Efficient Joint Random Number Generation for Secure Multi-party Computation

Erwin Hoogerwerf[1], Daphne van Tetering[1], Aslı Bay[2] and Zekeriya Erkin[1,3]
[1]*Cyber Security Group, Delft University of Technology, The Netherlands*
[2]*Department of Computer Engineering, Antalya Bilim University, Antalya, Turkey*
[3]*Digital Security Group, Radboud University Nijmegen, The Netherlands*

Keywords: Joint Random Number Generation, Secure Multi-party Computation, Data Aggregation.

Abstract: Large availability of smart devices and an increased number of online activities result in extensive personalized or customized services in many domains. However, the data these services mostly rely on are highly privacy-sensitive, as in pace-makers. In the last decades, many privacy breaches have increased privacy awareness, leading to stricter regulations on data processing. To comply with this legislation, proper privacy preservation mechanisms are required. One of the technological solutions, which is also provably secure, is Secure Multi-Party Computation (SMPC) that can compute any function with secret inputs. Mainly, in several SMPC solutions, such as data aggregation, we observe that secret values distributed among parties are masked with random numbers, encrypted and combined to yield the desired outcome. To ensure correct decryption of the final result, it is required that these numbers sum to a publicly known value, for instance, zero. Despite its importance, many of the corresponding works omit how to obtain such random numbers jointly or suggest procedures with high computational and communication overhead. This paper proposes two novel protocols for Joint Random Number Generation with very low computational and communication overhead. Our protocols are stand-alone and not embedded in others, and can therefore be used in data aggregation and other applications, for instance, machine learning algorithms, that require such random numbers. We first propose a protocol that relies on bit-wise sharing of individually generated random numbers, allowing parties to adapt random numbers to yield a public sum. Second, we propose a protocol that uses the sign function to generate a random number from broadcast numbers. We provide security and complexity analyses of our protocols.

1 INTRODUCTION

Recent developments in machine learning and data analytics have allowed more and more customized personalized online services, thanks to the availability of vast numbers and types of data collected all around us. On the one hand, the data collected can be used to improve machine learning algorithms and provide users with personalized services such as news notifications or product recommendations. On the other hand, data collection raises serious privacy concerns since often sensitive data is included (Jiang et al., 2019; Zhao et al., 2019). Over the last years, we have witnessed numerous privacy breaches (Privacy International, 2020), damaging individuals and also businesses. These privacy breaches have led to increased privacy awareness, not only among users but also in the political arena, resulting in stricter laws on data processing such as the General Data Protection Regulation in Europe and the Personal Data Protection Act in Singapore (European Parliament and Council of European Union, 2016; Singapore Parliament, 2014). To comply with legislation, proper privacy preservation mechanisms are required. Among many other technological solutions, Secure Multi-Party Computation (SMPC) techniques attract more attention from academia and business as it provides provable security in different adversarial models.

Firstly introduced by Yao, SMPC protocols are designed to evaluate a function using secret inputs distributed among different parties, thereby yielding a public result (Yao, 1982). Secure Multi-Party Computation is used to perform, among others, data clustering (Erkin et al., 2013), homomorphic encryption (Lyu, 2020) and data aggregation (Erkin and Tsudik, 2012). In this paper, we focus on a particular computation, namely Joint Random Number Generation (JRNG), which is notably used a lot in privacy-

Hoogerwerf, E., van Tetering, D., Bay, A. and Erkin, Z.
Efficient Joint Random Number Generation for Secure Multi-party Computation.
DOI: 10.5220/0010534804360443
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 436-443
ISBN: 978-989-758-524-1

preserving data aggregation protocols. In smart meter settings, for example, individual measurements are sent to a data aggregator, responsible for computing aggregate statistics such as sum or average. Previous research has shown that these measurements, when obtained by adversaries in an attack on data confidentiality, can be used to infer patterns revealing customer behaviour (Wang and Lu, 2013). To prevent such eavesdropping attacks from happening, smart meters are required to encrypt their measurements before sharing these with the aggregator. However, encryption is not sufficient to prevent malicious aggregators from inferring additional knowledge (Kursawe et al., 2011). One way to achieve this and protect against malicious aggregators is by requiring all meters to mask their measurements with random numbers. Jointly generating these among all participants in the protocol ensures they sum to a publicly known value, thereby guaranteeing the final result's correct decryption. Therefore, Joint Random Number Generation can be considered an important step in privacy-preserving data aggregation protocols.

