SC-DT: Scalable Constant Round Secure Comparison and its Application to Privacy Decision Tree Evaluation

Qinghui Zhang[1,2,3], Xiaojun Chen[1,2,3*], Yansong Zhang[1,2,3], Xudong Chen[1,2]

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
[2]State Key Laboratory of Cyberspace Security Defense, Beijing, China.
[3]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.

*Corresponding author(s). E-mail(s): chenxiaojun@iie.ac.cn;
Contributing authors: zhangqinghui@iie.ac.cn; zhangyansong@iie.ac.cn; chenxudong@iie.ac.cn;

Abstract

Numerous private decision tree evaluation (PDTE) protocols based on secure multi-party computation (MPC) have been proposed to protect sensitive data during evaluation. However, existing MPC-based PDTE protocols primarily focus on the two or three-party setting. Moreover, their core building block, secure threshold comparison, typically incurs logarithmic-round communication and dominates the online cost of tree evaluation. These limitations motivate the design of scalable and efficient secure comparison protocols for large-scale PDTE. In this paper, inspired by Falcon (PETs'2020), we propose a scalable constant round secure comparison protocol based on Shamir secret sharing in the honest-majority setting. Concretely, we leverage random shuffle to achieve zero detection with constant-round communication. Furthermore, we reduce random shuffle to random shift, thereby significantly decreasing the offline communication overhead. Besides, we reformulate feature selection and path selection in PDTE as private lookup table functionalities and integrate PLUT with our scalable comparison protocol to achieve scalable PDTE. Finally, we extend the above Shamir

secret sharing-based protocols to the packed secret sharing variants and further optimize the online communication efficiency of these packed variants. We instantiate the above protocols as a framework SC-DT and report their improved performance: i) For secure comparison, our packed secure comparison achieves a speedup of $\mathbf{1.7-2.3}\times$ and reduce communication by $\mathbf{1.7-2.5}\times$ in the online phase compared with Helix (Cryptology ePrint'2025). ii) For large-scale PDTE, our packed secret sharing-based protocol improves online communication efficiency by up to $\mathbf{1.9}\times$ over our Shamir-based protocol. In a 21-party WAN setting, it evaluates a 50-tree random forest with 42,550 nodes at an amortized latency of about one second per tree.

Keywords: secure multi-party computation, secure comparison, private lookup table, packed secret sharing, private decision tree evaluation

1 Introduction

Among various machine learning models, decision trees stand out for their efficiency and interpretability, leading to their extensive adoption in fields such as medical diagnosis [1] and financial analysis [2]. However, in applications including medical diagnosis, both the input feature vectors and the decision tree (DT) model may contain private information, which raises significant privacy concerns that require careful consideration. Secure multi-party computation (MPC) offers a promising solution to this challenge by enabling a set of n parties to jointly compute a function over their private inputs while preserving both input privacy and output correctness [3–6]. The integration of MPC with decision trees facilitates privacy-preserving collaborative model evaluation, establishing a paradigm known as private decision tree evaluation (PDTE).

PDTE process mainly relies on three steps, including feature selection, threshold comparison and path selection. The process starts at the root. At each node, a feature value is selected and then compared to a threshold; the comparison result determines whether to go the left path or the right path. The process will be repeated until reaching a leaf node. The value of a leaf node is the evaluation result of PDTE. In the context of PDTE, there have been significant research efforts to enhance evaluation

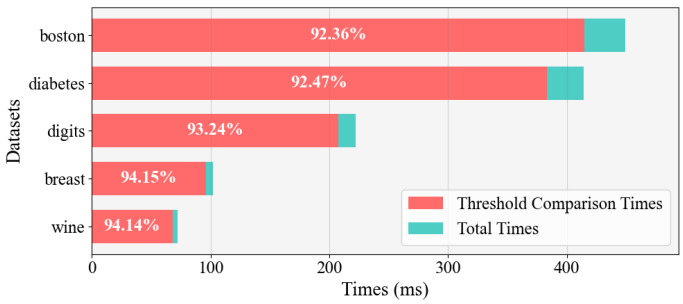


Fig. 1 Mostree's running time on different datasets in LAN. The percentage reflects the ratio of the threshold comparison latency to the total latency.

efficiency. These research efforts can be broadly classified into three categories based on their underlying cryptographic primitives. *The first category* leverages homomorphic encryption (HE) [7, 8] to realize non-interactive or low-interaction protocols. *The second category* turns to additive secret sharing in the 2-party computation (2PC) setting. For example, Ma et al. [9] paired garbled circuits (GC) with oblivious transfer (OT), while others [10, 11] optimized feature selection and path selection by combining one-hot vectors with vector inner-products. *The third category* considers replicated secret sharing in the 3PC setting. Ji et al. [12] and Cheng et al. [13] further leverage Function Secret Sharing (FSS) [14] to achieve constant-round PDTE. Subsequent works [15, 16] reduce communication costs by introducing oblivious selection and shuffling for feature selection and path selection.

Nevertheless, despite relying on different cryptographic primitives, all the aforementioned works suffer from scalability limitations when the number of participating entities becomes large. As a result, they face significant performance bottlenecks in applications such as industrial automation [17] and intelligent logistics systems. This motivates the development of scalable PDTE protocols that can efficiently support large-scale real-world deployments. Moreover, by rerunning Mostree [15] on a single machine and in 3PC local-area network (LAN) environment, we find that threshold comparison accounts for a substantial fraction of the total PDTE runtime (Figure 1). This overhead becomes even more pronounced in WAN settings. Consequently, efficient and scalable secure comparison is a key ingredient

for building scalable PDTE protocols. Early work by Octavian et al. [18] propose a constant-round and scalable secure comparison protocol based on Shamir secret sharing over finite fields. However, their construction requires a large gap between the secret domain and the share domain, which leads to substantial online communication overhead. Subsequent constant-round comparison protocols [19–21] have primarily been designed for the 2PC and 3PC settings, limiting their scalability. In contrast, scalable comparison protocols [22–24] can handle the general nPC setting, but typically require a logarithmic number of communication rounds. More recently, Yang et al. [25] and Feng et al. [26] both achieved constant-round comparison in the nPC setting. However, their constructions rely on the presence of a helper party, thereby requiring a stronger trust assumption. This reliance not only complicates deployment in adversarial environments but also hinders the extension of these protocols to general n-party settings without additional roles. These observations naturally raise the following question:

> *Can we design a communication-efficient and constant-round scalable secure comparison protocol without additional roles that leads to a practical scalable PDTE protocol?*

We propose a framework SC-DT to answer the above question affirmatively. It employs (packed) Shamir secret sharing scheme [27, 28] to support computations involving n participants in the honest-majority setting. The key technical insights are summarized as follows.

1.1 Technique Overview

Constant-Round Comparison. In existing scalable secure comparison protocols [22–24], the comparison task is typically reduced to most significant bit (MSB) extraction, a step that requires $O(\log \ell)$ rounds and dominates the overall runtime, where ℓ is the bit length of input. Our goal is to design a constant-round and scalable comparison protocol using Shamir secret sharing (SSS) with Mersenne prime fields. The key idea is to reduce MSB extraction to least significant bit (LSB)

extraction and unsigned comparisons. We build upon a key observation from SecureNN [29], which expresses MSB extraction as:

$$\mathrm{MSB}(x) = \mathrm{LSB}(2x) = \mathrm{LSB}(y) \oplus \mathrm{LSB}(r) \oplus 1\{(y < r)\}$$

where $y = 2x + r \pmod{p}, x, r \in \mathbb{Z}_p$, and p is an odd modulus (Mersenne primes are inherently odd). Moreover, $1\{(y < r)\}$ denotes the indicator function that evaluates to 1 if $y < r$, and 0 otherwise. $\mathrm{LSB}(y)$ can be computed locally, while $\mathrm{LSB}(r)$ is generated in the offline phase. The only remaining task in the online phase is to compute $1\{(y < r)\}$, which corresponds to a secure unsigned comparison. Falcon [30] proposes a DGK-style secure unsigned comparison protocol. However, it incurs $O(\log\ell)$ communication rounds due to the use of cumulative multiplication for zero testing and is restricted to the 3PC setting, limiting its scalability. To reduce interaction and enhance scalability, we replace the cumulative multiplication with random shuffle, enabling constant-round online comparison. However, this approach requires generating $\ell \times \ell$ random permutation matrix for random shuffle, which results in $O(n\ell^2)$ offline communication cost, rendering the approach impractical for a large number of parties. To address this challenge, we further reduce the problem of random shuffle to the random shift based on based on the following observation. Specifically, We observe that, *for a secret-shared vector with exactly one zero entry, determining whether a zero exists without leaking its location does not require shuffling the entire vector after masking before opening the result; it suffices to perform a random shift relative to the position of the zero and then open the result.* Leveraging this observation, we reduces $O(n\ell^2)$ offline communication cost to $O(\ell)$. Details can be found in §3.

Scalable PDTE. Beyond secure comparison, PDTE also requires feature selection and path selection. Inspired by the oblivious selection technique of Mostree [15], which combines one-hot vectors with vector inner products, we reformulate both tasks as instances of the private lookup table (PLUT) problem in nPC setting. Specifically, the parties hold a secret lookup index $u \in [0, m-1]$ and a size-m secret-shared private

lookup table $\mathbf{T}$, and generate secret-shared randomness $(\mathbf{e}_v, v)$, where $\mathbf{e}_v$ is a size-m one-hot vector that has value 1 at index v and value 0 elsewhere. The parties then compute and open the value $r = v - u \pmod m$, which is used as a public offset to construct a shifted random one-hot vector $\hat{\mathbf{e}}_j = \mathbf{e}_{(r+j) \bmod m}, j \in [0, m-1]$. Finally, each party can obtain a shared lookup result by computing shares of the inner product $\langle \hat{\mathbf{e}}_j, \mathbf{T} \rangle = \mathbf{T}[u]$. However, *field arithmetic over the Mersenne prime field $\mathbb{F}_p$ introduces a mismatch between the reconstructed value and the desired lookup index.* Specifically, in the typical setting where the lookup table size satisfies $m < |\mathbb{F}_p|$, let $[v]_d$ and $[u]_d$ be degree-d Shamir secret sharings over $\mathbb{F}_p$ of two values $u, v \in [0, m-1]$. After reconstructing the shared difference $[r]_d = [v]_d - [u]_d$, we can obtain $r = (v-u) \bmod p$. However, in general, $(v-u) \bmod p \neq (v-u) \bmod m$ and therefore the reconstructed value cannot be directly used as the lookup index. To achieve correct table lookup, we observe that when $u \leq v$, we can directly reveal $r = v - u \bmod p$; otherwise, we reveal $r = m + v - u \bmod p$. Consequently, we can obtain $r = 1\{u \leq v\} \cdot (v-u) + (1 - 1\{u \leq v\}) \cdot (m + v - u) \bmod p$ straightforwardly, and $1\{u \leq v\}$ can be derived from our constant round secure comparison. Details can be found in §4.

Improved Packed Secret Sharing Protocols. SSS serves as the key technology for implementing our scalable protocols. Nevertheless, when processing large-scale decision trees, such as random forests, the PDTE protocol based on SSS still encounters significant communication with an increasing number of participants. Franklin et al. introduced a parallelizable technique called packed Shamir secret sharing (PSS) [28], which can pack k secrets into a single secret sharing. Trivially replacing the SSS with PSS could give rise to an amortized reduction in communication cost by a factor of approximately k. Unfortunately, it still exhibits the following defect: XOR operation and PLUT operation require an additional degree reduce operation and packed DN protocol, respetively, which incur more online communication. To address these limitations, we exploit the *local collecting property between SSS and PSS*, which allows

multiple Shamir sharings to be packed into a single PSS without interaction. As a result, the additional online communication cost can be completely eliminated. Details can be found in §5.

1.2 Contributions

In a word, we outline our contributions as follows:

- **Scalable Constant-Round Secure Comparison.** We propose a scalable comparison protocol with constant-round complexity based on SSS. Inspired by [30], the key optimization of our protocol is leveraging random shuffle to realize zero detection, thereby reducing the round complexity from logarithmic to constant. Furthermore, we reduce the problem of random shuffle to the random shift, which reduces $O(n\ell^2)$ offline communication cost to $O(\ell)$. The protocol may be of independent interest.
- **Scalable Privacy Decision Tree Evaluation Framework.** Inspired by Mostree [15], we reformulate feature selection and path selection as instances of the private lookup table (PLUT) problem in nPC setting. Building upon our scalable constant-round secure comparison protocol, we further design an efficient PLUT protocol over prime fields. Together, the proposed PLUT protocol and our scalable constant-round secure comparison protocol enable an efficient and scalable PDTE framework.
- **Optimized Packed Secret Sharing Protocols.** We extend the scalable comparison and PLUT protocols based on SSS to their PSS variants, and further introduce PSS protocols optimized for more efficient online communication by utilizing the collecting property of PSS and SSS. In addition, we implement an efficient and scalable protocol for large-scale decision trees evaluation on the basis of these optimized protocols.

- **Implementation & Evaluation.** We implement SC-DT in C++ and conduct extensive experiments with the number of parties ranging from 9 to 21. For secure comparison, compared with the state-of-the-art scalable MPC framework Helix [24], our packed secure comparison reduces communication by $1.1-2.4\times$ and $1.7-2.5\times$ in the offline and online phases, respectively. It also achieves a $1.5-2.4\times$ reduction in online runtime. For PDTE, when evaluating a random forest consisting of 50 trees of depth 30 with 21 parties, the online runtime is only 1.46s, 8.27s, and 53.72s in LAN, MAN, and WAN settings, respectively.

2 Preliminary

2.1 Notations

The Mersenne prime p is the form of $p = 2^\ell - 1$ for some ℓ. In PDTE, ℓ is chosen to be 31 or 61. $\mathbb{F}_p$ denotes a Mersenne prime field. We use the lowercase letter x to denote a scalar and $x[i]$ to denote the i-th bit of x. The bold lower case letter $\mathbf{x}$ is a vector. $\mathbf{x}[i]$ is the i-th element of $\mathbf{x}$ and $\mathbf{x}_i[j]$ is the j-th element of the i-th vector $\mathbf{x}_i$. Particularly, $\mathbf{e}_v$ is the one-hot vector, which has value 1 at index v and value 0 elsewhere. Correspondingly, we use the uppercase bold letters such as $\mathbf{X}$ to denote tensors. In particular, for a two-dimensional tensor (matrix) $\mathbf{X}$, $\mathbf{X}[i,j]$ denotes the element in the i-th row and j-th column, and $\mathbf{X}[i,\cdot]$ denotes the i-th row of matrix. For a three-dimensional tensor $\mathbf{X}$, $\mathbf{X}[\mu,\rho,\sigma]$ denotes the element indexed by (μ,ρ,σ) and $\mathbf{X}\mu$ denotes the μ-th matrix slice of $\mathbf{X}$, and $\mathbf{X}[\cdot,\rho,\cdot]$ is the ρ-th row of all matrix. In addition, we denote by $[n]$ the set $\{0,1,\cdots,n-1\}$, where $n \in \mathbb{N}, \mathbb{N}$ is the set of natural numbers. For DT, we use $\mathcal{T},\mathcal{X},\mathcal{D},\mathcal{M},\mathcal{N}$ to represent the multi-dimensional tree array, feature vector, the max depth of tree, the total number of nodes including both leaf nodes and decision nodes, and length of feature vector, respectively. Finally, for the sake of simplicity, we denote the communication overhead of a given protocol by $\mathbf{Cost}(\cdot)$.

2.2 Secret Sharing

2.2.1 Shamir secret sharing (SSS) [27]

Let n be the number of parties and $|\mathbb{F}_p| > n+1$. For a secret $x \in \mathbb{F}_p$, we use $[x]_d$ to denote a degree-d SSS. It requires $d+1$ shares to reconstruct the whole secret and corresponds to a degree-d polynomial f such that the values $\{f(\boldsymbol{\alpha}[i])\}_{i=1}^{n}$ represent each share and $f(0) = x$. However, we do not always need to store the secret at point 0. We use $[x|_\beta]_d$ to denote a degree-d Shamir sharing whose secret is stored at the point $\beta, \beta \neq \boldsymbol{\alpha}[i]$.

2.2.2 Packed Shamir secret sharing (PSS) [28]

PSS is a vectorized version of Shamir secret sharing. It can pack k secrets into a single secret sharing. For a secret vector $\mathbf{x} \in \mathbb{F}_p^k$, we use $[\![\mathbf{x}]\!]_d$ to denote a degree-d PSS, where $k-1 \leq d \leq n-1$. It requires $d+1$ shares to reconstruct the whole secret and corresponds to a degree-d polynomial f such that the values $\{f(\boldsymbol{\alpha}[i])\}_{i=1}^{n}$ represent the each shares and $\{f(\boldsymbol{\beta}[i]) = \mathbf{x}[i]\}_{i=0}^{k-1}$, where $\{\boldsymbol{\beta}[i]\}_{i=0}^{k-1}$ are the default positions to store the secrets $\mathbf{x} = \{\mathbf{x}[i]\}_{i=0}^{k-1}$ and $\boldsymbol{\beta} \cap \boldsymbol{\alpha} = \emptyset$. Any $d-k+1$ shares are independent of the secrets.

2.2.3 Basic Protocols

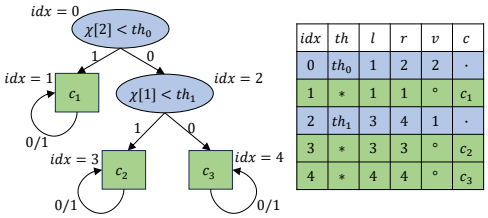
SSS and PSS support the following (semi-honest) addition and multiplication operations. Note that these operations are performed coordinate-wise on PSS in parallel.

