Efficient protocols for joint random number generation using multi-party computation techniques

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

Erwin Hoogerwerf

Delft, Nederland
Juni 2016

Copyright © 2016 door Erwin Hoogerwerf. Alle rechten voorbehouden.
BSc verslag TECHNISCHE WISKUNDE

“Efficiënte protocollen voor het genereren van samenhangende random getallen voor groepsberekeningen”

“Efficient protocols for joint random number generation using multi-party computation techniques”

Erwin Hoogerwerf

Technische Universiteit Delft

Begeleider

Dr.ir. Z. Erkin

Overige commissieleden

Prof.dr.ir. K. I. Aardal  Dr. J. G. Spandraw

Juni, 2016  Delft
Abstract

There exists several applications where it is desirable to produce random numbers such that adding these random numbers yields a publicly known sum. Efficient algorithms are necessary for generating such joint random numbers in an untrusted environment in settings such as, but not restricted to, multi-party computation (MPC). Applications range all the way to secure license plate identification or facial recognition. In this thesis, we give a brief introduction into the field of cryptography. Then, we focus on a core component in MPC protocols, namely on how to generate these joint random numbers. We first analyze the existing work. Then, we propose two new protocols to efficiently achieve joint random number generation, both in terms of computational demand and communication. We show that our second protocol can also omit the need for secure networks, meaning all the messages can be public. In addition to these protocols we also introduce an extension for general joint random number generation protocols in large networks based on a star topology, with explicit protocols for parties that are joining or leaving. We analyze all our proposals in terms of security, complexity and performance. Finally, we discuss questions that remain open after this thesis.
Acknowledgements

I would like to express my gratitude towards my supervisor, Dr. Zekeriya Erkin, for supporting me throughout this project, granting me the opportunity to publish a paper, and especially for showing me many of the aspects of cyber security. I also want to thank Prof. Karen Aardal for organizing the group meetings in the optimization group and reviewing an early version of this thesis.

I want to thank Kris Shrishak for working with me to produce a second paper, and I wish him good luck on his new position as a Ph.D at Technische Universität Darmstadt.

Finally I want to thank the Ph.D. students from the Cyber Security Group in Tu Delft for their help in finalizing this thesis and the pleasant company during this project. This includes in particular Majid Nateghizad, Gamze Tillem and Chibuike Ugwuoke.

Erwin Hoogerwerf
Delft, Juni 2016
1 Introduction

Cryptographic protocols for protecting sensitive data against untrustworthy service providers have shown significant progress in the last decade. Biometric data matching [13], data aggregation [8], recommender systems [2], and data mining [19] are just a few domains where researchers propose to protect the sensitive data against the untrusted service provider by means of cryptographic primitives, e.g. homomorphic encryption [3]. Protocols based on multi-party computation techniques investigated a number of different application scenarios, i.e. centralized, decentralized systems, presence or absence of semi-trusted entities, and security models, namely semi-honest and malicious. There is even a new research domain around processing encrypted signals known as Signal Processing in the Encrypted Domain (SPED) that studies privacy-preserving versions of signal processing algorithms such as registration plate detection [7], face recognition [9] or speech processing [11].

As processing sensitive data has found many application domains, we do expect more attention and thus, more research on developing cryptographic protocols. The main research challenge remains the efficiency of such protocols since the computational and communication overhead introduced is still considerable compared to the versions without protection. Especially when applying the protocols to larger groups, e.g. social networks or sensor networks, many of these protocols still are too expensive [16].

Particularly in data aggregation within the field of multi-party computation (MPC) solutions such as [10], [14], [17], the researchers assume that involved parties have random values with certain structure. For example, all participants in a group $G$ have random numbers $R_i$ such that $\sum_{i \in G} R_i = 0$ or $\prod_{i \in G} R_i = 1$. We refer to this structure as the underlying homomorphic property. This assumption is later used for building the cryptographic protocol and is not always explained, even though it is a crucial step. Therefore, we address the problem of generating random numbers with a public sum. More precisely, within a protocol with $K$ parties, we would like to have $K$ random and private numbers, generated locally by each party, that add up to a publicly known value.

To the best of our knowledge, there has not been any extensive study on this topic. The only other works found in literature that address joint random number generation (JRNG) are [10] and [14]. Both, however, assume a secure underlying network to send secret messages. Additionally, both solutions require a complete graph.

In this thesis we present two new protocols for JRNG. The first is a bit-sharing scheme under the same assumptions as the existing work, while the second is a scheme based on the discrete log problem that uses only a single broadcast message and is proven secure without the need for secure channels between the entities. In order to make our protocols scalable, we also present an extension for general JRNG protocols in large networks based on a star topology.

We envision that the protocols proposed in this work are highly beneficial for SPED and MPC in general. Both of the new JRNG schemes as well as the extension for large networks are being prepared for submission as a paper in the International Journal of Applied Cryptography [12].

The rest of the paper is organized as follows. In Section 2 we give a brief introduction to cryptography, introduce the notation and explain the Diffie-Hellman key exchange protocol. In Section 3 we discuss the related work and introduce our two new protocols for JRNG. Then, in Section 4 the extension to larger networks is discussed. In Section 5 we provide the complexity analysis of all the JRNG protocols as well as the network extension. Finally, we provide the discussion in Section 6 and conclusion in Section 7.
2 Preliminaries

Cryptography involves the art of secret sharing. That is, getting a secret message from one person to another without revealing the contents of the message. There are generally two ways to do this: one can either change the data, thus making it unreadable to anyone who does not know how to undo the changes, or hide the information such that it will not be found. From a mathematical perspective the latter is not very interesting, and thus we will only be considering changing the data, which we will refer to as encrypting for the remainder of this thesis. Within the field of cryptography there are 3 main research topics: Confidentiality, Integrity and Authentication. We briefly discuss these and then focus on the Diffie-Hellman problem, which lies at the heart of our JRNG scheme based on a single broadcast.

2.1 Confidentiality

In order to provide confidentiality, encryption functions are used. The most basic protocols for encryption are those concerned with protecting the input data, commonly referred to as plaintext, from parties that attempt to access data they are not supposed to acquire. If, for example, the first party, who is commonly known as Alice, wants to send a sentence to the second party, Bob, she could decide to use a cypher. A cypher is one of the oldest cryptosystems and moves each letter a couple of places up in the alphabet. If Alice and Bob have priorly agreed to move each letter by 5 places, then we can use the following table to encrypt and decrypt the message. Note that the empty spot means “space” (as in between words).

```
 a b c d e f g h i j k l m n o p q r s t u v w x y z
 f g h i j k l m n o p q r s t u v w x y z a b c d e
```

For example, “hi my name is alice” will be encrypted to “mnercesfrjenxefqnhj”. Notice that Bob can easily decrypt using the same table. It may be left to the reader that manually encrypting messages using such tables is not very efficient, considering a different table should be constructed for every key that can be used. Therefore we want to use encryption and decryption functions that avoid the use of tables. In the example above the encryption function would be $\epsilon_5(c_i) = c_i + 5$, and the decryption function $\delta_5(\epsilon_5(c_i)) = \epsilon_5(c_i) - 5$. Since there are only 27 possibilities, this obviously is not a very good way to protect your data, but if one allows functions such as $\epsilon_{3,4}(c_i) = 3c_i + 4$ then it might be a decent protocol for limited amounts of data. This becomes infeasible, however, if larger amounts of data need to be transported because a computer can simply check all the mappings of the alphabet on itself. The more data is known, the fewer valid plaintexts are viable, and the greater the chance someone guesses your encryption. This can be done even more efficiently, as is studied in crypto-analysis. The focus of this work, however, does not lie in this field but rather in the cryptographic protocols themselves.

In order to function, most cryptographic protocols need a secret shared key to work. In the example above the secret key was the number 5, i.e. the number of places that the letters were moved. In general, however, these keys cannot be pre-shared. For example, if you want to chat to someone new over the Internet, you do not yet have a pre-shared key. Thus there is need for key establishment protocols. Our work falls within this category, since we establish joint random numbers that in further application can be used to encrypt the data.
2.2 Authentication

Authentication is concerned with making sure that whenever Alice tries to send a message to Bob, Bob can be sure that the message was actually sent by Alice rather than some third party impersonating Alice. These protocols are generally not concerned with masking the data, and are thus not standalone protocols. They are generally used in conjunction with key agreement protocols so both parties know that the agreed key is known (only) by the other party. Since identity confirmation is outside the scope of this work, we will not go into further detail.

2.3 Integrity

Guaranteeing the integrity of a message is making sure that the contents of a message are authentic. This is different from confidentiality since an outsider may not be able to read the contents of a message, but he may be able to change it. Although we acknowledge the importance of data integrity for joint random numbers, it falls outside the scope of this work.