Remarkably, most research on privacy-preserving data aggregation omits how to obtain such numbers and assumes their existence. For example, Shi et al. assume that all keys during the setup of the private stream aggregation algorithm sum to zero (Shi et al., 2011; Shi et al., 2015). Lu et al. assume the same in their homomorphic encryption protocol (Lu et al., 2017). Moreover, several studies that do focus on JRNG have high computational and communication overhead. For example, Erkin and Tsudik assume a fully connected network is available, allowing each party to exchange random values with all others (Erkin and Tsudik, 2012). With large networks, this approach becomes infeasible. A similar approach by Kursawe et al. uses the notion of *leaders*, who are required to compute their random values such that all values together sum to zero (Kursawe et al., 2011). This reduces the complexity for all other nodes in the network since these only communicate with the set of leaders. However, the complexity of leader nodes remains the same as in (Erkin and Tsudik, 2012).

This paper presents two novel protocols to perform Joint Random Number Generation with minimal computational and communication overhead. The first protocol relies on bit-wise sharing of individually generated random numbers. By doing so, for each round, the information leaked to an adversary is minimized to only one bit per participant. The second protocol uses a single broadcast and is based on Diffie-Hellman key exchange (Merkle, 1978). We also present a version of the broadcast-based protocol that has reduced complexity. Finally, to compare all protocols, we conduct an analysis considering both communication and computation complexity.

Our contributions are two-fold. First, by formalizing two JRNG protocols explicitly, we provide clarity on how to efficiently perform Joint Random Number Generation. In contrast to previous work, our protocols are stand-alone and not embedded in others. Therefore, they can be used in any domain or application where JRNG is required. Finally, we show how to perform JRNG with minimal computational and communication overhead, thus improving the way JRNG is done for the whole research community.

The remainder of this paper is structured as follows. First, we discuss related work and preliminaries in section 2 and section 3. Next, we discuss the share-based protocol in section 4, before discussing the broadcast-based protocol in section 5, along with its security proof. Finally, discuss complexity in section 6 and conclude the paper in section 7.

2 RELATED WORK

Related work on Joint Random Number Generation (JRNG) can be divided into two categories: research that focuses on JRNG as a main goal, as does this one, and research that uses it to achieve a broader goal, and therefore embeds it in its protocol. First, we discuss work that embeds JRNG to, for example, perform privacy-preserving data aggregation: jointly generated random numbers are used to mask individual values before sending them to the aggregator. When the aggregator sums or multiplies all values, the masks cancel out and allow the aggregator to obtain the exact result without revealing individual values. Since in this work, we care about JRNG specifically, we only highlight those procedures and omit any other details. An example of such research is the work by Erkin and Tsudik, performing additive aggregation of plaintexts in a setting with at least three parties ($K \geq 3$) (Erkin and Tsudik, 2012). First, parties exchange random numbers in a pair-wise manner, i.e. parties p_i and p_j exchange values $r_{(p_i \to p_j)}$ and $r_{(p_j \to p_i)}$. Then, each party computes its random value R_{p_i} by summing over the $K-1$ received values:

$$R_{p_i} = \sum_{j=1, j \neq i}^{K} r_{(p_i \to p_j)} - r_{(p_j \to p_i)} \,. \tag{1}$$

It is trivial to see that the addition of these elements yields zero. While presented in a smart meter setting, this scheme can be used in any setting where JRNG is required. However, it should be noted that this work requires the presence of secure communication channels. The second work is an interactive protocol proposed by Kursawe, Danezis, and Kohlweiss

