\mathsf{MPC}}^{p_0}$ and 3244288 in $\Pi_{\mathsf{MPC}}^{p_1}$ and 16 rounds of communication.~~

[6] Of course in practice we generate the secure bits in batches and hence this is just the minimal number of rounds required.

Protocol Π_{KeyGen}

- [*Generate the various values required*]
1. $(\langle \mathbf{a} \rangle_{p_0}, \langle \mathbf{a} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Uniform}}(N)$.
2. $(\langle \mathbf{a}' \rangle_{p_0}, \langle \mathbf{a}' \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Uniform}}(N)$.
3. $(\langle \mathbf{s} \rangle_{p_0}, \langle \mathbf{s} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Hamming}}(h, N)$ or $(\langle \mathbf{s} \rangle_{p_0}, \langle \mathbf{s} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Binomial}}(N)$.
4. $(\langle \mathbf{e} \rangle_{p_0}, \langle \mathbf{e} \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Binomial}}(N)$.
5. $(\langle \mathbf{e}' \rangle_{p_0}, \langle \mathbf{e}' \rangle_{p_1}) \leftarrow \Pi_{\mathsf{Binomial}}(N)$.
6. For $i \in \{2, \ldots, n\}$
   (a) $\langle \mathfrak{sk}_{0,i} \rangle_{p_0} \leftarrow \mathsf{Input}(P_i)$ for a random vector $\mathfrak{sk}_{0,i} \in R_{p_0}$ selected by player P_i.
   (b) $\langle \mathfrak{sk}_{1,i} \rangle_{p_1} \leftarrow \mathsf{Input}(P_i)$ for a random vector $\mathfrak{sk}_{1,i} \in R_{p_1}$ selected by player P_i.
- [*Compute the square of the secret key*]
7. $\langle \mathbf{f} \rangle_{p_0} \leftarrow \mathsf{FFT}(\langle \mathbf{s} \rangle_{p_0})$.
8. $\langle \mathbf{f}' \rangle_{p_0} \leftarrow \langle \mathbf{f} \rangle_{p_0} \cdot \langle \mathbf{f} \rangle_{p_0}$ [This is the component wise product].
9. $\langle \mathbf{s}' \rangle_{p_0} \leftarrow \mathsf{FFT}^{-1}(\langle \mathbf{f}' \rangle_{p_0})$.
- [*Open the values* $\mathbf{a}$ *and* $\mathbf{a}'$]
10. $\mathbf{a}_0 \leftarrow \mathsf{Open}(\langle \mathbf{a} \rangle_{p_0})$.
11. $\mathbf{a}_1 \leftarrow \mathsf{Open}(\langle \mathbf{a} \rangle_{p_1})$.
12. $\mathbf{a}'_0 \leftarrow \mathsf{Open}(\langle \mathbf{a}' \rangle_{p_0})$.
13. $\mathbf{a}'_1 \leftarrow \mathsf{Open}(\langle \mathbf{a}' \rangle_{p_1})$.
14. $\mathbf{a} \leftarrow \mathsf{CRT}([\mathbf{a}_0, \mathbf{a}_1], [p_0, p_1])$.
15. $\mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{CRT}([\mathbf{a}'_0, \mathbf{a}'_1], [p_0, p_1])$.
- [*Compute* $\mathbf{b}$ *and* $\mathbf{b}'$]
16. $\langle \mathbf{b} \rangle_{p_0} \leftarrow \mathbf{a}_0 \odot \langle \mathbf{s} \rangle_{p_0} + p \cdot \langle \mathbf{e} \rangle_{p_0}$.
17. $\langle \mathbf{b} \rangle_{p_1} \leftarrow \mathbf{a}_1 \odot \langle \mathbf{s} \rangle_{p_1} + p \cdot \langle \mathbf{e} \rangle_{p_1}$.
18. $\langle \mathbf{b}' \rangle_{p_0} \leftarrow \mathbf{a}'_0 \odot \langle \mathbf{s} \rangle_{p_0} + p \cdot \langle \mathbf{e}' \rangle_{p_0} - p_1 \cdot \langle \mathbf{s}' \rangle_{p_0}$.
19. $\langle \mathbf{b}' \rangle_{p_1} \leftarrow \mathbf{a}'_1 \odot \langle \mathbf{s} \rangle_{p_1} + p \cdot \langle \mathbf{e}' \rangle_{p_1}$.
- [*Fix the final key for the sharing of the secret key*]
20. $\langle \mathfrak{sk}_{0,1} \rangle_{p_0} \leftarrow \langle \mathbf{s} \rangle_{p_0} - \sum_{i=2}^{n} \langle \mathfrak{sk}_{0,i} \rangle_{p_0}$.
21. $\langle \mathfrak{sk}_{1,1} \rangle_{p_1} \leftarrow \langle \mathbf{s} \rangle_{p_1} - \sum_{i=2}^{n} \langle \mathfrak{sk}_{1,i} \rangle_{p_1}$.
22. For all $i \in \{1, \ldots, n\}$ assign $\mathfrak{sk}_i \leftarrow \mathsf{CRT}([\mathfrak{sk}_{0,i}, \mathfrak{sk}_{1,i}], [p_0, p_1])$.
23. $P_1 \leftarrow \mathsf{Output}(\langle \mathfrak{sk}_{0,1} \rangle_{p_0})$.
24. $P_1 \leftarrow \mathsf{Output}(\langle \mathfrak{sk}_{1,1} \rangle_{p_1})$.
- [*Open the values* $\mathbf{b}$ *and* $\mathbf{b}'$]
25. $\mathbf{b}_0 \leftarrow \mathsf{Open}(\langle \mathbf{b} \rangle_{p_0})$.
26. $\mathbf{b}_1 \leftarrow \mathsf{Open}(\langle \mathbf{b} \rangle_{p_1})$.
27. $\mathbf{b}'_0 \leftarrow \mathsf{Open}(\langle \mathbf{b}' \rangle_{p_0})$.
28. $\mathbf{b}'_1 \leftarrow \mathsf{Open}(\langle \mathbf{b}' \rangle_{p_1})$.
29. $\mathbf{b} \leftarrow \mathsf{CRT}([\mathbf{b}_0, \mathbf{b}_1], [p_0, p_1])$.
30. $\mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2} \leftarrow \mathsf{CRT}([\mathbf{b}'_0, \mathbf{b}'_1], [p_0, p_1])$.
31. Output $\mathbf{a}, \mathbf{b}, \mathbf{a}_{\mathfrak{sk},\mathfrak{sk}^2}$ and $\mathbf{b}_{\mathfrak{sk},\mathfrak{sk}^2}$ to all players, and $\mathfrak{sk}_i$ to player P_i.

