



Research Article

<https://doi.org/10.1631/ENG.ITEE.2026.0223>

MTLPSI: multi-party threshold labeled private set intersection with dropout tolerance and passive collusion resistance

Shuai YANG, Yiliang HAN✉

The College of Cryptography Engineering, Engineering University of PAP, Xi'an 710086, China

Abstract: With increasingly stringent data privacy regulations and heightened public privacy awareness, secure multi-party computation (MPC) and its core component, private set intersection (PSI) technology, have garnered widespread attention. However, standard PSI protocols struggle to address the complex demands of practical applications concerning threshold conditions matching, robustness and security in multi-party environments. To simultaneously resolve these issues, this paper aims to construct an efficient and secure PSI variant protocol. First, we propose an efficient two-party threshold PSI protocol (TPSI) based on Bloom filters and threshold Paillier encryption, reducing computational overhead. Subsequently, by innovatively integrating Shamir's secret sharing with symmetric encryption techniques, we introduce the multi-party threshold labeled PSI protocol (MTLPSI) with dropout tolerance and resistance against passive collusion in the semi-honest setting. This protocol securely performs threshold private set intersection cardinality testing and label output in multi-party environments subject to node dropouts and passive collusion. Rigorous formal security analyses validate the proposed protocols, offering viable solutions for complex multi-party privacy computing scenarios such as smart cities and collaborative security systems.

Key words: Private set intersection (PSI); Threshold private set intersection (TPSI); Multi-party threshold labeled PSI (MTLPSI); Secure multi-party computation; Privacy protection

1 Introduction

With the widespread implementation of data privacy regulations such as the EU's General Data Protection Regulation (GDPR) (Voigt and von dem Bussche, 2017) and the awakening of societal privacy awareness, secure multi-party computation (MPC) technology, which enables collaborative computation while protecting raw data, has garnered unprecedented attention. As one of the core foundational components of MPC, the private set intersection (PSI) protocol enables two parties to compute the intersection of their datasets without revealing any information beyond the intersection. It demonstrates significant potential in fields such as medical data sharing, fraud detection, and privacy-preserving contact tracing (Rindal and Schoppmann, 2021; Raghuraman and Rindal, 2022; Chen YC and Huang, 2023; Feng et al., 2024; Wang et al., 2024; Wu ML

et al., 2024; Yang et al., 2024; Wu AX et al., 2025).

However, the standard PSI protocol reveals its limitations when addressing increasingly complex real-world scenarios. In many privacy-sensitive applications (e.g., collaborative threat intelligence, smart home emergency detection, and distributed IoT sensing), decision-making depends not on which specific elements overlap, but on whether the degree of overlap is sufficient to trigger an action. For instance, a security operations center should raise a "high-risk" alert only when a regional gateway matches at least t threat signatures from a central database; requiring all signatures to match would render the system overly rigid and unresponsive to partial evidence. This requirement has motivated the development of threshold private set intersection (TPSI) protocols (Ghosh and Simkin, 2019; Branco et al., 2021; Ghosh and Simkin, 2023; Liu FH et al., 2023), which disclose information only when the intersection cardinality exceeds a predetermined threshold. Although existing research has explored TPSI, its computational efficiency, particularly its deployability on resource-constrained IoT devices, remains an area for improvement.

A crucial yet underappreciated property of threshold mechanisms is their inherent resilience to data noise. In practice, client-side datasets are seldom pristine: IoT sensors suffer from measurement errors, network sampling intro-

✉ Yiliang HAN, hanyil@163.com

Shuai YANG, <https://orcid.org/0009-0002-2402-777X>

Yiliang HAN, <https://orcid.org/0000-0002-2116-5408>

CLC number: TP309

Received: July 21, 2026; Revision accepted: Oct. 1, 2026;

Crosschecked: Oct. 12, 2026

© The Authors 2026. Published by Zhejiang University Press Co., Ltd. This is an open access article distributed under the terms of the CC BY-NC-ND license (<https://creativecommons.org/licenses/by-nc-nd/4.0/>)

duces omissions, and threat databases contain transient faults. Under exact-match PSI, a single erroneous element can invalidate the entire intersection, causing a false negative. Our threshold mechanism elegantly circumvents this fragility: by requiring at least t matches, the protocol effectively disregards up to $n - t$ elements as potential noise. Formal analysis shows that even if a client's dataset contains e erroneous elements, the label is correctly disclosed as long as the genuine intersection size $s \geq t$, entirely independent of the exact value of e . This statistical error tolerance is precisely what makes threshold-based protocols compelling for real-world feature aggregation and collaborative monitoring.

While a single threshold t suffices to filter noise at the individual client level, practical multi-party scenarios demand an additional safeguard: the final decision should reflect a consensus among multiple independent observers, not the potentially compromised or malfunctioning judgment of a single client. To address this, our MTLPSI protocol introduces a second threshold t_L governing label reconstruction. The two thresholds serve distinct and complementary roles: 1. the intersection threshold t operates at the per-client level, ensuring that only clients with sufficient overlap receive valid label shares; 2. the label reconstruction threshold t_L operates at the system level, requiring at least t_L qualified clients to collectively reconstruct the final label via Shamir's secret sharing. This separation enables independent tuning of sensitivity and consensus strength, a flexibility absent from prior single-threshold designs.

More critically, real-world applications such as smart cities and collaborative security monitoring are inherently multi-party. While extending PSI from two parties to multiple parties unlocks broader scenarios, it also introduces severe security and robustness challenges: 1. Dynamic participants: participants may drop out unexpectedly due to network or device failures; 2. Enhanced adversary model: adversaries may collude to extract private information from honest parties. Existing TPSI and labeled PSI protocols are largely not designed to address the combined robustness and security challenges in such multi-party environments. Notably, the multi-party setting we consider is asymmetric: a central server interacts with multiple clients, and the goal is conditional label disclosure to the server rather than a global intersection among all parties. To the best of our knowledge, no existing protocol simultaneously supports threshold-labeled output, dropout tolerance, and passive collusion resistance in this setting.

To address these issues, we propose an efficient PSI variant that satisfies both threshold and label-output requirements while adapting to complex multi-party environments. Specifically, we first design an efficient two-party TPSI protocol based on Bloom filters (BF) and threshold Paillier encryption, performing the cardinality test with additive homomorphism and secure comparison—avoiding the heavy computational cost of fully homomorphic encryption (Hu et al., 2024). Building on this foundation, we introduce MTLPSI, which extends the functionality to the many-to-one setting via Shamir's secret sharing. This decouples per-client qualification (governed by t) from multi-client label reconstruction (governed by t_L), naturally supporting dropouts: the system requires only $\max(t_h, t_L)$ online responses rather than all $m - 1$ clients. Meanwhile, threshold Paillier ensures that fewer than

t_h colluding clients cannot decrypt any sensitive information.