(Kursawe et al., 2011). Similar to Erkin and Tsudik, it jointly generates random numbers yielding a zero-sum. However, to reduce communication complexity, it uses the notion of *leaders*. At the start of the protocol, each party generates a random number and shares it with the set of leaders P. Then, each leader generates its random number in such a way that summing it with all received shares yields zero. Similar to Erkin and Tsudik, this protocol requires at least three parties and secure communication channels. The use of leaders reduces communication complexity for non-leader parties, the complexity for leaders is unchanged. This protocol is adapted by Van de Kamp et al. to remove the notion of leaders: instead, parties share their random number with all other parties (Van De Kamp et al., 2019). In the same work, Kursawe et al. propose a JRNG scheme based on the Diffie-Hellman key exchange. The scheme yields a publicly known product rather than a sum, using the sign function $s_i(j)$.

$$s_i(j) = \begin{cases} 1 & \text{if } i > j\,, \\ -1 & \text{if } i < j\,. \end{cases} \tag{2}$$

In this protocol, each party broadcasts its public key $g^{k_{p_i}}$, for which $g \in \mathbb{Z}_p^*$ is a public generator and $k_{p_i} \in \mathbb{Z}_p^*$ is a random secret key. Each party uses these numbers to compute its random number R_{p_i} as follows:

$$R_{p_i} = \prod_{j=1, j\neq i}^{K} (g^{k_{p_j}})^{s_i(j)k_{p_i}} = g^{\sum_{j=1, j\neq i}^{K} s_i(j)k_{p_i}k_{p_j}}\,. \tag{3}$$

Verifying that this yields one is done by

$$\begin{aligned} \prod_{i=1}^{K} R_{p_i} &= g^{\sum_{i=1}^{K}\sum_{j=1, j\neq i}^{K} s_i(j)k_{p_i}k_{p_j}} \\ &= g^{\sum_{i=1}^{K}\sum_{j=1}^{i-1}(s_i(j)+s_j(i))k_{p_i}k_{p_j}} \\ &= g^0 \\ &= 1\,. \end{aligned}$$

When using this protocol for aggregation, each party masks its individual value $c_{i,j}$ using randomness R_{p_i} and sends this to the aggregator as $g^{c_{i,j}+R_{p_i}}$. As verified above, all random numbers cancel out in the final sum, leaving the aggregator with $g^{\sum_i c_{i,j}}$. Since it is not trivial to find the correct final sum from this, Kursawe et al. argue that the aggregator should launch a brute-force attack to obtain it (Kursawe et al., 2011). More recently, Bell et al. embedded JRNG in an aggregation protocol by requiring parties to establish shared keys with their neighbours to derive pairwise random masks, masking input c_i as follows (Bell et al., 2020):

$$c_i - \sum_{j<i} m_{i,j} + \sum_{j>i} m_{i,j}\,. \tag{4}$$

This method again ensures that all masks cancel out in the final sum. The protocol operates in a semi-honest security setting. It can be extended to a so-called semi-malicious security setting, meaning that the aggregation server is required to perform the first step of the protocol honestly, but is allowed to deviate from the protocol after that (Bell et al., 2020).

Finally, we close this section by discussing work that specifically focuses on Joint Random Number Generation. In this work, JRNG is denoted by its authors as modulo zero-sum randomness (Hayashi and Koshiba, 2018). As the name explains, modulo zero-sum randomness means that when summing individual numbers, the result is zero. To ensure zero-sum randomness, three conditions have to be met:

1. *Modulo Zero Condition*: the sum of all random numbers X_i has to be zero.
2. *Independence Condition*: all numbers $X_1, ..., X_m$ used have to be generated independently.
3. *Secrecy Condition*: each player i has knowledge of the random number it generated itself, X_i, and of the fact, the sum of all numbers will yield zero, but has no additional knowledge.

In this work, to generate random numbers, the authors impose a cyclic ordering on all parties. First, each party agrees on a secret random number with its two neighbors, Z_i and Z_{i-1}. Then, parties determine their final random by subtracting these two values:

$$R_{p_i} = Z_i - Z_{i-1}\,. \tag{5}$$

It is trivial to see that in summation, this yields zero. In follow-up work, the use of multiple access channels and verifiable quantum protocols to generate modulo zero-sum randomness is discussed (Hayashi, 2018; Hayashi and Koshiba, 2019).