- Addition: Given $[x]_d$ and $[y]_d$, the parties locally compute $[x]_d + [y]_d = [z]_d$. Similarly, for the size-k vector $\mathbf{x}$ and $\mathbf{y}$, the parties locally compute $[\![\mathbf{x}]\!]_d + [\![\mathbf{y}]\!]_d = [\![\mathbf{z}]\!]_d$.
- Multiplication: Given $[x]_d$ and $[y]_d$, the parties compute $[z]_d = [x]_d \cdot [y]_d$ by directly calling the DN protocol [31], i.e., $[z]_d \leftarrow \Pi_{\mathsf{DN}}([x]_d, [y_d])$. Similarly, we use Π_{PDN} to denote the extension of the DN protocol for PSS, i.e., $[\![\mathbf{z}]\!]_d \leftarrow \Pi_{\mathsf{PDN}}([\![\mathbf{x}]\!]_d, [\![\mathbf{y}]\!]_d)$.

- Linear operation: Given a public constant $c \in \mathbb{F}_p$, each party computes $c \cdot [x]_d, c + [x]_d$ locally. For PSS, $c + [\![\mathbf{x}]\!]_d$ and $c \cdot [\![\mathbf{x}]\!]_d$ denote adding c to each element of $\mathbf{x}$ and multiplying each element of $\mathbf{x}$ by c, respectively. In addition, when $d \leq n - k$ for PSS [32], all parties can locally multiply a public vector $\mathbf{c} \in \mathbb{F}_p^k$ with the share $[\![\mathbf{x}]\!]_d$. First, all parties locally compute $[\![\mathbf{c}]\!]_{k-1}$. Then all parties can locally compute $[\![\mathbf{c} * \mathbf{x}]\!]_{k-1+d} = [\![\mathbf{c}]\!]_{k-1} \cdot [\![\mathbf{x}]\!]_d$, where $*$ denote coordinate-wise multiplication.
- Random Shares & Random Bits: i) For random shares, we call the protocol $\Pi_{\mathsf{rand_p}}$ [33] and Π_{Random} [34] for generating uniformly random shares $[v]_d$ and $[\![\mathbf{v}]\!]_d$, respectively, where $v \in \mathbb{F}_p, \mathbf{v} \in \mathbb{F}_p^k$; ii) For random bits, we directly call the protocol $\Pi_{\mathsf{rand_2}}$ [33] and $\Pi_{\mathsf{RandomBits}}$ [34] for generating uniformly random shared bit $[b]_d$ and $[\![\mathbf{b}]\!]_d$, respectively, where $b \in \{0,1\}, \mathbf{b} \in \{0,1\}^k$.

2.3 Encoding Decision Tree

In PDTE, DT requires encoding. We adopt the encoding strategy of Mostree, which encodes a DT as an array, then transforms evaluation to a traversal algorithm over arrays. To be specific, we encode a DT as a multi-dimensional array $\mathcal{T}$ as in Figure 2. Each array item corresponding to a tree node contains five values: threshold value th, left child index l, right child index r, feature ID v, and classification label c (only valid for leaf nodes). The evaluation starts from $\mathcal{T}[0]$ and runs at most $\mathcal{D}$ iterations. During the secure evaluation, besides hiding each element of $\mathcal{T}$ and $\mathcal{X}$, we still need to hide the evaluation path. For leaf nodes at different depths, two circles are encoded by setting the child indices to point to the leaf itself. Thus, the parties always run at least $\mathcal{D}$ iterations for each evaluation and will be redirected back to itself once a leaf node with depth less than $\mathcal{D}$ is reached. At the same time, we adopt PLUT to hide the access pattern for secure feature selection and left or right path selection.



idx	th	l	r	v	c
0	th_0	1	2	2	·
1	*	1	1	°	c_1
2	th_1	3	4	1	·
3	*	3	3	°	c_2
4	*	4	4	°	c_3

Fig. 2 Encoding the tree as an array $\mathcal{T}$, where $*, \cdot \in \mathbb{F}_p, \circ \in [\mathcal{N}]$.

2.4 Model

We use $\mathcal{P} = \{P_1, \cdots, P_n\}$ to represent the n parties, where $n \geq 2d+1, d \in \mathbb{N}$. All input data, including the decision model and the feature vector, is secret-shared among all parties. n parties communicate through end-to-end networks to ensure the security and reliability of communication. We consider our protocols in the semi-honest model with the honest-majority setting. The number of corrupted parties t that are assumed to be poly-time bounded is at most $t = d$ (resp., $t = d - k + 1$) for SSS (resp., PSS), where $2 \leq k \leq d$. For a degree-d PSS which packs k secrets, k and default positions $\{\boldsymbol{\beta}[i]\}_{i=0}^{k-1}$ are publicly known to the n parties.

3 Scalable Constant-Round Comparison

Secure comparison protocols are typically reduced to MSB extraction in existing MPC frameworks [22–24]. MSB extraction is further reduced to LSB extraction and unsigned comparisons in the odd ring. However, unsigned comparisons incur high communication and round complexity. In this section, we propose a scalable constant-round comparison protocol using SSS with Mersenne prime fields. Specifically, we first recap the unsigned comparisons of Falcon [30] and introduce our strawman scalable protocol in §3.1. Then, we optimize our strawman protocol in §3.2. Finally, we present our scalable constant-round secure comparison protocol in §3.3.

3.1 Unsigned Comparisons of Falcon & Strawman Protocol

Given two values $x = x[\ell-1]\cdots x[0], y = y[\ell-1]\cdots y[0] \in \mathbb{F}_p$, when $x < y$, there is an index $\alpha \in [n]$ that satisfies $x[\ell-1] = y[\ell-1], \cdots, x[\alpha+1] = y[\alpha+1], x[\alpha] = 0$ and $y[\alpha] = 1$. In light of this, Falcon proposes an unsigned comparison protocol. Specifically, Falcon defines fellow formulas:

$$\mathbf{u}[i] = x[i] - y[i], \mathbf{w}[i] = x[i] \oplus y[i], \mathbf{c}[i] = \mathbf{u}[i] + 1 + \textstyle\sum_{j=i+1}^{\ell-1} \mathbf{w}[i].$$

If $x \geq y$, all elements of $\mathbf{c}$ are positive; in contrast, if $x < y$, only one element of $\mathbf{c}$ equals zero. Thus, the comparison reduces to checking for the existence of a zero in $\mathbf{c}$. Falcon cumulatively multiplies all entries in $\mathbf{c}$ for realizing the above functionality in 3PC settings, which requires $O(\log\ell)$ rounds. Subsequent study Mosformer [20] designs a constant-round protocol that replaces the cumulative multiplication with a distributed point function (DPF) [14], but it still lacks scalability.

Since the key is to check for the existence of a zero in $\mathbf{c}$, we consider performing a secure shuffle operation on the entire vector $\mathbf{c}$ for implementing a scalable constant-round protocol. To be specific, after computing $\mathbf{c}$, we first multiply each element $\mathbf{c}[i]$ of $\mathbf{c}$ by a non-zero value $\boldsymbol{\rho}[i]$ to get $\mathbf{g}$, where $\mathbf{g}[i] = \mathbf{c}[i] \cdot \boldsymbol{\rho}[i], i \in [\ell]$ and then conduct secure shuffle on $\mathbf{g}$. Finally, all parties reveal $\mathbf{g}$ and check whether any element of $\mathbf{g}$ is zero. We formally describe our strawman protocol in **Algorithm** 1.

In protocol Π_{SUCP}, $[\theta]_d$ is used to blind the output as θ' reveals information about the output $x \geq y$. In addition, as $r = r_1 \oplus r_2 = r_1 + r_2 - 2r_1r_2$, where $r, r_1, r_2 \in \{0, 1\}$, we compute $[\mathbf{w}[i]]_d = [x[i]]_d + y_i - 2 \cdot [x[i]]_d \cdot y[i], [b]_d = \theta' + [\theta]_d - 2 \cdot [\theta]_d \cdot \theta'$ for bit XOR $[\mathbf{w}[i]]_d = [x[i]]_d \oplus y[i], [b]_d = [\theta]_d \oplus \theta'$. For step 3, protocol Π_{RPM} (**Algorithm** 2) generates a $\ell \times \ell$ random permutation matrix for online random shuffle, as illustrated in Figure 3 (a). For step 14, all parties locally compute $[\mathbf{h}]_{2d} = [\mathbf{g}]_d \cdot [\mathbf{M}]_d + [\mathbf{0}]_{2d}$ and directly reveal $\mathbf{h}$ in one round. The parties can get a degree-$2d$ sharing $[\mathbf{0}]_{2d}$ locally by using the pseudo-random sharing of zero (PRZS [35]) technique.

Algorithm 1 Strawman Unsigned Comparison Π_{SUCP}

1: **Input:** $[x[i]]_d, i \in [\ell], x[i]$ is a bit and a public value y.
2: **Output:** $[x \geq y]_d = [b]_d, x = \sum_{i=0}^{\ell-1} 2^i x[i], b$ is a bit.
3: **Offline:**
4: Get ℓ non-zero random shares $[\boldsymbol{\rho}[i]]_d \leftarrow \Pi_{\mathsf{rand_p}}, i \in [\ell], \boldsymbol{\rho}[i] \in \mathbb{F}_p/\{0\}$.
5: Get a random bit share $[\theta]_d \leftarrow \Pi_{\mathsf{rand_2}}, \theta \in \{0, 1\}$.
6: Get a $\ell \times \ell$ random permutation matrix $[\mathbf{M}]_d \leftarrow \Pi_{\mathsf{RPM}}(\ell)$.
7: **Online:**
8: **for** $\{i = \ell - 1, \ell - 2, \cdots, 0\}$ **do**
9: Compute $[\mathbf{u}[i]]_d = (1 - 2 \cdot [\theta]_d) \cdot ([x[i]]_d - y[i])$.
10: Compute $[\mathbf{w}[i]]_d = [x[i]]_d + y[i] - 2 \cdot [x[i]]_d \cdot y[i]$.
11: Compute $[\mathbf{c}[i]]_d = [\mathbf{u}[i]]_d + 1 + \sum_{j=i+1}^{\ell-1} [\mathbf{w}[i]]_d$.
12: **end for**
13: Compute $[\mathbf{g}[i]]_d \leftarrow \Pi_{\mathsf{DN}}([\mathbf{c}[i]]_d, [\boldsymbol{\rho}[i]]_d), i \in [\ell]$.
14: Compute $[\mathbf{h}]_{2d} = [\mathbf{g}]_d \cdot [\mathbf{M}]_d + [\mathbf{0}]_{2d}$ and reveal $\mathbf{h} \in \mathbb{F}_p^\ell$.
15: Set $\theta' = 1$ if there is a zero in $\mathbf{h}$ and $\theta' = 0$ otherwise.
16: **return** $[b]_d = \theta' + [\theta]_d - 2 \cdot [\theta]_d \cdot \theta'$.

Algorithm 2 Random Permutation Matrix Π_{RPM}

1: **Input:** ℓ.
2: **Output:** $[\mathbf{M}]_d, \mathbf{M}$ is a $\ell \times \ell$ random permutation matrix and each row and column of $\mathbf{M}$ contains exactly one element equal to 1.
3: **for** $\{i = 1, 2, \cdots, n\}$ **do**
4: Each party P_i generates a ℓ-by-ℓ permutation matrix $\mathbf{M}_i$ and send secret-share $[\mathbf{M}_i]_d$ to other parties.
5: **end for**
6: All parties compute $[\mathbf{M}]_d = [\mathbf{M}_1]_d [\mathbf{M}_2]_d \cdots [\mathbf{M}_n]_d$ by Π_{DN}.
7: **return** $[\mathbf{M}]_d$.

Cost(Π_{SUCP}). For the offline phase: i) Steps 4, 5 need $2\ell + 4$ field elements per party in 2 rounds. ii) Steps 6 needs $7n\ell^2 - 6\ell^2$ field elements per party in $\lceil \log_2(n) \rceil + 1$ round. Additionally, the total offline cost of all DN multiplications in step 9 and 13 is 8ℓ field elements per party. Therefore, the offline phase needs $7n\ell^2 - 6\ell^2 + 10\ell + 4$ field elements per party in $\lceil \log_2(n) \rceil + 3$ round. For the online phase: i) Step 9 needs 2ℓ field elements per party in 1 round. ii) Steps 13 and 14 need 4ℓ field elements per party in 2 rounds. Therefore, the online phase needs 6ℓ field elements per party in 3 rounds.

3.2 Optimized Constant-Round Unsigned Comparison

Although our strawman protocol Π_{SUCP} achieves online constant-round communication complexity, it incurs an offline communication cost of $O(n\ell^2)$. This overhead stems primarily from the generation of the random permutation matrix, rendering the protocol impractical when handling a large number of comparisons.

In order to further reduce the offline communication cost, we review the online process of protocol Π_{SUCP}. We observe that if all elements of the vector $\mathbf{g}$ are non-zero, then $\mathbf{g}$ can be directly revealed because $\mathbf{g}$ is a random vector. Instead, we shuffle the vector $\mathbf{g}$ only when there exists a zero element in $\mathbf{g}$. Therefore, the sole purpose of shuffle is to hide the positions of the zero element. In light of this, we can reduce the secure shuffle to the secure shift on the vector $\mathbf{g}$ and propose the following proposition (with proof deferred to Appendix C):

Proposition 1 *For a secret-shared vector with exactly one zero entry in $\mathbb{Z}_p$, where p is a prime, determining whether a zero exists without leaking its location does not require shuffling the entire vector after masking before opening the result; it suffices to perform a random shift relative to the position of the zero and then open the result.*

Remark. *The **Proposition** 1 is not limited to the protocol presented herein; rather, it generalizes naturally to any protocol involving zero-tests over prime fields. As an example, Lu [19] proposes a constant-round comparison under 2PC. Our proposition can also be leveraged to reduce their offline cost.*

Analogous to the dependency of secure shuffle on a random permutation matrix, secure shift also depends on a random shift matrix. Specifically, each row and each column of the random shift matrix contains exactly one entry equal to 1. Figure 3 (b) describes a simple example. The 4×4 random shift matrix can be generated by applying a cyclic row shift with a randomly chosen offset to the 4×4 identity

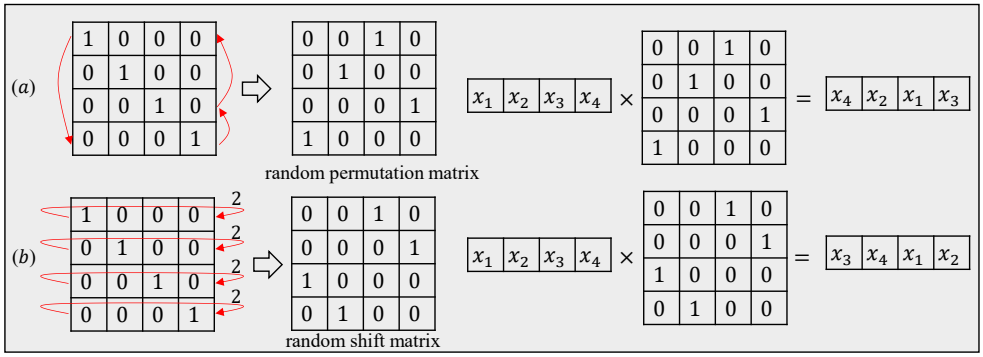


Fig. 3 An example about random shuffle and random shift. (a) A random permutation matrix is obtained by applying a random row permutation to the identity matrix. (b) A random shift matrix is obtained by applying a common cyclic shift to all rows of the identity matrix.

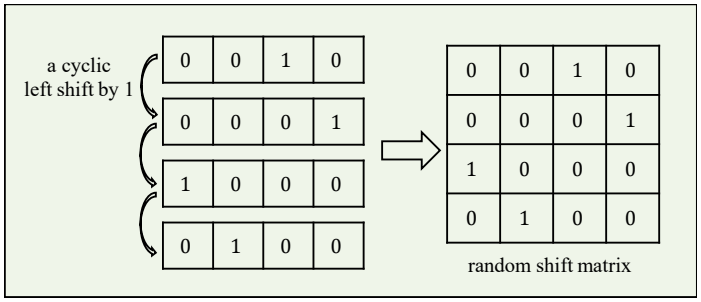


Fig. 4 From a random one-vector to a random shift matrix.

matrix. Hence, generating a random shift matrix can be further reduced to generating a random one-hot vector, which is then cyclic shifted locally, as illustrated in Figure 4.

Based on the above analysis, the critical step is the generation of the random one-hot vector. Fortunately, we can directly adopt the demux protocol proposed in [36], which generates a random one-hot vector whose length is a power of two. We present the protocol in **Algorithm** 3. Figure 5 describes a simple example. Based on this construction, we further design **Algorithm** 4 for generating a random shift matrix.

Cost(Π_{ROH})&**Cost**(Π_{RSM}). The communication cost of Π_{RSM} is the same as that of Π_{ROH}. For generating a size-2^m random one-hot vector (Π_{ROH}): i) Step 3 needs $4m$ field elements per party in 2 rounds. ii) Step 6 needs $6(2^m-2)$ field elements per party

Algorithm 3 Random One-Hot Vector Π_{ROH}

Input: m.
Output: $([\mathbf{e}_r]_d, [r]_d)$. $\mathbf{e}_r \in \{0,1\}^{2^m}$ is a random one-hot vector, and the r-th element is equal to 1, and all other elements are 0, where $r = \sum_{i=0}^{m-1} 2^i \cdot r_i, r_i \in \{0,1\}$.
Get m random bit shares $[r_i]_d \leftarrow \Pi_{\mathsf{rand}_2}, i \in [m]$ then compute $[r]_d = \sum_{i=0}^{m-1} 2^i \cdot [r_i]_d$.
All parties set $\mathbf{v}^{(0)} = (1 - [r_0]_d, [r_0]_d)$.
for $\{i = 1, 2, \cdots, m-1\}$ **do**
    All parties compute $\mathbf{v}^{(i)} = ((1 - [r_i]_d) \cdot \mathbf{v}^{(i-1)}, [r_i]_d \cdot \mathbf{v}^{(i-1)})$.
end for
All parties set $[\mathbf{e}_r]_d = \mathbf{v}^{(m-1)}$.
return $([\mathbf{e}_r]_d, [r]_d)$.

