# Replicated Additive Secret Sharing with the Optimized Number of Shares

Juanjuan Guo[1,2], Mengjie Shuai[1,2], Qiongxiao Wang[1,2], Wenyuan Li[1,2], and Jingqiang Lin[3(✉)]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100089, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100089, China
[3] School of Cyber Security, University of Science and Technology of China, Hefei 230027, Anhui, China
linjq@ustc.edu.cn

**Abstract.** Replicated additive secret sharing (RSS) schemes introduce the threshold for additive secret sharing, and are known for computational efficiency and flexibility. While the traditional RSS schemes usually have a huge storage overhead with each server holding multiple shares, recent variations have tried reducing storage overhead but at the expense of computational performance. In this work, we focus on optimizing the number of shares to reduce storage overhead without introducing excessive computational cost. First, we construct a 2-of-$n$ RSS (i.e., any two among $n$ servers could reconstruct the secret value), which generates secret shares incrementally so that the storage increases almost linearly with $n$, and achieves the optimal number of shares as we proved. Then, we extend 2-of-$n$ RSS to a general $t$-of-$n$ RSS. Moreover, the incrementally-generate mechanism makes our scheme support a server to join dynamically that refrain existing shares from being modified. Our empirical study across 60 servers supports that our scheme largely reduces the storage overhead while obtaining an efficient runtime. Storage efficiency shows an improvement of up to two orders of magnitude and online runtime is within microsecond scale in our experimental settings.

**Keywords:** Secret sharing · Replicated additive secret sharing · Secure multiparty computation · Data privacy

## 1 Introduction

Secret sharing (SS) [4,31] has been extensively studied since it is important not only for sensitive data storage but also as a fundamental building block for

This work was partially supported by National Key R&D Program of China under Grant No. 2020YFB1005800.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2023
Published by Springer Nature Switzerland AG 2023. All Rights Reserved
F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 371–389, 2023.
https://doi.org/10.1007/978-3-031-25538-0_20

multiparty computation (MPC) [34]. The basic idea of SS is to (1) split the secret value into shares, and (2) ensure that the value can be reconstructed only with the predefined threshold number of shares. In sensitive data storage, the shares are stored among a set of non-colluded servers, preventing the adversary from reconstructing the secret value. Further, in secure multiparty computation, these servers cooperatively process these shares to complete the computation over the secret values without reconstructing them.

Two split mechanisms have been proposed for secret sharing. One is Shamir's secret sharing scheme [31], which splits the secret value by setting it as the constant coefficient of a random polynomial. The other is additive secret sharing scheme [18], which splits the secret value with a number of random values, ensuring the summation of the shares equal to the secret value. Shamir's secret sharing scheme [31] requires heavy modular exponentiation (and multiplication) in data split and complicated process of degree reduction for computing multiplication of secret values, which results in rare adoption in the secure multiparty computation [5]. In contrast, the additive secret sharing scheme is more efficient, as it only needs modular addition for data split, and a lightweight algorithm (i.e., Beaver triples [2,3]) has been proposed to multiply the secret values.

Therefore, additive secret sharing is widely adopted in secure multiparty computation, e.g., privacy-preserving machine learning (PPML) [10,11,22,23,23–26,29,30,32] and threshold signature [9]. In PPML, the secret values (e.g., medical data, finance data) are split into shares, and processed (e.g., addition, multiplication, comparison) by integrating additive secret sharing with garbled circuit [27], oblivious transfer [33], and/or homomorphic encryption [7,12], to complete the training and/or inference in machine learning. In threshold signature [9,14–16,21], the private key is split with additive secret sharing scheme among a number of servers, and a threshold number of servers cooperatively process the shares with the to-be-signed message, to generate a valid signature of the message.

Replicated additive secret sharing (RSS) [18] is proposed to introduce the threshold in additive secret sharing, by replicating shares in multiple servers, i.e., one server maintains multiple shares of the secret value. For example, $t$-of-$n$ RSS only needs the shares from $t$ servers among the whole $n$ ones. ISN [18], the first RSS, designs an access structure to ensure any $t$ servers can obtain all the shares. However, ISN needs a large storage (i.e., $C_n^t$, the number of $t - sized$ subset of $n - sized$ set). CDI [13] and KCI [19] reduce the storage overhead, by converting Shamir's secret share into additive secret share. However, a large computational overhead is introduced during the conversions of shares.

Various PPML schemes attempt to integrate RSS for improving the flexibility. However, the effect is still limited due to the heavy computation or storage overhead of $t$-of-$n$ RSS when $t$ and $n$ is large. For example, AFL [1,17] adopts ISN and only supports 2-of-3 RSS, PPML schemes [10,24,26,29] only support 2-of-3 or 2-of-4 RSS.

In this paper, we propose a new RSS scheme for additive secret sharing scheme, which reduces the storage significantly, without introducing excessive

computational overhead. We first construct an optimal 2-of-$n$ RSS to split the value incrementally, i.e., first split the secret value in 2-of-2 setting, then 2-of-3 RSS until 2-of-$n$ RSS, which makes the storage increase almost linear with $n$, reduce the storage remarkably compared to ISN. Then, we extend 2-of-$n$ RSS to $t$-of-$n$ RSS, with a 3-of-$n$ RSS. Moreover, the proposed incrementally-generate mechanism supports a new server to join dynamically that refrain existing shares from being modified. We have implemented the propose RSS scheme, the evaluation demonstrates the significant decrease of storage and modest computation performance.

The main contributions are as follows.

- We design a new 2-of-$n$ RSS scheme and prove the optimal storage overhead theoretically.
- We extend the optimal 2-of-$n$ RSS to a general $t$-of-$n$ RSS, to support an arbitrary threshold. The analysis demonstrate that our scheme needs much less storage compared to ISN.
- Our scheme support the dynamical join of a new server in secret sharing, which generates the share while the shares remain unchanged.

## 2 Preliminaries

**Additive Secret Sharing.** Additive secret sharing(ASS) is a technique that splits the secret into shares that add up to the original secret. The additive secret sharing algorithm chooses $n$ strings ($s_1$, ..., $s_n$) uniformly at random subject to the requirement that $\sum_{i=1}^{n} s_i = s$ (this can be done by choosing $s_1$,...,$s_{n-1} \in \mathbb{Z}_{2^l}$ uniformly at random, and then setting $s_n = s - \sum_{i=1}^{n-1} s_i$. The reconstruction algorithm simply adds all the shares to reconstruct the secret. ASS guarantees that each share does not reveal any information about the secret. Traditional additive secret sharing is a full-threshold secret sharing that requires all parties (servers) cooperate to reconstruct the secret, and no single point of failure is allowed.