2.4 Multi-party computation

First introduced by Yao in [20], multi-party computation is concerned with finding the public output of a function with secret inputs. For example, say there are two millionaires, Alice and Bob. After speaking to one another and discussing how rich they are, they decide to figure out who is the richest. The problem is, they are reluctant to share just how much money they have. Mathematically, this is equivalent to finding \( x_1 > x_2 \) without any party knowing \( x_1 \) or \( x_2 \). Yao was the first to present a solution to this problem, and there has been a lot of research since. A more concrete application of MPC is private voting, where party members secretly cast their vote. This can be used as a means of countering corruption, since people can no longer check if the corrupted entity voted in their favor.

A real-world application of MPC is currently being used in Denmark [5]. Up to a few years ago several thousand farmers produced sugar beets that they all sold to Danisco, the only Danish sugar producer. Due to Danisco closing one of its factories, however, there came a need for a national market for selling contracts, which allows a farmer to produce a certain amount of sugar beets. This national market has taken the form of a secret auction, where all bids and offers are inputted securely and the market price is computed without ever decrypting any of the values.

A special case of MPC is secure addition, where a group of \( N \) entities with private values \( c_i \) want to determine the sum of their values \( \sum_{i=1}^{N} c_i \) without revealing their private values. We anticipate that secure addition is the main application for our JRNG protocols that are presented in Section 3.

2.5 Diffie-Hellman key exchange

We now introduce the Diffie-Hellman protocol (DH), which is at the basis of one of the protocols for JRNG in Section 3.4. Diffie and Hellman proposed an ingenious way of allowing two parties to establish a valid, cryptographically strong key, that is, hard to break, without a third party learning this key even if the third party learns all the information sent between the two parties [6]. Before we can explain the scheme however, we first introduce some of the definitions at heart of the protocol.
2.6 Mathematics in finite groups

DH works, like most other encryption protocols, in finite groups. This has several reasons, most of them involving the security of the protocol, and more specifically the presence of trapdoor functions. A trapdoor function is a function that is relatively easy to evaluate, but is really hard to calculate the inverse of if one does not know a certain property. This property is usually the secret key that can be used to decrypt the ciphertext. Trapdoor functions allow a user to easily encrypt their secret data, and only allow those that possess the secret key to decrypt the data.

We now quickly recap the mathematics at the base of the cryptographic models that we will look into.

**Definition 2.1.** A group \( G \) with operation \( \times \) is a set of elements such that the following properties hold:

1. \( \exists e \in G \) such that \( e \times g = g \times e = g \) for all \( g \in G \).
2. \( \forall f, g \in G; f \times g \in G \)
3. \( \forall g \in G \) there exists an unique \( g^{-1} \) such that \( g \times g^{-1} = g^{-1} \times g = e \)
4. \( a \times (b \times c) = (a \times b) \times c \) for all \( a, b, c \in G \)

**Definition 2.2.** A ring \( R \) is a group with both an additive and a multiplicative operation. That is, \((R, +) \) is an abelian group and for all \( a, b \in R \) we have \( ab \in R \). Multiplication is associative and distributive, but \( R \) with multiplication is not a group.

**Definition 2.3.** Two integers \( m, n \in \mathbb{Z} \) are called coprime if \( \gcd\{m, n\} = 1 \).

Recall that \( \mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z} \) is a ring with normal addition and multiplication modulo \( n \). Also, \( \mathbb{Z}_n^* \), the set of elements coprime to \( n \), is a group with respect to multiplication. When referring to \( \mathbb{Z}_n^* \) we will always imply multiplication unless explicitly mentioned otherwise. For the remainder of this thesis we will be working with either normal arithmetic or modulus some \( n \).

**Definition 2.4.** An element \( g \in G \) is called a generator for \( H \subset G \) if any element in \( H \) can be written as a power of \( g \).

**Definition 2.5.** An element \( g \) is called a primitive root modulo \( n \) if \( g \) is a generator for \( \mathbb{Z}_n^* \).

Now that we have a base understanding, let us go back to the Diffie-Hellman protocol as mentioned above. This is a scheme over \( \mathbb{Z}_p^* \) with primitive root \( g \), for some prime \( p \). We show how the scheme works for Alice and Bob.

1. Alice and Bob agree on primitive root \( g \in \mathbb{Z}_p^* \).
2. Alice and Bob generate random numbers \( r_1, r_2 \in \mathbb{Z}_p \).
3. Alice sends Bob \( g^{r_1} \) and Bob sends Alice \( g^{r_2} \).
4. Alice calculates \( R = g^{r_1 r_2} = (g^{r_2})^{r_1} \) and Bob calculates \( R = g^{r_1 r_2} = (g^{r_1})^{r_2} \).

Let us first consider an example. Assume Alice and Bob have agreed to work modulo 71. with primitive root 7. Alice chooses secret number \( r_1 = 20 \), Bob chooses \( r_2 = 58 \). Then Alice calculates \( 7^{20} \equiv 37 \mod 71 \). Bob finds \( 7^{58} \equiv 18 \mod 71 \). Bob sends 18 to Alice and Alice sends 37 Bob. Now Alice calculates \( 18^{20} \equiv 20 \mod 71 \).

1. Alice and Bob agree on primitive root \( g = 7 \in \mathbb{Z}_{71} \).
2. Alice and Bob generate random numbers \( r_1 = 20 \) and \( r_2 = 58 \).
3. Alice sends Bob \( g^{r_1} = 7^{20} \equiv 37 \mod 71 \) and Bob sends Alice \( g^{r_2} = 7^{58} \equiv 18 \mod 71 \).
4. Alice calculates \( R = g^{r_1 r_2} = 18^{20} = 20 \mod 71 \) and Bob calculates \( R = g^{r_1 r_2} = 37^{58} = 20 \mod 71 \).
Thus both Alice and Bob are capable of finding the shared secret $R$.

### 2.7 Security assumptions

It is clear that both Alice and Bob can calculate the shared secret $R = g^{r_1r_2}$. It may not be clear however whether an outsider can derive this secret. For this we distinguish between two types of attacks, which correspond to different settings.

**Definition 2.6.** A protocol is said to be in the semi-honest or honest-but-curious setting if all participating entities follow the protocol specifications, but may try to get hold of additional information they are not entitled to know.

An adversary in the semi-honest setting has access to all the public data as well as the protocol specifications, and is allowed to corrupt a limited amount of users. Upon corrupting a user, the adversary gains access to all the information this entity knows within the semi-honest setting. Such an adversary is usually called passive.

**Definition 2.7.** A protocol is said to be in the malicious setting any of the participants may deviate from the protocol specifications in order to break the protocol. An adversary that is deviating from the protocol is called an active adversary.

An adversary in the malicious setting is usually called active and has access to all the tools that a passive adversary has, but now in the malicious setting.

Diffie-Hellman is thought to be secure in the semi-honest setting. To our knowledge, however, there exists no formal proof that DH is secure. The assumption of the non-existence of any function that in polynomial time can derive $g^{r_1r_2}$ from $g^{r_1}$ and $g^{r_2}$ is called the Diffie-Hellman Problem assumption (DHP), and it still holds today. Other such assumptions are decisional Diffie-Hellman (DDH), which is very closely related to DHP where one, given $g, g^a, g^b$ and $g^c$, needs to determine if $ab = c$, which is the same as determining whether $g^{ab} = g^c$, instead of straightforwardly calculating $g^c$. It may be clear that once one can solve DHP, that he can also solve DDH, but not necessarily the other way around. Thus DDH is a stronger assumption than DHP. Finally there is the discrete log assumption (DL), which says that it is computationally hard to compute $r_1$ given $g$ and $g^{r_1}$. It may be left to the reader that DL is a weaker assumption than DDP and DHP. For the remainder of this thesis all the protocols are in the semi-honest setting.

\[
\begin{align*}
\text{DDH} & \quad \text{is stronger than} \quad \text{DHP} & \quad \text{is stronger than} \quad \text{DL}
\end{align*}
\]

![Security assumption hierarchy](image)

**2.8 Attacks on Diffie-Hellman**

Although no passive adversary can find the common key from Alice and Bob without breaking the DDH assumption, there exists a way around this. Suppose that a third entity, denoted as Charlie, wants to intercept the traffic between Alice and Bob and has the ability to alter communications. In this case, Charlie might want to execute a man-in-the-middle attack (see Figure 2.8). He does this by running two instances of DH: One with Alice and one with Bob. In the first instance Charlie will masquerade as Bob, and in the second as Alice. This way Alice now thinks she shares a secret key with Bob and so does Bob thinks he shares a secret with Alice, while in fact they both share a key with Charlie. Now whenever Alice wants to send a message to Bob using the encryption key just derived, Charlie can intercept this message and
decrypt it using the key he shares with Alice. Then he will encrypt it using the shared key with Bob and send it to Bob. This way Bob thinks he securely received a message from Alice, while Charlie has in fact access to the message. Thus Charlie can now read all the communications between Alice and Bob. This attack is illustrated in Figure 2.8.