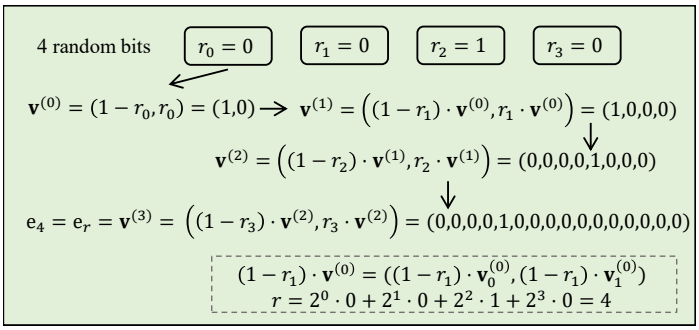


Fig. 5 A simple example about random one-hot vector generation.

in $m - 1$ rounds. Therefore, the Π_{ROH} and Π_{RSM} need $4m + 6(2^m - 2)$ field elements per party in $m + 1$ rounds for generating a $2^m \times 2^m$ random shift matrix.

Now, we present our optimized unsigned comparison protocol in **Algorithm** 5. Its main difference from Π_{SUCP} is that it uses Π_{RSM} instead of Π_{RPM} in offline phase. In addition, it is important to note that Π_{ROH} only generates a random one-hot vector whose length is a power of two. However, ℓ is chosen to be 31 or 61. To address this limitation, we can directly pad the vector $[\mathbf{g}]_d$ with $2^{\lceil \log_2(\ell) \rceil} - \ell$ non-zero values. Although we perform padding in the vector $[\mathbf{g}]_d$, it incurs negligible overhead. Concretely, when $\ell = 31$ (resp. $\ell = 61$), only 1 (resp. 3) value is padded. We report Π_{SUCP} and Π_{OUCP} theoretical communication overhead in Table 1.

Algorithm 4 Random Shift Matrix Π_{RSM}

Input: m.
Output: $[\mathbf{M}]_d, \mathbf{M}$ is a $2^m \times 2^m$ random shift matrix and each row and column of $\mathbf{M}$ contains exactly one element equal to 1.
Get size-2^m one-hot vector $[\mathbf{e}_r]_d \leftarrow \Pi_{\mathsf{ROH}}(m)$.
All parties set $[\mathbf{M}[0,\cdot]]_d = [\mathbf{e}_r]_d$.
for $\{i = 1, 2, \cdots, 2^m - 1\}$ **do**
    Get $[\mathbf{e}'_r]_d$ by cyclically left shifting $[\mathbf{e}_r]_d$ by i position locally and set $[\mathbf{M}[i,\cdot]]_d = [\mathbf{e}'_r]_d$.
end for
return $[\mathbf{M}]_d$.

Table 1 The amortized communication cost of Π_{SUCP} and Π_{OUCP} per party. ℓ is the bit length, n is the number of party.

Protocol	Offline		Online	
	Communication	Rounds	Communication	Rounds
Π_{SUCP}	$7n\ell^2 - 6\ell^2 + 10\ell + 4$	$\lceil \log_2(n) \rceil + 3$	6ℓ	3
Π_{OUCP}	$8 \cdot 2^{\lceil \log_2(\ell) \rceil} + 8\ell + 4\lceil \log_2(\ell) \rceil - 8$	$\lceil \log_2(\ell) \rceil - 1$	$4\ell + 2 \cdot 2^{\lceil \log_2(\ell) \rceil}$	3

3.3 Scalable Constant-Round Secure Comparison

Given two values $x, y \in \mathbb{F}_p$ in two's complement representation, $1\{x \leq y\}$ can be defined as:

$$1\{x \leq y\} = 1\{(x - y) \geq 0\} = \mathrm{DReLU}(x - y)$$

where DReLU is defined as:

$$\mathrm{DReLU}(a) = 1 - \mathrm{MSB}(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$$

It is observed in [29] that $\mathrm{MSB}(a) = \mathrm{LSB}(2a)$ if the multiplication is done over an odd ring. The Mersenne prime fields $\mathbb{F}_p$ are an odd ring, apparently. Hence, it holds that $\mathrm{DReLU}(a) = 1 - \mathrm{LSB}(2a)$.

In order to compute $\mathrm{LSB}(2a)$, we adopt the approach of [29]. More specifically, we first generate a random value $r \in \mathbb{F}_p$ and compute $z = 2a + r$. Then we can reveal z and compute $\mathrm{LSB}(z) = \mathrm{LSB}(2a) \oplus \mathrm{LSB}(r) \oplus 1\{z \leq r\}$ such that $\mathrm{LSB}(2a) =$

Algorithm 5 Optimized Unsigned Comparison Π_{OUCP}

Input: $[x[i]]_d, i \in [\ell], x[i]$ is a bit and a public value y.
Output: $[x \geq y]_d = [b]_d, x = \sum_{i=0}^{\ell-1} 2^i x[i], b$ is a bit.
Offline:
Get $2^{\lceil \log_2(\ell) \rceil}$ non-zero random shares $[\boldsymbol{\rho}[i]]_d \leftarrow \Pi_{\mathsf{rand_p}}, i \in [2^{\lceil \log_2(\ell) \rceil}], \boldsymbol{\rho}[i] \in \mathbb{F}_p/\{0\}$.
Get a random bit share $[\theta]_d \leftarrow \Pi_{\mathsf{rand_2}}, \theta \in \{0, 1\}$.
Get a $2^{\lceil \log_2(\ell) \rceil} \times 2^{\lceil \log_2(\ell) \rceil}$ random shift matrix $[\mathbf{M}]_d \leftarrow \Pi_{\mathsf{RSM}}(\lceil \log_2(\ell) \rceil)$.
Online:
for $\{i = \ell - 1, \ell - 2, \cdots, 0\}$ **do**
    Compute $[\mathbf{u}[i]]_d = (1 - 2 \cdot [\theta]_d) \cdot ([x[i]]_d - y[i])$.
    Compute $[\mathbf{w}[i]]_d = [x[i]]_d + y[i] - 2 \cdot [x[i]]_d \cdot y[i]$.
    Compute $[\mathbf{c}[i]]_d = [\mathbf{u}[i]]_d + 1 + \sum_{j=i+1}^{\ell-1} [\mathbf{w}[i]]_d$.
end for
Compute $[\mathbf{g}[i]]_d \leftarrow \Pi_{\mathsf{DN}}([\mathbf{c}[i]]_d, [\boldsymbol{\rho}[i]]_d), i \in [\ell]$.
for $\{i = \ell, \cdots, 2^{\lceil \log_2(\ell) \rceil} - 1\}$ **do**
    $[\mathbf{g}[i]]_d = [\boldsymbol{\rho}[i]]_d$.
end for
Compute $[\mathbf{h}]_{2d} = [\mathbf{g}]_d \cdot [\mathbf{M}]_d + [\mathbf{0}]_{2d}$ and reveal $\mathbf{h} \in \mathbb{F}_p^{2^{\lceil \log_2(\ell) \rceil}}$.
Set $\theta' = 1$ if there is a zero in $\mathbf{h}$ and $\theta' = 0$ otherwise.
Compute $[b]_d = \theta' + [\theta]_d - 2 \cdot [\theta]_d \cdot \theta'$.
return $[b]_d$.

$\mathrm{LSB}(z) \oplus \mathrm{LSB}(r) \oplus 1\{z \leq r\}$. As for $1\{z \leq r\}$, we can directly use our constant-round unsigned comparison protocol. We present our scalable constant-round comparison protocol in **Algorithm** 6.

Algorithm 6 Constant-Round Comparison Π_{CMP}

Input: $[x]_d, [y]_d$.
Output: $[x \geq y]_d = [b]_d, b$ is a bit.
Offline:
Get ℓ random bits share $[r_i]_d \leftarrow \Pi_{\mathsf{rand_2}}, r_i \in \{0, 1\}$, $i \in [\ell]$.
Compute $[r]_d = \sum_{i=0}^{\ell-1} 2^i \cdot [r_i]_d$.
Run the offline phase of Π_{OUCP}.
Online:
Compute $[z]_d = 2([x]_d - [y]_d) + [r]_d$ and reveal z.
Get $[v]_d \leftarrow \Pi_{\mathsf{OUCP}}(\{[r_i]_d\}_{i=0}^{\ell-1}, z)$.
Set $[u]_d = \mathrm{LSB}(z) \oplus [r_0]_d = [\mathbf{w}[0]]_d$. ▷ $[\mathbf{w}[0]]_d$ is from step 10 of Π_{OUCP}.
Compute $[b]_d = 1 - [u]_d \oplus [v]_d = 1 - [u]_d - [v]_d + 2 \cdot [v]_d \cdot [u]_d$.
return $[b]_d$.

$\mathbf{Cost}(\Pi_{\mathsf{CMP}})$. The offline cost is $4\ell + 2 + \mathbf{Cost}(\Pi_{\mathsf{OUCP}}^{\mathsf{offline}})$ field elements per party in $\lceil \log_2(\ell) \rceil - 1$ rounds. For the online phase: i) Step 8 needs 2 field elements per party in 1 round. ii) Step 9 needs $4\ell + 2 \cdot 2^{\lceil \log_2(\ell) \rceil}$ field elements per party in 2 rounds. It is worth noting that although Π_{OUCP} nominally requires 3 rounds of communication in the online phase, the multiplication $2 \cdot [\theta]_d \cdot [x[i]]_d$ in step 9 (**Algorithm** 5) can be precomputed in the offline phase when the ℓ input bit shares $[x[i]]_d = [r_i]_d, i \in [\ell]$ are generated. Consequently, the online phase can be reduced to 2 rounds of communication. iii) Step 11 needs 2 field elements per party in 1 round. Therefore, the online phase needs $4 + 4\ell + 2 \cdot 2^{\lceil \log_2(\ell) \rceil}$ field elements per party in 4 rounds.

4 Scalable Privacy Decision Tree Evaluation

In PDTE, in addition to threshold comparison, PDTE also requires feature selection and path selection. Mostree [15] proposes an oblivious selection protocol that combines one-hot vectors with vector inner products to support these operations. Building on this idea, we reformulate feature selection and path selection as instances of the private lookup table (PLUT) problem in nPC setting. Specifically, we first present the private lookup table protocol in §4.1. Then, we propose our scalable PDTE protocol in §4.2.

4.1 Private Lookup Table Protocol

Given a size-m lookup table $\mathbf{T}$ and an index $u \in [m]$ in secret-shared form, a secure private lookup typically proceeds as follows: i) Generate secret-shared randomness $(\mathbf{e}_v, v)$, where $\mathbf{e}_v$ is a size-m one-hot vector with a 1 at position v and 0 elsewhere. ii) Compute and open the value $r = v - u \pmod m$, which serves as a public offset. By leveraging r, we construct a shifted one-hot vector $\hat{\mathbf{e}}_j = \mathbf{e}_{(r+j) \bmod m}$ for $j \in [m]$. iii) Obtain the shared lookup result by computing the inner product $\langle \hat{\mathbf{e}}, \mathbf{T} \rangle = \mathbf{T}[u]$.

However, a subtle issue arises in this context, namely that *field arithmetic over the Mersenne prime field $\mathbb{F}_p$ introduces a mismatch between the reconstructed value*

and the desired lookup index. Specifically, we consider the typical setting where the lookup table size satisfies $m < |\mathbb{F}_p|$. After reconstructing the shared difference $[r]_d = [v]_d - [u]_d$, we can obtain $r = (v - u) \bmod p$. However, in general, $(v - u) \bmod p \neq (v - u) \bmod m$ and therefore the reconstructed value cannot be directly used as the lookup index. To obtain the correct lookup index, we observe that if $u \leq v$, the reconstructed field element can be revealed directly, i.e., $r = (v - u) \bmod p$, since $0 \leq v - u < m < p$ and $(v - u) \bmod p = (v - u) \bmod m$. Otherwise, we reveal $r = (m + v - u) \bmod p$. To see why, let $u > v, d = u - v > 0, m > d > 0, v - u = -d, (v - u) \bmod p = p - d, (m + v - u) \equiv (m + p - d) \bmod p = m - d$. Consequently, the above two cases can be unified as

$$r = (1\{u \leq v\} \cdot (v - u) + (1 - 1\{u \leq v\}) \cdot (m + v - u)) \bmod p.$$

In addition, $1\{u \leq v\}$ can be derived from our constant round secure comparison Π_{CMP} (**Algorithm** 6). It is worth noting that since Π_{ROH} (**Algorithm 3**) generates one-hot vectors only for power-of-two lengths, we assume that the lookup table size m is also a power of two. We present the secure private lookup protocol in **Algorithm** 7.

Algorithm 7 Private Lookup Table Π_{PLUT}

Input: $\{[\mathbf{T}[i]]_d\}_{i=0}^{m-1}, [u]_d, u \in [m], m < |\mathbb{F}_p|$.
Output: $[\mathbf{T}[u]]_d$.
Offline:
Get randomness $([\mathbf{e}_v]_d, [v]_d) \leftarrow \Pi_{\mathsf{ROH}}(\log_2 m)$.
Run the offline phase of Π_{CMP}.
Online:
Compute $[b]_d \leftarrow \Pi_{\mathsf{CMP}}([v]_d, [u]_d)$.
Compute $[r]_d = [b]_d \cdot ([v]_d - [u]_d) + (1 - [b]_d) \cdot (m + [v]_d - [u]_d)$ and reveal r.
Set $[\hat{\mathbf{e}}_j]_d = [\mathbf{e}_{(r+j) \bmod m}]_d, j \in [m]$.
Compute $[res]_d = \sum_{i=0}^{m} [\hat{\mathbf{e}}_i]_d \cdot [\mathbf{T}[i]]_d$. ▷ Π_{DN}.
return $[res]_d$.

Cost(Π_{PLUT}). The offline cost is **Cost**($\Pi_{\mathsf{ROH}} + \Pi_{\mathsf{CMP}}^{\mathsf{offline}}$). The online cost is $4 +$ **Cost**($\Pi_{\mathsf{CMP}}^{\mathsf{online}}$) fields elements per party in 6 rounds.

4.2 Scalable PDTE Protocol

Combining tree encoding §2.3, scalable constant-round comparison protocol Π_{CMP} and scalable private lookup table protocol Π_{PLUT}, we propose the scalable PDTE protocol formally described in **Algorithm** 8. It should be noted that the protocol Π_{PLUT} is restricted to inputs whose size is a power of two. Hence, we pad the input $\mathcal{T}$ (resp. $\mathcal{X}$) of size $\mathcal{M}$ (resp. $\mathcal{N}$) to length $2^{\ell_{\mathcal{M}}}$ (resp. $2^{\ell_{\mathcal{N}}}$), where $\ell_{\mathcal{M}}$ (resp. $\ell_{\mathcal{N}}$) is the smallest integer such that $2^{\ell_{\mathcal{M}}} \geq \mathcal{M}$ (resp. $2^{\ell_{\mathcal{N}}} \geq \mathcal{N}$).

Algorithm 8 Scalable PDTE Π_{SPDTE}

Input: $\{[\mathcal{T}[i]]_d\}_{i=0}^{\mathcal{M}-1}, \{[\mathcal{X}[j]]_d\}_{j=0}^{\mathcal{N}-1}$.
Output: $[res]_d, res$ is the evaluation result of DT.
Offline:
Run the offline phase of Π_{CMP} and Π_{PLUT}.
Online:
Initialize $[res]_d = 0$.
Parse $([th]_d, [l]_d, [r]_d, [v]_d, [c]_d) \leftarrow [\mathcal{T}[0]]_d$. ▷ $\mathcal{T}[0]$ is the root.
for $\{i = 0, 1, \cdots, \mathcal{D} - 1\}$ **do**
    $[\mathcal{X}[v]]_d \leftarrow \Pi_{\mathsf{PLUT}}(\{[\mathcal{X}[j]]_d\}_{j=0}^{\mathcal{N}-1}, [v]_d)$. ▷ Feature Selection.
    $[b]_d \leftarrow \Pi_{\mathsf{CMP}}([th]_d, [\mathcal{X}[v]]_d)$. ▷ Threshold Comparison.
    Compute $[idx]_d = (1 - [b]_d) \cdot [r]_d + [b]_d \cdot [l]_d$. ▷ Path Selection.
    $[\mathcal{T}[idx]]_d \leftarrow \Pi_{\mathsf{PLUT}}(\{[\mathcal{T}[i]]_d\}_{i=0}^{\mathcal{M}-1}, [idx]_d)$.
    $([th]_d, [l]_d, [r]_d, [v]_d, [c]_d) \leftarrow [\mathcal{T}[idx]]_d$.
    Update $[res]_d = [c]_d$.
end for
return $[res]_d$.

Cost(Π_{PDTE}). The offline cost is primarily determined by Π_{CMP} and Π_{PLUT}, requiring $O\big(\mathcal{D} \cdot \mathbf{Cost}(\Pi_{\mathsf{CMP}}^{\mathsf{offline}} + \Pi_{\mathsf{PLUT}}^{\mathsf{offline}})\big)$ field elements per party in $O(\log_2 \ell)$ rounds. In the online phase, the protocol incurs $(24 + 12\ell + 6 \cdot 2^{\lceil \log_2(\ell) \rceil}) \cdot \mathcal{D}$ field elements per party in $17 \cdot \mathcal{D}$ rounds.

5 Extension to Packed Secret Sharing Protocols

SSS is used as the fundamental primitive to realize our scalable protocols. However, with an increasing number of participants and comparison operations, the protocol

based on SSS still incurs significant communication cost. This also holds for large-scale decision trees, such as random forests. In theory, PSS [28] can pack k secrets into a single secret sharing, which gives rise to an amortized reduction in communication cost by a factor of approximately k. In this section, we focus on extending the above protocols to the PSS variant. Specifically, we first present the packed constant-round unsigned comparison protocol and private lookup table protocol in §5.1. Then we apply these techniques to the large-scale PDTE task in §5.2.