**Replicated Additive Secret Sharing.** Replicated additive secret sharing (RSS) [13,18,19] has an access structure $\Gamma$. An access structure is defined by qualified sets $Q \in \Gamma$, which are the sets of parties who are granted access, and the remaining sets of the parties are called unqualified sets. In the context of this work we only consider threshold structures in which any set of $t-1$ or fewer parties is not authorized to learn information about private values (i.e., they form unqualified sets), while any $t$ or more parties are able to jointly reconstruct the secret (and thus form qualified sets). RSS can be defined for any $n>t \geq 2$. To secret-share private $s$ using RSS, we treat $s$ as an element of a finite ring $\mathbb{Z}_{2^l}$ and additively split it into shares $s_T$ such that $s=\sum_{T\in\tau} s_T$ (in $\mathbb{Z}_{2^l}$), where $\tau$ consists of all maximal unqualified set of $\Gamma$ (i.e., all sets of $t-1$ parties in our case). Then each party $i \in [1,n]$ stores shares $s_T$ for all $T \in \tau$ subject to $i \notin T$.

In $t$-of-$n$ replicated additive secret sharing, the secret is shared among $n$ parties and the secret can be reconstructed when $t$ parties cooperate.

## 3 2-of-$n$ Replicated Additive Secret Sharing

Replicated additive secret sharing (RSS) with a small threshold has become popular over the last few years [15,24,26]. We design a 2-of-$n$ RSS scheme, which is proved to achieve the minimum number of shares in 2-of-$n$ RSS.

**The System Model.** In our secret sharing scheme there is one dealer and $n$ parties (servers). The dealer generates shares of the secret and distributes them to $n$ parties, only when $t$ parties cooperate will the secret be reconstructed. In our 2-of-$n$ RSS, $t$ is 2. We denote $n$ parties who receive shares after share generation as (share holder) parties, and $t$ parties who reconstruct the secret corporately as reconstruct parties.

**The Adversary Model.** We assume that an adversary in our scheme can be divided into two kinds. An eavesdropping adversary learns all the information stored at the corrupted parties. A halting adversary may also disturb corrupted parties to stop sending messages during the execution of the protocol. Our scheme can tolerate up to $t-1$ eavesdropping parties and up to $n-t$ halting parties. We assume that the dealer is a trusted party.

### 3.1 2-of-$n$ Replicated Share Generation

Our 2-of-$n$ replicated additive secret sharing generates shares incrementally by adding parties one by one in turn. In our 2-of-$n$ RSS, multiple share sets will be generated, and each share set contains two shares whose sum is equal to the secret $s$. We denote share as $s_a^b$, the subscript $a$ indicates the order of share set that this share belongs to, and the superscript $b$ indicates the order of a share in its share set. The value of $b$ is 1 or 2, and the value of $a$ is related to the number of parties $n$. The party whose id is $i$ is denoted as $P_i$.

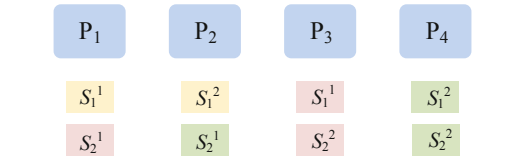



**Fig. 1.** 2-of-4 replicated share generation.

In our 2-of-$n$ replicated share generation, the dealer generates shares beginning with 2-of-2 additive secret sharing, then extends it to arbitrary 2-of-$n$ replicated share generation. We take 2-of-4 replicated share generation as an example, which shows in Fig. 1.

1) In 2-of-2 additive secret sharing, the dealer samples two additive shares $s_1^1$, $s_1^2$ in $\mathbb{Z}_{2^l}$ satisfying $s_1^1 + s_1^2 = s$, and sends $s_1^1$ to $P_1$, $s_1^2$ to $P_2$.

2) The 2-of-3 replicated share generation is based on 2-of-2 additive secret sharing, the dealer sends $P_1$'s share $s_1^1$ to $P_3$. Now $P_2$ can reconstruct the secret $s$ with either $P_1$ or $P_3$, but $P_1$ and $P_3$ cannot reconstruct the secret $s$ because they hold the same share. Thus we consider $P_1$ as the conflict party of $P_3$, or $P_1$ and $P_3$ are conflict parties. It is necessary for the dealer to generate the second share set $s_2^1$, $s_2^2$, and send $s_2^1$ to $P_1$, $s_2^2$ to $P_3$. So $P_1$ and $P_3$ can reconstruct the secret $s$ using $s_2^1$ and $s_2^2$.
3) The 2-of-4 replicated share generation is based on 2-of-3 replicated share generation. The dealer sends $P_2$'s share $s_1^2$ to $P_4$, now $P_4$ can reconstruct $s$ with either $P_1$ or $P_3$, but cannot reconstruct $s$ with $P_2$ because they hold the same share of the first share set. It is necessary for the dealer to send $s_2^1$ to $P_2$, $s_2^2$ to $P_4$ to enable $P_2$, $P_4$ reconstruct the secret.

In summary, $P_1$, $P_2$ hold $\{s_1^1\}$, $\{s_1^2\}$ respectively in 2-of-2 additive secret sharing; $P_1$, $P_2$, $P_3$ hold $\{s_1^1, s_2^1\}$, $\{s_1^2\}$, $\{s_1^1, s_2^2\}$ respectively in 2-of-3 RSS; $P_1$, $P_2$, $P_3$, $P_4$ hold $\{s_1^1, s_2^1\}$, $\{s_1^2, s_2^1\}$, $\{s_1^1, s_2^2\}$, $\{s_1^2, s_2^2\}$ respectively in 2-of-4 RSS.

Similar to 2-of-4 RSS, We design a 2-of-$n$ RSS incrementally through adding parties one by one. Since our goal is to make parties possess the minimum number of shares, and the dealer always distributes a new set of shares, each to the new party and its conflict party, we choose one of the parties holding the minimum number of shares as $P_{conflict}$, whose shares will be copied to the new party by the dealer. According to our rules, we find the conflict id of $P_i$ ($i \geq 3$) is $conflict = (i - 2^{(\lceil log_2 i \rceil - 1)})$ in 2-of-$n$ RSS. It can be seen that $P_{conflict}$ holds the minimum number of shares at present. For example, the conflict parties of $P_3$, $P_4$ are $P_1$, $P_2$ respectively, and the conflict parties of $P_5$, $P_6$, $P_7$, $P_8$ are $P_1$, $P_2$, $P_3$, $P_4$ respectively. The dealer will totally generate three share sets if $n$ ranges from 5 to 8, four share sets if $n$ ranges from 9 to 16, ..., according to the rules, we find that the dealer will generate $\lceil log_2 n \rceil$ share sets when the number of parties is $n$ where $\lceil \bullet \rceil$ represents ceiling function in math.