Our work makes three novel contributions beyond the composition of existing cryptographic tools. First, unlike existing T-MPSI protocols that require the server to compute the actual intersection set with each client, our protocol enables the server to obtain only a label value L when the threshold is satisfied—the server never learns which specific elements are in the intersection. Second, we introduce two decoupled thresholds: t for per-client qualification and t_L for system-level label reconstruction, enabling independent tuning of sensitivity and consensus strength while naturally supporting client dropouts. Third, to the best of our knowledge, no existing protocol simultaneously provides threshold-labeled output, dropout tolerance, and collusion resistance in the many-to-one client-server architecture.

The main contributions of this paper are as follows:

1. An efficient threshold PSI protocol: We design and implement a TPSI protocol based on Bloom filters and threshold Paillier encryption. Analysis and experiments demonstrate that this protocol significantly reduces computational overhead compared to existing schemes (Hu et al., 2024), making it suitable for edge computing environments.
2. A multi-party threshold labeled PSI protocol: We construct a multi-party threshold labeled PSI protocol with dropout tolerance and passive collusion resistance. It supports threshold-based set intersection with label output, while maintaining robustness against participant dropouts and resisting passive collusion attacks.
3. Rigorous formal security analysis: We provide formal security definitions and proofs for the proposed protocol under recognized security models, ensuring its robust security foundation.

2 Related works

The standard PSI protocol has been extensively studied over the past decade, with some approaches achieving intersection computations for large sets within seconds. Concurrently, several works have extended the PSI protocol to support multi-party scenarios. Various techniques have been employed to design PSI and MPSI protocols, including oblivious transfer (OT), permutation-based hashing, circuit-based computation, Bloom filters, cuckoo hashing, and oblivious programmable hashing (Kolesnikov et al., 2017; Chase and Miao, 2020; Garimella et al., 2021; Nevo et al., 2021; Rosulek and Trieu, 2021; Chielle and Maniatakos, 2025).

While studying the standard PSI protocol, researchers have also explored several variant protocols, including threshold PSI, cardinality PSI, labeled PSI, and fuzzy PSI. Table 1 compares the functionalities and techniques of representative protocols.

To address the requirement of only disclosing information when the intersection reaches a certain size, threshold PSI emerged. Ghosh and Simkin (2019) established the first upper and lower bounds for the communication complexity of TPSI and proposed the first protocol whose communication complexity depends primarily on the threshold parameter t and only logarithmically on the set size. Subsequently, Branco et al. (2021) constructed a TPSI protocol for multi-party scenarios

Table 1 Comparison of several variant private set intersection protocols

Protocol	Multi-party	Functionality	Dropout tolerance	Collusion resistance	Techniques used
Ghosh and Simkin (2019)	Yes	1. TPSI 2. MTPSI	No	No	AHE OLE Polynomial encoding (PE)
Branco et al. (2021)	Yes	MTPSI	No	No	TPKE PE
Ghosh and Simkin (2023)	Yes	MTPSI	No	No	AHE OT Secret Sharing
Liu FH et al. (2023)	Yes	pTPSI	No	Yes	Goldwasser-Sipser Protocol OPRF
Gao et al. (2024)	Yes	1. MPSI 2. MPSI-CA 3. MTPSI	No	Yes	ZKS OKVS PRF
Hu et al. (2024)	No	TPSI	-	-	FHE
Chen H et al. (2018)	No	LPSI	-	-	FHE OPRF
Cong et al. (2021)	No	LPSI	-	-	AHE
Bienstock et al. (2024)	No	LPSI	-	-	Oblivious Ciphertext Compression
Uzun et al. (2021)	No	FLPSI	-	-	FHE Secret Sharing
Gao et al. (2025)	No	1. FPSI 2. FLPSI	-	-	AHE OT OKVS Spatial Additive Sharing
van Baarsen and Pu (2024)	No	1. FPSI 2. FLPSI	-	-	OKVS OPPRF PHE
Ours	Yes	MTLPSI	Yes	Yes (semi-honest)	TAHE BF SCP

For our protocol, “Yes” denotes passive collusion resistance in the semi-honest model only and does not imply malicious security

using threshold public-key encryption based on additive homomorphic encryption (AHE). To optimize communication complexity, Ghosh and Simkin (2023) developed a protocol with quasi-linear dependence on the threshold parameter t , based on simple assumptions such as AHE and OT. This protocol can transform any such scheme with secret-sharing outputs and communication complexity $O(\lambda mt)$ (where λ is the security parameter and m is the number of participants) into one with communication complexity $O(\lambda^2 mt \log(t))$. To enhance the efficiency and security of TPSI, Liu FH et al. (2023) pioneered the concept of probabilistic TPSI (pTPSI), where parties learn the intersection I with a probability proportional to the size of I . This capability sacrifices a small probability of adverse events (i.e., parties learning insufficiently large I) to open new avenues for efficient and scalable designs. Gao et al. (2024) proposed executing PSI protocols on a bicentric

network architecture. By introducing bicentric zero-knowledge sharing (ZKS) and Oblivious Key Value Storage (OKVS), they constructed a new efficient and scalable MPSI protocol. Additionally, based on bicentric zero-knowledge sharing, they proposed the multi-party private set intersection cardinality (MPSI-CA) and multi-party threshold private set intersection (MTPSI) protocols. Hu et al. (2024) pioneered the implementation of TPSI protocols using fully homomorphic encryption (FHE). This protocol generates a shared secret token using programmable bootstrapping techniques within FHE, activated only when the intersection size meets threshold conditions. Combining this secret token generation with any standard PSI protocol yields TPSI with the same asymptotic communication complexity as the selected plain PSI. Currently, most TPSI protocols consist of two sub-protocols: a cardinality test protocol and a standard PSI protocol. In this paper, we introduce

a multi-party secure comparison protocol and construct the cardinality test protocol using BF and threshold Paillier encryption, followed by executing the standard PSI protocol, thereby approximating the communication complexity of standard PSI.