It is important to note that all related work in this section focuses on yielding a public sum of zero. Our approach is a more generalized one, where instead we facilitate the joint generation of random numbers that sum up to a publicly known value $N \geq 0$.

3 PRELIMINARIES

In this section, we introduce preliminaries, the security assumption, and notation used in this work.

3.1 Decisional Diffie-Hellman Problem

The security of the broadcast-based scheme presented in section 5 relies on the Decisional Diffie-Hellman Problem (DDH). This problem is defined as follows: given a cyclic group G of prime order q, its public generator g, the following two distinguishers are computationally indistinguishable (in the security parameter q); (g^x, g^y, g^{xy}), where x, y are chosen uniformly at random from $[0, \ldots q-1]$ and (g^x, g^y, g^z), where x, y, z are chosen uniformly at random from $[0, \ldots q-1]$.

3.2 Security Setting

We assume an environment composed of a static group of K entities wanting to jointly establish random numbers with a publicly known sum. Similar to related work, we assume all entities to be semi-honest (honest-but-curious). This means they follow the protocol but are not prevented from inferring additional knowledge from received information. Each entity is capable of performing modular operations, such as modular addition, and of generating strong random numbers. We do not assume the presence of any other active entity, such as a trusted third party or an aggregator, to perform computations. Protecting against active adversaries can be done at the cost of additional communication and computation complexity, but is considered out of the scope of this work.

For the bit-wise sharing protocol, we assume the existence of secure communication channels. These ensure that any message sent over the network can only be retrieved by its sender and recipient.

3.3 Notation

The notation used is summarized in Table 1.

Table 1: Notation Summary.

Symbol	Definition
p	large prime
$r_{p_i \to p_j}$	random value generated by p_i send to p_j
g	generator for $\mathbb{Z}_p^*$
K	number of entities in the protocol
l	security parameter
m	bit-length of random numbers
C_i	carry-over from bit i
N	publicly known target value
R_{p_i}	random number generated by entity p_i
k_{p_i}	temporary values used to construct R_{p_i}
x_{ij}	bit j of random number R_{p_i}
x_{Nj}	bit j of publicly known target value N
y_i	updated bit i replacing x_{ii} of R_i
d_i	scaled carry-over for key entity i
$s_i(j)$	sign function
Z_a	short for $\mathbb{Z}/a\mathbb{Z}$

4 SHARE-BASED JRNG PROTOCOL

In this section, we describe a Joint Random Number Generation protocol that uses the bit-wise sharing of individually generated random numbers. The random numbers generated by the protocol sum up to a publicly known integer $0 \leq N \leq 2^m$, for some $m \in \mathbb{N}$. Initially, all entities generate a random number R_{p_i}, and a subset of all entities is selected as m key entities. Then, the acting key entity adjusts the j'th bit of its random number such that it equals the j'th bit of the target N. To do so, each entity sends its j'th bit to the acting key entity, which sums these to determine if its j'th bit should change. Finally, to prevent carry-over from affecting subsequent computations, the acting key entity subtracts this from its random number. This process is repeated for m iterations until each key entity has adjusted its random number once.

The protocol is formalized as follows:

1. All K entities generate random numbers $R_{p_i} \in Z_{2^m}$, such that $R_{p_i} = \sum_{j=1}^{m} x_{ij} 2^{j-1}, x_{ij} \in \{0,1\}$. m entities are assigned key entity as in (Kursawe et al., 2011).
2. Then, for m iterations:
   2.1. All entities i send x_{ij} to the acting key entity using the underlying secure network.
   2.2. The acting key entity calculates $d_j = \sum_{i=1, i\neq j}^{K} x_{ij}$. If $d_j \equiv x_{Nj} \mod 2$, it sets its j'th coefficient to $y_j = 0$. Otherwise, it sets its j'th coefficient to $y_j = 1$.
   2.3. Finally, the acting key entity calculates $C_j = \lfloor \frac{d_j + y_j}{2} \rfloor 2^j$ and updates its random number to $R_j \leftarrow R_j - C_j \mod 2^m$.