**Algorithm 1.** Shr_2ofn: 2-of-$n$ replicated share generation

**Input:** The number of parties $n$.
**Output:** Shares for all parties.

1: Generate $\lceil log_2 n \rceil$ share sets $(s_1^1, s_1^2)$, $(s_2^1, s_2^2)$,...,$(s_{\lceil log_2 n \rceil}^1, s_{\lceil log_2 n \rceil}^2)$.
2: Set an array $rawshare_1$ to store the first share of share sets $s_1^1$, $s_2^1$, ...,$s_{\lceil log_2 n \rceil}^1$, and set an array $rawshare_2$ to store the second share of share sets $s_1^2$, $s_2^2$, ...,$s_{\lceil log_2 n \rceil}^2$.
3: Set an array $vshare$ containing $n$ sub-arrays, and each corresponds to a party.
4: Add $rawshare_1[1]$ to $vshare[1]$, add $rawshare_2[1]$ to $vshare[2]$.
5: **for** $3 \leq i \leq n$ **do**
6: Compute the conflict id of $P_i$ as $conflict = i - 2^{\lceil log_2 i \rceil - 1}$.
7: Copy $vshare[conflict]$ to $vshare[i]$.
8: Add $rawshare_1[\lceil log_2 i \rceil]$ into $vshare[conflict]$ and add $rawshare_2[\lceil log_2 i \rceil]$ into $vshare[i]$.
9: **end for**
10: Output $vshare[i]$ to share holder $P_i$, where $1 \leq i \leq n$.

We describe 2-of-$n$ replicated share generation Shr_2ofn executed by the dealer in Algorithm 1. In order to reduce communication rounds, the dealer sends shares in one round after generating all shares for $n$ parties. Each party receives shares that have been arranged in ascending order of share sets. Specifically, $P_i$ holds at most $\lceil log_2 n \rceil$ shares in $vshare[i]$. These shares, in turn, belong to the first share set, the second share set, ..., the $\lceil log_2 n \rceil$th share set, so the index of a share in $vshare[i]$ represents the share sets to which it belongs.

**Proposition 1.** *In 2-of-n RSS that splits the secret into multiple sets with each set containing 2 shares, the number of shares of the party holding the most shares in our scheme reaches the minimum.*

*Proof.* Let $Min(n)$ denotes the minimum number of shares of the party holding the most shares, we prove by mathematical induction and contradiction that $Min(n)$ is $\lceil log_2 n \rceil$ in 2-of-$n$ RSS.

Base case: Show that the statement holds for the smallest $n$ in 2-of-$n$ secret sharing: $n = 2$, $Min(2) = \lceil log_2 2 \rceil = 1$ is clearly true because the dealer has to generate one share set in this case.

Inductive step: Show that for any $k > 2$, if $Min(k) = \lceil log_2 k \rceil$ holds, then $Min(k+1) = \lceil log_2(k+1) \rceil$ also holds. We consider two cases:

1) $k$ satisfies $\lceil log_2(k+1) \rceil = \lceil log_2 k \rceil$. According to our share generation algorithm, $P_{k+1}$ will get all shares of $P_{conflict}$, then the dealer distributes one share of the $\lceil log_2(k+1) \rceil$th share sets to $P_{conflict}$, and distributes the other share to $P_{k+1}$, now these two conflict parties hold $\lceil log_2(k+1) \rceil$ shares. Min(k) is $\lceil log_2 k \rceil$, it means that the rest parties except two conflict parties still hold $\lceil log_2 k \rceil$ shares. Because $\lceil log_2(k+1) \rceil = \lceil log_2 k \rceil$, all parties hold $\lceil log_2 k \rceil$ shares at most. As $\lceil log_2 k \rceil$ is the minimum for $k$ parties, it is obviously for $(k+1)$ parties. Thus, $Min(k+1) = \lceil log_2(k+1) \rceil$ holds.
2) $k$ satisfies $\lceil log_2(k+1) \rceil = \lceil log_2 k \rceil + 1$. We prove this by contradiction. We first suppose that $\lceil log_2 k \rceil$ shares are enough for $k+1$ parties. In this case, $\lceil log_2 k \rceil = log_2 k$. The secret can be reconstructed only if the shares held by any two parties are not all the same. Each party holds $log_2 k$ shares and $log_2 k$ share sets each contains 2 shares in our method, so at most $2^{log_2 k} = k$ share combinations can be generated according to combination theorem. When the dealer distributes $k$ combinations of shares to $k+1$ parties, there will definitely be two parties holding the same combination of shares, making it impossible for them to reconstruct the secret, which violates our suppose. Therefore, a new set of shares needs to be generated, $Min(k+1) = \lceil log_2 k \rceil + 1 = \lceil log_2(k+1) \rceil$ holds.

### 3.2 2-of-$n$ Replicated Secret Reconstruction

In our 2-of-$n$ replicated additive secret sharing, each party holds at most $\lceil log_2 n \rceil$ secret shares. Before reconstructing the secret, two reconstruct parties exchange their id $id_1, id_2$ that $id_2 > id_1$. Each reconstruct party determines which share

will be used to reconstruct the secret according to the id of the other party. Finally, the secret can be reconstructed by adding the two selected shares.

Since our 2-of-$n$ share generation is an incremental scheme, we design the secret reconstruction algorithm following the inverse idea of share generation described in Subsect. 3.1. The conflict id of the larger party $P_{id_2}$ can be computed as $conflict = id_2 - 2^{(\lceil log_2(id_2) \rceil - 1)}$.

1) If $conflict = id_1$, it means that shares of $P_{id_1}$ were copied to $P_{id_2}$ during share generation phase, so $P_{id_1}$ and $P_{id_2}$ can reconstruct the secret using their first different share, which is $\lceil log_2(id_2) \rceil$th share they hold.
2) If $conflict \neq id_1$, according to our share generation algorithm, the dealer copies the shares of $P_{conflict}$ to $P_{id_2}$, so $P_{id_2}$ can use these copied shares to reconstruct the secret with any other party except $P_{conflict}$. Now that $id_1 \neq conflict$, $P_{id_2}$ will only use the shares copied from $P_{conflict}$ to reconstruct the secret. Therefore, $P_{id_2}$ can be regarded as $P_{conflict}$ in this case. Now the secret reconstruction between $P_{id_1}$ and $P_{id_2}$ has transformed into the secret reconstruction between $P_{id_1}$ and $P_{conflict}$, so $P_{id_2}$ rolls its id $id_2$ back to $conflict$. Then each reconstruct party finds the larger id between $id_1$ and $conflict$, and computes the conflict id of it.