Another weakness is that Alice cannot confirm that it is Bob she is talking to, thus being an example of a protocol without identity confirmation.

Figure 2: Normal protocol run verses run during man-in-the-middle attack

Man-in-the-middle attack:

Normal run: Alice Bob

Charlie masquerading as Bob

Charlie masquerading as Alice

Charlie

2.9 Notation

The notation presented in Table I is used throughout the thesis.

3 Joint random number generation protocols

As mentioned earlier, existing research in JRNG is very limited. Besides using any secure addition protocol to obtain the sum from randomly generated values, which defeats the purpose of this work, the only works found in the literature that address JRNG are described by Erkin and Tsudik in [10] and by Kursawe, Danezis and Kohlweiss in [14]. Both are designed to provide random values for a group of at least three entities.

In this chapter we first give a brief overview of the protocols mentioned above. We then propose the use of random variables combined with the central limit theorem and show some of the practical and mathematical restrictions for such a protocol. Then we present two new protocols for JRNG: a share-based protocol and a scheme based on a single broadcast. For both protocols numerical examples are provided. We finalize by showing an application of the broadcast scheme for secure addition.
Table 1: Notation used throughout the thesis.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>large prime</td>
</tr>
<tr>
<td>$r_{i\to j}$</td>
<td>random value generated by $i$ sent to $j$</td>
</tr>
<tr>
<td>$g$</td>
<td>generator for $\mathbb{Z}_p^*$</td>
</tr>
<tr>
<td>$K$</td>
<td>number of entities participating in the protocol</td>
</tr>
<tr>
<td>$m$</td>
<td>bitlength of random numbers</td>
</tr>
<tr>
<td>$C_i$</td>
<td>carry-over from bit $i$ to bit $j$</td>
</tr>
<tr>
<td>$N$</td>
<td>publicly known target value</td>
</tr>
<tr>
<td>$R_i$</td>
<td>random number generated by entity $i$</td>
</tr>
<tr>
<td>$k_i$</td>
<td>temporary values used to construct $R_i$</td>
</tr>
<tr>
<td>$x_{ij}$</td>
<td>bit $j$ of random number $R_i$</td>
</tr>
<tr>
<td>$x_{Nj}$</td>
<td>bit $j$ of target value $N$</td>
</tr>
<tr>
<td>$y_i$</td>
<td>updated bit $i$ replacing $x_{ii}$ of $R_i$</td>
</tr>
<tr>
<td>$d_i$</td>
<td>scaled carry-over for key entity $i$</td>
</tr>
<tr>
<td>$D$</td>
<td>random variable distribution</td>
</tr>
<tr>
<td>$s_i(j)$</td>
<td>sign function</td>
</tr>
<tr>
<td>$G$</td>
<td>set of leaves requiring a secondary cohort</td>
</tr>
<tr>
<td>$n$</td>
<td>set of leaves requiring a secondary cohort</td>
</tr>
<tr>
<td>$l$</td>
<td>maximum number of entities that the adversary can corrupt</td>
</tr>
<tr>
<td>$k$</td>
<td>maximum of children a parent can have</td>
</tr>
<tr>
<td>$K$</td>
<td>number of entities in tree $T$</td>
</tr>
<tr>
<td>$T$</td>
<td>the tree induced by the star topology</td>
</tr>
<tr>
<td>$E_i$</td>
<td>number of leaves in branch $i$ [Do I need to mentioned this]</td>
</tr>
</tbody>
</table>

### 3.1 Related works

Initially all nodes pair-wise exchange two random values which they will use to compute their private values later, i.e. $r_{i\to j}$ and $r_{j\to i}$. Here $r_{i\to j}$ denotes the random value generated by $i$ and sent to $j$. Then they sum over the incoming and outgoing elements so that the values, once aggregated, will cancel out. Formally, the proposed scheme works as follows:

1. Each node $i$ generates $K - 1$ random numbers $r_{i\to j}$ for all $j \neq i$. Node $i$ then sends these values over the underlying (secure) network to their corresponding nodes.

2. Each node $i$ now calculates $R_i = \sum_{j=1, j \neq i}^{K} (r_{i\to j} - r_{j\to i})$

It is trivial to see that addition of all these elements yields zero. The original scheme also added a parameter $n$ (RSA modulus in the original work), such that the total added up to $nK$. Furthermore, as was also stated by the authors, it is sufficient to send the seed of the random values that were generated rather than random numbers themselves, thus reducing the amount of network traffic.

The protocol provided by Kursawe, Danezis and Kohlweiss in [14] is effectively the same scheme, but rather than sending messages to all entities, a subset of “leaders” is chosen, and values are only sent to leaders. This greatly reduces the amount of communication required. From now on, however, we refer to the work by Erkin and Tsudik to be the reference protocol. Additionally, Kursawe et al. proposed a JRNG scheme where joint values had a public product rather than sum. Since our broadcast scheme as proposed in Section 3.4 is very similar to this protocol we briefly introduce it here. For this we introduce the sign function:

$$s_i(j) = \begin{cases} 
1 & \text{if } i > j \\
-1 & \text{if } i < j
\end{cases}$$
Note that \( s_i(j) + s_j(i) = 0 \) for all \( i \neq j \). Now entities establish their joint random numbers \( R_i \) via the following protocol:

1. A public generator \( g \in \mathbb{Z}_p^* \) is agreed upon. Each entity generates a random number \( k_i \in \mathbb{Z}_p^* \) and calculates \( g^{k_i} \).
2. All entities publicly broadcast \( g^{k_i} \) to all other entities.
3. Each entity \( i \) calculates \( R_i = \prod_{j=1, j \neq i}^{K} (g^{k_j})^{s_i(j)k_i} = g^{\sum_{j=1, j \neq i}^{K} s_i(j)k_i} \).

Multiplying all these joint random numbers yields publicly known value \( 1 \) since
\[
\prod_{i=1}^{K} R_i = g^{\sum_{i=1}^{K} \sum_{j=1, j \neq i}^{K} s_i(j)k_i} = g^{\sum_{j=1}^{K} \sum_{i=1, j \neq i}^{K-1} (s_i(j) + s_j(i))k_ik_j} = g^0 = 1
\]

Although this protocol allows for multiplicative aggregation of these random values, Kursawe showed that retrieving the total sum of data elements using this protocol is not trivial, and requires a brute-force attack. In our broadcast protocol we present an adaptation such that the sum of the joint random values equals \( 0 \), rather than their product. Because this omits the need for brute-forcing, we will only discuss our additive protocol in the complexity analysis.

### 3.2 Random variable protocols

Our first attempt at generating joint random numbers is by using variables that are randomly drawn from a known distribution \( D \). We then use the central limit theorem, which states that the average of randomly distributed variables will converge to a normal distribution with decreasing deviation as the number of samples drawn from this distribution increases. The major advantage such a scheme has is that there is no way to predict an individual reading. Such a scheme works in the following fashion:

1. Each entity generates random value \( R_i \sim D \).
2. Each entity uses the consequent protocol to encrypt message using the secret (random) key \( R_i \).
3. The aggregator now decrypts using the public key \( pk = \frac{1}{K} \sum R_i \), where \( pk \) is normally distributed, i.e. \( pk \sim N(\text{mean}(D), \mathcal{O}(\frac{1}{\sqrt{K}})) \).

If used in practice and the total cannot be decrypted, the scheme should be repeated until a valid outcome is found. There are a number of problems such schemes however:

#### 3.2.1 Practical restrictions

In practice random schemes may not always suffice since in the worst case scenario the scheme may need to run several times before a valid outcome has been achieved. Especially in cases where communication is expensive this may not be desirable.

Also, a participant may unwillingly reveal additional information regarding their message if an adversary gets access to multiple encryptions of the same message encrypted with a different key, sampled from the same distribution \( D \). The same applies to the total of smaller subsets.
This is especially problematic since the total converges to a normal distribution with reducing standard deviation for a large number of readings, easily allowing for an exploit.

A random variable scheme also requires a large number of entities to guarantee a reasonably fast convergence, since the convergence is of $O\left(\frac{1}{\sqrt{K}}\right)$. This means the scheme cannot be applied for small groups.

### 3.2.2 Mathematical restrictions

Besides practical, these protocols also suffer from mathematical problems. The central limit theorem states that the average of these random values tends to a normal distribution with decreasing deviation. Since we usually do not want to calculate the average of the messages, we need some way to divide the cryptographic keys by $K$. This obviously cannot be done before aggregation since this is equivalent to encrypting with an negligible distortion. Thus any such division should be done by the aggregator before or after the aggregation without affecting the messages. We do not see how this can de done in such a way that any adversary cannot just copy this.

In all cases one can generally also exclude certain measurements to derive the total of a subset of participating entities. The adversary can simply test for combining subsets of the data while accounting for subset size until a value is found that falls within the expected range.

Because of these problems we considered random variables unsuitable for joint random number generation schemes.