Figure 13. The distributed BGV key generation protocol.

Theorem 6.1. *The protocol Π_{KeyGen} UC-securely realises the functionality $\mathcal{F}_{\mathsf{KeyGen}}$ against a static, active adversary corrupting at most $n-1$ parties in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$-hybrid model.*

Proof. We define the simulator $\mathcal{S}$ as follows. The simulator emulates the behaviour of honest parties exactly, but additionally does the following:

- At the start of the execution, the simulator initialises a local copy of $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ and sends the message *start* to $\mathcal{F}_{\mathsf{KeyGen}}$ and awaits the public key $\mathfrak{pk} = (\bar{\mathbf{a}}, \bar{\mathbf{b}}, \bar{\mathbf{a}}_{\mathfrak{sk},\mathfrak{sk}^2}, \bar{\mathbf{b}}_{\mathfrak{sk},\mathfrak{sk}^2})$ in response.
- When the adversary and simulator execute Π_{Uniform}, the simulator replaces the values $\mathbf{a} \mod p_0$, $\mathbf{a} \mod p_1$, $\mathbf{a}' \mod p_0$ and $\mathbf{a}' \mod p_1$ stored in the instances of $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$, respectively, with $\bar{\mathbf{a}} \mod p_0$, $\bar{\mathbf{a}} \mod p_1$, $\bar{\mathbf{a}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_0$ and $\bar{\mathbf{a}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_1$.
- In Step 6, for each $j \in [n]$, if j is corrupt and $j > 2$ then the simulator awaits the input $\mathfrak{sk}_{j,0}$ and $\mathfrak{sk}_{j,1}$ for each corrupt party P_j and constructs $\mathfrak{sk}_j \leftarrow \mathsf{CRT}(\mathfrak{sk}_{j,0}, \mathfrak{sk}_{j,1})$, and sends these to $\mathcal{F}_{\mathsf{KeyGen}}$.
- Just before opening $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_0'$ and $\mathbf{b}_1'$ the simulator replaces these values stored in the instances of $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ with $\bar{\mathbf{b}} \mod p_0$, $\bar{\mathbf{b}} \mod p_1$, $\bar{\mathbf{b}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_0$ and $\bar{\mathbf{b}}_{\mathfrak{sk},\mathfrak{sk}^2} \mod p_1$.