Executing the above operations for two reconstruct ids in a loop which stops until two ids are in conflict or the larger id is 2. If the larger id is 2, the smaller id must be 1 so that two reconstruct parties choose the first share to reconstruct the secret.

The algorithm Rec_2ofn executed by each reconstruct party to select the correct share is described in Algorithm 2, where $recparty$ is an array containing reconstruct id in ascending order, each reconstruct party $P_i (i \in \{id_1, id_2\})$ executes the algorithm, and obtain the correct share that will be used to reconstruct the secret.

**Algorithm 2.** Rec_2ofn: 2-of-$n$ replicated secret reconstruction

**Input:** $recparty$.
**Output:** The share that used to reconstruct the secret.

```
1: while True do
2:   if recparty[2] == 2 then
3:     output vshare[i][1].
4:   else
5:     conflict=recparty[2]-2^(⌈log2(recparty[2])⌉−1)
6:     if conflict == recparty[1] then
7:       output vshare[i][⌈log2(recparty[2])⌉]
8:     else
9:       Replace recparty[2] with conflict in recparty, and sort ids in recparty.
10:    end if
11:  end if
12: end while
```

## 4 *t*-of-*n* Replicated Additive Secret Sharing

The threshold of RSS introduced in Sect. 3 is limited to 2, however, there are also some scenarios that requires larger threshold [16,28]. We extend the threshold to an arbitrary number, achieving $t$-of-$n$ RSS.

### 4.1 *t*-of-*n* Replicated Share Generation

In $t$-of-$n$ replicated share generation, the dealer generates secret shares in an incremental way, similar to 2-of-$n$ RSS. It starts from $t$-of-$t$ additive secret sharing, then to $t$-of-$(t+1)$ RSS, ... , $t$-of-$n$ RSS.

**t-of-t Additive Secret Sharing.** The dealer begins with $t$-of-$t$ additive secret sharing, it generates shares $s_1^1, s_1^2, ..., s_1^t$ in $\mathbb{Z}_{2^l}$, satisfying $s_1^1+s_1^2+...+s_1^t=s$. Then allocates $s_1^1$ to $P_1$, $s_1^2$ to $P_2$, ... , $s_1^t$ to $P_t$ so that the $t$ parties can reconstruct the secret $s$.

**t-of-(t+1) Replicated Share Generation.** The dealer generates $t$-of-$(t+1)$ RSS based on $t$-of-$t$ additive secret sharing above.

The dealer allocates the share $s_1^1$ of $P_1$ to $P_{t+1}$, we consider $P_1$ as the conflict party of $P_{t+1}$, or $P_1$, $P_{t+1}$ are conflict parties. Now $t$ parties $\{P_1, P_2, ..., P_t\}$ or $\{P_2, P_3, ..., P_{t+1}\}$ can reconstruct the secret $s$ using $s_1^1$, $s_1^2$, ..., $s_1^t$. However, any $t$ parties including both $P_1$ and $P_{t+1}$ cannot reconstruct the secret $s$ because they hold the same share $s_1^1$ at present.

To solve this, the dealer will generate and allocate shares to enable any $t$ parties including both $P_1$ and $P_{t+1}$ to reconstruct the secret $s$. The dealer samples two shares $s_2^1$, $s_2^2$ in $\mathbb{Z}_{2^l}$, then allocates $s_2^1$ to $P_1$ and allocates $s_2^2$ to $P_{t+1}$. Next, the dealer computes $\delta_1 = s - s_2^1 - s_2^2$. The new secret $\delta_1$ needs to be shared among the rest $t-1$ parties, achieving any $t-2$ parties from the rest parties can reconstruct the new secret $\delta_1$, and further any $t$ parties among $\{P_1, P_2, ..., P_{t+1}\}$ can reconstruct the secret $s$. In other words, when reconstructing the secret $s$, $P_1$, $P_{t+1}$ uses $s_2^1$, $s_2^2$ respectively, and the rest parties use their share which can reconstruct $\delta_1$.

When we generate shares for $(t-2)$-of-$(t-1)$ RSS of the new secret $\delta_1$, it is a sub-problem of generating shares for $t$-of-$(t+1)$ RSS of the original secret $s$. Thus, we design a recursive algorithm to solve sub-problem $(t-2)$-of-$(t-1)$ RSS, in which sub-problem $(t-4)$-of-$(t-3)$ RSS will be called, ..., until the recursion reaches its bound 2-of-3 or 3-of-4 determined by whether $t$ is even or odd respectively since $t$ reduces by 2 in each recursion. The 2-of-$n$ replicated share generation algorithm has been given in Sect. 3. The 3-of-$n$ replicated share generation algorithm is similar to 2-of-$n$ replicated share generation, which we will give at the end of this section.

***t*-of-*n* Replicated Share Generation.** Our scheme adds $P_{t+2}$ based on $t$-of-$(t+1)$ share generation described above, then adds $P_{t+3}$, ..., $P_n$ similarly to achieve $t$-of-$n$ RSS. When $P_i$ $(i \geq t+1)$ is added, the dealer copies the shares of its conflict party $P_{i-t}$ to $P_i$, then samples two shares $s_*^1$ and $s_*^2$ and sends each of them to $P_{i-t}$, $P_i$ respectively. Next, the dealer computes $\delta = s - s_*^1 - s_*^2$, and

generates shares for $(t-2)$-of-$(i-2)$ RSS of the new secret $\delta$, it is a sub-problem of generating shares for $t$-of-$i$ RSS of the original secret $s$. Thus, we design a recursive algorithm to solve sub-problem $(t-2)$-of-$(i-2)$ RSS, in which sub-problem $(t-4)$-of-$(i-4)$ RSS will be called, ..., until the recursion reaches its bound that the threshold of secret sharing to be called is 2 or 3.

**Algorithm 3.** $\mathcal{F}_{maxlevel}(t, n)$

```
if t==2 then
    return maxlevel = ⌈log2 n⌉
else if t==3 then
    return maxlevel = n − 2
else if t==n then
    return maxlevel = 1
else then
    return maxlevel = F_maxlevel(t, n − 1) + F_maxlevel(t − 2, n − 2)
endif
```

In our $t$-of-$n$ replicated additive secret sharing, the number of share sets generated by the dealer can be calculated using $t$ and $n$ as described in Algorithm 3. The cases that $t$ is 2 and $t$ is 3 have proved before. When considering the $n$th party $P_n$, the dealer allocates shares of $P_{n-t}$ to $P_n$ on the basis of $t$-of-$(n-1)$ share generation, and executes $(t-2)$-of-$(n-2)$ RSS for the other $n-2$ parties excluding $P_{n-t}$ and $P_n$. Therefore, the total number of share sets of $t$-of-$n$ secret sharing is $\mathcal{F}_{maxlevel}(t, n-1) + \mathcal{F}_{maxlevel}(t-2, n-2)$.