In practice, usually $N = 0$. However, to keep the final outcome secret, different values for N can be used. Then, the final outcome can only be determined by the entities that know N, since these can subtract it from the final outcome. While in this work the protocol is demonstrated in a binary setting, it can be used in any base-k setting for some $k \in \mathbb{N}$.

A distinguishing feature of this protocol is that instead of communicating full-sized random numbers in each round, this protocol uses bit-wise communication. This reduces communication complexity to a total of $K-1$ bits per iteration. This reduction becomes more significant for large numbers and high amounts of entities. Since only key entities are required to perform modular operations, the scheme can be used in situations where not all entities have equal or sufficient computing power. Even more important, the use of key entities ensures that computation complexity is dominated by the number of key entities, not by the number of entities in general. Compared to the work by Erkin and Tsudik, (Erkin and Tsudik, 2012), communication complexity is reduced and a fully connected network is not required.

It is important to note that this scheme should not be used for applications where certain bits are more

relevant than others, as is the case in a bank transfer. If this protocol would be used to mask a bank transfer with encryption $c = m + R \mod 2^m$, this would allow the adversary to distinguish between high-valued and low-valued transactions by only corrupting one entity.

Finally, it should be noted that the scheme can also be used for situations where $m > K$. This can be done as follows: for each j, where $K < j \leq m$, we require entity K to act during all remaining iterations, thereby continuously adjusting its random number.

4.1 Security

When assessing the share-based protocol in a semi-honest security setting, it becomes clear that the protocol is vulnerable to collusion among key entities. In each round, the acting key entity receives an entity's j'th bit, thereby obtaining partial information about each random number R_{p_i}. Because of this, collusion allows key entities to partially reconstruct the random number held by other entities. While this seemingly renders the protocol unusable, situations exist in which computational speed is more important and where non-privacy-preserving Joint Random Number Generation suffices. Such a situation is found in the work by Garay, Schoenmakers, and Villegas (Garay et al., 2007). Here, a random number of r is jointly generated, before being encrypted using fresh randomness. By adding fresh randomness to r, it is effectively masked. Therefore, in this case, our bit-wise sharing protocol provides sufficient security.

In situations where privacy of individual bits is required, secure multi-party computation techniques, such as (additive) secret sharing can be added to the protocol (Van De Kamp et al., 2019). This can be achieved as follows: during the second step of the protocol, instead of directly sending its bit x_{ij} to the acting key entity, each entity first generates $K-1$ shares s_{ij} of its bit, such that $x_{ij} = \sum_{j=1}^{K-1} s_{ij} \mod N$, similar to the work by Garcia and Jacobs (Garcia and Jacobs, 2010). Of these, it holds one by itself and distributes the others among other non-acting key entities. Finally, instead of sending entire bits, each entity sends the received shares to the acting key entity, together with its own share. Because of this, the acting key entity receives all shares of all bits at the same time. This allows it to reconstruct the original bits, without knowing which bit originally belonged to which entity, thereby preserving the privacy of each bit. While this allows the privacy protection of individual bits, it also leads to an increase in communication complexity. Therefore, using a privacy-preserving version of this protocol should be justified.

5 SINGLE BROADCAST-BASED JRNG PROTOCOL

In this section, we present a protocol, which is inspired by (Kursawe et al., 2011), that does not require a secure network and only needs one broadcast per entity to jointly establish random numbers with a publicly known sum of N. We again assume a group of at least three entities. Additionally, we assume a publicly known ordering exists. Similar to the protocol in (Kursawe et al., 2011), this protocol is based on Diffie-Hellman key exchange and uses the sign function (Kursawe et al., 2011). However, different from the protocol by Kursawe et al. our protocol performs additive aggregation rather than multiplicative aggregation. Thus, we provide the additive variant here.