5.1 Improved Packed Secret Sharing Protocols

5.1.1 Packed Comparison Protocol

For packed unsigned comparison protocol $\Pi^{\mathsf{PSS}}_{\mathsf{UCP}}$ in **Algorithm** 9, we follow the basic steps of **Algorithm** 5 by replacing SSS with PSS. We use $\{[\![\mathbf{x}_i]\!]_d\}_{i=0}^{\ell-1}$ to denote the packed bitwise representation of $\mathbf{x} \in \mathbb{F}_p^k$, where $\mathbf{x}_i$ contains the i-th bits of all elements in $\mathbf{x}$. Similarly, the public vector $\mathbf{y} \in \mathbb{F}_p^k$ is packed into $[\![\mathbf{y}_i]\!]_{k-1}$ locally. It should be noted that the computations of $\mathbf{w}_j = \mathbf{x}_j \oplus \mathbf{y}_j, j \in [\ell]$ and $\mathbf{b} = \boldsymbol{\theta} \oplus \boldsymbol{\theta}'$ in steps 12 and 22, respectively, raise the polynomial degree to $d + k - 1$. As a result, an extra 2 rounds of communication are required to reduce the degree back to d ($\Pi_{\mathsf{DegreeReduce}}$ in Appendix A). To further eliminate the extra communication round, we exploit *the collecting property between SSS and PSS*, which allows multiple Shamir sharings to be packed into a single PSS without interaction. Concretely, given the k degree-t SSS $\{[\mathbf{x}[i]|_{\boldsymbol{\beta}[i]}]_t\}_{i=0}^{k-1}$, we can compute $[\![\mathbf{x} * \mathbf{y}]\!]_d = \sum_{i=0}^{k-1} [\![\mathbf{e}_i]\!]_{k-1} \cdot \mathbf{y}[i] \cdot [\mathbf{x}[i]|_{\boldsymbol{\beta}[i]}]_t$ locally. The main reason is that the degree-d PSS tolerates up to $t = d-k+1$ corrupted parties and the degree-t SSS tolerates up to t corrupted parties. The terms $[\![\mathbf{e}_i]\!]_{k-1} \cdot \mathbf{y}[i] \cdot [\mathbf{x}[i]|_{\boldsymbol{\beta}[i]}]_t$ directly output the degree-d PSS values, i.e., $d = t + k - 1$. We present our optimized packed unsigned comparison protocol in the blue-highlighted portions of **Algorithm** 9 and packed constant-round comparison protocol $\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$ in **Algorithm** 10. In addition, we show the PSS variant $\Pi^{\mathsf{PSS}}_{\mathsf{RSM}}$ of Π_{RSM} and $\Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$ in Appendix A.

Algorithm 9 Packed Unsigned Comparison $\Pi^{\mathsf{PSS}}_{\mathsf{UCP}}$ (Optimized version $\Pi^{\mathsf{PSS}}_{\mathsf{OUCP}}$)

1: **Input:** $\{[\![\mathbf{x}_i]\!]_d\}_{i=0}^{\ell-1}$ and a public vector $\mathbf{y}$ of length k, where $\mathbf{x}_i$ is a bit string of length k and $\mathbf{x}[j] = \sum_{i=0}^{\ell-1} 2^i \cdot \mathbf{x}_i[j], \mathbf{y}[j] = \sum_{i=0}^{\ell-1} 2^i \cdot \mathbf{y}_i[j], , j \in [k]$.
2: **Input of optimized version:** $(\{[\![\mathbf{x}_i]\!]_d\}_{i=0}^{\ell-1}, \{[\mathbf{x}_i[j]|_{\boldsymbol{\beta}[j]}]_t\}_{\ell-1}^{i=0}, j \in [k]$ and $\mathbf{y})$.
3: **Output:** $[\![\mathbf{b}]\!]_d, \mathbf{b}[j] = (\mathbf{x}[j] \geq \mathbf{y}[j])$ is a bit.
4: **Offline:**
5: Get $[\![\boldsymbol{\rho}_i]\!]_d \leftarrow \Pi_{\mathsf{Random}}, i \in [2^{\lceil \log_2(\ell) \rceil}]$, $\boldsymbol{\rho}_i$ is a non-zero vector of length k.
6: Get $[\![\boldsymbol{\theta}]\!]_d \leftarrow \Pi_{\mathsf{RandomBits}}, \boldsymbol{\theta} \in \{0,1\}^k$.
7: **Optimized version**: $([\![\boldsymbol{\theta}]\!]_d, \{[\boldsymbol{\theta}[i]|_{\boldsymbol{\beta}[i]}]_t\}_{i=0}^{k-1}) \leftarrow \Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$.
8: Get $[\![\mathbf{M}]\!]_d \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{RSM}}(\lceil \log_2(\ell) \rceil)$, where the tensor $\mathbf{M}$ contains k random shift matrices and each matrix is of size $2^{\lceil \log_2(\ell) \rceil} \times 2^{\lceil \log_2(\ell) \rceil}$.
9: **Online:**
10: **for** $\{j = \ell - 1, \ell - 2, \cdots, 0\}$ **do**
11: $\quad [\![\mathbf{u}_j]\!]_d = (1 - 2 \cdot [\![\boldsymbol{\theta}]\!]_d) \cdot ([\![\mathbf{x}_j]\!]_d - [\![\mathbf{y}_j]\!]_{k-1})$.
12: $\quad [\![\mathbf{w}_j]\!]_d = [\![\mathbf{x}_j]\!]_d + [\![\mathbf{y}_j]\!]_{k-1} - 2[\![\mathbf{x}_j]\!]_d [\![\mathbf{y}_j]\!]_{k-1}$. ▷ $\mathbf{w}_j = \mathbf{x}_j \oplus \mathbf{y}_j$.
13: $\quad$ **Optimized version:** $[\![\mathbf{w}_j]\!]_d = [\![\mathbf{x}_j]\!]_d + [\![\mathbf{y}_j]\!]_{k-1} - 2\sum_{i=0}^{k-1} [\![\mathbf{e}_i]\!]_{k-1} \cdot [\mathbf{x}_j[i]|_{\boldsymbol{\beta}[i]}]_t \cdot \mathbf{y}_j[i]$.
14: $\quad [\![\mathbf{c}_j]\!]_d = [\![\mathbf{u}_j]\!]_d + 1 + \sum_{i=j+1}^{\ell-1} [\![\mathbf{w}_j]\!]_d$.
15: **end for**
16: Compute $[\![\mathbf{g}_i]\!]_d \leftarrow \Pi_{\mathsf{PDN}}([\![\mathbf{c}_i]\!]_d, [\![\boldsymbol{\rho}_i]\!]_d), i \in [\ell]$.
17: **for** $\{j = \ell, \cdots, 2^{\lceil \log_2(\ell) \rceil} - 1\}$ **do**
18: $\quad$ Set $[\![\mathbf{g}_j]\!]_d = [\![\boldsymbol{\rho}_j]\!]_d$.
19: **end for**
20: Compute $[\![\mathbf{H}]\!]_{2d} = [\![\mathbf{g}]\!]_d \cdot [\![\mathbf{M}]\!]_d + [\![\mathbf{0}]\!]_{2d}$ and reveal $\mathbf{H} \in \mathbb{F}_p^{k \times 2^{\lceil \log_2(\ell) \rceil}}$.
21: Set $\boldsymbol{\theta}'[i] = 1$ if there is a zero in i-th row of $\mathbf{H}$ and $\boldsymbol{\theta}'[i] = 0$ otherwise, $i \in [k]$.
22: $[\![\mathbf{b}]\!]_d = [\![\boldsymbol{\theta}]\!]_d + [\![\boldsymbol{\theta}']\!]_{k-1} - 2[\![\boldsymbol{\theta}']\!]_{k-1} [\![\boldsymbol{\theta}]\!]_d$. ▷ $\mathbf{b} = \boldsymbol{\theta} \oplus \boldsymbol{\theta}'$.
23: **Optimized version:** $[\![\mathbf{b}]\!]_d = [\![\boldsymbol{\theta}]\!]_d + [\![\boldsymbol{\theta}']\!]_{k-1} + 2\sum_{i=0}^{k-1} [\![\mathbf{e}_i]\!]_{k-1} \cdot [\boldsymbol{\theta}[i]|_{\boldsymbol{\beta}[i]}]_t \cdot \boldsymbol{\theta}'[i]$.
24: **return** $[\![\mathbf{b}]\!]_d$.

Cost($\Pi^{\mathsf{PSS}}_{\mathsf{OUCP}}$). The amortized offline cost of $\Pi^{\mathsf{PSS}}_{\mathsf{OUCP}}$ is **Cost**($\Pi^{\mathsf{PSS}}_{\mathsf{RSM}} + \Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$) + $\frac{n}{k(n-t)}(3+2^{\lceil \log_2(\ell) \rceil}+4\ell)+\frac{4}{k}$ field elements amortized per party in $\lceil \log_2(\ell) \rceil + 2$ rounds. In the online phase, the protocol incurs $\frac{(6\ell+2^{\lceil \log_2(\ell) \rceil+1}+2)}{k}$ field elements amortized per party in 3 rounds. In contrast, $\Pi^{\mathsf{PSS}}_{\mathsf{UCP}}$ requires 5 rounds of online communication.

Cost($\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$). The amortized offline cost is $\mathsf{cost}(\Pi^{\mathsf{PSS\text{-}offline}}_{\mathsf{OUCP}} + \Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}) + \frac{2n}{(n-t)k}$ field elements per party in $\lceil \log_2(\ell) \rceil + 2$ rounds. Similar to Π_{CMP}, although $\Pi^{\mathsf{PSS}}_{\mathsf{OUCP}}$ nominally requires 3 rounds of communication in the online phase, the multiplication $2 \cdot [\theta]_d \cdot [x[i]]_d$ in step 11 (**Algorithm** 9) can be precomputed in the offline phase when the ℓ input bit shares $[\![\mathbf{x}_j]\!]_d = [\![\mathbf{r}_j]\!]_d, j \in [\ell]$ are generated. Hence, the online cost only needs $\frac{(4\ell+2^{\lceil \log_2(\ell) \rceil+1}+k+3)}{k}$ field elements amortized per party in 4 rounds.

Algorithm 10 Packed Comparison $\Pi_{\mathsf{CMP}}^{\mathsf{PSS}}$

1: **Input:** $[\![\mathbf{x}]\!]_d, [\![\mathbf{y}]\!]_d$.
2: **Output:** $[\![\mathbf{b}]\!]_d, \mathbf{b}[i] = (\mathbf{x}[i] \geq \mathbf{y}[i])$ is a bit, $i \in [k]$.
3: **Offline:**
4: Get random bits shares $([\![\mathbf{r}_i]\!]_d, \{[\mathbf{r}_i[j]|_{\boldsymbol{\beta}[j]}]_t\}_{j=0}^{k-1}) \leftarrow \Pi_{\mathsf{RandomBits}}^{\mathsf{Mixed}}$, where $i \in [\ell], \mathbf{r}_i$ is bit string of length of k.
5: Compute $[\![\mathbf{r}]\!]_d = \sum_{i=0}^{\ell-1} 2^i \cdot [\![\mathbf{r}_i]\!]_d$.
6: Run the offline phase of $\Pi_{\mathsf{OUCP}}^{\mathsf{PSS}}$.
7: **Online:**
8: Compute $[\![\mathbf{z}]\!]_d = 2([\![\mathbf{x}]\!]_d - [\![\mathbf{y}]\!]_d) + [\![\mathbf{r}]\!]_d$ and reveal $\mathbf{z}$.
9: Get $[\![\mathbf{v}]\!]_d \leftarrow \Pi_{\mathsf{OUCP}}^{\mathsf{PSS}}([\![\mathbf{r}_i]\!]_d, \{[\mathbf{r}_i[j]|_{\boldsymbol{\beta}[j]}]_t\}_{j=0}^{k-1}, \mathbf{z}), i \in [\ell]$.
10: Set $[\![\mathbf{u}]\!]_d = \mathrm{LSB}(\mathbf{z}) \oplus [\![\mathbf{r}_0]\!]_d = [\![\mathbf{w}_0]\!]_d$. ▷ $[\![\mathbf{w}_0]\!]_d$ is from step 13 of $\Pi_{\mathsf{OUCP}}^{\mathsf{PSS}}$.
11: Compute $[\![\mathbf{b}]\!]_d = 1 - [\![\mathbf{u}]\!]_d \oplus [\![\mathbf{v}]\!]_d = 1 - [\![\mathbf{u}]\!]_d - [\![\mathbf{v}]\!]_d + 2 \cdot [\![\mathbf{u}]\!]_d \cdot [\![\mathbf{v}]\!]_d$.
12: **return** $[\![\mathbf{b}]\!]_d$.

5.1.2 Packed Private Lookup Table Protocol

For the packed private lookup table protocol $\Pi_{\mathsf{PLUT}}^{\mathsf{PSS}}$ in **Algorithm** 11, we follow the basic steps of **Algorithm** 7 by replacing SSS with PSS. Let $\mathbf{e}_i \in \{0,1\}^k$ denote the one-hot vector whose i-th entry equals 1. Since each $[\![\mathbf{r}_i]\!]_d, i \in [m]$ packs k random one-hot vector, we need to perform k cyclic shifts $\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{m-1}$ in online step 9 and then recombine the resulting packed sharings using $\mathbf{e}_i$ in online step 15 to obtain the final correct output.

However, the above process incurs extra k packed DN multiplications (Π_{PDN}) and a single invocation of the degree reduction operation ($\Pi_{\mathsf{DegreeReduce}}$ in Appendix A). To further reduce the online communication cost, we observe that given the degree-t SSS $\{[\mathbf{e}_{\mathbf{v}[i]}|_{\boldsymbol{\beta}[i]}]_t\}_{i=0}^{k-1}$ of the vectors $(\mathbf{e}_{\mathbf{v}[0]}, \mathbf{e}_{\mathbf{v}[1]}, \cdots, \mathbf{e}_{\mathbf{v}[k-1]})$, we can compute $[\![\langle \mathbf{T}_i, \mathbf{e}_{\mathbf{v}[i]} \rangle]\!]_{2d} = \sum_{i=0}^{k-1} [\![\mathbf{e}_i]\!]_{k-1} \cdot [\![\mathbf{T}]\!]_d \cdot [\mathbf{e}_{\mathbf{v}[i]}|_{\boldsymbol{\beta}[i]}]_t$ locally. Then we only need to perform degree reduction on $[\![\langle \mathbf{T}_i, \mathbf{e}_{\mathbf{v}[i]} \rangle]\!]_{2d}$. We present our optimized protocol in **Algorithm** 12. In addition, we show the PSS variant $\Pi_{\mathsf{ROH}}^{\mathsf{PSS}}$ of Π_{ROH} in Appendix A.

Cost($\Pi_{\mathsf{OPLUT}}^{\mathsf{PSS}}$). The amortized offline cost of $\Pi_{\mathsf{OPLUT}}^{\mathsf{PSS}}$ is $\frac{2n}{(n-t)k}$ + **Cost**(Π_{ROH} + $\Pi_{\mathsf{CMP}}^{\mathsf{PSS-offline}}$) field elements per party in $\lceil \log_2(\ell) \rceil + 2$ rounds. The online cost needs $\frac{(4\ell + 2^{\lceil \log_2(\ell) \rceil + 1} + 2k + 6)}{k}$ field elements amortized per party in 6 rounds.

Algorithm 11 Packed Private Lookup Table $\Pi^{\mathsf{PSS}}_{\mathsf{PLUT}}$

Input: $\{[\![\mathbf{T}[i]]\!]_d\}_{i=0}^{m-1}, [\![\mathbf{u}]\!]_d$, where $\mathbf{u}[i] \in [m], \mathbf{T}[i]$ denotes the collection of the i-th item of all k lookup table $\mathbf{T}_j, j \in [k]$, each table is of size m, and $m < |\mathbb{F}_p|$.
Output: $[\![\mathbf{T}_0[\mathbf{u}[0]], \mathbf{T}_1[\mathbf{u}[1]], \cdots, \mathbf{T}_{k-1}[\mathbf{u}[k-1]]]\!]_d$.
Offline:
Get randomness $(\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{m}, [\![\mathbf{v}]\!]_d) \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{ROH}}(\log_2 m)$.
Run the offline phase of $\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$.
Online:
Compute $[\![\mathbf{b}]\!]_d \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{CMP}}([\![\mathbf{v}]\!]_d, [\![\mathbf{u}]\!]_d)$.
Compute $[\![\mathbf{w}]\!]_d = [\![\mathbf{b}]\!]_d \cdot ([\![\mathbf{v}]\!]_d - [\![\mathbf{u}]\!]_d) + (1 - [\![\mathbf{b}]\!]_d) \cdot (m + [\![\mathbf{v}]\!]_d - [\![\mathbf{u}]\!]_d)$ and reveal $\mathbf{w}$.
for $\{i = 0, 1, \cdots, k-1\}$ **do**
    for $\{j = 0, 1, \cdots, m-1\}$ **do**
        $[\![\hat{\mathbf{r}_j}]\!]_d = [\![\mathbf{r}_{(\mathbf{w}[i]+j) \bmod m}]\!]_d$.
    end for
    $[\![\mathbf{res}_i]\!]_d = \sum_{j=0}^{m-1} [\![\hat{\mathbf{r}_j}]\!]_d \cdot [\![\mathbf{T}[j]]\!]_d$. ▷ Π_{PDN}.
end for
Compute $[\![\mathbf{res}]\!]_{d+k-1} = \sum_{i=0}^{k-1} [\![\mathbf{res}_i]\!]_d \cdot [\![\mathbf{e}_i]\!]_{k-1}$.
Get $[\![\mathbf{res}]\!]_d \leftarrow \Pi_{\mathsf{DegreeReduce}}([\![\mathbf{res}]\!]_{d+k-1})$.
return $[\![\mathbf{res}]\!]_d$.