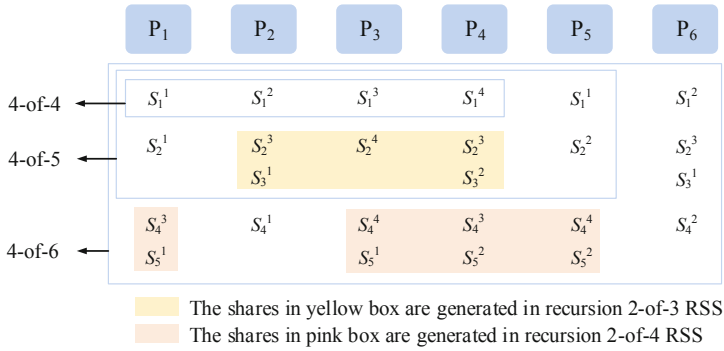



**Fig. 2.** 4-of-6 replicated share generation.

Figure 2 shows an example of 4-of-6 replicated share generation. We describe our $t$-of-$n$ share generation Shr_tofn in Algorithm 4 and define the following parameters:

- 2-tuple(*sharelevel*, *sharevalue*): It describes a share, *sharelevel* is the order of the share set to which it belongs, and *sharevalue* is its value.
- *vparty*: an array containing ids of all parties (share holders).

- *vshare*: a two-dimensional array that records the 2-tuples of $n$ parties. After the share generation algorithm ends, $vshare[i]$ contains all 2-tuples of $P_i$. *vshare* is initialized as an empty two-dimensional array, and because it will be modified in every recursion, it is set as a global variable.
- *maxlevel*: an integer indicating maximum order of share sets that have been generated so far, *maxlevel* is initialized to 1. It will be modified in every recursion, so we set it as a global variable.
- *genlevel*: an integer indicating the *sharelevel* of share that is generated during the present recursion, it is initialized to 1. As our share generation is incremental, the *sharelevel* of new shares must not be the same as the shares that have already been generated, thus *genlevel* is always set to $maxlevel+1$.

The algorithm Shr_tofn takes threshold $t$, secret $s$, *genlevel* and *vparty* as input. It has no output, because all shares that are generated will be added into array *vshare*.

**Algorithm 4.** Shr_tofn($t$, $s$, *genlevel*, *vparty*)

**Input:** threshold $t$, secret $s$, *genlevel*, *vparty*.

1: **if** $t$==2 **then**
2: Shr_2ofn($s$, *genlevel*, *recurparty*)
3: **else if** $t$=3 **then**
4: Shr_3ofn($s$, *genlevel*, *recurparty*)
5: **else**
6: Sample shares $s_1, s_2, ..., s_t \xleftarrow{\$} \mathbb{Z}_{2^l}$ such that $\sum_{i=1}^{t} s_i = s$.
7: Set $sharelevel = genlevel$, add 2-tuple($sharelevel$, $s_i$) to $vshare[vparty[i]]$ where $1 \leq i \leq t$.
8: **for** $i = t+1; i \leq$ sizeof($vparty$); $i++$ **do**
9: Traverse 2-tuples in $vshare[vparty[i-t]]$ to find $vshare[vparty[i-t]][ord]$ such that $vshare[vparty[i-t]][ord].sharelevel = genlevel$.
10: Copy the shares from $ord$th to the last in $vshare[vparty[i-t]]$ to $vshare[vparty[i]]$.
11: **if** $genlevel > maxlevel$ **then**
12: $maxlevel = genlevel$.
13: **end if**
14: Sample $s'_1, s'_2$ in $\xleftarrow{\$} \mathbb{Z}_{2^l}$, add 2-tuple($maxlevel+1, s'_1$) into $vshare[vparty[i-t]]$, add 2-tuple($maxlevel+1, s'_2$) into $vshare[vparty[i]]$.
15: Add $vparty[m]$ into an empty array *recurparty* where $1 \leq m < i$ and $m! = i-t$.
16: Shr_tofn($t-2$, $s - s'_1 - s'_2$, $maxlevel$+1, *recurparty*)
17: **end for**
18: **end if**

The 3-of-$n$ replicated share generation Shr_3ofn called in Shr_tofn is also incrementally generated, similar to 2-of-$n$ share generation. The dealer begins with 3-of-3 additive secret sharing, sends $s_1^1$, $s_1^2$, $s_1^3$ to $P_1$, $P_2$, $P_3$ respectively. When adding $P_i$ ($i \geq 4$), the dealer generates a new set of shares $s_2^1, s_2^2, s_2^3$, and sends $s_2^1$ to $P_{i-3}$, $s_2^2$ to $P_i$, $s_2^3$ to $P_j$ where $1 \leq j < i$ and $j \neq i-3$. The dealer needs

to generate $n-2$ sets of shares in total in our 3-of-$n$ replicated share generation. Considering the parameter compatibility when Shr_3ofn and Shr_2ofn are called in Shr_tofn, it is necessary to introduce parameters *sharelevel*, *genlevel*, *vparty*, etc. based on 3-of-$n$ and 2-of-$n$ replicated share generation we described before.

### 4.2 $t$-of-$n$ Replicated Secret Reconstruction

In $t$-of-$n$ replicated additive secret sharing, each party holds multiple shares. The $t$ reconstruct parties exchange their ids, and each party selects a correct share according to $t$ reconstruct ids, then the secret can be reconstructed by computing the sum of $t$ shares that have been selected.

Our $t$-of-$n$ secret reconstruction follows the reverse strategy of share generation. Now that share generation is in an incremental way, reconstruction will start from maximum reconstruct id to minimum. After exchanging id with other reconstruct parties, each party holds the ids of all reconstruct parties as $\{id_1, id_2, ..., id_t\}$ in ascending order. The correct share selection starts from maximum id $id_t$, and computes the conflict id of $id_t$ as $conflict = id_t - t$.

1) If $P_{conflict}$ is also a reconstruct party, it means that the dealer copied shares of $P_{conflict}$ to $P_{id_t}$ during share generation. Therefore, $P_{conflict}$ and $P_{id_t}$ shall choose the first different share of them to reconstruct the secret. As their first different share is generated on the basis of $t$-of-$(id_t - 1)$ RSS, the *sharelevel* of which is $\mathcal{F}_{maxlevel}(t, id_t - 1) + 1$. The rest parties execute the secret reconstruction algorithm in a recursion, and with a threshold $t - 2$ after two conflict parties are removed.
2) If $P_{conflict}$ is not a reconstruct party, shares of $P_{id_t}$ that are generated after the conflict with $P_{conflict}$ will not be used. Instead, $P_{id_t}$ will choose one from shares that are copied from $P_{conflict}$. Therefore, $P_{id_t}$ can be regarded as $P_{conflict}$ in this case, similar to 2-of-$n$ secret reconstruction. Thus, $P_{id_t}$ rolls its id back to $conflict$. Then reconstruct parties sort ids of refreshed reconstruct parties in ascending order.