Before we formalize the protocol, it is important to note that in this work we assume a fully connected network. This is, however, not required since all messages can be publicly known. Therefore, different network topologies, such as a star network where all messages are relayed through a semi-honest entity, can also be used. Additionally, in this work, we set $N = 0$, while in practice any $N \in \mathbb{N}$ can be used.

1. A public generator $g \in \mathbb{Z}_p^*$ is agreed upon. All entities generate a random $k_{p_i} \in \mathbb{Z}_p^*$ and $g^{k_{p_i}}$.
2. All entities publicly broadcast $g^{k_{p_i}}$.
3. Each entity p_i computes its final random number:

$$R_{p_i} = \sum_{j=1, j\neq i}^{K} s_i(j)(g^{k_{p_j}})^{k_{p_i}} = \sum_{j=1, j\neq i}^{K} s_i(j) g^{k_{p_i} k_{p_j}}.$$

Correctness is easily verified since $s_i(j) + s_j(i) = 0$ for all $i \neq j$, and thus

$$\begin{aligned}\sum_{i=1}^{K} R_{p_i} &= \sum_{i=1}^{K} \sum_{j=1, j\neq i}^{K} s_i(j) g^{k_{p_i} k_{p_j}} \\ &= \sum_{i=1}^{K} \sum_{j=1}^{i-1} (s_i(j) + s_j(i)) g^{k_{p_i} k_{p_j}} = 0\,.\end{aligned}$$

5.1 Security

The security of this protocol is based on the hardness of Decisional Diffie-Hellman Assumption. As the protocol is a direct application of the Diffie-Hellman Key exchange protocol, its security can be reduced to Decisional Diffie-Hellman problem as R_p's are constructed from the public-key of Diffie-Hellman Exchange protocol and can not be distinguished from random R_p's.

5.2 Adapted Broadcast-based Protocol

To reduce the complexity of the broadcast-based scheme, a ring-topology can be assumed. This requires the assumption that any entity p_i is only able to communicate with its neighbours: the entities ranked above and below it. Since this rank is seen in $(\text{mod } K)$, the lowest and highest ranked entities are also neighbors. The scheme is formalized as follows:

1. A public generator $g \in \mathbb{Z}_p^*$ is agreed upon. All entities generate a random $k_{p_i} \in \mathbb{Z}_p^*$ and $g^{k_{p_i}}$.
2. Each entity publicly sends $g^{k_{p_i}}$ to both neighbors.
3. Each entity p_i computes

$$R_{p_i} = \sum_{j \in N_i} s_i(j)(g^{k_{p_j}})^{k_{p_i}} = \sum_{j \in N_i} s_i(j) g^{k_{p_i} k_{p_j}}.$$

Correctness follows since $i \in N_j$ implies that $j \in N_i$. Therefore, correctness of this scheme can be verified in the same way as the original scheme is verified.

6 COMPLEXITY ANALYSIS

In this section, we analyze the complexity of our protocols. For computation complexity, we base our analysis on the number of modular additions and modular exponentiation performed. For communication complexity, we base our analysis on the number of messages exchanged and their lengths. We conclude this section by comparing our protocols.

6.1 Computation Complexity

The share-based protocol is asymmetric since the number of operations performed by key entities differs from the number of operations performed by non-key entities. This underlines the protocol's versatility: non-key entities require significantly less computing power than key entities. Its computation complexity is dominated by modular addition, performed each round by the acting key entity using all $K-1$ received values. For m iterations, this results in $m \cdot (K-1)$ modular additions. It is important to note that in this protocol modular addition is performed using single bits. In contrast to the share-based protocol, the single broadcast-based protocol is symmetric, meaning that the number of operations performed is the same for all entities. Besides modular addition, the complexity of the broadcast-based protocol is dominated by modular exponentiation. Each entity performs both operations once during the protocol: modular exponentiation during the first step and modular addition during the final step. This results in K modular exponentiations and $K \cdot (K-1)$ modular additions, since modular addition is performed using $(K-1)$ numbers. In contrast to the first protocol, the addition is performed using complete numbers. Compared to the original version of the broadcast-based protocol, its adapted version has lower computation complexity. This is caused by the fact that entities only send messages to their two neighbors. As a consequence, the amount of numbers used in modular addition is reduced to 2 and computation complexity to $2K$. The number of modular exponentiation remains the same.