Algorithm 12 $\Pi^{\mathsf{PSS}}_{\mathsf{OPLUT}}$

Input: $\{[\![\mathbf{T}[i]]\!]_d\}_{i=0}^{m-1}, [\![\mathbf{u}]\!]_d$, where $\mathbf{u}[i] \in [m]$ and $m < |\mathbb{F}_p|$.
Output: $[\![\mathbf{T}_0[\mathbf{u}[0]], \mathbf{T}_1[\mathbf{u}[1]], \cdots, \mathbf{T}_{k-1}[\mathbf{u}[k-1]]]\!]_d$.
Offline:
Get k SSS random one-hot vector $\{([\mathbf{e}_{\mathbf{v}[j]}|_{\boldsymbol{\beta}[j]}]_t, [\mathbf{v}[j]|_{\boldsymbol{\beta}[j]}]_t)\}_{j=0}^{k-1} \leftarrow \Pi_{\mathsf{ROH}}(\log_2 m)$, where $\mathbf{e}_{\mathbf{v}[j]}$ is a size-m one-hot vector with a 1 at position $\mathbf{v}[j]$ and 0 elsewhere, $\mathbf{v}$ is a size-k vector.
Compute $[\![\mathbf{v}]\!]_d = \sum_{j=0}^{k-1} [\![\mathbf{e}_j]\!]_{k-1} \cdot [\mathbf{v}[j]|_{\boldsymbol{\beta}[j]}]_t$. ▷ $d = t - k + 1$.
Run the offline phase of $\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$.
Online:
Compute $[\![\mathbf{b}]\!]_d \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{CMP}}([\![\mathbf{v}]\!]_d, [\![\mathbf{u}]\!]_d)$.
Compute $[\![\mathbf{r}]\!]_d = [\![\mathbf{b}]\!]_d \cdot ([\![\mathbf{v}]\!]_d - [\![\mathbf{u}]\!]_d) + (1 - [\![\mathbf{b}]\!]_d) \cdot (m + [\![\mathbf{v}]\!]_d - [\![\mathbf{u}]\!]_d)$ and reveal $\mathbf{r}$.
for $\{i = 0, 1, \cdots, k-1\}$ **do**
    for $\{j = 0, 1, \cdots, m-1\}$ **do**
        $[\hat{\mathbf{e}}_j|_{\boldsymbol{\beta}[i]}]_t = [\mathbf{e}_{(\mathbf{r}[i]+j) \bmod m}|_{\boldsymbol{\beta}[i]}]_t$.
    end for
    $[\![\mathbf{res}_i]\!]_{d+t} = \sum_{j=0}^{m-1} [\![\hat{\mathbf{e}}_j]\!]_t \cdot [\![\mathbf{T}[j]]\!]_d$.
end for
Compute $[\![\mathbf{res}]\!]_{2d} = \sum_{i=0}^{k-1} [\![\mathbf{res}_i]\!]_{d+t} \cdot [\![\mathbf{e}_i]\!]_{k-1}$.
Get $[\![\mathbf{res}]\!]_d \leftarrow \Pi_{\mathsf{DegreeReduce}}([\![\mathbf{res}]\!]_{2d})$.
return $[\![\mathbf{res}]\!]_d$.

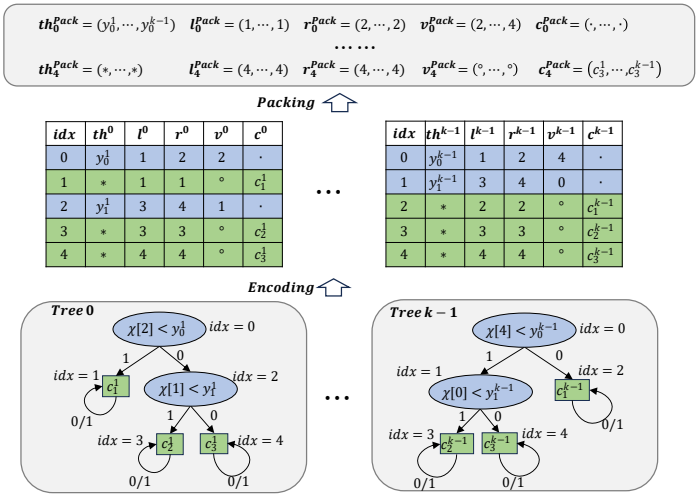


Fig. 6 A simple example for k trees. Each tree has 5 nodes.

5.2 PDTE For Large-Scale Tree Model

When dealing with large-scale tree models, such as random forests, a large number of trees must be evaluated. Since each tree can be evaluated independently, we can use PSS for parallel processing. Every k trees are packed into a multi-dimensional array $\mathcal{T}$. Figure 6 depicts a simple example. It is worth noting that not all trees contain the same number of nodes. Therefore, we take the tree with the largest number of nodes as the reference and pad the remaining trees with randomly generated dummy nodes so that all trees have the same number of nodes before packing. The padding strategy is shown in Figure 7. At the same time, we pack k replicated copies of each element of the feature vector $\mathcal{X}$ into a PSS value. Similar to protocol Π_{SPDTE}, we need to pad the input to the nearest power-of-two length. We present our packed scalable PDTE protocol in **Algorithm** 13.

Cost($\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$). The offline cost is primarily determined by $\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$ and $\Pi^{\mathsf{PSS}}_{\mathsf{OPLUT}}$, requiring $O\big(\mathcal{D}\cdot\mathsf{cost}(\Pi^{\mathsf{PSS\text{-}offline}}_{\mathsf{CMP}}+\Pi^{\mathsf{PSS\text{-}offline}}_{\mathsf{OPLUT}})\big)$ field elements per party in $O(\log_2\ell)$ rounds. In the online phase, the protocol incurs $\frac{(19+6k+12\ell+3\cdot 2^{\lceil\log_2(\ell)\rceil+1})}{k}\cdot\mathcal{D}$ field elements per party over $17\cdot\mathcal{D}$ rounds.

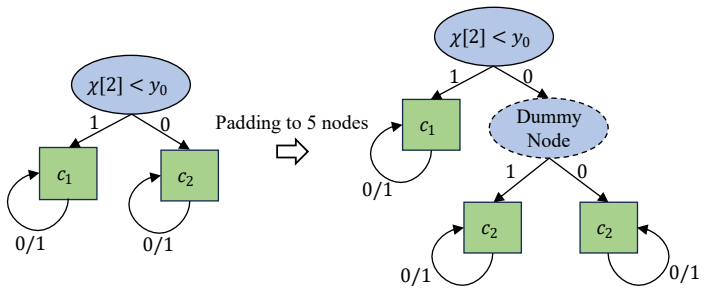


Fig. 7 A simple example for dummy nodes padding.

Algorithm 13 Packed Scalable PDTE $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$

Input: $\{[\![\mathcal{T}[i]]\!]_d\}_{i=0}^{\mathcal{M}-1}, \{[\![\mathcal{X}_u]\!]_d\}_{u=0}^{\mathcal{N}-1}$, where $\mathcal{T}[i]$ denotes the collection of the i-th node of all k trees $\mathcal{T}_j, j \in [k]$ and $\mathcal{X}_u$ denotes a vector consisting of k copies of the u-th attribute.
Output: $[\![\mathbf{res}]\!]_d$, **res** contains the evaluation result of k DT.
Offline:
Run the offline phase of $\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$ and $\Pi^{\mathsf{PSS}}_{\mathsf{OPLUT}}$.
Online:
Initialize $[\![\mathbf{res}]\!]_d = 0$.
Parse $([\![\mathbf{th}]\!]_d, [\![\mathbf{l}]\!]_d, [\![\mathbf{r}]\!]_d, [\![\mathbf{v}]\!]_d, [\![\mathbf{c}]\!]_d) \leftarrow [\![\mathcal{T}[0]]\!]_d$.
for $\{i = 0, 1, \cdots, \mathcal{D} - 1\}$ **do**
    $[\![\mathcal{X}_{\mathbf{v}[0]}, \cdots, \mathcal{X}_{\mathbf{v}[k-1]}]\!]_d \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{OPLUT}}(\{[\![\mathcal{X}_u]\!]_d\}_{j=0}^{\mathcal{N}-1}, [\![\mathbf{v}]\!]_d)$.
    $[\![\mathbf{b}]\!]_d \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{CMP}}([\![\mathbf{th}]\!]_d, [\![\mathcal{X}_{\mathbf{v}[0]}, \cdots, \mathcal{X}_{\mathbf{v}[k-1]}]\!]_d)$.
    $[\![\mathbf{idx}]\!]_d = (1 - [\![\mathbf{b}]\!]_d) \cdot [\![\mathbf{r}]\!]_d + [\![\mathbf{b}]\!]_d \cdot [\![\mathbf{l}]\!]_d$.
    $[\![\mathcal{T}[\mathbf{idx}[0]], \cdots, \mathcal{T}[\mathbf{idx}[k-1]]]\!]_d \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{OPLUT}}(\{[\![\mathcal{T}[i]]\!]_d\}_{i=0}^{\mathcal{M}-1}, [\![\mathbf{idx}]\!]_d)$.
    $([\![\mathbf{th}]\!]_d, [\![\mathbf{l}]\!]_d, [\![\mathbf{r}]\!]_d, [\![\mathbf{v}]\!]_d, [\![\mathbf{c}]\!]_d) \leftarrow [\![\mathcal{T}[\mathbf{idx}[0]], \cdots, \mathcal{T}[\mathbf{idx}[k-1]]]\!]_d$.
    Update $[\![\mathbf{res}]\!]_d = [\![\mathbf{c}]\!]_d$.
end for
return $[\![\mathbf{res}]\!]_d$.

6 Experiment

In this section, we implement the proposed protocol and compare the performance of our comparison protocol with that of Helix [24]. In addition, to demonstrate the scalability of our PDTE framework, we further evaluate the performance of the proposed SSS-based PDTE and PSS-based PDTE in large-scale multi-party settings.

6.1 Experiments Environment

We implement our protocol on top of hmmpc in C++ and run our experiments on 3 cloud servers equipped with a 32-core processor with 64GB of RAM. We use the Eigen library to accelerate matrix multiplication and parallelize the computations using multithreading. To simulate different network conditions, we use Linux Traffic Control (tc) to simulate wide-area network (WAN, RTT: 80 ms, 40 Mbps), metropolitan-area network (MAN, RTT: 6 ms, 100 Mbps), and local-area network (LAN, RTT: 0.1 ms, 1 Gbps). RTT denotes Round-Trip Time. We completed performance evaluations for constant-round comparison and PDTE in various settings involving different numbers of parties, including 9-party computation (9PC), 15PC and 21PC. All evaluations are executed 5 times, and the average is recorded.

6.2 Performance on Comparison

We first re-run Helix in the semi-honest setting, and compare its comparison protocol with the SS-based protocol Π_{CMP} in SC-DT under an adversary threshold of $t = (n-1)/2$, where n is the number of parties. We then evaluate both Π_{CMP} and $\Pi_{\mathsf{CMP}}^{\mathsf{PSS}}$ against Helix under different adversary corruption thresholds.

Adversary Threshold of $t = (n-1)/2$. The online running time of secure comparison for $\ell = 61$-bit inputs and 10000 comparison operations is depicted in Figure 8. In Table D1 of Appendix D, we present the runtime and amortized per-party communication costs under different network settings, input bit lengths, and numbers of parties. Compared with Helix, our protocol achieves improvements in online running time and online communication cost. Specifically, it reduces communication by 9% and 20% for $\ell = 31$ and $\ell = 61$-bit inputs, respectively. It also achieves a 19% and 13% speedup for online running time with $n = 21$ parties and $\ell = 61$-bit inputs under MAN and WAN settings, respectively. In particular, we observe that the performance improvement of our protocol becomes more significant as the input bit length and the

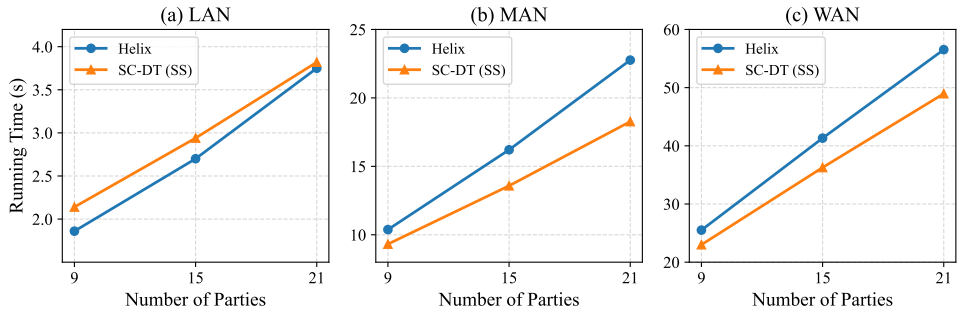


Fig. 8 Online running time comparison between Π_{CMP} and Helix [24] in LAN, MAN, and WAN settings with an adversary threshold of $(n-1)/2$, for $\ell = 61$-bit inputs and 10000 secure comparison operations.

number of parties increase, as shown in Figure 8. It is worth noting that our protocol performs less efficiently in the LAN setting and in the offline phase, mainly due to higher online computation overhead and increased offline communication cost.

Adversary Threshold of $t < (n-1)/2$. The online running time of secure comparison under an adversary threshold of $t = n/3$ for $\ell = 61$-bit inputs and 10000 comparison operations is depicted in Figure 9. In Table D2 - D5 of Appendix D, we present the runtime and amortized per-party communication costs across different network settings, input bit lengths, the number of parties, and adversary corruption thresholds. We observe that the performance of Π_{CMP} and Helix is similar to that under an adversary threshold of $t = (n-1)/2$; however, the PSS-based protocol $\Pi_{\mathsf{CMP}}^{\mathsf{PSS}}$ in SC-DT provides significant advantages. Specifically, compared with Helix, $\Pi_{\mathsf{CMP}}^{\mathsf{PSS}}$ reduces communication by $1.1-2.4\times$ and $1.7-2.5\times$ in offline and online, respectively. It also achieves $1.7-2.4\times$ and $1.7-2.3\times$ speedup for online running time under MAN and WAN settings, respectively. Even in the LAN setting, our protocol maintains $1.5\times$ online advantages with comparable offline time when the number of parties is $n = 21$. Furthermore, we observe that the benefits of our protocol become more significant as

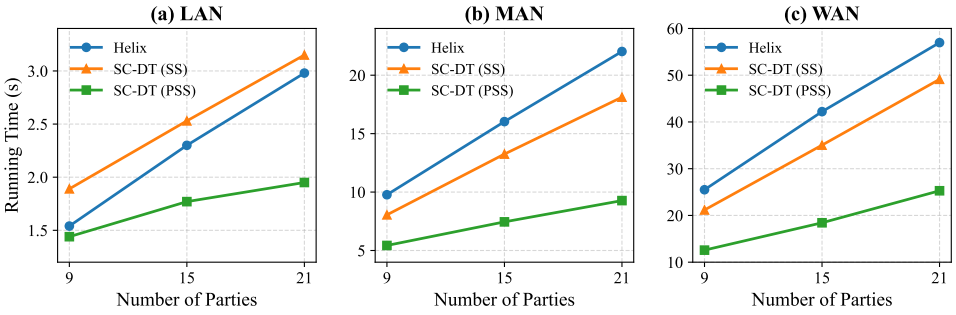


Fig. 9 Online running time comparison among Π_{CMP}, $\Pi^{\mathsf{PSS}}_{\mathsf{CMP}}$ and Helix [24] in LAN, MAN, and WAN settings with an adversary threshold of $n/3$, for $\ell = 61$-bit inputs and 10000 secure comparison operations.

the input bit length and the number of parties increase. In particular, as the number of participating parties increases, we can choose larger packing parameters, which further amplifies the performance gains.

6.3 Performance on PDTE

Datasets and Tree Models. We compare Π_{SPDTE} with $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$ under different adversary corruption thresholds for large-scale decision tree evaluation. More specifically, we use 3 representative datasets from the UCI repository (see Table 2) and directly adopt the pre-trained model from Kangaroo [8] for PDTE.

Table 2 Parameters of Datasets.

Dataset	Depth $\mathcal{D}$	Features $\mathcal{N}$	Nodes $\mathcal{M}$
digits	15	47	337
diabetes	28	10	787
boston	30	13	851

Performance. We construct random forests with 50 trees using the Digits, Diabetes, and Boston datasets. The online running time of PDTE is depicted in Figure 10. In

Table D6 - D8 of Appendix D, we present the runtime and amortized per-party communication costs under different network settings and adversary corruption thresholds. Compared with Π_{SPDTE}, $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$ achieves a $1.2-1.4\times$ and $1.5-1.9\times$ reduction in offline and online communication, respectively. For $n=21$ and $t=7$, the runtime of the online phase is reduced by $27\%, 28\%$, and 39% for Digits, Diabetes, and Boston, respectively, under the LAN setting. The corresponding reductions are $33\%, 30\%$, and 31% in the MAN setting, and $17\%, 17\%$, and 15% in the WAN setting. For the offline phase, $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$ exhibits no runtime benefit over Π_{SPDTE} in the LAN setting, owing to its greater computational overhead. In the MAN and WAN settings, however, $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$ is slightly faster than Π_{SPDTE}, as its communication overhead is much lower. Furthermore, we observe that for $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$, given n, a larger k does not necessarily lead to better performance. This can be attributed to two main factors. First, increasing k incurs higher computational overhead. Second, a larger k reduces the number of adversaries that the protocol can tolerate. In fact, when the number of adversaries is small, Π_{SPDTE} tends to be more advantageous. Nevertheless, the advantages of our protocol become more pronounced as the number of participating parties grows, since a larger n allows for the use of a larger packing parameter k, which further amplifies the performance gains from communication reduction.

7 Related Work

In this section, we organize the related work into two categories: secure comparison and privacy decision tree evaluation.