If no parties are in conflict, each reconstruct party chooses the first share in the present recursion to reconstruct the secret.

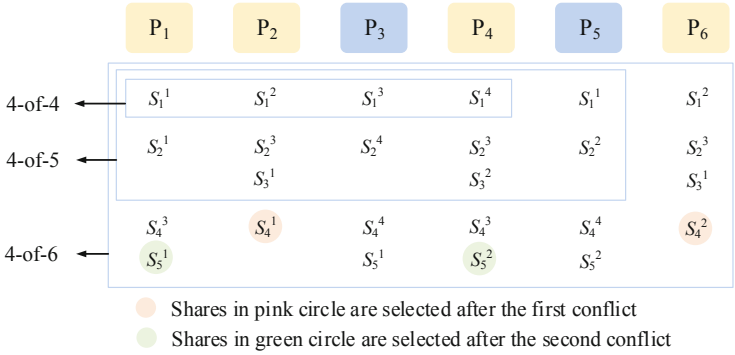



**Fig. 3.** 4-of-6 replicated secret reconstruction.

Figure 3 shows an example of 4-of-6 replicated secret reconstruction that the reconstruct parties are $\{P_1, P_2, P_4, P_6\}$, The conflict party of maximum party $P_6$ is $P_2$, so they choose their first different share $s_4^1$ and $s_4^2$. The rest reconstruct parties $\{P_1, P_4\}$ call 2-of-4 secret reconstruction, where share holders are $\{P_1, P_3, P_4, P_5\}$, as $P_2$ and $P_4$ are excluded. In recursion 2-of-4 called by 4-of-6 secret reconstruction, the conflict party of maximum party $P_4$ is $P_1$, so they choose their first different share $s_5^1$ and $s_5^2$. Because $s_5^1 + s_5^2 = s - s_4^1 - s_4^2$ holds during share generation, it is obviously that four shares selected can reconstruct the secret $s$.

**Algorithm 5.** Rec_tofn($vparty$, $recparty$, $reclevel$, $recid$, $vshare[recid]$)

**Input:** $vparty$, $recparty$, $reclevel$, $recid$, $vshare[recid]$.
**Output:** The share that used to reconstruct the secret .

Set threshold $t$=sizeof($recparty$)
**if** $t$==2 **then**
  Rec_2ofn($vparty$, $recparty$, $reclevel$, $recid$, $vshare[recid]$)
**else if** $t$==3 **then**
  Rec_3ofn($vparty$, $recparty$, $reclevel$, $recid$, $vshare[recid]$)
**else**
  **while** True **do**
    **if** $recparty[t] == party[t]$ **then**
      Output 2-tuple.$sharevalue$ if 2-tuple.$sharelevel = reclevel$.
    **else**
      Generate an array $ind$ containing $t$ parameters indicating the indexes of $t$ reconstruct parties in $vparty$, compute $conflict = vparty[ind[t] - t]$.
      **if** $conflict$ **in** $recparty$ **then**
        Set $reclevel$=$\mathcal{F}_{maxlevel}(t, recparty[t] - 1)$+1.
        **if** $recid == conflict$ or $recid == recparty[t]$ **then**
          Output 2-tuple.$sharevalue$ if 2-tuple.$sharelevel = reclevel$.
        **else**
          Remove id $recparty[t]$ and $conflict$ from $vparty$ and $recparty$.
          Rec_tofn($vparty$, $recparty$, $reclevel$, $recid$, $vshare[recid]$)
        **end if**
      **else**
        Replace id $recparty[t]$ with id $conflict$ in $recparty$.
        Sort ids of $recparty$ in ascending order, and refresh $ind$.
        If $recid = recparty[t]$, set $rectid = conflict$.
      **end if**
    **end if**
  **end while**
**end if**

More specifically, we describe our $t$-of-$n$ secret reconstruction Rec_tofn in Algorithm 5, which is executed by each reconstruct party independently. We define the following parameters:

– $vparty$: an array containing ids of all parties (share holders) in ascending order.

- *recparty*: an array containing reconstruct party ids in ascending order.
- *ind*: an array containing indexes of $t$ reconstruct parties in *vparty*. Conflict id of maximum reconstruct party $recparty[t]$ is $vparty[ind[t] - t]$.
- *reclevel*: an integer, when two reconstruct parties are in conflict, they will choose shares whose *sharelevel* is equal to *reclevel* to reconstruct the secret. Furthermore, suppose $id_1$ is conflict with $id_2$, they will choose their first different share which is generated after $t$-of-$(id_2 - 1)$ RSS. Since maximum *sharelevel* of $t$-of-$(id_2 - 1)$ RSS is $\mathcal{F}_{maxlevel}(t, id_2 - 1)$, *reclevel* is set to $\mathcal{F}_{maxlevel}(t, id_2 - 1) + 1$. *reclevel* is initialized to 1.
- *recid*: an id of reconstruct party who executes secret reconstruction algorithm. $P_{recid}$ holds $vshare[recid]$ as its own shares, which is the input of secret reconstruction algorithm.

The 3-of-$n$ replicated secret reconstruction Rec_3ofn called in Rec_tofn follows inverse intuition of Shr_3ofn, which is similar to 2-of-$n$ RSS. Three reconstruct ids are $id_1$, $id_2$, $id_3$ where $id_1 < id_2 < id_3$. Firstly reconstruct parties computing conflict id of maximum party $P_{id_3}$ as $conflict = id_3 - 3$. If $conflict$ is equal to $id_1$ or $id_2$, reconstruct parties use the $(id_3 - 2)$th share to reconstruct the secret. If $conflict$ is not equal to $id_1$ or $id_2$, replace $id_3$ with id $conflict$ and rearrange three reconstruct ids in ascending order, and find new maximum id of them. Then execute the above operations for three reconstruct ids in a loop and stop until any two of three reconstruct parties are in conflict or maximum id is 3, then reconstruct parties choose the first share if maximum id is 3. Considering the parameter compatibility when Rec_3ofn and Rec_2ofn are called in Rec_tofn, it is necessary to introduce parameters *sharelevel*, *reclevel*, *vparty*, etc. based on 3-of-$n$ and 2-of-$n$ secret reconstruction we described before.

**Security Analysis.** Our $t$-of-$n$ RSS is a semi-honest scheme, privacy will be violated if less than $t$ parties can reconstruct the secret. During share generation, dealer generates multiple independent sets of shares, and each set contains $t$ shares whose sum is the secret $s$. Since each party only holds at most one share in the same set in our scheme, less than $t$ parties cannot reconstruct the secret, thus privacy is guaranteed.