6.2 Communication Complexity

For the share-based JRNG protocol, communication complexity strongly depends on the number of iterations m performed during one run of the protocol. For each iteration, each entity sends one message to the acting key entity. Consequently, for m iterations, this results in m messages send by non-key entities and $m-1$ messages send by key entities. For K entities, this yields a communication complexity of $O(mK)$. This is comparable to any other system using the notion of l leaders, as discussed in Related Work. However, since our protocol requires entities to share one bit of their random number only, communication complexity is reduced, even if m far exceeds l. The single broadcast-based protocol uses broadcasting to communicate. Therefore, its communication complexity is rather high. Each entity sends one message to all others, resulting in $K \cdot (K-1)$ messages being sent during one execution of the protocol, simplified as K^2. Additionally, compared to the share-based protocol, messages are longer, since numbers are communicated entirely. In the adapted version of the single broadcast-based protocol, each entity only sends two messages, thereby reducing communication complexity to $2K$. Similar to the original version of the protocol, numbers are communicated entirely.

6.3 Comparison

Computation and communication complexity are shortly summarized in Table 2. When looking at communication complexity, it becomes visible that both the share-based and the adapted protocol require a number of messages that is linear with K. The share-based protocol requires the shortest messages and therefore has the lowest communication complexity. When taking into account computation complexity, it becomes visible that, again, the share-based and adapted protocol have the lowest complexity. Since the share-based protocol does not require modular ex-

Table 2: Computation and communication complexity of existing and our works.

	Modular Addition	Modular Exp.	Number of Messages	Message length (bits)	Encryption/ Decryption	Key Agreement
(Erkin and Tsudik, 2012)	$K \cdot (K-1)$	-	K^2	m	-	-
(Kursawe et al., 2011) DH key exchange based	$K \cdot (K-1)$	$2K$	K^2	m	-	-
(Kursawe et al., 2011) Interactive	$P \cdot (K-1)$	-	KP	m	KP/KP	-
(Van De Kamp et al., 2019)	$K \cdot (K-1)$	-	K^2	m	-	-
(Bell et al., 2020)	$K \cdot (K-1)$	-	$2K$	m	-	$2K$
(Hayashi and Koshiba, 2018)	K	-	$2K$	m	-	$2K$
Our Share-based pro.	$m \cdot (K-1)$	-	K	1	-	-
Our Broadcast-based pro.	$K \cdot (K-1)$	K	K^2	m	-	-
Our Adapted broadcast-based pro.	$2K$	K	$2K$	m	-	-

ponentiation, while the others do, it has the best computation complexity. However, it is important to note that it does require the existence of secure communication channels, while messages in the broadcast-based protocol can be shared publicly.

In Table 2, we also include computation and communication complexity of related works. Comparing these to the share-based protocol shows that the share-based protocol has better computation and communication complexity, as it requires less modular additions and sends fewer and shorter messages. Similar to (Erkin and Tsudik, 2012), it requires secure communication channels. Table 2 also shows the computation and communication complexity of the broadcast-based protocol are similar to those of the works by (Erkin and Tsudik, 2012; Kursawe et al., 2011) and (Van De Kamp et al., 2019). Additionally, while the works by (Bell et al., 2020) and (Hayashi and Koshiba, 2018) have better communication complexity, they require additional key agreements to be performed. The same holds for the interactive protocol by (Kursawe et al., 2011), while computation and communication complexity are similar, it requires additional encryption and decryption operations.