Labeled PSI extends the output of PSI from specific intersection elements to associated labels, significantly broadening its application scenarios. Chen H et al. (2018) addressed the scenario of unbalanced PSI (where the receiver set is much smaller than the sender set and receivers are low-power devices). They constructed and implemented an unbalanced LPSI protocol based on FHE, enhancing the security model by incorporating an oblivious pseudorandom function (OPRF) during the preprocessing phase. Building upon the previous work, Cong et al. (2021) optimized both computational and communication complexities. Computational complexity was reduced from $O(|X|)$ to $O(\sqrt{|X|})$, while communication complexity decreased from $O(\log|X|)$ to nearly constant. Bienstock et al. (2024) proposed LSOvCompress and LSOvDecompress—two Oblivious Ciphertext compression and decompression algorithms based on Random Band Matrices. These algorithms achieve near-optimal compression relying only on additive homomorphism and were applied to LPSI, significantly optimizing communication costs.

In scenarios like feature matching, the inherent noise characteristics of data have driven the demand for fuzzy PSI. Uzun et al. (2021) pioneered the concept of FLPSI to address privacy-preserving queries for biometric data. This protocol combines AES block cipher, homomorphic encryption, garbled circuits, and t -out-of- T secret sharing, achieves communication complexity that scales sublinearly with database size and demonstrates strong practical applicability. Subsequently, (Gao et al., 2025; van Baarsen and Pu, 2024) expanded research on FPSI, exploring its extension to LPSI. While the aforementioned LPSI/FLPSI protocols have achieved significant performance improvements, they remain confined to two-party settings. When extending to multi-party scenarios, additional challenges such as collusion attacks and participant dropouts emerge. Among the few works that address multi-party threshold PSI, Bay et al. (2022) proposed a practical T-MPSI protocol using additive homomorphic encryption and secure comparison protocol (SCP). Their protocol allows a set of parties to compute the intersection only if its size exceeds a threshold, and the result is revealed to all participants. However, it does not support labeled output, nor does it explicitly handle client dropouts or collusion attacks. Moreover, the symmetric nature of their setting diverges from the many-to-one architecture common in applications such as threat intelligence sharing and IoT monitoring.

In summary, while existing works address subsets of these requirements—threshold functionality, labeled output, multi-party setting, dropout tolerance, or collusion resistance—none simultaneously provides all five in the many-to-one architecture. Our MTLPSI protocol fills this gap by combining lightweight TPSI with a novel separation of per-client threshold test and global label reconstruction via Trusted Third Party (TTP)-assisted secret sharing. Our work provides a concrete instantiation that simultaneously satisfies these requirements, with the protocol design explicitly tailored to the many-to-one

setting.

3 Preliminaries

3.1 Notations

This section defines the notations and cryptographic building blocks employed in our constructions. For clarity, Table 2 catalogs the primary symbols alongside their formal definitions.

Table 2 Notations and symbols

Notation	Description
m	Number of parties
P_i	i^{th} party, where $i \in \{1, 2, \dots, m-1\}$ denotes a client and $i = m$ denotes the server
S_i	Dataset of the i^{th} party
n_i	Size of dataset S_i
n	Size of the largest dataset among all parties
L	Label value from the server
t_h	Threshold for homomorphic threshold decryption
t_L	Threshold for reconstructing L
t	Threshold for intersection cardinality
(pk, sk)	Public-private key pair
sk_i	Secret share of sk among m participants
key	Symmetric encryption and decryption key
$Enc_{pk}(\cdot)$, $Dec_{sk}(\cdot)$	Public-key encryption and decryption functions (abbreviated as $Enc(\cdot)$ and $Dec(\cdot)$)
$SymEnc(M, key)$, $SymDec(C, key)$	Symmetric encryption of message M and decryption of ciphertext C
$\mathbf{BF}_i$	Bloom filter of size ℓ for set S_i
$\mathbf{EBF}$	Entry-wise encryption of a Bloom filter
${}_h \Sigma$	Homomorphic summation operation on ciphertexts
l_i	Label share generated from L for the i^{th} party
el_i	Symmetric encryption of l_i
$Share(\cdot)$, $Recon(\cdot)$	Secret share generation and secret value reconstruction functions

3.2 Bloom filter

A Bloom filter (Bloom, 1970) is a highly space-efficient probabilistic data structure designed to support fast membership queries. Its core consists of a bit array of length ℓ , initially set to zero, and k independent hash functions. When adding an element, it is mapped to k specific positions in the bit array via the k hash functions and set to 1 at those positions. When querying an element, the corresponding k positions are checked to see if they are all 1: if so, the element is deemed “possibly present”; if any is 0, it is “definitely absent”. This design prevents false negative errors but incurs a certain false positive rate p . This misclassification probability is closely tied to the parameters. Given the expected number of elements n , the optimal bit array size ℓ and number of hash functions k required to achieve a target false positive rate can be approximated as $\ell = -\frac{n \ln p}{(\ln 2)^2}$ and $k = \frac{\ell}{n} \ln 2$. This controlled trade-off between space efficiency and accuracy makes it an ideal building block for constructing efficient privacy-preserving protocols.

This paper adopts the encrypted Bloom filter defined in Davidson and Cid (2017), specifically defined as follows: Encrypted Bloom filter (EBF): For the Bloom filter $\mathbf{BF}_i$

corresponding to the dataset S_i of client i , its corresponding $\mathbf{EBF}_i$ is defined as $\mathbf{EBF}_i[j] = \text{Enc}(\mathbf{BF}_i[j])$.

3.3 Threshold Paillier encryption

Our protocol requires an additively homomorphic threshold public-key encryption scheme. We employ the threshold Paillier encryption scheme proposed in Fouque et al. (2001). The threshold Paillier encryption scheme combines the original Paillier encryption with Shamir's secret sharing and threshold decryption, achieving the property that "decryption requires collaboration from at least t_h participants, while fewer than t_h participants cannot decrypt", while preserving the additive homomorphism of the original Paillier scheme. The threshold Paillier scheme consists of the following four core algorithms:

1. Key generation algorithm

First, select safe primes $p = 2p' + 1$ and $q = 2q' + 1$, and compute $u = pq$ and $v = p'q'$. Next, choose random elements $a, b \in \mathbb{Z}_u^*$ to generate the public parameter $g = (1 + u)^a \cdot b^n \pmod{u^2}$. Then, select a random $\beta \in \mathbb{Z}_u^*$ to set the global secret key as $\text{sk} = \beta v$, and use a random polynomial of degree $t_h - 1$, $f(X) = \sum_{i=0}^{t_h-1} a_i X^i$ (where $a_0 = \beta v$ and the remaining coefficients are uniformly chosen from $\{0, 1, \dots, uv - 1\}$), to distribute the secret share $s_i = f(i) \pmod{uv}$ to each participant. Finally, construct the public key $\text{pk} = (g, u, \theta)$, where $\theta = av\beta \pmod{u}$, which will be used for plaintext recovery in the subsequent threshold decryption phase.