Since the only inputs to the protocol are randomly sampled by parties, the simulator can perfectly emulate the behaviour of honest parties throughout, as the environment does not observe the random tape of honest parties or the simulator. Indeed, since $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ are used as black boxes and the only communication between parties occurs via these functionalities, which are emulated locally honestly by the simulator, the replacements made in the simulation outlined are executed without being observed by the environment (at this point). Moreover, the inputs of corrupt parties can be extracted trivially and passed on to $\mathcal{F}_{\mathsf{KeyGen}}$ so that the final outputs have the correct distribution.

It only remains to show that the environment cannot observe a difference between the transcripts in a real execution and an execution in which the replacements described above are made. While the final outputs of the real and ideal worlds is the same, the distribution of the transcript differs since communication generated in the execution of the sampling subprotocols depends on the secret $\bar{\mathbf{s}}$ which is (implicitly) generated by the functionality $\mathcal{F}_{\mathsf{KeyGen}}$ when executing $\mathsf{KeyGen}()$ and cannot be computed by the simulator from the public key without breaking the LWE assumption for the security of the key. We must show that the amount by which the distributions differ is negligible.

The only time information stored in $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ or $\mathcal{F}_{\mathsf{MPC}}^{p_1}$ is either revealed to the parties or is generated by parties is in the bits shared in $\Pi_{\mathsf{RandomBits}}$ and in Π_{Uniform}. In the protocols Π_{Binomial} and Π_{Hamming} the parties obtain bits from $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$, and in the remainder of the protocol Π_{KeyGen}, the correctness of the computations is guaranteed by the security of the black boxes $\mathcal{F}_{\mathsf{MPC}}^{p_0}$ and $\mathcal{F}_{\mathsf{MPC}}^{p_1}$. In Π_{Uniform}, every party contributes to every secret, and since there is at least one honest party (that samples uniformly), the output is always uniform. Thus no distinguishing environment can exist, by the choice of parameters for the subset sum, and therefore Π_{KeyGen} UC-securely realises $\mathcal{F}_{\mathsf{KeyGen}}$ in the $\mathcal{F}_{\mathsf{MPC}}^{p,p_0,p_1}.\mathsf{RandomBits}()$-hybrid model. □

7 Implementation

In our implementation we use MASCOT-SPDZ [34] as the base protocol used for our one-time setup phase for SPDZ. We could have selected here BDOZ [8] using LowGear [35] for the pre-processing. Both give $O(n^2)$ protocols with LowGear about a factor of four times faster for small values of n. Since our solution is built on top of the `SCALE-MAMBA` [2] framework, we re-used a lot of their already existing codebase and hence selected MASCOT-SPDZ as the underlying protocol. Our implementation of this key generation protocol is now included in the `SCALE-MAMBA` code base. As explained in the introduction our key generation protocol, is inherently $O(n^2)$ in nature. This seems to be unavoidable as the only practical $O(n)$ MPC protocol known is SPDZ, which is exactly the MPC protocol we are trying to instantiate with our key generation protocol.

Selection of FHE parameters We recall that the two-leveled BGV key generation procedure requires us to choose two prime moduli p_0 and p_1 and a polynomial degree N to define the ciphertext space $R_q \cong R_{p_0} \times R_{p_1}$ and a prime p which defines the size of our plaintext space. In addition we require that the relations presented in Section 3.2 hold.

The size of the plaintext space p defines the modulus of the underlying secret sharing scheme in the SPDZ protocol. Different values of p will be needed for different application scenarios. In this paper we focus on $p \approx 2^{64}$ and $p \approx 2^{128}$. The precise sizes of the other parameters are derived from a noise analysis of how the resulting encryption scheme is used, which takes into account the circuit being computed, the zero-knowledge proofs required, and the distributed decryption procedure, and the computational difficulty of the Ring-LWE problem. This analysis is quite involved and we referred to the `SCALE-MAMBA` documentation [2] to obtain the required parameters.