### 3.3 Share-based protocol

We will now propose a new scheme which rather than communication among all $K$ entities, will only involve communication with a select group of $m$ entities, which we call the key entities. The goal of the scheme is to establish joint random numbers that add up to some target $0 \leq N \leq 2^m$, for some $m \in \mathbb{N}$. The protocol relies on using the binary representation from each random number and then fix one bit at a time. This will be done such that the total will match the binary representation of the target $N = \sum_{j=1}^{m} x_{Nj}2^{i-1}$, $x_{Nj} \in \{0, 1\}$, while locally correcting for the effects of carry-over, i.e. avoid impacting the next bit. Here $x_{Nj}$ denote the $j$’th bit of $N$. The carry-over will be denoted by $C_j \in \mathbb{Z}$, where we want to note that we are using scaled carry-over. Hence $01 + 01$ results in a carry-over of 2. For explaining the basic protocol we assume to be working modulo $2^m$. How to deal with cases where we want to generate numbers in other finite groups is considered in section 3.3.3. $y_i$ is the bit that we use to replace $x_{ij}$ for $1 \leq i \leq m$. Finally we use $\lfloor x \rfloor$ to denote the largest integer that is less than or equal to $x$. Then the protocol is as follows:

1. All $K$ entities generate random numbers $R_i = \sum_{j=1}^{m} x_{ij}2^{j-1}$, $x_{ij} \in \{0, 1\}$. We also assign numbers 1 to $m$ between the different entities.
2. Each entity $i$ sends $x_{ij}$ to key entity $j \pmod{K}$ for small groups) using the underlying secure network.
3. Key entity $j$ calculates $d_j = \sum_{i=1, i\neq j}^{K} x_{ij}$. If $d_j \equiv x_{Nj} \pmod{2}$, entity $j$ sets its $j$’th coefficient to $x_{jj} = 0$. Otherwise it updates its $j$’th coefficient to $x_{jj} = 1$. Finally key entity $j$ calculates $C_j = \lfloor (d_j + x_{jj})/2 \rfloor 2^{j}$ and updates its random number to $R_j \leftarrow R_j - C_j \pmod{2^m}$.

Since we work modulo $2^m$, we do not consider negative numbers. Furthermore, entities $1 \leq i \leq m$ now have updated (semi)-random number $R_i = y_i2^{i-1} + \sum_{j=1, j\neq i}^{m} x_{ij}2^{j-1} - C_i \pmod{2^m}$. Thus,
we have replaced \( x_{ii} \) by \( y_i \), and then dealt with the carry-over by subtracting it locally. Now we show that the scheme works when we have at least \( m \) entities \((K \geq m)\). We prove this in the following lemma, where we use the binary sum representation \( \sum_{j=1}^{K} R_i = \sum_{i=1}^{m} a_i 2^{i-1}, a_i \in \{0,1\} \) and \( C_i = \sum_{j=1}^{m} c_{ij} 2^{i-1}, c_{ij} \in \{0,1\} \) for all \( 1 \leq i \leq m \). We use \( x'_{ij} \) to denote binary coefficients for each \( R_i \) after the protocol run, and \( x_{ij} \) as the binary coefficients after step 1 in the protocol.

**Lemma 1.** For \( 1 \leq j \leq m-1 \) it holds that \( \left\lfloor \frac{d_j + y_j}{2} \right\rfloor 2^{i-1} = (d_j + y_j - x_{Nj}) 2^{i-1} \). It now follows that \( N = \sum_{i=1}^{K} \sum_{j=1}^{m} x'_{ij} \mod 2^m \).

**Proof.** The first formula holds since \( x_{Nj} = d_j + y_j \mod 2 \). The second statement follows since

\[
\sum_{i=1}^{K} \sum_{j=1}^{m} x'_{ij} 2^{i-1} = \sum_{i=1}^{m} \sum_{j=1}^{m} x'_{ij} 2^{i-1} + \sum_{i=m+1}^{K} \sum_{j=1}^{m} x'_{ij} 2^{i-1} \mod 2^m
\]

\[= \sum_{j=1}^{m} \left( \sum_{i=1}^{m} x_{ij} 2^{i-1} - C_j + (y_j - x_{jj}) 2^{i-1} \right) + \sum_{i=m+1}^{K} \sum_{j=1}^{m} x_{ij} 2^{i-1} \mod 2^m
\]

\[= \sum_{j=1}^{m} K \sum_{i=1}^{m} x_{ij} 2^{i-1} + \sum_{j=1}^{m} (y_j - x_{jj}) 2^{i-1} - \sum_{j=1}^{m} C_j \mod 2^m
\]

\[= \sum_{j=1}^{m} \left( y_j - x_{jj} + \sum_{i=1}^{K} x_{ij} \right) 2^{i-1} - \sum_{j=1}^{m} (d_j + y_j - x_{Nj}) 2^{i-1} \mod 2^m
\]

\[= \sum_{j=1}^{m} \left( y_j - x_{jj} + \sum_{i=1}^{K} x_{ij} \right) 2^{i-1} - \sum_{j=1}^{m} (d_j + y_j - x_{Nj}) 2^{i-1} \mod 2^m
\]

\[= \sum_{j=1}^{m} (y_j - x_{jj} + \sum_{i=1}^{K} x_{ij}) 2^{i-1} - \sum_{j=1}^{m} (d_j + y_j - x_{Nj}) 2^{i-1} \mod 2^m
\]

\[= \sum_{j=1}^{m} x_{Nj} 2^{i-1} \mod 2^m
\]

Here the second step is justified due to the replacement of bit \( x_{ii} \) by \( y_i \) in addition to the carry-over for key entities.

Just like [10] this requires few computational resources, since it relies merely on additive operations, while not suffering in the number of rounds. The proposed protocol does, however, drastically reduce the amount of communication required in comparison to the previous approaches for larger groups. The reduction in comparison to [14] is mainly in terms of bandwidth, i.e. the size of the messages sent. This is because instead of requiring entities to send large random numbers (depending on the subsequent application), all communication is done bit-wise while only requiring a minor increase in the number of messages. The key entities are also the only entities doing any calculations. A more extensive comparison between all the protocols is provided in Section 5.

Note that the scheme here is posed in a binary setting, but it can be used in any base-\( k \) setting for some \( k \in \mathbb{N}_{\geq 2} \).

### 3.3.1 Example

We will illustrate the protocol using the following example, with all values displayed in Table 2. First all entities choose a random number of 4 bits. For convenience we choose key entity \( i \) equal to entity \( i \) for \( 1 \leq i \leq 4 \). We assume the (public) target to be \( N = 1111 \). In the first round
all entities send their first bit to key entity 1. Thus he receives 0,0,0,1 not taking into account
his own first bit. Since the target bit equals one, this means entity 1 sets \( y_1 = 0 \). Since his first
bit is already 1, key entity 1 does not have to change his first bit. Also there is no carry-over,
thus \( C_1 = 0 \) and key entity 1 is done. Now entity 2 receives values 0,0,1,1. The target is 1, thus
\( y_2 = 1 \), and key entity 2 changes his second bit to 1. He then subtracts the resulting carry-over
\( C_2 = 4 \) from his number, thus his final number equals \( R_2 = 1010 - 0100 = 0110 \). Likewise do
dentity 3 and 4 change their third and fourth bit respectively, and after correcting for carry-over
they find their final numbers 1000 and 0110. Note that key entity 4 does not need to correct for
carry-over since this is always a multiple of 16.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Start</th>
<th>( d_i )</th>
<th>Change bit?</th>
<th>Carry-over</th>
<th>Final number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1100</td>
<td>1</td>
<td>no</td>
<td>0</td>
<td>1100</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>2</td>
<td>yes</td>
<td>4</td>
<td>0110</td>
</tr>
<tr>
<td>3</td>
<td>0100</td>
<td>3</td>
<td>yes</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1110</td>
<td>3</td>
<td>yes</td>
<td>-</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1111</td>
</tr>
</tbody>
</table>

3.3.2 Security

The security of the share-based protocol relies on every entity receiving at most partial infor-
mation regarding the values of the random numbers generated in addition to the assumption
that the underlying network is secure. To be more precise, if there are sufficient key entities,
any key entity only has access to one bit from every other entity. We want to stress, however,
that this scheme is prone to a maleficent coalition among the \( m \) key entities, since knowing \( l \)
bits from the secret random number of an entity reduces the number of possible values that this
number can attain, usually referred to as the entropy, of this number by a factor \( 2^l \). Thus if \( l \)
bits are compromised, any joint random number only contains \( 2^{m-l} \) bits of secret information.
Note that this scheme works for any key-length, but the key-length should at least be \( l \) bits
more than the minimum required entropy for the subsequent application of these numbers.