2. Encryption algorithm

Identical to the original Paillier encryption $c = g^M \times r^u \pmod{u^2}$ ($M \in \mathbb{Z}_u, r \in \mathbb{Z}_u^*$).

3. Share decryption algorithm

The i^{th} participant uses its secret share s_i to compute $c_i = c^{2\Delta s_i} \pmod{u^2}$, ($\Delta = N!$), where N is the number of participants. In this paper, we denote this as $\text{partDec}(c, \text{sk}_i)$.

4. Combining algorithm

If fewer than t_h valid shares are available, the algorithm fails immediately. Otherwise, select t_h valid shares to form set S , then execute the following steps:

Compute Lagrange Coefficients: For each $j \in S$, compute the adjusted Lagrange coefficient $\mu_j^S = \Delta \times \prod_{j' \in S, j' \neq j} \frac{j'}{j' - j}$.

Recover Plaintext:

$$M = L \left(\prod_{j \in S} c_j^{2\mu_j^S} \pmod{u^2} \right) \times \frac{1}{4\Delta^2\theta} \pmod{u}, \text{ where}$$

$\prod_{j \in S} c_j^{2\mu_j^S} \pmod{u^2}$ is equivalent to $c^{4\Delta^2\beta m} \pmod{u^2}$. After extracting the linear term via $L(\cdot)$, multiply by the inverse of θ to recover M . In this paper, we denote this as $\text{combine}(\cdot)$.

3.4 Rerandomization algorithm

In additively homomorphic public-key encryption (AH-PKE) schemes, the rerandomization algorithm is a crucial component for ensuring ciphertext security. Its core function is to refresh the randomness of a ciphertext without changing the underlying plaintext, thereby breaking the randomness correlation between ciphertexts at different operational stages.

Specifically, consider an AH-PKE scheme consisting of key generation algorithm KeyGen, encryption algorithm Enc, and decryption algorithm Dec. The scheme satisfies additive homomorphism: for any plaintexts m_1, m_2 , the ciphertext addition $c_1 +_h c_2$ (where $c_1 = \text{Enc}(m_1)$, $c_2 = \text{Enc}(m_2)$, and $+_h$

represents the scheme-defined ciphertext addition operation) decrypts to $m_1 + m_2$.

Based on the above AH-PKE scheme, the Rerandom algorithm is defined as follows:

Given a public key pk and a ciphertext $c = \text{Enc}(m)$ (where m is the plaintext), the rerandomization algorithm Rerandom outputs:

$$\text{Rerandom}(\text{pk}, c) = c +_h \text{Enc}(0).$$

3.5 Secure comparison protocol

In our protocol, we employ a multi-party secure comparison protocol. This protocol compares two values x_0 and x_1 in a multi-party setting, with the core objective that no participant can obtain any information about x_0 and x_1 during protocol execution, and the protocol is equally applicable in the two-party case. Its ideal functionality is illustrated in Fig. 1.

Parameters: Number of parties m ; homomorphic encryption scheme $\text{HE} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ with public key pk ; public parameters pp .

Input: Each party P_i ($i = 1, \dots, m$) receives as input two ciphertexts $\text{Enc}(x_0)$ and $\text{Enc}(x_1)$ under the public key pk .

Output: Each party P_i obtains a ciphertext $\text{Enc}(b)$, where b is a bit satisfying:

$$b = \begin{cases} 1 & \text{if } x_0 \leq x_1, \\ 0 & \text{otherwise.} \end{cases}$$

Fig. 1 Ideal functionality for secure multi-party comparison protocol $\mathcal{F}_{\text{SCP}}$

In this paper, we use the scheme proposed by Kerschbaum et al. (2009) to achieve the corresponding security requirements. The specific execution steps are as follows:

1. Initial Ciphertext Generation (P_1 executes): Firstly, generate large random numbers r_i and $r'_i < r_i$ that satisfy the multiplicative hiding requirement. Then compute using homomorphism $\text{Enc}(c) = (\text{Enc}(x_0) \cdot \text{Enc}(-x_1))^{r_i} \cdot \text{Enc}(-r'_i)$, where $\text{Enc}(-x_1)$ is obtained via the multiplicative inverse of $\text{Enc}(x_1)$. Finally, send the triple $(a_1^1, a_1^2, a_1^3) = (\text{Enc}(1), \text{Enc}(0), \text{Enc}(c))$ to P_2 .

2. Ciphertext Transmission (P_2 to P_m execute): Generate two large random numbers r_i and $r'_i < r_i$, and randomly flip a coin to obtain a bit $b_i \in \{0, 1\}$. Then based on the value of b_i , rerandomize the triple $(a_{i-1}^1, a_{i-1}^2, a_{i-1}^3)$ received from the previous participant P_{i-1} , computing the new triple (a_i^1, a_i^2, a_i^3) : If $b_i = 0$: $a_i^1 = a_{i-1}^1 \cdot \text{Enc}(0)$, $a_i^2 = a_{i-1}^2 \cdot \text{Enc}(0)$, $a_i^3 = (a_{i-1}^3)^{r_i} \cdot \text{Enc}(-r'_i)$. If $b_i = 1$: $a_i^1 = a_{i-1}^1 \cdot \text{Enc}(0)$, $a_i^2 = a_{i-1}^2 \cdot \text{Enc}(0)$, $a_i^3 = (a_{i-1}^3)^{r_i} \cdot \text{Enc}(r'_i)$. Finally, send the rerandomized triple to the next participant P_{i+1} (when $i = m$, P_m sends to the initial participant P_1 , forming a circular transmission).

3. Joint Decryption and Result Determination (All participants collaborate)

All participants P_1 to P_m execute the threshold decryption protocol to jointly decrypt the final transmitted a_m^3 , obtaining the plaintext $\text{Dec}(a_m^3)$, and determine the comparison result

based on its sign: If $\text{Dec}(a_m^3) < 0$, it indicates that $x_0 \leq x_1$, and in this case $a_m^1 = \text{Enc}(1)$; otherwise, $a_m^1 = \text{Enc}(0)$. For more specific details, please refer to Kerschbaum et al. (2009).

3.6 Security model

We adopt the standard semi-honest (honest-but-curious) adversary model, where all parties faithfully follow the protocol specification but may attempt to infer additional information from the messages they receive. Static corruption is assumed, meaning the adversary chooses the corrupted set before the protocol starts. We emphasize that our claimed “collusion resistance” is confined to the semi-honest setting and refers specifically to passive collusion, not to malicious deviations from the protocol.

Within this model, we consider two specific adversarial capabilities:

Passive collusion resistance. The adversary may passively corrupt up to $t_h - 1$ clients and, optionally, the server. Although corrupted clients may collude by pooling their views, the threshold Paillier scheme guarantees that no plaintext can be recovered beyond what is implied by the output, as fewer than t_h decryption shares are available.