## 5 Related Work

Existing solutions have some inherent flaws, such as huge storage overhead, poor computational overhead and fixed little threshold.

**Storage Inefficient RSS.** ISN [18] realizes $t$-of-$n$ replicated additive secret sharing with each party holding $C_{n-1}^{t-1}$ shares, which causes huge storage overhead. CT12 [8] presents a scheme for the evolving 2-threshold access structure in which the share size of party $t$ is linear in $t$. KNY16 [20] designs a secret sharing for the evolving 2-threshold access structure and an $l$-bit secret in which the share size of party $t$ is $logt + (l + 1)loglogt + 4l + 1$ bits. The storage overheads of CT12 and KNY16 are unbalanced, and the share size held by the party with a larger id is much larger than that of the party with a smaller id.

**Computational Inefficient RSS.** CDI [13] and KCI [19] achieve threshold additive secret sharing by converting Shamir secret share into additive secret share. The share conversion process needs to calculate Lagrange interpolation, which brings huge computational overhead. DKLs18 [15] and DKLs19 [16] are two threshold schemes that convert Shamir secret shares of private key into additive secret shares, and then combine additive secret sharing and oblivious transfer to accomplish threshold signature.

**RSS with a Fixed Threshold.** AFL [1,17] follows the share generating method of ISN to design a 2-of-3 MPC protocol, then TYAO [17] adapts AFL to a fully malicious case. Besides, ABY3 [26] and Sharemind [6] are 2-of-3 MPC schemes based on 2-of-3 replicated secret sharing. In these 2-of-3 MPC schemes based on replicated additive secret sharing, each party holds 2 shares. There are also some schemes [10,24,26,29] applied in PPML that only realize RSS with a fixed small threshold $t$ and a fixed number of parties $n$, such as 2-of-3 or 2-of-4 RSS.

From the perspective of extendibility, some existing replicated additive secret sharing schemes [10,11,18,26,29] split the secret into one share set, and each party holds multiple shares of the set. PrivPy [24] is sightly different from the previous schemes, which splits the secret into multiple sets of shares, and each party holds one share of each share set.

## 6 Evaluation

We evaluate the storage and computational overhead of our scheme. Our storage efficiency is up to two orders of magnitude faster than the best prior work, and the online runtime of the protocol is in the microsecond level. We test replicated share generation (offline) executed by the dealer, and replicated secret reconstruction (online) executed by reconstruct party. In order to measure performance, we implement a prototype in C++. We run single-threaded simulations on Ubuntu 18.04 with an Intel Xeon(R) Silver 4216 CPU 2.10 GHz, with 32 GB of RAM.

### 6.1 Storage Cost

**Table 1.** Comparision of threshold and storage.

| Algorithm | Threshold | Storage(bits) |
|---|---|---|
| ABY3 [26],etc | 2-of-3 | $2l$ |
| PrivPy [24] | 2-of-4 | $2l$ |
| ISN [18] | $t$-of-$n$ | $C_{n-1}^{t-1} \cdot l$ |
| Our Shr_2ofn | 2-of-$n$ | $\lceil log_2 n \rceil \cdot l$ |
| Our Shr_3ofn | 3-of-$n$ | $(n-2) \cdot l$ |
| Our Shr_tofn | t-of-$n$ | $\leq \mathcal{F}_{maxlevel}(t,n) \cdot (l+\sigma)$ |

We compare the storage overhead of our Shr_2ofn, Shr_3ofn, Shr_tofn with existing schemes in Table 1, where the length of *sharevalue*, *sharelevel* is $l$, $\sigma$ bits respectively. The existing schemes [1,6,10,17,24,26,29] realize 2-of-3, 2-of-4 RSS by having each party hold 2 shares. Our Shr_2ofn (Shr_3ofn) replicated share generation expands the number of parties to an arbitrary $n$ by having each party hold $\lceil log_2 n \rceil$ shares ($(n-2)$ shares). Our scheme obtains the same storage cost as existing schemes when $n$ is 3 or 4.

In our replicated share generation Shr_tofn, the number of shares held by distinct parties is different. The maximum number of shares held by a party is $\mathcal{F}_{maxlevel}(t,n)$, which is described in Algorithm 3. Each party not only stores the shares, but also stores their corresponding *sharelevel*. Therefore, the party who holds the most shares needs to store $\mathcal{F}_{maxlevel}(t,n) \cdot (l+\sigma)$ bits. In order to compare the storage overhead with ISN intuitively, we set $l$ as 64 like most schemes (i.e., ABY3, PrivPy), and set $\sigma$ as 32, which is much larger than the size of the maximum *sharelevel* in most cases (i.e., the length of *sharelevel* of 4-of-60 or 5-of-60 is only 10 or 12 respectively). We test the cases when $t$ is 4 and $n$ ranges from 10 to 50, and depict storage comparison in Fig. 4. The result shows our storage efficiency is up to two orders of magnitude faster than ISN. When we increase $t$, the storage gap between ISN and our Shr_tofn will be larger.