3.3.3 Finding elements in other finite groups

Thus far we have only shown how to provide joint random numbers module \( 2^m \), for some \( m \in \mathbb{N} \).
In general, however, we want to generate numbers in groups of an order other than \( 2^m \), say
some \( M \in \mathbb{Z} \). We claim that in this case the protocol is still applicable, albeit with some minor
adaptations. For this we assume that the entities are capable of generating elements modulo
\( M \). Let \( m \) be the bitlength of \( M \). Then we propose the following adaptation:

1. All \( K \) entities generate random numbers \( R_i = \sum_{j=1}^{m} x_{ij} 2^{j-1} \), \( x_{ij} \in \{0,1\} \), such that the \( R_i \)
are uniformly distributed over \( \mathbb{Z}_M \). We also assign numbers 1 to \( m \) between the different
entities.

2. Each entity \( i \) sends \( x_{ij} \) to key entity \( j \) (mod \( K \) for small groups) using the underlying
secure network.

3. Key entity \( j \) calculates \( d_j = \sum_{i=1}^{k} x_{ij} 2^{i-1} \mod M \). Key entity \( j \) updates its random
number to \( R_j \leftarrow R_j - d_j \).

4. Key entity \( m \) calculates \( d_m = \sum_{i=1}^{k} x_{im} 2^{m-1} \mod M \). Key entity \( m \) updates its random
number to \( R_m \leftarrow R_j + N - d_j \).
The correctness of this scheme follows since for each bit, we add a term such that the total yields 0 modulo $M$. Only for the final bit the total is corrected to $N$ modulo $M$ by key entity $m$. Thus $\sum_{i=1}^{K} R_i = N \mod M$.

Although in this scheme any bit from a key entity $i$ may change when updating its random number modulo $M$, we want to stress that this will not introduce a significant amount of additional randomness over the original scheme. This is because it will only affect key entities $i$ where $R_i - d_i < 0$. Since key sizes are generally large, this typically is not the case and thus the security of this extension is equal to that of the original scheme.

### 3.4 JRNG protocols based on single broadcast

In our second protocol, we once again assume a group of $K$ entities wanting to establish joint random numbers with a known sum. Rather than exchanging random numbers with other entities, we now propose to broadcast a single value to all other entities which they can use to form their joint random number. Here it should hold that $K \geq 3$. Addition of all these numbers will, when aggregated, result in all terms canceling out. All operations are performed in $\mathbb{Z}_p^*$, where $p = kq + 1$ is prime for some other large prime $q$. We assume there to exist a publicly known ordering so we can define the sign function $s_i$:

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

Now the proposed scheme works as follows:

1. A public generator $g \in \mathbb{Z}_p^*$ is agreed upon. Each entity generates a random number $k_i \in \mathbb{Z}_p^*$ and calculates $g^{k_i}$.

2. All entities publicly broadcast $g^{k_i}$ to all other entities.

3. Each entity $i$ calculates $R_i = \sum_{j=1,j \neq i}^{K} s_i(j) (g^{k_j})^{k_i} = \sum_{j=1,j \neq i}^{K} s_i(j) g^{k_i k_j}$.

The final joint random number for each entity is $R_i$ as obtained above. The correctness of this scheme follows since $s_i(j) + s_j(i) = 0$ for all $i \neq j$, and thus

$$\sum_{i=1}^{K} R_i = \sum_{i=1}^{K} \sum_{j=1,j \neq i}^{K} s_i(j) g^{k_i k_j} = \sum_{i=1}^{K} \sum_{j=1}^{K-1} (s_i(j) + s_j(i)) g^{k_i k_j} = 0.$$

We also want to acknowledge the fact that this protocol is not restricted to a fully connected network, but can also be used in a star-topology where all messages are relayed through a single semi-honest entity. This is possible since all messages can be publicly known.

#### 3.4.1 Example

Let us consider the prime $p = 107 = 2 \cdot 53 + 1$ with generator $g = 2$ for $\mathbb{Z}_p^*$. We assume 3 entities, Alice, Bob and Charlie, corresponding to 1,2 and 3, want to run the broadcast protocol. They respectively pick random numbers $r_1 = 34$, $r_2 = 28$ and $r_3 = 84$ and broadcast $2^{r_i}$. Now Alice calculates $-g^{r_1 r_2} = 80$ and $-g^{r_1 r_3} = 5$. Adding both these values results in her secret key $R_1 = 85$. Bob and Charlie do the same. Notice that in the end we have $R_1 + R_2 + R_3 = 0 \mod 107$ as required.
Table 3: Intermediate values corresponding to Alice, Bob and Charlie.

<table>
<thead>
<tr>
<th>Entity</th>
<th>$r_i$</th>
<th>$g^{r_i}$</th>
<th>$s_i(1)g^{r_1r_i}$</th>
<th>$s_i(2)g^{r_2r_i}$</th>
<th>$s_i(3)g^{r_3r_i}$</th>
<th>$R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice (1)</td>
<td>34</td>
<td>9</td>
<td>-</td>
<td>80</td>
<td>5</td>
<td>85</td>
</tr>
<tr>
<td>Bob (2)</td>
<td>28</td>
<td>62</td>
<td>27</td>
<td>-</td>
<td>24</td>
<td>51</td>
</tr>
<tr>
<td>Charlie (3)</td>
<td>84</td>
<td>39</td>
<td>102</td>
<td>83</td>
<td>-</td>
<td>78</td>
</tr>
</tbody>
</table>

$$R_1 = -g^{r_1r_2} - g^{r_1r_3} = 42$$

$$R_2 = g^{r_1r_2} - g^{r_2r_3} = 39$$

$$R_3 = g^{r_1r_3} + g^{r_2r_3} = 13$$

Figure 3: Graphical illustration, here the arrow denotes the sign of $s_i(j)$.

### 3.4.2 Security

The security of the algorithm is based on DDH. The scheme is secure against any passive adversaries if we assume at least 2 honest participants $i$ and $j$. This way no outsider can calculate $g^{k_ik_j}$ without knowledge of either $k_i$ or $k_j$ (and retrieving those values is infeasible by the discrete log assumption), and thus cannot retrieve any information regarding the keys from $i$ and $j$.

Since this protocol makes use of Diffie-Hellman key exchange, $p$ should be at least 2048 bits and $q$ should be 224 bits, as is recommended by the U.S. National Institute of Standards and Technology (NIST) in [4]. In order to reduce key size and computational complexity while providing the same level of security we note that this method can also be applied using elliptic curve cryptography. For details on elliptic curve cryptography we refer to Miller in [15].

Note that this schemes does not require a secure network since all messages are public. Since this scheme is based on Diffie-Hellman key exchange, we assume that any adversary is not capable of executing a man-in-the-middle attack. This can, for example, be avoided by requiring entities to authenticate themselves.

### 3.4.3 Ring topology

In case we may assume a ring topology, we can reduce the computational complexity of this scheme even further. For this we assume any entity $i$ can communicate with the $[l + 1]$ entities ranked below and above $i$, where $l$ once again is the maximum size of the adversarial coalition.

We refer to these entities as neighbours, where $N_i$ denotes the set of neighbours of $i$. Naturally this is seen modulo $K$, i.e. the lowest and highest ranked entities are also neighbours. Then the scheme works as follows.
1. A public generator \( g \in \mathbb{Z}_p^* \) is agreed upon. Each entity generates a random number \( k_i \in \mathbb{Z}_p^* \) and calculates \( g^{k_i} \).

2. All entities publicly broadcast \( g^{k_i} \) to all of its neighbours.

3. Each entity \( i \) calculates \( R_i = \sum_{j \in N_i} s_i(j)(g^{k_i})^{k_i} = \sum_{j \in N_i} s_i(j)g^{k_ik_j} \).

Correctness follows since \( i \in N_j \) implies that \( j \in N_i \), thus aggregation works in the same manner as in the original scheme. Security follows since \( l + 1 \) entities need to be corrupted before a single key is compromised. The advantage of this adaptation is that it reduces computational complexity to \( \mathcal{O}(l) \), which for large networks is much less than \( \mathcal{O}(K) \), which is the computational complexity of the original scheme as presented in Section 5. Since in general we do not assume such a structure however, we do not consider this in the complexity analysis.

3.5 Applications

As mentioned earlier the most obvious use of the these protocols lies MPC. In this setting a number of parties want to calculate the sum of their individual secrets \( c_i \), without revealing their individual input. We will now give an example of an encryption protocol based on the single broadcast scheme. This protocol is also being prepared for submission for International Workshop on Information Forensics and Security 2016 (WIFS 2016) [18]. Let \( p \) be a safe prime, and \( R_i \) be determined by the broadcast scheme with known sum 0 over \( \mathbb{Z}_p \). We now propose to encrypt value \( c_i \) via

\[
\epsilon_i(c_i) = c_i + R_i \mod p.
\]

Since no cryptographic operation other than addition are performed, this is usually referred to as masking the measurements. The scheme is statistically secure since the random values \( R_i \) used to mask are 2048 values, and in general far exceed the minimum requirements of 112 bits of security, as specified by NIST [4]. Informally speaking, statistical security means that given a cyphertext \( \epsilon(c) \) for some message \( c \), any algorithm cannot retrieve the message with a significantly higher probability than random guessing. One can now find the sum over all the measurements via

\[
\sum_{i=1}^{K} \epsilon_i(c_i) = \sum_{i=1}^{K} c_i + R_i = \sum_{i=1}^{K} c_i + \sum_{i=1}^{K} R_i = \sum_{i=1}^{K} c_i
\]

Thus the scheme is correct. Notice that the scheme is especially useful in a setting with expensive communication where the underlying network is not secure and/or multiple parties are corrupted.