Dropout tolerance. We assume honest-but-crashed failures: clients may go offline unexpectedly due to network issues, but do not send adversarially crafted messages. The protocol guarantees successful completion as long as the number of active honest clients is at least $\max(t_h, t_L)$.

To summarize, the protocol guarantees semi-honest security, passive collusion resistance against up to $t_h - 1$ corrupted clients, and dropout tolerance whenever $\max(t_h, t_L)$ clients remain online. It does not, however, address malicious adversaries, adaptive corruptions, or intentional abort attacks. These limitations are inherent to the current design and are left as future work.

4 Our threshold private set intersection protocol

To address the problem of threshold private set intersection, we propose a novel protocol that combines encrypted Bloom filters with threshold Paillier encryption. Our protocol ensures that the intersection is revealed only when the cardinality of the intersection meets a predefined threshold t for intersection cardinality, thereby providing enhanced privacy guarantees compared to standard PSI protocols.

The key insight of our approach is to perform a secure cardinality test before revealing any information about the intersection elements. This is achieved through a series of homomorphic operations and secure comparisons that allow the parties to determine whether $|X \cap Y| \geq t$ without learning the actual intersection at this stage. Its ideal functionality is illustrated in Fig. 2.

4.1 Protocol overview

The protocol operates in five phases, as illustrated in Fig. 3. First, the parties initialize the protocol and generate cryptographic parameters. The sender then constructs an encrypted Bloom filter representing their set. The core

Parameters: Intersection threshold t , security parameter λ , public parameters of the threshold Paillier encryption scheme (including public key pk and key generation parameters).

Input:

1. Sender S inputs a set $X = \{x_1, x_2, \dots, x_n\}$.
2. Receiver R inputs set $Y = \{y_1, y_2, \dots, y_n\}$.

Output:

1. R obtains a bit e , where $e = 0$ if and only if $|X \cap Y| \geq t$. Receiver R additionally obtains the intersection $I = X \cap Y$; otherwise, R obtains the empty set $\emptyset$.

Fig. 2 Ideal functionality for threshold private set intersection protocol $\mathcal{F}_{\text{TPSI}}$

innovation lies in the cardinality testing phase, where the parties securely determine whether the intersection meets the threshold. Before executing the standard PSI protocol, each participant’s set is masked by applying a bitwise XOR operation with the threshold label generated from the cardinality test result. Its core mechanism lies in using this label to control whether both parties can successfully obtain intersection information.

The formal specification of our threshold PSI protocol is presented in Fig. 3, with detailed steps for each phase.

4.2 Correctness analysis

The receiver and the sender obtain the ciphertext a via the secure comparison protocol. If the intersection cardinality is larger than the threshold, a is the ciphertext of 0; otherwise, a is the ciphertext of 1. The two parties jointly decrypt a to obtain the result e . Only when $e = 0$ do they perform the XOR operation on their respective sets with an identical mask value, and then retrieve the authentic intersection through the standard PSI protocol. Otherwise, they adopt different mask values to perturb their sets, and no valid information about the intersection can be obtained even if the standard PSI protocol is executed.

Since the sender represents its set using a Bloom filter, the correctness of the threshold test is subject to the filter’s inherent false positive rate p . A false positive occurs when an element $y \notin X$ is incorrectly reported as present, which may cause the protocol to overestimate the true intersection cardinality and erroneously conclude $|X \cap Y| \geq t$ when in fact $s = |X \cap Y| < t$.

Let $F \sim \text{Bin}(n - s, p)$ denote the number of false positives among the $n - s$ non-intersecting elements. The protocol effectively evaluates $s + F \geq t$ instead of $s \geq t$. For $s < t$, the error probability is:

$$P_{\text{err}}(s, t, n, p) = \Pr[s + F \geq t] = \sum_{j=t-s}^{n-s} \binom{n-s}{j} p^j (1-p)^{n-s-j}.$$

The worst case occurs at $s = t - 1$, where a single false positive triggers a false acceptance:

$$P_{\text{err}}^{\max} = \Pr[F \geq 1] = 1 - (1-p)^{n-t+1} \approx (n-t+1) \cdot p.$$

In our experimental setup, we configure the Bloom filter to achieve $p \leq 1\%$, keeping this error small for moderate n . For applications demanding stronger guarantees, the false positive

Parameters: Intersection threshold t , security parameter λ .

Input: The receiver R and sender S hold decryption key shares sk_1, sk_2 for the threshold Paillier algorithm respectively, with their common public key being pk . Both parties hold their respective sets $Y = \{y_1, y_2, \dots, y_n\}$ and $X = \{x_1, x_2, \dots, x_n\}$.

Protocol execution:

Initialization (by receiver R):

The receiver selects k hash functions $(h_1, h_2, \dots, h_k)$ for generating the **BF**, then sends them to the sender.

EBF generation (by sender S):

1. Construct the **BF** corresponding to their own set data $X = \{x_1, x_2, \dots, x_n\}$.
2. Encrypt each element in the **BF** using public key pk to obtain **EBF**.
3. Send **EBF** to the receiver.

Cardinality test:

1. For each element $y_j \in Y$, R computes the corresponding encrypted positions in the **EBF** as $C_j^i = \mathbf{EBF}(H_i(y_j))$, $(i = 1, \dots, k)$, and aggregates them via homomorphic addition $C_j = \sum_{i=1}^k C_j^i$.
2. R rerandomizes each C_j to obtain $C'_j = \text{Rerandom}(C_j)$.
3. R and S engage in n parallel instances of a secure two-party comparison protocol $\text{SCP}(C'_j, \text{Enc}(k)) \rightarrow T$, where each $T_j \in \{\text{Enc}(0), \text{Enc}(1)\}$, forming a vector $T = [T_1, \dots, T_n]$.
4. S homomorphically sums the comparison results and rerandomizes $T' = \text{Rerandom}(\sum_{i=1}^n T_i)$.
5. S and R run one instance of the secure comparison protocol: $\text{SCP}(T', \text{Enc}(t)) \rightarrow a$.

Threshold label generation and set updates:

1. S partially decrypts a : $b = \text{partDec}(a, sk_2)$ and transmits b to R . S then masks each element in X as $X' = \{x_i \oplus b \mid x_i \in X\}$.
2. R also partially decrypts a : $c = \text{partDec}(a, sk_1)$, then performs joint decryption to recover the plaintext result: $e = \text{combine}(b, c)$.
3. If $e = 1$: $Y' = \{y_i \oplus r \mid y_i \in Y\}$, $r \xleftarrow{\$} \{0, 1\}^\lambda$.
If $e = 0$: $Y' = \{y_i \oplus b \mid y_i \in Y\}$.