7.1 Secure Comparison

We review the representative works as follows: i) Secure comparison was initially realized using GC [37]. Since then, numerous secure comparison protocols [38–40]

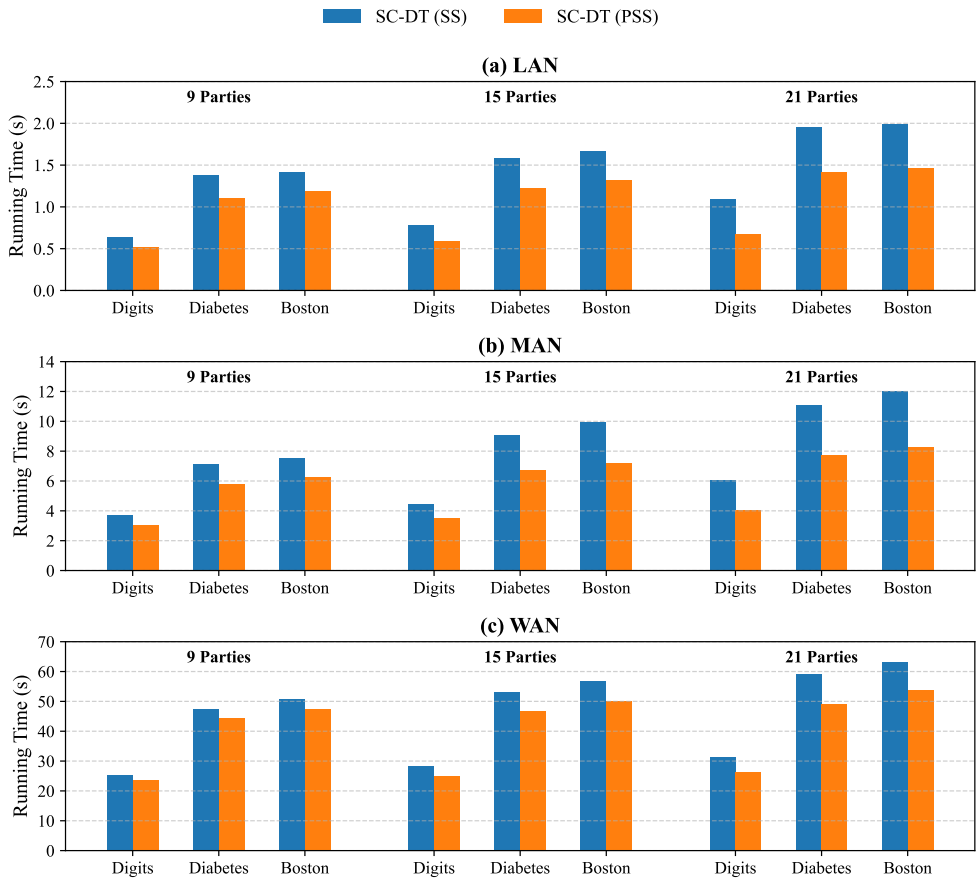


Fig. 10 Online running time comparison between Π_{SPDTE} and $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$ in LAN, MAN, and WAN settings with an adversary threshold of $n/3$, for $\ell = 61$-bit inputs and random forests consisting of 50 trees trained on the Digits, Diabetes, and Boston datasets.

have continued to adopt GC as their underlying technique, while leveraging optimizations including free-XOR [41] and point-and-permute [42] to improve efficiency. More recently, WINK [26] proposed a scalable comparison protocol based on GC. Although GC-based protocols achieve constant-round complexity, they generally incur substantial communication overhead. ii) FSS [14] enables parties to locally evaluate a function using correlated keys and obtain secret-shared outputs, typically relying on a dealer to generate the corresponding keys. In particular, the distributed comparison

function (DCF) [43–45] has been widely used to realize secure comparison protocols. More recently, Mosformer [20] leveraged the distributed point function (DPF) to construct a constant-round secure comparison protocol in the three-party setting. However, existing FSS-based approaches generally suffer from limited scalability, as the key generation grows significantly with the number of parties. iii) SecureNN [29] and a series of works [19, 21, 46] combine random shuffling with the DGK-style comparison protocol [47] to achieve constant-round secure comparison. In addition, Bicoptor [48] employs random shuffling together with probabilistic truncation to construct a constant-round comparison protocol. Although these protocols achieve constant-round complexity, they are primarily designed for two/three-party settings and do not scale naturally to a large number of parties. iv) CrypTFlow2 [49] and subsequent works such as Cheetah [50] achieve secure comparison by recursively comparing sub-integers using OT, resulting in logarithmic-round complexity. v) Lu et al. [51] encodes the integers to be compared into the degree of a polynomial and leverages homomorphic encryption (HE) to perform secure comparison. vi) Many work [52, 53] realize secure comparison or MSB extraction by first converting arithmetic shares into boolean shares, performing bit-wise operations, and then converting the result back to arithmetic shares. These approaches typically incur logarithmic-round communication complexity. vii) Falcon [30] and Rabbit [54] employ random masking techniques: they mask the secret value with a random share, reveal the masked value, and then derive the comparison result from the public masked value and the secret mask. These protocols also achieve logarithmic-round complexity. However, they are primarily designed for a small number of parties and do not scale naturally to large multi-party settings. viii) Octavian et al. [18] proposed a constant-round and scalable secure comparison protocol based on Shamir secret sharing over finite fields. However, their construction requires a large gap between the secret domain and the share domain, resulting in substantial online communication overhead. More recently, Helix [24] leverages the

DN multiplication protocol to construct a four-input multiplication gate, achieving $O(\log_4(\ell))$ rounds secure comparison, where ℓ is the bit length of input. As a result, it offers a more favorable trade-off between online round complexity and communication cost.

7.2 Privacy Decision Tree Evaluation

There are currently many works on PDTE, mainly divided into two categories: Constant-round PDTE and $O(\mathcal{D})$ rounds PDTE, where $\mathcal{D}$ is maximum depth of the tree.

Constant-round PDTE. Constant-round PDTE protocols require comparisons to be performed at all decision nodes in the tree, after which the final prediction is derived via Path Conjugation or SumPath [7]. On the one hand, a line of work [7, 51, 55, 56] has developed non-interactive PDTE protocols based on HE, and further leveraged SIMD techniques to improve throughput for large-scale decision tree evaluation [8]. However, these HE-based approaches incur substantial computational costs, which limit their practical scalability. On the other hand, benefiting from the communication-efficient comparisons enabled by DCF [14], Cheng et al. [57] proposed a constant-round PDTE protocol in the 3PC setting based on replicated secret sharing (RSS) and DCF. Nevertheless, the dependence on DCF introduces a scalability bottleneck in the protocol.

$O(\mathcal{D})$ rounds PDTE. $O(\mathcal{D})$-round PDTE protocols start from the root node and iteratively perform a secure comparison at each level, obliviously selecting the next node according to the comparison result until a leaf node is reached. In the 2PC setting, Tueno et al. [58] utilize GC to implement secure comparisons, and employ OT and ORAM to realize oblivious array indexing (OAI). This OAI primitive is used for both feature selection and path selection during decision tree evaluation. Bai et

al. [10] achieve small communication overhead for feature selection and path evaluation, but the computational overhead remains high due to complex cryptographic operations. The state-of-the-art work [9] utilizes OT for feature selection and GC for threshold comparison. Subsequently, some protocols are proposed in the 3PC setting. Mostree [15] proposes two low communication oblivious selection (OS) protocols by leveraging the advantages of RSS and DCF, and further extends them to the malicious-adversary setting. The recent work [16] utilizes shuffle to reduce the communication cost of both feature selection and path selection. Despite their high efficiency, the above protocols remain limited in scalability.

8 Conclusion & Future Work

In this paper, we propose a scalable constant-round secure comparison protocol and further develop a PSS-based variant. Additionally, building upon PSS, we present a parallel and scalable PLUT protocol, which can be combined with secure comparison to enable scalable PDTE. Extensive evaluations also present our improvements.

Future Work. In this paper, we focus primarily on the semi-honest setting. An important direction for future research is to extend our protocols to achieve security against malicious adversaries. Another promising direction is to generalize our constant-round secure comparison techniques to support more complex nonlinear functions, such as Softmax and GeLU, which are widely used in modern neural networks and large language models.

References

[1] Podgorelec, V., Kokol, P., Stiglic, B., Rozman, I.: Decision trees: An overview and their use in medicine. Journal of Medical Systems **26**, 445–463 (2002)

[2] Koh, H.C., Tan, W.C., Goh, C.P.: A two-step method to construct credit scoring

models with data mining techniques. The International Journal of Business and Information **1** (2006)

[3] Yao, A.C.-C.: Protocols for secure computations. 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), 160–164 (1982)

[4] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. Proceedings of the nineteenth annual ACM symposium on Theory of computing (1987)

[5] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. Proceedings of the twentieth annual ACM symposium on Theory of computing (1988)

[6] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. Proceedings of the twentieth annual ACM symposium on Theory of computing (1988)

[7] Mahdavi, R.A., Ni, H., Linkov, D., Kerschbaum, F.: Level up: Private non-interactive decision tree evaluation using levelled homomorphic encryption. Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (2023)

[8] Xu, W., Zhu, H., Zheng, Y., Bian, S., Sun, N., Yuan, H., Feng, D., Li, H.: Kangaroo: A private and amortized inference framework over wan for large-scale decision tree evaluation. Proceedings 2025 Network and Distributed System Security Symposium (2025)

[9] Ma, J.P.K., Tai, R.K.H., Zhao, Y., Chow, S.S.M.: Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. Proceedings 2021 Network and Distributed System Security Symposium (2021)

[10] Bai, J., Song, X., Cui, S., Chang, E.-C., Russello, G.: Scalable private decision tree evaluation with sublinear communication. Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (2022)

[11] Zhang, Q., Chen, X., Chen, X., Zhao, X., Jing, W., Fan, T.: Oceptree: An online communication efficient privacy-preserving decision tree evaluation. In: Security and Privacy in Communication Networks (2024). https://api.semanticscholar.org/CorpusID:285313784

[12] Ji, K., Zhang, B., Lu, T., Li, L., Ren, K.: Uc secure private branching program and decision tree evaluation. IEEE Transactions on Dependable and Secure Computing **20**, 2836–2848 (2023)

[13] Cheng, N., Gupta, N., Mitrokotsa, A., Morita, H., Tozawa, K.: Constant-round private decision tree evaluation for secret shared data. Proc. Priv. Enhancing Technol. **2024**, 397–412 (2023)

[14] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (2016)

[15] Bai, J., Song, X., Zhang, X., Wang, Q., Cui, S., Chang, E.-C., Russello, G.: Mostree: Malicious secure private decision tree evaluation with sublinear communication. Proceedings of the 39th Annual Computer Security Applications Conference (2023)

[16] Lu, T., Zhang, B., Li, H., Ren, K.: A new paradigm for privacy-preserving decision tree evaluation. IACR Cryptol. ePrint Arch. **2025**, 1447 (2025)

[17] Sangaiah, A.K., Medhane, D.V., Han, T., Hossain, M.S., Muhammad, G.: Enforcing position-based confidentiality with machine learning paradigm through

mobile edge computing in real-time industrial informatics. IEEE Transactions on Industrial Informatics **15**, 4189–4196 (2019)

[18] Catrina, O., Hoogh, S.: Improved primitives for secure multiparty integer computation. In: International Conference on Security and Cryptography for Networks (2010). https://api.semanticscholar.org/CorpusID:7104878

[19] Lu, T., Kang, X., Zhang, B., Ma, Z., Zhang, X., Liu, Y., Ren, K., Chen, C.: Efficient {2PC} for constant round secure equality testing and comparison. In: 34th USENIX Security Symposium (USENIX Security 25), pp. 1985–2004 (2025)

[20] Cheng, K., Xia, Y., Song, A., Fu, J., Qu, W., Shen, Y., Zhang, J.: Mosformer: Maliciously secure three-party inference framework for large transformers. Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (2025)

[21] Dong, Y., Chen, X., Song, X., Yang, Y., Zhang, T., Dong, J.-S.: Mizar: Boosting secure three-party deep learning with co-designed sign-bit extraction and gpu acceleration. 2025 IEEE Annual Computer Security Applications Conference (ACSAC), 905–921 (2025)

[22] Yuan, B., Yang, S., Zhang, Y., Ding, N., Gu, D., Sun, S.: Md-ml: Super fast privacy-preserving machine learning for malicious security with a dishonest majority. In: USENIX Security Symposium (2024). https://api.semanticscholar.org/CorpusID:271325612

[23] Zhang, Y., Chen, X., Dong, Y., Zhang, Q., Hou, R., Liu, Q., Chen, X.: Md-sonic: Maliciously-secure outsourcing neural network inference with reduced online communication. IEEE Transactions on Information Forensics and Security **20**, 3534–3549 (2025)

[24] Zhang, Y., Chen, X., Zhang, Q., Dong, Y., Chen, X.: Helix: Scalable multi-party machine learning inference against malicious adversaries. IACR Cryptol. ePrint Arch. **2025**, 347 (2025)

[25] Yang, M., Du, J., Zhou, Z., Guo, D., Meng, Q.: MCU: Algebraic and Transcendental Function Evaluation in MPC without Preprocessing. Cryptology ePrint Archive, Paper 2026/750 (2026). https://eprint.iacr.org/2026/750

[26] Feng, Y., Bai, T., Lu, Z., Wu, J., Tang, X., Lu, S.: Wink:a semi-honest secure multi-party computation framework with efficient comparison protocol. 2025 44th International Symposium on Reliable Distributed Systems (SRDS), 380–391 (2025)

[27] Shamir, A.: How to share a secret (1979). (2021). https://api.semanticscholar.org/CorpusID:234065255

[28] Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: Symposium on the Theory of Computing (1992). https://api.semanticscholar.org/CorpusID:7486402

[29] Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proceedings on Privacy Enhancing Technologies **2019**, 26–49 (2019)

[30] Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies **2021**, 188–208 (2020)

[31] Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference (2007). https://api.semanticscholar.org/CorpusID:14441263

[32] Goyal, V., Polychroniadou, A., Song, Y.: Sharing transformation and dishonest majority mpc with packed secret sharing. IACR Cryptol. ePrint Arch. **2022**, 831 (2022)

[33] Kiltz, E.: Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. In: IACR Cryptology ePrint Archive (2006). https://api.semanticscholar.org/CorpusID:8721130

[34] Zhang, Q., Chen, X., Zhang, Y., Chen, X.: High-throughput and scalable secure inference protocols for deep learning with packed secret sharing. ArXiv **abs/2601.13041** (2026)

[35] Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Theory of Cryptography Conference (2005). https://api.semanticscholar.org/CorpusID:40617859

[36] Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster secure multi-party computation of aes and des using lookup tables. In: International Conference on Applied Cryptography and Network Security (2017). https://api.semanticscholar.org/CorpusID:24162013

[37] Yao, A.C.-C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986), pp. 162–167 (1986). IEEE

[38] Demmler, D., Schneider, T., Zohner, M.: Aby - a framework for efficient mixed-protocol secure two-party computation. In: Network and Distributed System Security Symposium (2015). https://api.semanticscholar.org/CorpusID:14639818

[39] Patra, A., Schneider, T., Suresh, A., Yalame, H.: Aby2.0: Improved mixed-protocol secure two-party computation. IACR Cryptol. ePrint Arch. **2020**, 1225 (2020)

[40] Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. 2017 IEEE Symposium on Security and Privacy (SP), 19–38 (2017)

[41] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. In: International Colloquium on Automata, Languages and Programming (2008). https://api.semanticscholar.org/CorpusID:7746167

[42] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security Symposium (2004). https://api.semanticscholar.org/CorpusID:5515370

[43] Wagh, S.: Pika: Secure computation using function secret sharing over rings. Proceedings on Privacy Enhancing Technologies (2022)

[44] Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: Fss-based secure training and inference with gpus. 2024 IEEE Symposium on Security and Privacy (SP), 597–616 (2024)

[45] Gupta, K., Chandran, N., Gupta, D., Katz, J., Sharma, R.: Shark: Actively secure inference using function secret sharing. 2025 IEEE Symposium on Security and Privacy (SP), 2472–2490 (2025)

[46] Lu, T., Zhang, B., Li, L., Ren, K.: Aegis: A lightning fast privacy-preserving machine learning platform against malicious adversaries. IACR Cryptol. ePrint Arch. **2023**, 1890 (2023)

[47] Damgård, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for online auctions. In: Australasian Conference on Information Security and Privacy (2007). https://api.semanticscholar.org/CorpusID:778897

[48] Zhou, L., Wang, Z., Cui, H., Song, Q., Yu, Y.: Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 534–551 (2023). IEEE

[49] Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020)

[50] Huang, Z., Lu, W.-j., Hong, C., Ding, J.: Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 809–826 (2022)

[51] Lu, W.-j., Zhou, J., Sakuma, J.: Non-interactive and output expressive private comparison from homomorphic encryption. Proceedings of the 2018 on Asia Conference on Computer and Communications Security (2018)

[52] 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)

[53] 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)

[54] Makri, E., Rotaru, D., Vercauteren, F., Wagh, S.: Rabbit: Efficient comparison for secure multi-party computation. In: International Conference on Financial Cryptography and Data Security, pp. 249–270 (2021). Springer

[55] Tueno, A., Boev, Y., Kerschbaum, F.: Non-interactive private decision tree evaluation. In: Database Security (2019). https://api.semanticscholar.org/CorpusID:202660615

[56] Cong, K., Das, D., Park, J., Pereira, H.V.L.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (2022)

[57] Cheng, N., Gupta, N., Mitrokotsa, A., Morita, H., Tozawa, K.: Constant-round private decision tree evaluation for secret shared data. Proceedings on Privacy Enhancing Technologies (2024)

[58] Tueno, A., Kerschbaum, F., Katzenbeisser, S.: Private evaluation of decision trees using sublinear cost. Proceedings on Privacy Enhancing Technologies **2019**, 266–286 (2019)

Appendix A The protocol $\Pi^{\mathsf{PSS}}_{\mathsf{ROH}}$, $\Pi^{\mathsf{PSS}}_{\mathsf{RSM}}$, $\Pi_{\mathsf{DegreeReduce}}$, and $\Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$

We present the $\Pi^{\mathsf{PSS}}_{\mathsf{ROH}}$ and $\Pi^{\mathsf{PSS}}_{\mathsf{RSM}}$in **Algorithm** 14 and **Algorithm** 15, respectively.

Cost($\Pi^{\mathsf{RSS}}_{\mathsf{ROH}}$). For generating a size$-2^m$ random one-hot vector: i) Step 3 needs $(\frac{3n}{k(n-t)} + \frac{4}{k})m$ field elements amortized per party in 2 rounds. ii) Step 6 needs $\frac{2(2^m-2)}{k} + \frac{2n(2^m-2)}{(n-t)k}$ field elements amortized per party in $m-1$ rounds. Therefore, the $\Pi^{\mathsf{RSS}}_{\mathsf{ROH}}$ needs $\frac{n}{k(n-t)}(3m+2(2^m-2)) + \frac{1}{k}(4m+2(2^m-2))$ field elements amortized per party in $m+1$ rounds for generating k size-2^m random one-hot vectors.