4 Star topology extension

In this section we first introduce the basic protocol for extending a JRNG protocol \( P \) to a larger network. The high level idea is instead of applying the JRNG scheme to all nodes at the same time, we only require \( P \) to be run in small subgroups of a few entities in which the scheme can run efficiently. Then by the underlying homomorphic property we can add up the keys acquired in the different runs to obtain the final key valid for the entire network.

Initially we present the following scheme, where we assume all nodes to be distributed in some tree such that each parent has at most \( k \) children. here \( k + 1 \) is the maximum size for efficiently running the underlying JRNG protocol. All definitions and requirements are made exact in Section 4.2.

1. Each parent node runs local protocol run \( P \) with all of its children.
2. Each entity now combines the random numbers it established during any protocol runs it
took part in using \( P \)'s homomorphic property.

Since this scheme is basically several runs of the underlying protocol where keys obtained in
different runs are combined, the scheme holds. That is, if the protocol is executed properly all
keys will once again cancel out during aggregation.

There is, however, an attack possible that allows for the total of subsets to become compromised.
This is the main focus of the remainder of the chapter. We introduce 3 different ways of dealing
with this attack suitable for different networks, mainly dependent on the connectivity properties
of the network.

For clarification, we use the following notation, where local refers to the restriction to the subset.

**Definition 4.1.** With a cohort we will refer to any subset that performs a local run of the
underlying JRNG protocol \( P \).

### 4.1 Security concerns

The protocol as presented thus far suffers from two major security concerns. We explain both
problems using our broadcast scheme as local reference protocol for convenience. The first
problem revolves around the need for two honest nodes during each protocol. Since the protocol
only runs in cohorts, the secret value \( R_i \) may be compromised if \( i \) is a leaf node in cohort \( T \)
and the other entities that are in \( T \) are all compromised. The same goes for the lead node.
Intermediate nodes require twice as many corrupted nodes before its key becomes compromised,
since it takes part in 2 runs, and are thus assumed safe. The second security issue will be
explained in a binary tree setting, but also holds for the general case of base \( k \). This issue
occurs when a node \( h \), parent to two leaf nodes \( i \) and \( i + 1 \), becomes corrupted. This means
the adversary controlling \( h \) now has access to \( K = g^{hr_i + hr_{i+1}} \), as well as the encryption of the
plaintext from \( h \) using \( K \). This means that total of the encrypted value of \( h \) using \( K \) and the
normally encrypted values of entities \( i \) and \( i + 1 \) can be decrypted. Hence the total aggregated
value of \( h \), \( i \) and \( i + 1 \) is compromised. Since it knows \( h \), \( i + j \) is also compromised. Notice that
this problem also applies to nodes on higher levels, but this reveals less information, since this
node can only find the total of a larger subset. We will refer to this as a compromised subset or
subset attack.

### 4.2 Definitions and requirements

In order to properly explain the solutions that are presented to avoid compromised subsets, we
first introduce some definitions.

**Definition 4.2.** A parent node together with its children is called a primary cohort. With a
secondary cohort we mean any other subgroup running the underlying JRNG protocol \( P \).

**Definition 4.3.** A tree \( T \) is called base-\( k \) if each node has at most \( k \) children.

**Definition 4.4.** A tree is called complete if every leaf is located in the two bottom layers of the
tree.

For the remainder of this section we are using the following notion of security.

**Definition 4.5.** An extension scheme for JRNG in the semi-honest setting is called secure with
security parameter \( l \) if a passive adversary is allowed to corrupt at most \( l \) entities such that:

1. The secret joint random number of any individual node cannot be retrieved.
2. There are no compromised subsets.

For any solution to satisfy the first requirement the joint random number of any individual node requires a share of at least \( l + 1 \) other nodes, and the underlying protocol must also be sufficiently secure. In order to guarantee this we require the tree structure to satisfy the following requirements:

**Requirement 4.1.** Any tree satisfies property \([4.7.1]\) if it satisfies the following properties

1. Any cohort consists of at least \( \lceil \frac{l}{2} \rceil \) entities.
2. The underlying JRNG protocol \( P \) for each entity provides security against at least \( \lceil \frac{l}{2} \rceil \) entities in each cohort.

This means that in general a tree is base-\( k \) and must look somewhat like the following:

![Graphical representation of tree structure](image)

**Figure 4: Graphical representation of tree structure**

With the lead cohort we mean the cohort containing the highest node in the tree and its children. The second property from requirement \([4.7.1]\) depends on the underlying JRNG protocol. We want to acknowledge that the additive broadcast always satisfies this property if the first requirement holds. The share-based scheme satisfies the second requirement if there are at least \( \lceil \frac{l}{2} \rceil \) key entities. For the cohorts containing leaves there may be extra requirements, but those are explained with the corresponding solution again subset attacks.

It turns out it is much harder to avoid compromised subsets without inducing too many restrictions. In general, this property is satisfied for a cohort \( i \) if there are at least \( l + 1 \) entities that are not leaves, since the adversary cannot corrupt all those nodes by the assumption of security parameter \( l \). Otherwise, a different solution is necessary.

For solution 1 we assume there to be a complete tree. Solution 2 is viable for any tree without extreme number of users in a single branch. Since we want this work to be as general as possible, we do not go into further details on how to establish such a tree. Two solutions also use 2party protocol runs. These are defined as follows:

**Definition 4.6.** A 2-party protocol run is performing the underlying protocol \( P \) with only two parties.

Here we want to stress that the values obtained in this protocol are known to both participating parties, since the total is public. The values obtained are, however, thought random for any third party that does not have access to either of the random numbers obtained in the protocol run. As a final definition we introduce branches, which are important in order to guarantee the security of the protocol.

**Definition 4.7.** A child node \( a \in T \) from the lead cohort together with all of the nodes below it are called a branch.
With Requirement 4.1 this implies there must be at least \( l + 1 \) branches.

We also need an ordering such that \( s_i(j) \) is well-defined, in the same way as in the broadcast protocol. Finally,

### 4.3 Solutions

In order to avoid compromised subsets, we will provide different solutions depending on the properties of the underlying network. This means that (some of the) leaves have to perform additional protocol runs. This way there are no longer branches connected to the rest of the tree via a single node. We now rephrase the protocol to account for this additional operation.

1. Each primary cohort \( T \) runs local protocol run \( P \).
2. Each leaf node runs a local protocol run \( P \) with all of its secondary cohorts.
3. Each entity now combines the random numbers it established during any protocol runs it took part in using \( P \)'s homomorphic property.

We will now explain each solution, and briefly discuss the strengths and weaknesses of the given approach. A more in-depth discussion is provided in Section 5. We refer to any of the methods below as joining a secondary cohort for convenience, but with this we always imply the method corresponding to the chosen solution to avoid compromised subsets.

#### 4.3.1 Solution 1: Shifting leaves

We propose to have all cohorts required to take part in a secondary cohort to participate in a pairing to connect the different branches. We propose the \( \lceil \frac{l}{2} \rceil \) highest ranked entities engages in a 2-party protocol run with the respective \( \lceil \frac{l}{2} \rceil \) lowest ranked entities from the next subset using the global ranking. Note that for this reason we need some way to order subsets. This could for example by setting this equal to highest ranked entity at the start of the protocol. We recommend using a static ranking throughout the protocol run only to be updated when a primary cohort is added or removed. This way if the highest ranked entity leaves or malfunctions we can avoid several cohorts having to perform calculations. The final key will once again be calculated using \( P \)'s underlying homomorphic property.

“Shifting” the leaves in this manner ensures that at least \( 2\lceil \frac{l}{2} \rceil + 1 \geq l + 1 \) entities need to be corrupted before a subset attack becomes possible. Note that the +1 is due to the presence of a parent in each cohort which must also be compromised.