Final PSI computation:

1. S and R execute standard PSI protocol on $(X', Y') \rightarrow I'$
2. R computes $I = \{z \oplus b \mid z \in I'\}$.

Fig. 3 Our TPSI protocol Π_{TPSI}

rate can be reduced exponentially by increasing the filter length $\ell = -\frac{n \ln p}{(\ln 2)^2}$, at the cost of linear communication overhead. Thus, the false positive effect is a controllable trade-off.

4.3 Security analysis

Theorem 1 In the random oracle model, assuming that the threshold Paillier encryption is IND-CPA secure, the SCP protocol and the standard PSI protocol are semi-honest secure, then the protocol Π_{TPSI} securely realizes the ideal functionality $\mathcal{F}_{\text{TPSI}}$ in the semi-honest adversary model.

For every real-world adversary $\mathcal{A}$, we construct an ideal-world simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$,

$$\text{IDEAL}_{\mathcal{F}_{\text{TPSI}}, \mathcal{S}, \mathcal{Z}} \approx_c \text{REAL}_{\Pi_{\text{TPSI}}, \mathcal{A}, \mathcal{Z}}.$$

We consider the following two cases separately: (1) the adversary corrupts the sender S , and (2) the adversary corrupts the receiver R .

Case 1: adversary corrupts the sender S . In this case, the simulator $\mathcal{S}_S$ operates as follows:

1. $\mathcal{S}_S$ submits the real input X of S to the ideal functionality $\mathcal{F}_{\text{TPSI}}$.
2. $\mathcal{S}_S$ simulates the protocol execution: it randomly selects k hash functions; computes a Bloom filter **BF** based on X , and uses Paillier encryption to generate a simulated **EBF**.
3. In the simulated Cardinality Test phase, since $\mathcal{S}_S$ does not know the receiver's set Y , it generates random Paillier ciphertexts to simulate C'_j for each y_j , and invokes the SCP simulator to produce n comparison results T'_j .
4. Then, $\mathcal{S}_S$ generates a random partial decryption value b

to simulate the partial decryption result that S sends to R in the real protocol.

5. Finally, when simulating the standard PSI protocol, $\mathcal{S}_S$ invokes the simulator of this sub-protocol with the input X' masked by b .

The indistinguishability of the simulation in this case follows from:

1. By the IND-CPA security of Paillier encryption, the real and simulated **EBF** are computationally indistinguishable.
2. By the homomorphic and rerandomization properties of Paillier encryption and the security of the SCP protocol, the random ciphertexts C'_j and comparison results T_j generated by the simulator are computationally indistinguishable from those in the real protocol.
3. In the real protocol, b is the partial decryption of a_1 under sk_2 . Due to the security of threshold Paillier decryption, the partial decryption values are random; hence, b is computationally indistinguishable from the randomly generated value in the simulator.
4. In the final intersection computation, since b is computationally indistinguishable, the masked set X' is also computationally indistinguishable.

Therefore, for all PPT environments $\mathcal{Z}$, we have

$$\text{IDEAL}_{\mathcal{F}_{\text{TPSI}}, \mathcal{S}_S, \mathcal{Z}} \approx_c \text{REAL}_{\Pi_{\text{TPSI}}, \mathcal{A}_S, \mathcal{Z}},$$

where $\mathcal{S}_S$ is the simulator for Case 1 and $\mathcal{A}_S$ is the adversary corrupting S .

Case 2: adversary corrupts the receiver R . In this case, the simulator $\mathcal{S}_R$ operates as follows:

1. $\mathcal{S}_R$ submits the real input Y of R to the ideal functionality $\mathcal{F}_{\text{TPSI}}$ and directly obtains the output: if the threshold condition is satisfied, it receives the real intersection I ; otherwise, it receives the empty set $\emptyset$.

2. At the beginning of the simulation, $\mathcal{S}_R$ randomly selects k hash functions and sends them to the simulated S .

3. When simulating the step of receiving **EBF**, since $\mathcal{S}_R$ does not know the sender's real set X , it randomly generates an **EBF'**.

4. Then, $\mathcal{S}_R$ invokes the SCP sub-protocol simulator to generate comparison results T_j , ensuring that $T_j = \text{Enc}(1)$ if and only if $y_j \in I$.

5. Next, it generates a random partial decryption value b and adjusts c so that the joint decryption result is consistent with e .

6. Finally, based on the value of e , $\mathcal{S}_R$ masks Y accordingly: if $e = 1$, it masks Y with a new random number r to obtain Y' ; if $e = 0$, it masks Y with b to obtain Y' . Then, it invokes the standard PSI protocol simulator to complete the computation.

The indistinguishability of the simulation in this case follows from:

1. Due to the pseudo-randomness of the hash function and the IND-CPA security of Paillier encryption, the simulated **EBF'** is computationally indistinguishable from the real **EBF**.

2. By the security of the SCP protocol and the IND-CPA security of Paillier, the simulated T_j are computationally indistinguishable from the real ones.

3. Due to the security of threshold Paillier decryption, the partial decryption values are randomly distributed; hence, the simulated b is computationally indistinguishable from the real one received.

4. Since b is computationally indistinguishable, when $e = 0$, the simulated Y' is computationally indistinguishable from the real one. When $e = 1$, both the real protocol and the simulator use a random number to XOR with Y , so the two are computationally indistinguishable.

Therefore, for all PPT environments $\mathcal{Z}$, we have

$$\text{IDEAL}_{\mathcal{F}_{\text{TPSI}}, \mathcal{S}_R, \mathcal{Z}} \approx_c \text{REAL}_{\Pi_{\text{TPSI}}, \mathcal{A}_R, \mathcal{Z}},$$

where $\mathcal{S}_R$ is the simulator for Case 2 and $\mathcal{A}_R$ is the adversary corrupting R .

Combining Case 1 and Case 2, we have constructed simulators for both corruption scenarios and shown that their outputs are computationally indistinguishable from the views of the real protocol. Hence, the protocol Π_{TPSI} securely realizes the ideal functionality $\mathcal{F}_{\text{TPSI}}$ in the semi-honest adversary model.