Cost($\Pi^{\mathsf{RSS}}_{\mathsf{RSM}}$). The communication cost of $\Pi^{\mathsf{PSS}}_{\mathsf{RSM}}$ is the same as that of $\Pi^{\mathsf{PSS}}_{\mathsf{ROH}}$.

We present the $\Pi_{\mathsf{DegreeReduce}}$ from [34] in **Algorithm** 16.

Algorithm 14 Packed Random One-Hot Vector $\Pi^{\mathsf{PSS}}_{\mathsf{ROH}}$

Input: m.
Output: $(\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{2^m-1}, [\![\mathbf{v}]\!]_d)$, where $\mathbf{r}_i = (\mathbf{e}_{\mathbf{v}[0]}[i], \mathbf{e}_{\mathbf{v}[1]}[i], \cdots, \mathbf{e}_{\mathbf{v}[k-1]}[i])$ and $\mathbf{e}_{\mathbf{v}[i]}, i \in [k]$ is a size-2^m one-hot vector with a 1 at position $\mathbf{v}[i]$ and 0 elsewhere.
Get $[\![\mathbf{b}_i]\!]_d \leftarrow \Pi_{\mathsf{RandomBits}}, i \in [m], \mathbf{b}_i$ is a bit string of length k, and then compute $[\![\mathbf{v}]\!]_d = \sum_{i=0}^{m-1} 2^i \cdot [\![\mathbf{b}_i]\!]_d$.
All parties set $\mathbf{u}^{(0)} = (1 - [\![\mathbf{b}_0]\!]_d, [\![\mathbf{b}_0]\!]_d)$.
for $\{i = 1, 2, \cdots, m-1\}$ **do**
    All parties compute $\mathbf{u}^{(i)} = ((1 - [\![\mathbf{b}_i]\!]_d) \cdot \mathbf{u}^{(i-1)}, [\![\mathbf{b}_i]\!]_d \cdot \mathbf{u}^{(i-1)})$.
end for
All parties set $\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{2^m-1} = \mathbf{u}^{(m-1)}$.
return $(\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{2^m-1}, [\![\mathbf{v}]\!]_d)$.

Algorithm 15 Packed Random Shift Matrix $\Pi^{\mathsf{PSS}}_{\mathsf{RSM}}$

Input: m.
Output: $[\![\mathbf{M}]\!]_d$, where the tensor $\mathbf{M}$ contains k random shift matrices and each matrix is of size $2^m \times 2^m$.
Get packed random one-hot vector $\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{2^m-1} \leftarrow \Pi^{\mathsf{PSS}}_{\mathsf{ROH}}(m)$.
All parties set $[\![\mathbf{M}[\cdot, 0, \cdot]]\!]_d = \{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{2^m-1}$.
for $\{i = 1, 2, \cdots, 2^m - 1\}$ **do**
    Get $\{[\![\mathbf{r}'_i]\!]_d\}_{i=0}^{2^m-1}$ by cyclically left shifting $\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{2^m-1}$ by i position locally and set $[\![\mathbf{M}[\cdot, i, \cdot]]\!]_d = \{[\![\mathbf{r}'_i]\!]_d\}_{i=0}^{2^m-1}$.
end for
return $[\![\mathbf{M}]\!]_d$.

Cost($\Pi_{\mathsf{DegreeReduce}}$). The communication complexity of the protocol is 2 round with $\frac{2n}{k}(\frac{1}{n-t} + \frac{1}{n})$ field elements amortized per party (resp. $2n(\frac{1}{n-t} + \frac{1}{n})$ field elements for Shamir secret sharing).

We present the simple $\Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$ in **Algorithm** 17. The parties generate a bits $r \in \{0, 1\}$ by computing $S(a) = \frac{\frac{a}{\sqrt{a^2}}+1}{2}$, where $a \in \mathbb{F}_p^*$ is a uniformly random non-zero element. $\sqrt{a^2}$ is formulated to equal the unique element in $[1, (p-1)/2]$ and $a/\sqrt{a^2}$ is 1 or -1. It can be further optimized using a Vandermonde matrix.

Cost($\Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$). The communication complexity of the protocol is 3 round with $\frac{n}{n-t}(\frac{2}{k} + 1 + k) + \frac{4}{k}$ field elements amortized per party.

Algorithm 16 Degree Reduce $\Pi_{\mathsf{DegreeReduce}}$

Input: $[\![\boldsymbol{x}]\!]_{d_1}$ (reps. $[x|_\beta]_{d_2}$), $d \le d_1 \le n-1, d-k+1 \le d_2 \le n-1$.
Output: $[\![\boldsymbol{x}]\!]_{d_3}$ (reps. $[x|_\beta]_{d_4}$), $d \le d_3 \le n-1$, $d-k+1 \le d_4 \le n-1, d_1 \ne d_3, d_2 \ne d_4$.
Get $([\![\boldsymbol{r}]\!]_{d_1}, [\![\boldsymbol{r}]\!]_{d_3}) \leftarrow \Pi_{\mathsf{Random}}$(resp. $([r|_\beta]_{d_2}, [r|_\beta]_{d_4}) \leftarrow \Pi_{\mathsf{rand}_p}$).
All parties compute $[\![\boldsymbol{x'}]\!]_{d_1} = [\![\boldsymbol{x}]\!]_{d_1} + [\![\boldsymbol{r}]\!]_{d_1}$ (resp. $[x'|_\beta]_{d_2} = [x|_\beta]_{d_2} + [r|_\beta]_{d_2}$) and send their shares to P_{king}.
P_{king} reconstructs the secret $\boldsymbol{x'}$ (resp. x') and distributes $[\![\boldsymbol{x'}]\!]_{d_3}$ (resp. $[x'|_\beta]_{d_4}$) to all parties.
All parties locally compute $[\![\boldsymbol{x}]\!]_{d_3} = [\![\boldsymbol{x'}]\!]_{d_3} - [\![\boldsymbol{r}]\!]_{d_3}$ (resp. $[x|_\beta]_{d_4} = [x'|_\beta]_{d_4} + [r|_\beta]_{d_4}$).
return $[\![\boldsymbol{x}]\!]_{d_3}$ (reps. $[x|_\beta]_{d_4}$).

Algorithm 17 Mixed RandomBits $\Pi^{\mathsf{Mixed}}_{\mathsf{RandomBits}}$

Input: m.
Output: $([\![\mathbf{r}]\!]_d, \{[\mathbf{r}[j]|_{\boldsymbol{\beta}[j]}]_t\}_{j=0}^{k-1})$.
All P_i samples a random size$-k$ vector $\mathbf{v}_i$ and then distributes a d-sharing $[\![\mathbf{v}_i]\!]_d$ and k t-sharing $[\mathbf{v}_i[j]|_{\boldsymbol{\beta}[j]}]_t$.
Compute $[\![\mathbf{v}]\!]_d = \sum_{i=1}^n [\![\mathbf{v}_i]\!]_d$ and $[\mathbf{v}[j]|_{\boldsymbol{\beta}[j]}]_t = \sum_{i=1}^n [\mathbf{v}_i[j]|_{\boldsymbol{\beta}[j]}]_t$.
Compute $[\![\mathbf{v'}]\!]_d \leftarrow \Pi_{\mathsf{PDN}}([\![\mathbf{v}]\!]_d, [\![\mathbf{v}]\!]_d)$ and parties open $\mathbf{v'}$. If there exists a zero in $\mathbf{v'}$, then abort. Otherwise, proceed as follows.
Compute $\mathbf{b}[i] = \sqrt{\mathbf{v'}[i]}, i \in [k]$, $[\![\mathbf{b}]\!]_{k-1}$ and $[\![\mathbf{c}]\!]_{d+k-1} = [\![\mathbf{b}]\!]_{k-1} \cdot [\![\mathbf{v}]\!]_d$ locally.
Compute $[\mathbf{u}[j]|_{\boldsymbol{\beta}[j]}]_t = \mathbf{b}[j] \cdot [\mathbf{v}[j]|_{\boldsymbol{\beta}[j]}]_t$ locally.
Get $[\![\mathbf{c}]\!]_d \leftarrow \Pi_{\mathsf{DegreeReduce}}([\![\mathbf{c}]\!]_{d+k-1})$.
Compute $[\![\mathbf{r}]\!]_d = \frac{[\![\mathbf{c}]\!]_d + 1}{2}$, $\frac{[\mathbf{u}[j]|_{\boldsymbol{\beta}[j]}]_t + 1}{2} = [\mathbf{r}[j]|_{\boldsymbol{\beta}[j]}]_t$.
return $([\![\mathbf{r}]\!]_d, \{[\mathbf{r}[j]|_{\boldsymbol{\beta}[j]}]_t\}_{j=0}^{k-1})$.

Appendix B Functionality Description

In this section, we focus on the single-instance setting and provide formal definitions of our main functionalities. The batched setting can be defined analogously.

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

Input: The functionality receives inputs $\{[x[i]]_d\}_{i=1}^{\ell-1}, y$, where $x = \sum_{i=0}^{\ell-1} 2^i x[i], x[i] \in \{0,1\}$.

Output: Compute the following

1. Reconstruct $x[i], i \in [\ell]$.
2. Compute

$$\mathbf{u}[i] = x[i] - y[i], \mathbf{w}[i] = x[i] \oplus y[i],$$
$$\mathbf{c}[i] = \mathbf{u}[i] + 1 + \textstyle\sum_{j=i+1}^{\ell-1} \mathbf{w}[i].$$

3. Set $b = (x \geq y) = 0$, If all elements of $\mathbf{c}$ are positive; otherwise, set $b = (x \geq y) = 1$.
4. Distribute shares of $[b]_d$ to the parties.

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

Input: The functionality receives inputs $[x]_d$ and $[y]_d$.

Output: Compute the following

1. Reconstruct x, y.
2. Randomly sample a random value $r \in \mathbb{F}_p$ and compute $z = 2(x - y) + r$.
3. Compute $v = \mathcal{F}_{\mathsf{OUCP}}(z, r)$ and $u = \mathrm{LSB}(z) \oplus [r_0]$.
4. Compute $b = 1 - u \oplus v$.
5. Distribute shares of $[b]_d$ to the parties.

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

Input: The functionality receives inputs $\{[\mathbf{T}[i]]_d\}_{i=0}^{m}, [u]_d, u \in [m], m < |\mathbb{F}_p|$.

Output: Compute the following

1. Reconstruct $\mathbf{T}, u$.
2. Distribute shares of $[\mathbf{T}[u]]_d$ to the parties.

Appendix C Proof

In this section, we focus on the single-instance setting and provide security proofs for our main protocols. The proofs for the batched setting are analogous. In addition, we give a proof of our proposed **Proposition** 1.

Definition 1 (Semi-Honest Security) Let Π be a n-party protocol running in real-world and $\mathcal{F} : (\{0,1\}^{\lambda_1})^n \to (\{0,1\}^{\lambda_2})^n$ be the ideal randomized functionality. We say Π securely computes $\mathcal{F}$ in the presence of a subset $\mathcal{C} \subset \{P_1, P_2, \cdots, P_n\}$ of the parties can be corrupted by semi-honest adversaries if for every corrupted $P_i, i \in \mathcal{C}$ and every input $\mathbf{x} \in (\{0,1\}^{\lambda_1})^n$, there exists a simulator $\mathcal{S}$ such that:

$$\{view_{i,\Pi}(\mathbf{x}), output_{\Pi}(\mathbf{x})\} \overset{c}{\approx} \{\mathcal{S}(\mathbf{x}[i], \mathcal{F}_i(\mathbf{x})), \mathcal{F}(\mathbf{x})\},$$

where $view_{i,\Pi}(\mathbf{x})$ is the view of P_i in the execution of Π on $\mathbf{x}$, $output_{\Pi}(\mathbf{x})$ is the output of all parties and $\mathcal{F}_i(\mathbf{x})$ denotes the i-th output of $\mathcal{F}(\mathbf{x})$.

Theorem 1 Π_{OUCP} *securely realizes* $\mathcal{F}_{\mathsf{OUCP}}$ *in the* $(\mathcal{F}_{\mathsf{rand}_p}, \mathcal{F}_{\mathsf{rand}_2}, \mathcal{F}_{\mathsf{DN}})$*-hybrid model with abort, in the presence of a fully semi-honest adversary controlling* d *corrupted parties.*

Proof We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. The simulator $\mathcal{S}$ works as follows.

In the beginning, $\mathcal{S}$ receives the input shares of $\{[x[i]]_d\}_{i=0}^{\ell-1}$ held by corrupted parties. When invoking $\mathcal{F}_{\mathsf{rand}_p}$ and $\mathcal{F}_{\mathsf{rand}_2}$, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{rand}_p}$ and $\mathcal{F}_{\mathsf{rand}_2}$, and receives from the adversary the shares of $[\boldsymbol{\rho}[i]]_d, i \in [2^{\lceil \log_2(\ell) \rceil}]$ and $[\theta]_d$ held by corrupted parties.

In online step 1, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{DN}}$ and receives from the adversary the shares of $[\theta \cdot x[i]]_d, i \in [\ell]_d$. For each honest party, $\mathcal{S}$ samples a random vectors as its shares of $\{[\mathbf{u}[i]]_d\}_{i=0}^{\ell-1}$. For each corrupted party, $\mathcal{S}$ computes its shares of $\{[\mathbf{u}[i]]_d\}_{i=0}^{\ell-1}$. $\{[\mathbf{w}[i]]_d\}_{i=0}^{\ell-1}$ and $\{[\mathbf{c}[i]]_d\}_{i=0}^{\ell-1}$ are obtained through local computations.

In online steps 2, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{DN}}$ and receives from the adversary the shares of $[\mathbf{g}[i]]_d, i \in [\ell]$. For each honest party, $\mathcal{S}$ samples the random vectors as its shares of $[\mathbf{g}[i]]_d$. For each corrupted party, $\mathcal{S}$ computes its shares of $[\mathbf{g}[i]]_d$. The online step 3 is performed locally.

In online step 4, depending on whether P_{king} is a corrupted party, there are two cases:

- If P_{king} is an honest party, $\mathcal{S}$ uses these shares to reconstruct the secret $\mathbf{h} = \mathbf{g} \cdot \mathbf{M}$, and sends $\mathbf{h}$ back to corrupted parties.
- If P_{king} is a corrupted party, $\mathcal{S}$ sends the shares of $\mathbf{h} = \mathbf{g} \cdot \mathbf{M}$ of honest parties to P_{king}. P_{king} also receives the shares from corrupted parties. Then P_{king} can reconstruct $\mathbf{h}$ and send $\mathbf{h}$ back to corrupted parties.

In the following steps, all computations are performed locally.

In the real view, $\mathbf{h}$ is uniformly random since $\boldsymbol{\rho}$ is uniformly random. Hence, the view generated by the simulator $\mathcal{S}$ is indistinguishable from the real view underlying the security of $\mathcal{F}_{\mathsf{rand}_p}, \mathcal{F}_{\mathsf{rand}_2}$, and $\mathcal{F}_{\mathsf{DN}}$. □

Theorem 2 Π_{CMP} *securely realizes* $\mathcal{F}_{\mathsf{CMP}}$ *in the* $(\mathcal{F}_{\mathsf{rand}_2}, \mathcal{F}_{\mathsf{OUCP}})$*-hybrid model with abort, in the presence of a fully semi-honest adversary controlling* d *corrupted parties.*

Proof We construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. The simulator $\mathcal{S}$ works as follows.

In the beginning, $\mathcal{S}$ receives the input shares of $[x]_d$ and $[y]_d$ held by corrupted parties. When invoking $\mathcal{F}_{\mathsf{rand}_2}$, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{rand}_2}$ and receives from the adversary the shares of $\{[r_i]_d\}_{i=0}^{\ell-1}$ held by corrupted parties.

In online step 1, depending on whether P_{king} is a corrupted party, there are two cases:

- If P_{king} is an honest party, $\mathcal{S}$ uses these shares to reconstruct the secret $z = 2(x - y) + r$, and sends z back to corrupted parties.

- If P_{king} is a corrupted party, $\mathcal{S}$ sends the shares of $z = 2(x-y)+r$ of honest parties to P_{king}. P_{king} also receives the shares from corrupted parties. Then P_{king} can reconstruct z and send z back to corrupted parties.

In online step 2, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{OUCP}}$ and receives from the adversary the shares of $[v]_d$. For each honest party, $\mathcal{S}$ samples a random value as its shares of $[v]_d$. For each corrupted party, $\mathcal{S}$ computes its shares of $[v]_d$. The online step 3 is performed locally.

In the following steps, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{DN}}$ and receives from the adversary the shares of $[uv]_d$. For each honest party, $\mathcal{S}$ samples a random value as its shares of $[uv]_d$. For each corrupted party, $\mathcal{S}$ computes its shares of $[uv]_d$. Then $[b]_d$ is performed locally.

In the real view, b is uniformly random since $[uv]_d$ is uniformly random. Hence, the view generated by the simulator $\mathcal{S}$ is indistinguishable from the real view underlying the security of $\mathcal{F}_{\mathsf{rand}_2}, \mathcal{F}_{\mathsf{OUCP}}$. □

Theorem 3 Π_{PLUT} *securely realizes* $\mathcal{F}_{\mathsf{PLUT}}$ *in the* $(\mathcal{F}_{\mathsf{ROH}}, \mathcal{F}_{\mathsf{DN}})$*-hybrid model with abort, in the presence of a fully semi-honest adversary controlling* d *corrupted parties.*

Proof We construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. The simulator $\mathcal{S}$ works as follows.

In the beginning, $\mathcal{S}$ receives the input shares of $\{[\![\mathbf{T}[i]]\!]_d\}_{i=0}^{m}$ and $[u]_d$ held by corrupted parties. When invoking $\mathcal{F}_{\mathsf{ROH}}$, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{ROH}}$ and receives from the adversary the shares of $([\mathbf{e}_v]_d, [v]_d)$ held by corrupted parties.