7 CONCLUSION

While Joint Random Number Generation can be found in several Secure Multi-Party Computation solutions, such as data aggregation, most research omits how to obtain such numbers and simply assumes their existence. In this paper, we have presented two computationally efficient protocols for Joint Random Number Generation. One of these is share-based, using bit-wise updates to ensure that all individual numbers yield the correct sum. While this protocol does not provide semi-honest security, situations exist in which the current security guarantees suffice and where computational speed is more important. For completeness, we also demonstrate measures to increase the security of the share-based protocol. The other protocol is broadcast-based and uses the sign function to ensure that all generated numbers yield a public sum. This protocol relies on the computational Diffie-Hellman Problem to provide semi-honest security. Additionally, we have proposed an adapted version of the broadcast-based protocol that has lower computation and communication complexity. By comparing all three protocols based on computation and communication complexity, it becomes clear that the share-based protocol is most efficient. However, the fact that it requires secure communication channels hinders its applicability. Therefore, when choosing which protocol to employ, the desired security guarantees and the additional cost of establishing secure channels should be taken into account.

ACKNOWLEDGMENT

This work was partly supported by SECREDAS project that has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 783119. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Austria, Belgium, Czech Republic, Germany, Spain, Finland, France, Hungary, Italy, Poland, Portugal, Romania, Sweden, Tunisia, United Kingdom.

REFERENCES

Bell, J. H., Bonawitz, K. A., Gascón, A., Lepoint, T., and Raykova, M. (2020). Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269.

Erkin, Z. and Tsudik, G. (2012). Private computation of spatial and temporal power consumption with smart meters. In *International Conference on Applied Cryptography and Network Security*, pages 561–577. Springer.

Erkin, Z., Veugen, T., Toft, T., and Lagendijk, R. L. (2013). Privacy-preserving distributed clustering. *EURASIP Journal on Information Security*, 2013(1):4.

European Parliament and Council of European Union (2016). Regulation (eu) 2016/679. Availableat:https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN.

Garay, J., Schoenmakers, B., and Villegas, J. (2007). Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer.

Garcia, F. D. and Jacobs, B. (2010). Privacy-friendly energy-metering via homomorphic encryption. In *International Workshop on Security and Trust Management*, pages 226–238. Springer.

Hayashi, M. (2018). Secure modulo sum via multiple access channel. *arXiv preprint arXiv:1812.10862*.

Hayashi, M. and Koshiba, T. (2018). Secure modulo zero-sum randomness as cryptographic resource.

Hayashi, M. and Koshiba, T. (2019). Verifiable quantum secure modulo summation. *arXiv preprint arXiv:1910.05976*.

Jiang, L., Lou, X., Tan, R., and Zhao, J. (2019). Differentially private collaborative learning for the iot edge. In *EWSN*, pages 341–346.

Kursawe, K., Danezis, G., and Kohlweiss, M. (2011). Privacy-friendly aggregation for the smart-grid. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 175–191. Springer.

Lu, R., Heung, K., Lashkari, A. H., and Ghorbani, A. A. (2017). A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot. *IEEE Access*, 5:3302–3312.

Lyu, L. (2020). Lightweight crypto-assisted distributed differential privacy for privacy-preserving distributed learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Merkle, R. C. (1978). Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299.

Privacy International (2020). Data breach. Availableat:https://privacyinternational.org/examples/data-breach.

Shi, E., Chan, T. H., Rieffel, E., Chow, R., and Song, D. (2011). Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17. Citeseer.

Shi, Z., Sun, R., Lu, R., Chen, L., Chen, J., and Shen, X. S. (2015). Diverse grouping-based aggregation protocol with error detection for smart grid communications. *IEEE Transactions on Smart Grid*, 6(6):2856–2868.

Singapore Parliament (2014). Personal data protection act overview. Availableat:pdpc.gov.sg/Overview-of-PDPA/The-Legislation/Personal-Data-Protection-Act.

Van De Kamp, L., Ugwuoke, C., and Erkin, Z. (2019). Economy: Ensemble collaborative learning using masking. In *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, pages 1–6. IEEE.

Wang, W. and Lu, Z. (2013). Cyber security in the smart grid: Survey and challenges. *Computer networks*, 57(5):1344–1371.

Yao, A. C. (1982). Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE.

Zhao, L., Wang, Q., Zou, Q., Zhang, Y., and Chen, Y. (2019). Privacy-preserving collaborative deep learning with unreliable participants. *IEEE Transactions on Information Forensics and Security*, 15:1486–1500.