4.4 Complexity analysis

4.4.1 Computational complexity

The computational complexity of our protocol primarily stems from cryptographic primitive operations and can be decomposed into the following key components: The sender needs to perform $O(k \cdot n)$ hash operations to construct the Bloom filter, conduct $O(\ell)$ Paillier encryption operations to generate the EBF, and execute $O(n)$ homomorphic operations and secure comparison protocols; the receiver requires $O(k \cdot n)$ hash computations and $O(n)$ secure comparison protocols. Here, n

represents the size of both parties' sets, k is the number of hash functions, and ℓ is the length of the Bloom filter. The protocol also involves $O(1)$ partial decryptions and $O(n)$ set masking operations. Overall, the computational complexity of the protocol is $O(k \cdot n)$.

4.4.2 Communication complexity

The communication complexity of the protocol is mainly reflected in two aspects: data transmission volume and number of interaction rounds: In terms of data transmission, the protocol needs to transmit $O(\ell)$ Paillier ciphertexts, $O(n)$ secure comparison protocol ciphertexts, and $O(1)$ partial decryption results; in terms of interaction rounds, the protocol requires constant rounds for initialization, 1 round for EBF transmission, $O(n)$ rounds for secure comparison protocol interactions, and 1 round for partial decryption interaction, plus the communication rounds required by the standard PSI protocol. The overall communication complexity is $O(\ell)$, where ℓ is proportional to the sender's set size n .

5 Our multi-party threshold labeled private set intersection protocol

In privacy-sensitive practical applications, there often arises a class of collaborative computing requirements: all parties wish to perform joint computations while protecting their local private data, disclosing only aggregated results without leaking individual data details. While traditional multi-party PSI protocols assume symmetric roles and aim for a joint intersection, many emerging applications adopt a client-server architecture where the server aggregates information from multiple clients without revealing individual data. This many-to-one paradigm, though highly practical, has received limited attention in the PSI literature. To fill this gap, we focus on the following scenario: multiple clients need to hide the elements of their respective private sets from a server, and the server can compute corresponding label values only when the size of the intersection with each client's set reaches an independently preset threshold.

We formally define the problem as follows: Consider $m - 1$ clients, each holding a private set, and one server. Only when the size of the intersection between a client's set and the server's set reaches a preset threshold t , will the protocol enable that client to compute its corresponding sub-label. These sub-labels $\{l_1, l_2, \dots, l_{m-1}\}$ are then securely aggregated to jointly generate the final decision label, without leaking any real set elements throughout the entire process.

To address this problem, we propose the MTLPSI protocol. Unlike traditional multi-party threshold PSI, which aims to determine whether the overall intersection of all participating parties reaches a threshold and output that intersection, MTLPSI allows each client to independently compare with the server. Only when the intersection size between a client and the server exceeds its preset threshold t does the client output the correct label. Its ideal functionality is defined as shown in Fig. 4.

The protocol features multi-party privacy protection, a threshold-based label reconstruction mechanism, passive

collusion resistance in the semi-honest model, and support for participant dropout, making it suitable for scenarios with high demands for both privacy and collaboration, including:

1. IoT Threat Collaborative Detection: Multiple regional gateways compare device anomaly behavior characteristics. When the number of cross-gateway detections reaches a threshold, a “high-risk device” label is triggered, initiating an isolation mechanism without exposing specific device locations.

2. Cybersecurity Collaborative Defense: Enterprises, cloud service providers, and operators share threat features (e.g., malicious IP addresses). When multiple parties simultaneously detect the same threat, a “global high-risk threat” label is generated to promote joint defense, while allowing participants to selectively exit for commercial reasons.

3. Smart Home Privacy-Aware Collaboration: Multiple devices within a household (e.g., cameras, health monitors) automatically trigger an “emergency event” label and notify contacts when the intersection of detected abnormal user states (e.g., falls, abnormal physiological signs) satisfies predefined conditions, without leaking specific behavioral data.

In summary, the MTLPSI protocol extends traditional threshold PSI in both functional model and security mechanisms, making it suitable for collaborative scenarios that require balancing privacy protection and conditional triggering.

5.1 Protocol overview

The MTLPSI protocol enables a server to securely compute labels for multiple clients based on a threshold set intersection condition, without disclosing the actual elements of any participant’s set. The protocol proceeds in six phases:

1. Initialization: The server establishes cryptographic parameters and distributes hash functions to all clients for Bloom filter construction.

2. Encrypted Label Generation: The TTP precomputes label shares protected by symmetric encryption and secret sharing, ensuring that labels remain confidential unless the threshold condition is met. The TTP is assumed to be fully trusted only during this phase, after which it remains offline and takes no part in the online protocol. In the event of TTP compromise, the adversary could learn label shares and secret key, but crucially, no client’s set elements or the server’s set are exposed, as the TTP never receives any set data. In practice, the TTP can be instantiated as a hardened, air-gapped system or replaced by a distributed setup protocol.

3. EBF Generation: Each client constructs an encrypted Bloom filter representing their private set, providing an efficient and privacy-preserving representation for subsequent computations.

4. Cardinality Threshold Test: Through homomorphic operations and secure multi-party comparisons, the protocol verifies whether each client-server pair satisfies the intersection cardinality threshold t without revealing the actual intersection.

5. Label Computation: This phase is executed conditionally upon a successful threshold test. It reconstructs the final label through a secure sequence of homomorphic aggregation, distributed threshold decryption, and secret-sharing-

Parameters: Number of parties m , intersection threshold t , security parameter λ , label reconstruction threshold t_L , decryption threshold t_h , symmetric key security parameter μ .

Input:

1. Clients $P_1, \dots, P_{m-1}$ each input a private set S_i (of size n_i).
2. Server P_m inputs a private set S_m .

Output:

1. If there exist at least $\max(t_h, t_L)$ clients P_i (for $i = 1, \dots, m-1$), and among these clients, more than t_L satisfy $|S_i \cap S_m| \geq t$, then P_m outputs the label L ; otherwise, P_m outputs a random number.
2. The other parties $P_1, \dots, P_{m-1}$ receive no output.

Fig. 4 Ideal functionality for multi-party threshold labeled private set intersection protocol $\mathcal{F}_{\text{MTLPSI}}$

based recovery, ensuring correctness even if some participants drop out during the process.

6. Output: Depending on whether sufficient threshold conditions are met, the server obtains either the valid label value L or a random value, which prevents leakage of information about the underlying sets.

The complete protocol specification is presented in Fig. 5.

5.2 Correctness analysis

Only when the intersection size between client P_i and the server, $|S_i \cap S_m| \geq t$, the cardinality threshold test produces a valid ciphertext judgment. This result enables the client to obtain a valid secret share ciphertext. If there are no fewer than t_L qualifying clients, the original label L can be correctly reconstructed from these valid shares. Conversely, for clients that do not satisfy the intersection size condition, the protocol ensures that they receive randomized invalid shares, which provide no useful information during reconstruction. If the number of invalid shares exceeds $m - t_L + 1$, the label cannot be correctly recovered.