In online step 1, depending on whether P_{king} is a corrupted party, there are two cases:

- If P_{king} is an honest party, $\mathcal{S}$ uses these shares to reconstruct the secret $r = v - u$, and sends r back to corrupted parties.
- If P_{king} is a corrupted party, $\mathcal{S}$ sends the shares of $r = v-u$ of honest parties to P_{king}. P_{king} also receives the shares from corrupted parties. Then P_{king} can reconstruct r and send r back to corrupted parties.

In online step 2, all computations are performed locally.

In online step 3, $\mathcal{S}$ invokes the simulator of $\mathcal{F}_{\mathsf{DN}}$ and receives from the adversary the shares of $\sum_{i=0}^{m}[\hat{\mathbf{e}}_i]_d \cdot [\mathbf{T}[i]]_d$. For each honest party, $\mathcal{S}$ samples a random value as its shares of $\sum_{i=0}^{m}[\hat{\mathbf{e}}_i]_d \cdot [\mathbf{T}[i]]_d$. For each corrupted party, $\mathcal{S}$ computes its shares of $\sum_{i=0}^{m}[\hat{\mathbf{e}}_i]_d \cdot [\mathbf{T}[i]]_d$.

In the real view, res is uniformly random since the output of $\mathcal{F}_{\mathsf{DN}}$ is uniformly random. Hence, the view generated by the simulator $\mathcal{S}$ is indistinguishable from the real view underlying the security of $\mathcal{F}_{\mathsf{ROH}}, \mathcal{F}_{\mathsf{DN}}$. □

Proposition 1 *For a secret-shared vector with exactly one zero entry in $\mathbb{Z}_p$, where p is a prime, determining whether a zero exists without leaking its location does not require shuffling the entire vector after masking before opening the result; it suffices to perform a random shift relative to the position of the zero and then open the result.*

Proof Let $\mathbf{x}$ be a secret-shared vector of length m over a prime field $\mathbb{F}_p$.

Case 1: Provided that all entries of $\mathbf{x}$ are nonzero, multiplying each entry by an independent random element chosen uniformly from $\mathbb{F}_p$ and then publishing the resulting vector ensures that each entry is uniformly distributed over $\mathbb{F}_p$.

Case 2: If there exists a zero element in $\mathbf{x}$, then multiplying each entry by an independent random element chosen uniformly from $\mathbb{F}_p$ yields $\mathbf{x}'$. After applying the secure shuffle to $\mathbf{x}'$, we publish the resulting vector. Then for any position, the probability that the zero element appears in the resulting vector is exactly $\frac{1}{m}$. Similarly, after applying the secure shift to $\mathbf{x}'$, we publish the resulting vector. Then for any position, the probability that the zero element appears in the resulting vector is also exactly $\frac{1}{m}$.

□

Appendix D Detailed Benchmarks

Table D1 The running time and amortized per-party communication costs of Helix and Π_{CMP} over different network settings with an adversary threshold of $(n-1)/2$, for 10000 secure comparison operations. Communication is in MB, and running time is in seconds. n is the number of parties, and Bits refers to the input length.

Bits	n	protocol	Comm.(MB)			LAN.(s)		
			offline	online	total	offline	online	total
31	9	Helix	7.43	5.03	12.46	0.97	0.63	1.60
		SS	12.73	4.60	17.33	1.43	0.70	2.13
	15	Helix	7.80	5.28	13.09	1.36	0.98	2.34
		SS	13.37	4.83	18.20	2.24	1.05	3.29
	21	Helix	7.96	5.39	13.35	1.92	1.26	3.18
		SS	13.64	4.93	18.57	3.02	1.15	4.17
61	9	Helix	33.85	22.44	56.28	2.80	1.86	4.66
		SS	49.71	18.03	67.73	4.67	2.14	6.81
	15	Helix	35.54	23.56	59.10	4.35	2.70	7.05
		SS	52.19	18.93	71.12	6.42	2.94	9.36
	21	Helix	36.27	24.04	60.31	5.72	3.75	9.47
		SS	53.26	19.31	72.57	8.95	3.82	12.77

Bits	n	protocol	MAN.(s)			WAN.(s)		
			offline	online	total	offline	online	total
31	9	Helix	2.42	2.36	4.78	7.46	7.06	14.52
		SS	4.30	2.42	6.72	12.78	6.75	19.53
	15	Helix	4.15	3.76	7.91	11.90	10.50	22.40
		SS	7.10	3.91	11.01	20.99	10.04	31.03
	21	Helix	5.68	5.31	10.99	15.85	14.36	30.21
		SS	10.45	5.01	15.46	25.93	13.70	39.63
61	9	Helix	10.95	10.38	21.33	27.40	25.51	52.91
		SS	49.71	9.32	26.35	45.83	23.00	68.83
	15	Helix	18.84	16.21	35.05	43.35	41.32	84.67
		SS	28.80	13.58	42.38	71.70	36.27	107.97
	21	Helix	26.03	22.76	48.79	61.92	56.53	118.45
		SS	40.75	18.27	59.02	98.23	48.96	147.19

Table D2 The running time and amortized per-party communication costs of Helix, Π_{CMP} and $\Pi_{\mathsf{CMP}}^{\mathsf{PSS}}$ under different network settings and adversary thresholds for 10000 secure comparison operations. Communication is in MB, and running time is in seconds. n is the number of parties, Bits refers to the input length, and k is the size of the package in PSS.

Bits	n	t	protocol	k	Comm.(MB)			LAN.(s)		
					offline	online	total	offline	online	total
31	9	2	Helix	-	8.73	3.78	12.51	0.62	0.27	0.89
			SS	-	13.35	3.46	16.81	0.98	0.38	1.36
			PSS	3	7.97	2.22	10.19	1.13	0.30	1.43
		3	Helix	-	8.51	4.40	12.91	0.65	0.40	1.05
			SS	-	13.51	4.03	17.54	1.20	0.52	1.72
			PSS	2	10.34	2.68	13.02	1.38	0.37	1.75
	15	2	Helix	-	9.20	3.40	12.61	0.78	0.28	1.06
			SS	-	13.69	3.11	16.81	1.18	0.38	1.56
			PSS	6	5.56	1.81	7.37	1.78	0.43	2.21
		5	Helix	-	8.87	4.53	13.40	1.01	0.78	1.79
			SS	-	14.06	4.15	18.20	1.97	0.80	2.77
			PSS	3	8.08	2.29	10.37	1.84	0.55	2.39
	21	2	Helix	-	9.39	3.24	12.64	1.05	0.38	1.43
			SS	-	13.82	2.97	16.79	1.49	0.48	1.97
			PSS	9	4.69	1.65	6.35	1.67	0.53	2.20
		7	Helix	-	9.03	4.58	13.61	1.55	1.01	2.56
			SS	-	14.29	4.20	18.49	2.61	1.00	3.61
			PSS	4	6.83	2.07	8.90	2.03	0.62	2.65

Table D3 Table D2 (continued).

Bits	n	t	protocol	k	Comm.(MB)			LAN.(s)		
					offline	online	total	offline	online	total
61	9	2	Helix	-	40.12	16.84	56.95	2.11	1.16	3.27
			SS	-	52.31	13.53	65.84	3.54	1.63	5.17
			PSS	3	30.56	8.73	39.29	2.90	1.23	4.13
		3	Helix	-	39.00	19.64	58.64	2.50	1.54	4.04
			SS	-	52.89	15.78	68.67	4.10	1.89	5.99
			PSS	2	39.72	10.51	50.23	3.65	1.44	5.09
	15	2	Helix	-	42.35	15.16	57.51	2.79	1.37	4.16
			SS	-	53.69	12.18	65.88	4.25	1.98	6.23
			PSS	6	21.25	7.09	28.34	3.24	1.31	4.55
		5	Helix	-	40.67	20.20	60.86	3.90	2.30	6.20
			SS	-	55.02	16.23	71.25	5.93	2.53	8.46
			PSS	3	30.95	9.00	39.95	4.72	1.77	6.49
	21	2	Helix	-	43.25	14.44	57.69	3.50	1.82	5.32
			SS	-	54.23	11.60	65.83	5.45	2.33	7.78
			PSS	9	17.87	6.50	24.37	3.77	1.60	5.37
		7	Helix	-	41.38	20.44	61.82	5.24	2.98	8.22
			SS	-	55.94	16.42	72.36	7.93	3.15	11.08
			PSS	4	26.13	8.14	34.27	5.35	1.95	7.30

Table D4 Table D2 (continued).

Bits	n	t	protocol	k	MAN.(s)			WAN.(s)		
					offline	online	total	offline	online	total
31	9	2	Helix	-	2.90	2.09	4.99	8.22	6.41	14.63
			SS	-	4.47	2.15	6.62	13.56	5.92	19.48
			PSS	3	3.45	1.24	4.69	10.35	3.46	13.81
		3	Helix	-	2.63	2.33	4.96	8.35	6.91	15.26
			SS	-	4.54	2.41	6.95	12.90	6.67	19.57
			PSS	2	4.09	1.35	5.44	12.04	4.04	16.08
	15	2	Helix	-	4.21	2.46	6.67	10.45	8.54	18.99
			SS	-	6.80	2.38	9.18	16.56	5.00	21.56
			PSS	6	3.66	1.59	5.25	10.72	4.39	15.11
		5	Helix	-	4.61	3.74	8.35	14.98	10.34	25.32
			SS	-	7.94	3.60	11.54	21.62	9.82	31.44
			PSS	3	5.53	1.90	7.43	16.16	5.24	21.40
	21	2	Helix	-	5.83	3.29	9.12	14.40	8.37	22.77
			SS	-	9.47	2.93	12.40	23.92	7.36	31.28
			PSS	9	4.38	2.01	6.39	12.84	5.38	18.22
		7	Helix	-	6.53	5.35	11.88	17.69	13.81	31.50
			SS	-	11.15	5.03	16.18	28.32	13.40	41.72
			PSS	4	6.25	2.40	8.65	19.09	6.54	25.63

Table D5 Table D2 (continued).

Bits	n	t	protocol	k	MAN.(s)			WAN.(s)		
					offline	online	total	offline	online	total
61	9	2	Helix	-	12.55	8.61	21.16	32.12	22.58	54.70
			SS	-	19.15	7.42	26.57	45.15	18.15	63.30
			PSS	3	11.97	4.84	16.81	30.28	11.13	41.41
		3	Helix	-	12.33	9.77	22.10	31.41	25.50	56.91
			SS	-	19.10	8.05	27.15	45.32	21.17	66.49
			PSS	2	15.00	5.43	20.43	37.82	12.58	50.40
	15	2	Helix	-	19.62	11.45	31.07	47.10	27.37	74.47
			SS	-	26.96	9.86	36.82	63.50	23.75	87.25
			PSS	6	13.18	6.21	19.39	35.89	16.30	52.19
		5	Helix	-	21.46	16.03	37.49	51.87	42.21	94.08
			SS	-	31.55	13.25	44.80	79.42	35.04	114.46
			PSS	3	18.94	7.45	26.39	51.01	18.43	69.45
	21	2	Helix	-	26.18	15.71	41.89	64.56	38.54	103.10
			SS	-	36.88	13.50	50.38	89.62	32.42	122.04
			PSS	9	15.28	7.80	23.08	41.10	19.38	60.48
		7	Helix	-	30.28	22.03	52.31	71.73	56.97	128.70
			SS	-	45.71	18.12	63.83	108.14	49.12	157.26
			PSS	4	24.56	9.27	33.83	60.61	25.28	85.89

Table D6 The running time and amortized per-party communication costs of Π_{SPDTE} and $\Pi^{\mathsf{PSS}}_{\mathsf{SPDTE}}$ under different network settings and adversary thresholds for $\ell = 61$-bit inputs and random forests consisting of 50 trees trained on the Digits, Diabetes, and Boston datasets. Communication is in MB, and running time is in seconds. n is the number of parties, k is the size of the package in PSS, and t is the adversary thresholds.

Model	**n**	**t**	**protocol**	**k**	**Comm.(MB)**			**LAN.(s)**		
					offline	**online**	**total**	**offline**	**online**	**total**
Digits	9	2	SS	-	21.87	3.10	24.97	1.31	0.59	1.90
			PSS	3	16.54	2.03	18.57	1.77	0.48	2.25
		3	SS	-	22.11	3.62	25.72	1.88	0.64	2.52
			PSS	2	18.03	2.40	20.43	2.14	0.52	2.66
	15	2	SS	-	22.43	2.80	25.22	1.79	0.33	2.12
			PSS	6	16.52	1.75	18.26	2.41	0.54	2.95
		5	SS	-	22.95	3.72	26.68	3.14	0.78	3.92
			PSS	3	16.87	2.10	18.96	2.99	0.59	3.58
	21	2	SS	-	22.68	2.66	25.35	2.26	0.40	2.66
			PSS	9	17.47	1.60	19.07	2.98	0.59	3.57
		7	SS	-	23.39	3.77	27.16	4.13	1.09	5.22
			PSS	4	16.45	1.93	18.38	4.03	0.67	4.71
Diabetes	9	2	SS	-	55.64	5.80	61.43	3.07	1.19	4.26
			PSS	3	44.81	3.80	48.61	3.25	1.10	4.35
		3	SS	-	56.25	6.76	63.01	4.02	1.38	5.40
			PSS	2	46.91	4.48	51.39	4.80	1.10	5.91
	15	2	SS	-	57.03	5.22	62.25	4.12	0.73	4.85
			PSS	6	46.41	3.26	49.67	5.34	1.12	6.46
		5	SS	-	58.41	6.95	65.36	6.12	1.58	7.70
			PSS	3	45.58	3.91	49.49	6.46	1.23	7.69
	21	2	SS	-	57.63	4.97	62.61	5.15	0.88	6.03
			PSS	9	48.54	2.99	51.53	6.26	1.34	7.60
		7	SS	-	59.47	7.03	66.50	8.31	1.96	10.28
			PSS	4	45.35	3.60	48.95	8.29	1.42	9.72

Table D7 Table D6 (continued).

Model	n	t	protocol	k	Comm.(MB)			LAN.(s)		
					offline	online	total	offline	online	total
Boston	9	2	SS	-	59.61	6.21	65.82	3.33	1.31	4.65
			PSS	3	48.01	4.07	52.08	3.81	1.12	4.93
		3	SS	-	60.27	7.24	67.51	4.24	1.42	5.66
			PSS	2	50.26	4.80	55.06	4.69	1.19	5.88
	15	2	SS	-	61.10	5.59	66.69	4.34	0.78	5.13
			PSS	6	49.72	3.49	53.22	5.17	1.30	6.47
		5	SS	-	62.58	7.45	70.03	6.79	1.67	8.45
			PSS	3	48.83	4.19	53.02	6.87	1.32	8.20
	21	2	SS	-	61.75	5.33	67.08	5.96	0.89	6.85
			PSS	9	52.01	3.20	55.21	6.91	1.43	8.34
		7	SS	-	63.72	7.54	71.26	8.73	1.99	10.72
			PSS	4	48.59	3.86	52.45	9.12	1.46	10.58

Model	n	t	protocol	k	MAN.(s)			WAN.(s)		
					offline	online	total	offline	online	total
Digits	9	2	SS	-	7.43	3.26	10.69	22.09	24.88	46.97
			PSS	-	37.05	2.90	9.95	21.14	23.31	44.45
		3	SS	-	7.68	3.69	11.37	23.04	25.44	48.48
			PSS	2	7.21	3.04	10.25	21.66	23.68	45.33
	15	2	SS	-	11.56	3.03	14.59	26.59	24.13	50.71
			PSS	6	9.58	3.29	12.87	26.96	24.28	51.25
		5	SS	-	13.07	4.84	17.90	37.22	28.44	65.66
			PSS	3	10.94	3.51	14.45	31.78	24.92	56.70
	21	2	SS	-	15.64	3.34	18.98	39.98	24.36	64.34
			PSS	9	12.20	3.70	15.90	33.95	25.35	59.31
		7	SS	-	18.46	6.06	24.51	49.31	31.49	80.80
			PSS	-	415.00	4.03	19.03	42.74	26.16	68.90

Table D8 Table D6 (continued).

Model	n	t	protocol	k	MAN.(s)			WAN.(s)		
					offline	online	total	offline	online	total
Diabetes	9	2	SS	-	20.06	6.40	26.46	54.39	46.47	100.86
			PSS	3	17.97	5.57	23.54	49.78	43.70	93.48
		3	SS	-	19.90	7.12	27.02	53.67	47.36	101.03
			PSS	2	17.83	5.81	23.64	50.06	44.29	94.34
	15	2	SS	-	28.86	6.00	34.86	74.19	44.81	119.00
			PSS	6	25.73	6.40	32.13	68.88	45.66	114.55
		5	SS	-	33.22	9.11	42.33	90.73	53.02	143.76
			PSS	3	29.05	6.75	35.79	78.66	46.66	125.32
	21	2	SS	-	39.12	6.28	45.41	95.12	45.86	140.98
			PSS	9	33.87	7.41	41.01	89.38	47.56	136.94
		7	SS	-	47.39	11.07	58.46	123.64	59.08	182.72
			PSS	4	39.85	7.74	47.58	107.14	49.06	156.19
Boston	9	2	SS	-	20.77	6.72	27.49	58.96	48.80	107.76
			PSS	3	19.82	5.94	25.76	53.13	46.78	99.91
		3	SS	-	21.40	7.52	28.92	60.00	50.83	110.83
			PSS	2	19.46	6.28	25.74	56.26	47.48	103.74
	15	2	SS	-	31.26	6.30	37.57	78.98	48.10	127.08
			PSS	6	27.88	6.88	34.76	74.46	48.90	123.35
		5	SS	-	36.54	9.93	46.47	96.00	56.71	152.71
			PSS	3	31.20	7.22	38.42	83.91	49.98	133.88
	21	2	SS	-	42.33	6.77	49.11	104.71	49.23	153.94
			PSS	9	36.28	7.58	43.86	95.63	50.95	146.57
		7	SS	-	50.60	12.02	62.62	132.04	63.07	195.11
			PSS	4	43.22	8.27	51.49	111.95	53.72	165.67