This gives us (for example), to guarantee a computational security of 128 bits with a polynomial degree $N = 32768$, that our ciphertext modulus q has to satisfy $q < 2^{883}$. For such parameters, the trusted setup of `SCALE-MAMBA`, as we have discussed earlier, produces a secret key with Hamming Weight exactly 64, and uses noise vectors distributed according to a centred binomial distribution with standard deviation of $3.16 = \sqrt{10}$. As a first set of experiments we use exactly the same methodology to select the secret key, but we pick a secret key with *expected* Hamming Weight 64. This does not change the noise analysis (which is done using an expected noise methodology for the secret key in any case), and thus we end up with the same system parameters as used in `SCALE-MAMBA`. In particular for a plaintext modulus of 128 bits this leads us to use a p_0 modulus of 345 bits and a p_1 modulus of 225 bits. So in order to run the key generation protocol presented above, we will need to run two instances of the MASCOT protocol; one for the 345 bits prime p_0 and another for the 225 bits prime p_1.

We also tried different sets of parameters in our experiments, which provide BGV keys for a SPDZ ~~modulus~~ moduli of 64 and 128 bits, always taking the same parameters as the Setup phase for `SCALE-MAMBA`. In Table 1 we report the prime sizes, in bits, required for each set of parameters for two and three parties.

Number of Parties, n	2				3			
FHE Plaintext Size, $\log_2 p$	64		128		64		128	
Polynomial Degree	16384		32768		16384		32768	
$p_0 \vert p_1$ bit length	216	164	345	225	217	163	346	224

Table 1. Parameter Sizes.

With these size of primes, we can now set the value m for our batch size of shared bits which has to respect the bound stated in Section 4. In practice we want to have m a divisor of the total number of shared bits that we will need, and also small enough to avoid RAM or network overflow. Empirically we found that taking $50000 \leq m \leq 100000$ (depending on the setting) gives us good results. Although for smaller values of p_0 (if we wish to use p_0 as the auxillary prime p) we need to restrict m so that $m < p_0/(W \cdot 2^{\mathsf{stat}}) = p_0/2^{208}$.

Extended Random Oblivious Transfer: When starting this work, the `SCALE-MAMBA` framework did not have an implementation of the offline phase of MASCOT, and an implementation of the extended Random Oblivious Transfer for a prime field $\mathbb{F}_p$, thus we needed to implement these[7]. The triple generation method of MASCOT [34][Protocol 4] makes use of an extended Correlated Oblivious Transfer (COT) protocol. For this we used the passively secure protocol of Frederiksen et al. [24][Full Version, Figure 19] (which is essentially the protocol of Ishai et al [29]). Such a passively secure COT protocol is sufficient due to how it is used in the MASCOT offline phase. This COT protocol was already implemented in `SCALE-MAMBA`. For two parties P_i and P_j with the former acting as the sender and the later as the receiver, calls to the COT protocol outputs values $\{M_0^k, M_1^k = M_0^k + \Delta_i\}_{k\in[n]}$ to P_i and $\{M_{\mathbf{b}_k}^k\}_{k\in[n]}$ to P_j, where $\Delta_i \in \mathbb{F}_{2^{128}}$ is the input from P_i and $\mathbf{b} \in \{0,1\}^n$ is the choice vector from P_j, and $\forall k \in [n]\ M_0^k \leftarrow_\$ \mathbb{F}_{2^{128}}$

To obtain extended Random Oblivious Transfer (ROT), from these extended COTs, we used the decorrelation technique presented in [12][Figure 15] which consists in both parties hashing the output of the extended COT. This gives us an extended ROT in $\mathbb{F}_{2^{128}}$. However, we want to run the MASCOT protocol on prime fields $\mathbb{F}_{p_0}$ and $\mathbb{F}_{p_1}$, so to translate our extended ROT from $\mathbb{F}_{2^{128}}$ to $\mathbb{F}_p$, we take $\lceil \frac{\log_2(p)+128}{128} \rceil$ outputs in $\mathbb{F}_{2^{128}}$, concatenate them together and take the result mod p.