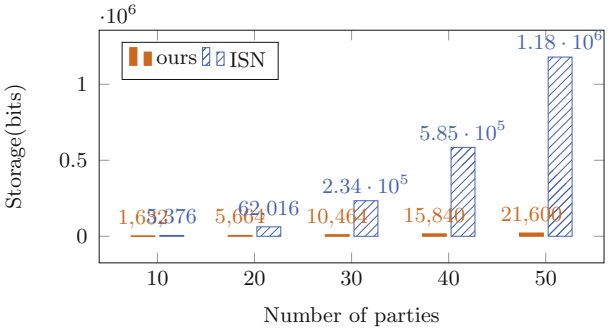


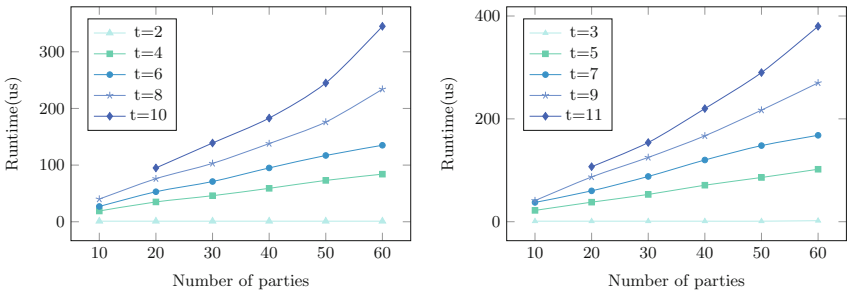

**Fig. 4.** Storage comparison between our Shr_tofn and ISN.

### 6.2 Computational Cost

**Table 2.** Runtime of share generation executed by the dealer(ms).

| | $n = 10$ | $n = 20$ | $n = 30$ | $n = 40$ | $n = 50$ | $n = 60$ |
|---|---|---|---|---|---|---|
| $t = 2$ | 0.03 | 0.04 | 0.06 | 0.07 | 0.08 | 0.09 |
| $t = 3$ | 0.03 | 0.08 | 0.12 | 0.21 | 0.28 | 0.36 |
| $t = 4$ | 0.06 | 0.21 | 0.46 | 0.82 | 1.17 | 1.63 |
| $t = 5$ | 0.06 | 0.42 | 1.24 | 2.60 | 4.50 | 7.60 |
| ... | | | | | | |
| $t = 9$ | 0.02 | 4.31 | 77.19 | 694.21 | $3 \times 10^3$ | $16 \times 10^3$ |
| $t = 10$ | 0.01 | 8.04 | 535.02 | $11 \times 10^3$ | $103 \times 10^3$ | $103 \times 10^3$ |

Runtime of replicated share generation is shown in Table 2. Only millisecond generation time is required when $t$ is 2 or 3, while runtime explodes when both $t$ and $n$ are large, for example, runtime of 10-of-60 share generation is 10 min. In fact, other schemes [13,18] also incur huge overhead when $t$ and $n$ are large.




(a) Secret reconstruction (t is even) (b) Secret reconstruction (t is odd)

**Fig. 5.** Runtime of secret reconstruction executed by reconstruct party.

Runtime of replicated secret reconstruction is plotted in Fig. 5. We test runtime of selecting the correct share, excluding exchanging ids among reconstruct parties and adding shares to reconstruct the secret. Runtime of exchanging ids depends on network performance and adding shares is fast so it can be ignored. We test the runtime in two cases: Rec_tofn will call Rec_2ofn if $t$ is even, and call Rec_3ofn if $t$ is odd. Runtime of distinct reconstruct subsets are different, so we test a large number of distinct subsets and finally take the average. It only takes 1 or 2 microsecond (us) for each party to select the correct share when $t$ is 2 or 3, and only 400 us selection time is required in 10-of-60 secret reconstruction.

## 7 Conclusion

In this paper, we first propose a 2-of-$n$ RSS scheme including a share generation algorithm, which generates shares incrementally and a secret reconstruction algorithm, which follows inverse intuition of share generation. Our 2-of-$n$ scheme is theoretically proved to achieve the optimal number of shares in 2-of-$n$ RSS. Then we design a more general $t$-of-$n$ RSS based on our 2-of-$n$ RSS, which supports an arbitrary threshold. It achieves a far better storage performance than existing schemes by making each share capable of reconstructing the secret in more subsets of parties. Since our RSS scheme generates shares in an incremental way, it realizes a extendibility that existing shares should not be changed when a new party joins in RSS. However, our RSS scheme can only compute addition operations in MPC, not multiplication operations in MPC. In the future, we will combine our RSS scheme with oblivious transfer to realize threshold multiplication operation in MPC, which can be directly applied in practical scenarios such as privacy-preserving machine learning and threshold signature.

## References

1. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 805–817 (2016)
2. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the eighth annual ACM Symposium on Principles of Distributed Computing, pp. 201–209 (1989)
3. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
4. Blakley, G.R.: Safeguarding cryptographic keys. In: Managing Requirements Knowledge, International Workshop on, pp. 313–313. IEEE Computer Society (1979)
5. Blakley, G.R., Meadows, C.: Security of ramp schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 242–268. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_20
6. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: a framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88313-5_13
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theor. (TOCT) **6**(3), 1–36 (2014)
8. Cachin, C.: On-line secret sharing. In: Boyd, C. (ed.) Cryptography and Coding 1995. LNCS, vol. 1025, pp. 190–198. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60693-9_22
9. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold EC-DSA. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 266–296. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_10

10. Chaudhari, H., Choudhury, A., Patra, A., Suresh, A.: Astra: high throughput 3pc over rings with application to secure prediction. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, pp. 81–92 (2019)
11. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: efficient 4pc framework for privacy preserving machine learning. arXiv preprint arXiv:1912.02631 (2019)
12. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
13. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_19
14. Dalskov, A., Orlandi, C., Keller, M., Shrishak, K., Shulman, H.: Securing DNSSEC keys via threshold ECDSA from generic MPC. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 654–673. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_32
15. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 980–997. IEEE (2018)
16. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ECDSA from ECDSA assumptions: the multiparty case. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1051–1066. IEEE (2019)
17. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 225–255. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_8
18. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electr. Commun. Japan (Part III: Fundamental Electronic Science) **72**(9), 56–64 (1989)
19. Kikuchi, R., Chida, K., Ikarashi, D., Ogata, W., Hamada, K., Takahashi, K.: Secret sharing with share-conversion: Achieving small share-size and extendibility to multiparty computation. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **98**(1), 213–222 (2015)
20. Komargodski, I., Naor, M., Yogev, E.: How to share a secret, infinitely. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 485–514. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_19
21. Kondi, Y., Magri, B., Orlandi, C., Shlomovits, O.: Refresh when you wake up: proactive threshold wallets with offline devices. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 608–625. IEEE (2021)
22. Koti, N., Pancholi, M., Patra, A., Suresh, A.: {SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 2651–2668 (2021)
23. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: secure tensorflow inference. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 336–353. IEEE (2020)
24. Li, Y., Xu, W.: Privpy: general and scalable privacy-preserving data mining. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1299–1307 (2019)

25. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 2505–2522 (2020)
26. Mohassel, P., Rindal, P.: Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 35–52 (2018)
27. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: the garbled circuit approach. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 591–602 (2015)
28. Ohara, K., Watanabe, Y., Iwamoto, M., Ohta, K.: Multi-party computation for modular exponentiation based on replicated secret sharing. IEICE Trans. Fund. Electron. Commun. Comput. Sci. **102**(9), 1079–1090 (2019)
29. Patra, A., Suresh, A.: Blaze: blazing fast privacy-preserving machine learning. arXiv preprint arXiv:2005.09042 (2020)
30. Poddar, R., Kalra, S., Yanai, A., Deng, R., Popa, R.A., Hellerstein, J.M.: Senate: a {Maliciously-Secure}{MPC} platform for collaborative analytics. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 2129–2146 (2021)
31. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
32. Wagh, S., Gupta, D., Chandran, N.: Securenn: efficient and private neural network training. Cryptology ePrint Archive (2018)
33. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: fast extension for correlated OT with small communication. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1607–1626 (2020)
34. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), pp. 162–167. IEEE (1986)