The advantage of this scheme is that there is no secret information other than any secret information used in \( P \), and it easily allows for adding or removing users. Assuming the numbering of the primary cohorts follows a spatial distribution, i.e. cohort \( i \) and cohort \( i + 1 \) are located close to one another, this also means that communication between the pairings should be relatively efficient, since your pair is always nearby. A minor downside is that nodes that are not leaves in cohorts that also contain leaves may be required to take part in a secondary cohort, where in the other solutions the leaves are the only entities having to perform such. This is also the reason we have chosen to restrict this solution to a complete tree. Alternatively, one can restrict any cohort containing leaves to not also have children that are not leaves. This also avoids certain nodes from having to perform too many computations, but we will not consider any such possibility in the remainder of this thesis.
4.3.2 Solution 2: Regrouping

We propose to put all the leaves requiring a secondary cohort in one set $G$ and run another instance of the broadcast protocol there. We then divide $G$ in subsets $H_i, i \in \mathbb{N}$ such that the protocol run once again becomes feasible and perform a protocol run within each subset. Here $H_i$ should be chosen in such a way that it contains participants from at least $l + 1$ different branches. Participants from different branches are necessary to avoid the problem presented in Figure 4.3.2. Here we assume the shown nodes to be the only nodes in secondary cohorts 1, 2 and 3. We find that the node denoted in red is the only corrupted entity in the tree.

![Diagram](image)

Figure 5: Example where primary cohorts are not connected to different branches with security parameter $l = 2$. Black lines refer to connections in the tree, and the corresponding secondary cohorts are denoted in blue. The red circle denotes the only corrupted entity.

Now we can clearly no longer have compromised subsets as obtained above, since a key from a leaf now has input from $l + 1$ different branches (including its own).

In general such a division may not always exist. Upon inducing certain restrictions however, we can guarantee the existence of secondary cohorts. For this lemma we order the branches according to size, such that branch 1 contains most of the leaves.

**Lemma 2.** Let $N$ be the number of entities in $G$. Let $E_i$ be the number of leaves in branch $i$, and $T$ the number of branches. If for an integer $n \geq N/k$ it holds that

$$
\sum_{i=1}^{T} E_i \geq (l - j + 1) \cdot n
$$

for $1 \leq j \leq l$, then there exists pairwise disjoint $H_1, H_2, ..., H_n \subset G$ with $\bigcup_{i=1}^{n} H_i = G$ such that each $H_i$:

- contains entities from at least $l + 1$ different branches.
- contains at most $k$ entities.

**Proof.** We now prove the lemma by constructing $H_i$ with the required properties. In order to do so we assign entities to subsets until all subsets satisfy property 1. Then we can distribute the remaining entities over the subsets at will. There are sufficient secondary cohorts such that every leaf can uniquely be assigned to a secondary cohort since $N \leq nk$, thus if we achieve property 1 the proof is finished.
We prove that there exist sufficient entities recursively. Since \( \sum_{i=l+1}^{T} E_i \geq n \), there exist sufficient entities to assign one entity not from the first \( l \) branches to each cohort. Furthermore \( \sum_{i=1}^{T} E_i \geq 2n \), and thus there are also sufficient entities to assign 2 entities not from the first \( l - 1 \) branches to each cohort. Since \( E_i \geq E_k \) for all \( k > l \), we conclude that we can assign two different entities not from the first \( l - 1 \) branches to each cohort.

In general, from knowing we can assign \( m \leq l \) different entities to each cohort, that \( \sum_{i=l+1}^{T} E_i \geq mn \) and that \( E_m \geq E_k \) for all \( k > m \) we can deduce that we can assign \( m + 1 \) different entities to each cohort. Thus we can assign \( l \) entities from different branches to each cohort. The biggest advantage of this scheme is that it also satisfies [4.3.1]. This means that this scheme just requires any primary cohort to consist of at least \( \lceil \frac{l}{2} \rceil \) entities. A downside is that a division satisfying the construction as specified above may not always exist.

4.3.3 Solution 3: Secret pairings

Alternatively, we suggest that each leaf secretly pairs with some other leaf from a different branch. The security of this solution relies on an adversary not knowing whether or not it has corrupted a subset. We do this by running a protocol run with only these two entities such that that the aggregated total yields the identity for the homomorphic property. A subset attack is now infeasible since an adversary node does not know which entities formed a pairing.

With respect to the other solutions pairing allows for more freedom in graph structure since a cohort can contain any number of leaves. The major downside is that leaves need some way to secretly/randomly contact a leaf from a different branch without the adversary finding out. This can for example be done by a trusted third party (TTP), or at least a semi-honest party that does not collude with any of the entities participating in the protocol. Note that the TTP does not actually need to participate in the protocol for JRNG, and is only contacted upon node addition or deletion. Thus such a TTP does not defeat the point of using JRNG schemes.

How to establish such secret pairings without a trusted third party will be left as an open problem.

4.4 Node addition and deletion

We now discuss how a node can enter or leave the protocol run. Since all of the previous solutions rely on leaves also having secondary cohorts, we provide different solutions for joining a primary cohort, joining a secondary cohort, leaving a primary cohort and finally for leaving a secondary cohort. For each solution for subset attacks explicit protocols are provided for all the different possibilities.

4.4.1 Join cohort

The most obvious way to add nodes is to simply queue them until the start of the next protocol run, since we anticipate the underlying protocol should be run regularly. For example, in the application presented in Section 3.5 the underlying broadcast protocol should be run once every measurement. In case these joint random numbers are used for a longer period of time and a new protocol run is expensive we present 3 options for node addition during the protocol run: 1 for addition in a full tree and 2 for addition in general trees.
Complete tree addition
This solution assumes a complete tree and is designed to maintain this structure. Let i be the entity wanting to join in the protocol run. It joins any primary cohort in the following order:

1. i joins as a child node in any primary cohort that isn’t full already. It will also join a secondary cohort.
2. i joins a full primary cohort from the lowest layer. This cohort splits itself and the part that is now without parent joins a leaf j from the second-lowest layer to form a new primary cohort, with j becoming its parent. All entities involved run a leave and join protocol for their secondary cohort if necessary.
3. i joins a full primary cohort from the lowest layer. This cohort will than split itself and the part that is now without parent joins a leaf j from the lowest layer to form a new primary cohort in a new layer, with j becoming its parent. All entities involved run a leave and join protocol for their secondary cohort if necessary.

In order to prove correctness we only need to show that the resulting tree is once again complete. This is guaranteed because we cannot create extra leaves in a level that is not the last or second-last. Thus any problems can only occur upon creating a new level. The only way to create a new level is if the node joins via step 3, but this can only happen if the second-lowest layer does not contain any leaves. The cohort that splits comes from the last layer, meaning the resulting tree can only have leaves in the final two layers.

Furthermore we chose to let i join a full cohort from the lowest layer

General tree addition
Suppose entity i wants to join a cohort in the protocol run.

Initially i will try to join any cohort H that is not yet full, i.e. consists of k entities or less. It joins as a child node, and an instance of the underlying JRNG protocol is perform in H.

If it is infeasible for i to join such a cohort, it may try to join a cohort H such that it becomes too full. In this case split H into T and T’, where T’ consists of \( \lfloor \frac{k}{2} \rfloor \) entities. Note that for this to be feasible, \( l + 1 < k \), i.e. we can only split cohorts if the resulting cohorts still satisfies property 1 requirement. T retains the old host of H and all nodes in T remove the shares from nodes in T’.

From this point on there exist several solutions. For the first two solutions we assume \( i \in T’ \). The third option is a special case of the first two, where we assume \( i \in T \).

1. T’ can now join any leaf j to form a new cohort, with j acting as its new host. j will transfer any secondary cohorts it took part in to i and T’ needs to establish a new key. Since T’ may end up in a new branch, some of the leaves may need to join a new secondary cohort.
2. T’ can now join any leaf j to form a new cohort, with i acting as its new host. j becomes a leaf in T’, which needs to establish a new key. Since T’ may end up in a new branch, some of the leaves may need to join a new secondary cohort.
3. i joins cohort T. T’ becomes a new cohort, with i acting as its new parent. T’ needs to establish a new key with i in the same manner as T.

The difference between the first two options is whether it is easier for j to replace its primary cohort or its secondary cohort, but there is no major difference in computational complexity. Although the third option requires less entities to rerun the protocol since it does not affect any other cohorts, it may create very long branches, which could negatively impact the performance of the protocol.
4.4.2 Node removal

Leaving can be due to several reasons. The most obvious occurrence of a leaving node is upon
node malfunction, i.e. the node is incapable to complete the protocol run. Other reasons might
be unplanned events that force a node to leave etc.

In general, if entity $i$ wants to leave the tree, we want to establish new random numbers where
necessary such that the sum over the remaining nodes becomes known, while minimizing the
number of users having to perform new protocol runs. This is important since otherwise we
might as well rerun the entire protocol. Removing nodes is reasonably straightforward. In case
$i$ is not a leaf node there are two options. $i$ leaves according to the following protocol:

1. $i$ is a leaf node in a cohort consisting of more than $l + 1$ nodes. In this case $i$ will leave
   its cohort, and each other node within this cohort removes the share of $i$ in its random
   number.

2. $i$ is a leaf node in at least one cohort consisting of $l$ nodes or less from the bottom layer.
   If possible this leaf is replaced with another leaf from the first case. Alternatively the
   remaining leaves needs to perform a join protocol run. The old parent should join a
   secondary cohort, since it is now a leaf.