Results of our experiments Our implementation of the key generation protocol was tested in a LAN setting, with each party running on an Intel i7-7700K CPU with 32GB of RAM over a 10Gb/s network switch. In our experiments we found that executing more threads than the available number of cores (in our case eight) gave a performance improvement. This is because the computation between receiver and sender in the OT protocols is asymmetric, resulting in each party sometimes waiting for the other to perform some computation.

For each prime size, we report the throughput of our MASCOT implementation regarding triple generation in Table 2. We experimented with different number of threads, and for

[7] Our implementations are now included in the `SCALE-MAMBA` code-base.

each case we give our results for 1, 5 and 10 threads. By comparison to [34][Figure 9], our implementation of the MASCOT protocol for triple generation seems to have room for improvement. In particular for a 128 bit field, we are about 7 times slower than MP-SPDZ [21]. Therefore, we believe that a better implementation of MASCOT using less CPU resources would give us a significant amelioration of our BGV key generation benchmark.

n	2				3			
$\log_2 p$	64		128		64		128	
1 Thread	126.77	177.1	70.01	133.29	58.78	79.01	28.7	57.22
5 Thread	587.77	784.69	292.18	567.62	276.38	376.65	134.62	279.27
10 Thread	759.1	1032.57	365.93	740.05	389.19	547.27	185.32	379.34

Table 2. Triple Throughput for $\mathbb{F}_{p_0}$ and $\mathbb{F}_{p_1}$ in Triples per Second.

Finally in Table 3 we give figures related to the throughput of the secure randomly shared bits, namely the throughput of the algorithm in Figure 8, in the four different scenarios we experimented with. For each case, we give our result while also running the MASCOT triple generation, and also assuming that this offline phase has already been done beforehand. In the same table we eventually give the total runtime for the distributed BGV key generation procedure, which includes the time to perform the MASCOT offline phase.

Number of Parties	2		3	
FHE Plaintext size	64	128	64	128
Shared bit throughput including offline time (b/s)	416.67	263.16	194.55	124.69
Shared bit throughput excluding offline time (b/s)	1515.15	1449.28	1149.43	1041.67
Expected Expected BGV Key Generation Wall Clock	~~47m17s~~2 hours	~~2h35m34s~~5 hours	~~1h30m24s~~3 hours	~~4h55m44s~~9 hours

Table 3. Shared bit throughput and expected total time for the KeyGen protocol.

We ~~can observe~~ estimated that for two parties it ~~takes between 47 minutes and five hours~~ would take roughly five hours hours to complete the distributed key generation for two parties, and nine hours for three parties for 128-bit FHE plaintext size. Although it may seem inefficient, we argue that this protocol needs only to be done once in order to enable future computation of the offline phase of SPDZ. Moreover, we have shown that our protocol is highly parallelizable. So in practice, if such a protocol was to be run on high end servers owned by cloud service providers, the total runtime could be drastically reduced.

We pause to compare these run-times to the covertly secure distributed key generation protocol presented in [18]. This protocol did not produce public and secret keys with the same distribution as the non-distributed version. For a plaintext size of 64-bits the authors of [18] report 12 and 16 seconds key generation time for $n = 2$ and $n = 3$, for a covert

security of 1/10, i.e. an adversary can cheat with probability 1/10. For a plaintext size of 128-bits the times are 33 and 44 seconds. The execution time of this covertly secure protocol is linear in c, where the covert security measure is $1/c$. Thus to obtain comparable security to our protocol, the protocol in [18] would be utterly inpractical.

Changing the Standard Deviation Our protocol can also be run when we select the standard deviation for the centred binomial distributions to be equal to $\sigma = \sqrt{1/2} = 0.707$. The analysis of the parameters, given in Table 1 is roughly the same. However, the associated run times for key generation become faster as we no longer need to generate as many doubly authenticated bits. This is reflected in Table 4. We see that by choosing $\sigma = 0.707$, we need only two bits instead of 40 for the sampling from the centred binomial distribution. We thus obtain a factor of at least 2.5 improvement overall, compared to the previous setting.

Number of Parties	2		3	
FHE Plaintext size	64	128	64	128
Expected BGV Key Generation Wall Clock	~~16m10s~~30 mins	~~1h03m19s~~2 hours	~~28m46s~~1 hour	~~1h52m54s~~4 hours

Table 4. Shared bit throughput and expected total time for the KeyGen protocol for $\sigma = 0.707$.