3. $i$ is a leaf node in at least one cohort consisting of less than $k$ nodes not from the bottom
   layer. The remaining entities pick any leaf node from the bottom layer to replace $i$,
   preferably from the first case. $i$ is now treated as a leaving leaf from either of the first two
   cases.

4. $i$ is not a leaf node. Either of the following two options is chosen:
   • Replacement: Any leaf $j$ that is capable of replacing $i$ is chosen to replace $i$ (This
     can simply be the lowest or highest indexed entity). $j$ is now treated as a leaving leaf
     from either of the above cases.
   • Promotion: The children from $i$ choose a new parent $j$ amongst them to replace $i$
     (This can simply be the lowest or highest indexed entity). Now $j$ runs a leave protocol
     run.

If none of the above solutions are valid within the framework of the chosen solution against subset
attacks, e.g. when Lemma 2 shows no solution exists or there are no nodes that can physically
replace node $i$, the entire protocol should be rerun anyway. Such an event is generally rare,
although no proof will be given.

Notice that both replacement and promotion are applicable to all the solutions for compromised
subsets. Only in the full tree setting a problem might occur if the promoted entity is from
a cohort in the second lowest layer of minimal size. Thus this instance should be avoided or
replacement can be used instead.

4.5 Joining and leaving secondary cohorts

Since the protocol for joining secondary cohorts depends on the solution chosen to deal with
subset attacks, we provide a different protocol for each of them.

4.5.1 Shifting leaves

Upon leaving the protocol there will only be a problem if the leaving node was part of a pairing
or when its cohort $H$ is disbanded. The first case is solved just as above, and all pairings its
the cohort should be reevaluated. In the latter $H$ is simply removed and the previous and next cohort have to reestablish all the pairings with $H$.

4.5.2 Regrouping

Suppose entity $i$ wants to join a secondary cohort. If Lemma 2 still holds for some $n \in \mathbb{N}$, $i$ attempts to join any secondary cohorts in the following order.

1. $i$ joins any secondary cohort that contains entities from at least $l + 1$ different branches that is not yet full.
2. $i$ creates a new cohort and requests entities from other secondary cohorts such that all secondary cohorts remain with entities from at least $l + 1$ different branches (including this new cohort).
3. New secondary cohorts are assigned such that every cohort contains leaves from $l + 1$ different branches. This requires all entities to perform a new instance of $P$ in their new secondary cohort.

If Lemma 2 no longer holds, a different solution to compromised subset should be considered, or some of the entities from branches with many entities should move to branches with few.

If node $i$ wants to leave or malfunctions action should only be taken if there are entities from fewer than $l + 1$ branches remaining in cohort $H$. If possible, a node from a different secondary cohort replaces $i$ such that all cohorts end up with nodes from $l + 1$ different branches. Otherwise all nodes in $H$ leave $H$ and then rejoin via the protocol above.

4.5.3 Form pairs

Upon joining or leaving, the TTP is contacted and the TTP randomly assigns a new secret pairing to all parties that no longer share a pairing, and informs all affected parties of the changes.

5 Complexity analysis

We base our analysis on the number of operations that need to be performed by each individual entity. Since the broadcast scheme as well as [10] are symmetric, the number of operations are the same for all entities. The share-based protocol is asymmetric, but involves no computationally demanding operations (only addition). The number of messages is equal for all entities. With a message we mean a private, secure message between two entities, whereas broadcasts are (public) messages to at least 2 different entities. Finally we note that [10] as well as our share-based scheme require a secure network via which entities can sent their messages, while the broadcast scheme allows all messages to be public. All results are presented in Table 5.

During this analysis we do not take into account any additional communication required to determine the element $g$ in the single-broadcast schemes. It may be noted though that only key entities are required to be capable of receiving messages, while the other entities only need to send. Although not visible in the table, messages sent in the share-based scheme are much smaller, thus require less bandwidth, as only (encrypted) bits are sent over the network. Also, if a single entity is the key entity for multiple bits, these messages can be combined, reducing the number of required messages even further.
Table 4: Number of operations needed per entity

<table>
<thead>
<tr>
<th></th>
<th>Share-based</th>
<th>Broadcast scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponentiations</td>
<td>-</td>
<td>$K$</td>
</tr>
<tr>
<td>Broadcast messages</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>2-party messages</td>
<td>$2K - 2$</td>
<td>$m$</td>
</tr>
<tr>
<td>Rounds</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Secure network?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

All in all we conclude that whenever the JRNG scheme based on a single broadcast is computationally viable it should be applied over the other schemes. Depending on the size of $m$ and the efficiency of [10] either of the other solutions can be applied.

5.1 Extension

Comparison between the presented protocols in terms of security and the number entities that have to perform computations during addition and removal of nodes is given in Table 5. Here $l$ is the security parameter the protocols are designed for, and $k$ the maximum size for any cohort. $G$ is the set of leaves that requires a secondary cohort.

Table 5: Complexity analysis for secondary cohorts

<table>
<thead>
<tr>
<th></th>
<th>Shifting leaves</th>
<th>Regrouping</th>
<th>Secret pairing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum # corrupted users to allow for subset attacks</td>
<td>$l + 1$</td>
<td>$l + 1$</td>
<td>$K - 1$</td>
</tr>
<tr>
<td>Minimum # corrupted users to allow for individual key compromise</td>
<td>$l + 1$</td>
<td>$l + 1$</td>
<td>$l + 1$</td>
</tr>
<tr>
<td># entities having to (re)do computations upon joining secondary cohort</td>
<td>0 up to $2l$</td>
<td>$l + 1$ up to $#G$</td>
<td>1</td>
</tr>
<tr>
<td># entities having to (re)do computations upon leaving secondary cohort</td>
<td>0 up to $3l$</td>
<td>$l + 1$ up to $#G$</td>
<td>1 up to $2t$</td>
</tr>
<tr>
<td>Minimum # entities in primary cohorts containing leaves besides parent.</td>
<td>$l + 1$</td>
<td>$\lceil \frac{l}{2} \rceil$</td>
<td>$\lceil \frac{l}{2} \rceil$</td>
</tr>
</tbody>
</table>

In addition to just computational power however, we also want to acknowledge the structures that each solution best fits into. As mentioned earlier, shifting leaves is very efficient when subsequent cohorts are physically close to one another, while the other solutions demand that secondary cohorts bind entities between different cohorts that may suffer from inefficient communication. A downside of this method is that shifting leaves requires extra structure within the tree. By only allowing leaves in the bottom two layers we prevent entities that are not leaves from having to perform extra instances of the underlying protocol $P$.

The main advantage of regrouping over the other solutions is that the secondary cohorts also function via regular protocol runs. Since there need to be at least $l + 1$ entities in secondary cohorts, the secondary cohorts already make sure that no individual key can be compromised. This means that cohorts containing leaves can consist of $\lceil \frac{l}{2} \rceil$ entities just like the other cohorts. A downside is that entities are performing $P$ in groups of more than $l + 1$ entities in total. This implies that more computations are performed than strictly necessary. Additionally, this solution is only applicable if Lemma 2 holds, and managing the secondary cohorts can be very costly upon nodes leaving or joining in extreme cases, as is shown in Table 5.
Where the other solutions have restrictions on the tree structure, secret pairing does not, since its security relies on an adversary not knowing when it has compromised a subset. Also addition and deletion of nodes is easy. The main drawback is that a TTP is required to establish the secret pairings, and there needs to be a way for these entities to constantly renew their secret keys without any revealing the pairing.

6 Discussion and future work

Since research on this topic is minimal, we recommend more studies to be conducted. For example, solutions to malicious adversaries should be considered. Although in the current state an active adversary cannot retrieve any individual random numbers, he can alter the total of the random numbers by simply subtracting some value from its own. This way the adversary knows the new total but the rest does not. Also some form of verification should be given the single broadcast protocol to confirm from whom they received messages, to avoid problems when both Alice and Bob are broadcasting. If in this case Alice receives a value from Bob but Bob did not receive a value from Alice, the resulting random values no longer cancel out and once again the total is not known.

Additional research may also reveal how to secretly and randomly pair leaves in order to satisfy secret pairing. Any solution found should also accommodate for addition and deletion of nodes during the protocol run.

7 Conclusion

In this work we have presented two new methods for JRNG. The JRNG scheme based on a single broadcast is the best scheme whenever exponentiation is computationally feasible. If this is infeasible, either the share-based scheme or [10] should be used. The difference is that the share-based scheme uses less bandwidth due to it only sending bits, while the latter sends larger messages but does not suffer from any data leakage. In general the latter also requires slightly fewer messages to be sent since usually $m < 2^K$.

Additionally, an extension for the previous methods has been given to accommodate for larger networks. Here we introduced the problem of compromised subsets and have presented three solutions to it. Finally, we have shown several ways to add or remove users from the network while minimizing the number of entities that have to perform computations.

References