7.1 Secret Keys Generated According to a Centred Binomial Distribution

Finally we examine the case of using the centred binomial distribution for generating the secret keys, and not just the noise values, with standard deviation selected to be $\sigma = \sqrt{1/2} = 0.707$, as opposed to using a low Hamming weight based distribution for the secret keys. This pushes the parameter sizes for the underlying BGV scheme up a little, as we need to cope with more potential noise growth due to the 'heavier' secret key. Using the same analysis as before we find the parameter sizes given in Table 5, with the resulting run times for distributed key generation given in Table 6. This time we performed the experiments also for $n = 4$ and $n = 5$ so as to show how the times grow with n; recall the overall method is $O(n^2)$ as mentioned earlier.

Number of Parties, n	2				3				4				5			
FHE Plaintext Size, $\log_2 p$	64		128		64		128		64		128		64		128	
Polynomial Degree	16384		32768		16384		32768		16384		32768		16384		32768	
$p_0 \vert p_1$ bit length	222	159	352	219	223	158	353	218	223	158	353	228	223	158	353	228

Table 5. Parameter Sizes for secret keys distributed according to a centred binomial distribution, and Gaussian error distribution of $\sigma = 0.707$.

Number of Parties	2		3		4		5	
FHE Plaintext size	64	128	64	128	64	128	64	128
Expected BGV Key Generation	~~5m08s~~10 mins	~~18m20s~~40 mins	~~8m12s~~16 mins	~~26m35s~~50 mins	~~11m23s~~25 mins	~~52m48s~~2 hours	~~16m14s~~30 mins	~~2h11m42s~~4.5 hours

Table 6. ~~Total~~ Expected total time for the KeyGen protocol for $\sigma = 0.707$ and secret keys generated according to a centred binomial distribution.

Acknowledgements

The authors would like to thank Carsten Baum~~and Emmanuela Orsini~~, Emmanuela Orsini and Victor Treinsoutrot for suggestions in relation to the work in this paper, and Claudio Orlandi in elaborating on the early history of the BDOZ and Tiny-OT work.

The work in this paper was carried out whilst Dragos Rotaru and Tim Wood were PhD students at the University of Bristol, and were employed by KU Leuven.

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070 and FA8750-19-C-0502, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via Contract No. 2019-1902070006, by the FWO under an Odysseus project GOH9718N, and by CyberSecurity Research Flanders with reference number VR20192203.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, ODNI, United States Air Force, IARPA, DARPA, the US Government or FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: Holz, T., Savage, S. (eds.) USENIX Security 2016: 25th USENIX Security Symposium. pp. 327–343. USENIX Association, Austin, TX, USA (Aug 10–12, 2016)
2. Aly, A., Cong, K., Cozzo, D., Keller, M., Orsini, E., Rotaru, D., Scherer, O., Scholl, P., Smart, N.P., Tanguy, T., Wood, T.: SCALE-MAMBA v1.14: Documentation (2021), `https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf`
3. Aly, A., Orsini, E., Rotaru, D., Smart, N.P., Wood, T.: Zaphod: Efficiently combining LSSS and garbled circuits in SCALE. In: Brenner et al. [11], pp. 33–44, `https://doi.org/10.1145/3338469`
4. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 483–501. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
5. Baum, C., Cozzo, D., Smart, N.P.: Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 11959, pp. 274–302. Springer, Heidelberg, Germany, Waterloo, ON, Canada (Aug 12–16, 2019)
6. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology – CRYPTO'91. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1992)

7. Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: Micciancio, D. (ed.) TCC 2010: 7th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 5978, pp. 201–218. Springer, Heidelberg, Germany, Zurich, Switzerland (Feb 9–11, 2010)
8. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 169–188. Springer, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011)
9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 489–518. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019)
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012: 3rd Innovations in Theoretical Computer Science. pp. 309–325. Association for Computing Machinery, Cambridge, MA, USA (Jan 8–10, 2012)
11. Brenner, M., Lepoint, T., Rohloff, K. (eds.): Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019. ACM (2019), https://doi.org/10.1145/3338469
12. Burra, S.S., Larraia, E., Nielsen, J.B., Nordholt, P.S., Orlandi, C., Orsini, E., Scholl, P., Smart, N.P.: High-performance multi-party computation for binary circuits based on oblivious transfer. J. Cryptol. 34(3), 34 (2021), https://doi.org/10.1007/s00145-021-09403-1
13. Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., Wang, R.: Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 590–607 (2021), https://doi.org/10.1109/SP40001.2021.00025
14. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 769–798. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
15. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 280–299. Springer, Heidelberg, Germany, Innsbruck, Austria (May 6–10, 2001)
16. Curtis, B.R., Player, R.: On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In: Brenner et al. [11], pp. 1–10, https://doi.org/10.1145/3338469
17. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 5443, pp. 160–179. Springer, Heidelberg, Germany, Irvine, CA, USA (Mar 18–20, 2009)
18. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013: 18th European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 8134, pp. 1–18. Springer, Heidelberg, Germany, Egham, UK (Sep 9–13, 2013)
19. Damgård, I., Orlandi, C.: Multiparty computation for dishonest majority: From passive to active security at low cost. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010. Lecture Notes in Computer Science, vol. 6223, pp. 558–576. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)
20. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
21. Data61: MP-SPDZ (2019), https://github.com/data61/MP-SPDZ
22. Franklin, M.K., Haber, S.: Joint encryption and message-efficient secure computation. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 266–277. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)
23. Franklin, M.K., Haber, S.: Joint encryption and message-efficient secure computation. Journal of Cryptology 9(4), 217–232 (Sep 1996)
24. Frederiksen, T.K., Keller, M., Orsini, E., Scholl, P.: A unified approach to MPC with preprocessing using OT. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9452, pp. 711–735. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015)
25. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)

26. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
27. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099 (2012), https://eprint.iacr.org/2012/099
28. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. Journal of the Association for Computing Machinery 21, 277–292 (1974)
29. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 145–161. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003)
30. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Sahai, A. (ed.) TCC 2013: 10th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 7785, pp. 600–620. Springer, Heidelberg, Germany, Tokyo, Japan (Mar 3–6, 2013)
31. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 20: 27th Conference on Computer and Communications Security. pp. 1575–1590. ACM Press, Virtual Event, USA (Nov 9–13, 2020)
32. Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster secure multi-party computation of AES and DES using lookup tables. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 17: 15th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 10355, pp. 229–249. Springer, Heidelberg, Germany, Kanazawa, Japan (Jul 10–12, 2017)
33. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 724–741. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
34. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security. pp. 830–842. ACM Press, Vienna, Austria (Oct 24–28, 2016)
35. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 158–189. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018)
36. Koiliaris, K., Xu, C.: A faster pseudopolynomial time algorithm for subset sum. In: Klein, P.N. (ed.) ACM-SIAMSymposium on Discrete Algorithms, SODA2017. pp. 1062–1072 (2017)
37. Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Computing across trust boundaries using distributed homomorphic cryptography. Cryptology ePrint Archive, Report 2019/961 (2019), https://eprint.iacr.org/2019/961
38. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 681–700. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
39. Orsini, E., Smart, N.P., Vercauteren, F.: Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. Lecture Notes in Computer Science, vol. 12006, pp. 254–283. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 24–28, 2020)
Pan, Y., Zhang, F.: A note on the density of the multiple subset sum problems. Cryptology ePrint Archive, Report 2011/525 (2011),
Pisinger, D.: Linear time algorithms for knapsack problems with bounded weights. Journal of Algorithms 33, 1–14 (1999)
40. Rivest, R.: Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer (1999), https://people.csail.mit.edu/rivest/Rivest-commitment.pdf
41. Rotaru, D., Wood, T.: MArBled circuits: Mixing arithmetic and Boolean circuits with active security. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) Progress in Cryptology - INDOCRYPT 2019: 20th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 11898, pp. 227–249. Springer, Heidelberg, Germany, Hyderabad, India (Dec 15–18, 2019)
42. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security. pp. 39–56. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017)
43. Zhu, R., Ding, C., Huang, Y.: Practical MPC+FHE with applications in secure multi-PartyNeural network evaluation. Cryptology ePrint Archive, Report 2020/550 (2020), https://eprint.iacr.org/2020/550