During protocol execution:

1. t controls the intersection cardinality; a client obtains a valid share only if the intersection size is greater than t .
2. t_h specifies the minimum number of clients required for threshold decryption.
3. The label threshold t_L is the minimum number of valid clients required to jointly reconstruct the final label.

Although these three thresholds are not directly coupled, to correctly recover the label L , the number of online valid clients must be no less than $\max(t_L, t_h)$.

5.3 Security analysis

Theorem 2 In the random oracle model, assuming that the threshold Paillier encryption and the symmetric encryption scheme are IND-CPA secure, the SCP is semi-honest secure, and Shamir’s secret sharing is information-theoretically secure, the protocol Π_{MTLPSI} securely realizes the ideal functionality $\mathcal{F}_{\text{MTLPSI}}$ in the semi-honest adversary model.

For every real-world adversary $\mathcal{A}$, we construct an ideal-world simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$,

$$\text{IDEAL}_{\mathcal{F}_{\text{MTLPSI}}, \mathcal{S}, \mathcal{Z}} \approx_c \text{REAL}_{\Pi_{\text{MTLPSI}}, \mathcal{A}, \mathcal{Z}}.$$

Parameters: Intersection threshold t , security parameter λ , label reconstruction threshold t_L , decryption threshold t_h , symmetric key security parameter μ .

Input: $m - 1$ clients $P_1, \dots, P_{m-1}$ and a server P_m , each holding a private dataset S_i of size n_i . Let pk be the public key of the threshold Paillier encryption scheme, whose private key sk is shared among all m participants, with each party holding a decryption key share sk_i .

Protocol execution:

Initialization: The server selects k hash functions $\{h_1, h_2, \dots, h_k\}$ for generating the **BF**, then sends them to the clients.

Encrypted label generation (by TTP):

1. Generate the label value L and a symmetric encryption key key .
2. Use threshold Shamir secret sharing to generate $m - 1$ label shares: $l_i = \text{Share}(L, t_L)$ for $i = 1, \dots, m - 1$.
3. Encrypt each label share: $el_i = \text{SymEnc}(l_i, key)$.
4. Send el_i secretly to client P_i , and send (L, key) secretly to the server P_m .

EBF generation (by clients P_1-P_{m-1}):

1. Compute **BF** _{i} corresponding to their own set data.
2. Encrypt each element in **BF** _{i} using public key pk to obtain **EBF** _{i} .
3. Send **EBF** _{i} to P_m .

Cardinality threshold test:

1. For each **EBF** _{i} and each $s_j^m \in S_m$, P_m computes $C_{ij}^z = \text{EBF}_i(h_z(s_j^m))$, and homomorphically sums $C_{ij} = \sum_{z=1}^k C_{ij}^z$.
2. P_m rerandomizes: $C'_{ij} = \text{Rerandom}(C_{ij})$.
3. P_m and P_i run $n(m - 1)$ instances of secure multi-party comparison $\text{SCP}(C'_{ij}, \text{Enc}(k)) \rightarrow T = \{T_1, T_2, \dots, T_{m-1}\}$ where $T_i = [\dots]_n \in \{\text{Enc}(0), \text{Enc}(1)\}$.
4. P_m computes $h \sum_{j=1}^n T_{ij}$. This value will be used after rerandomization to determine whether the intersection size between P_i and P_m exceeds the threshold.
5. P_m rerandomizes: $sT_i = \text{Rerandom}(h \sum_{j=1}^n T_{ij})$.
6. P_m and P_i run $m - 1$ instances of secure multi-party comparison: $\text{SCP}(sT_i, \text{Enc}(t)) \rightarrow E$, where each $E_j \in \{\text{Enc}(0), \text{Enc}(1)\}$, forming a vector $E = [E_1, \dots, E_{m-1}]$.

Label computation:

1. P_i performs homomorphic addition: $pel_i = (E_i +_h \text{Enc}(el_i))$ and sends pel_i to each P_j ($j \neq i$).
2. P_j ($j = 1, 2, \dots, m$) partially decrypts pel_i : $pel_i^j = \text{partDec}(pel_i, sk_j)$. Then P_j ($j \neq m$) sends result to P_m .
3. P_m combines decryption: $el'_i = \text{combine}(pel_i^1, pel_i^2, \dots, pel_i^m)$, then decrypts: $l'_i = \text{SymDec}(el'_i, key)$.
4. Execute secret recovery: $L' = \text{Recon}(l'_1, l'_2, \dots, l'_{m-1})$.

Output: Output L' .

Fig. 5 Our MTLPSI protocol Π_{MTLPSI}

We consider the following two corruption cases separately: (1) the adversary corrupts the server P_m and a subset of clients; (2) the adversary corrupts only a subset of clients (the server remains honest).

Case 1: adversary corrupts the server P_m and a subset of clients. Let the corrupted client set be $\mathcal{C}_{\text{corr}}$ with $|\mathcal{C}_{\text{corr}}| < \min(t_h, t_L)$. The simulator $\mathcal{S}_1$ proceeds as follows:

Firstly, $\mathcal{S}_1$ receives from the environment $\mathcal{Z}$ the inputs of the corrupted parties: the server input (S_m, L) and the corrupted client inputs $\{S_i\}_{P_i \in \mathcal{C}_{\text{corr}}}$. It forwards these inputs to the ideal functionality $\mathcal{F}_{\text{MTLPSI}}$ and obtains the output: if the condition is satisfied, it receives the label L ; otherwise, it receives a random number.

Secondly, $\mathcal{S}_1$ simulates the protocol execution as follows:

1. Initialization: Randomly select k hash functions and send them to all clients.
2. Encrypted label generation: If the ideal functionality outputs L , set $L^* = L$; otherwise generate a random L^* . Generate a random symmetric key key^* . Using Shamir's secret sharing, generate shares l_i^* and compute $el_i^* = \text{SymEnc}(l_i^*, key^*)$. Send el_i^* to each simulated client, and send (L^*, key^*) to the simulated server.
3. EBF generation: For each honest client $P_j \notin \mathcal{C}_{\text{corr}}$, construct an all-zero Bloom filter and encrypt it to produce a simulated **EBF** _{j} . For each corrupted client, generate **EBF** _{i}

using its real input.

4. Cardinality threshold test: For each client P_i , $\mathcal{S}_1$ simulates the SCP protocol between the server and the client. First, $\mathcal{S}_1$ determines a set $\mathcal{C}'$ of clients that satisfy the condition: if the ideal functionality outputs L , then $|\mathcal{C}'| \geq t_L$; otherwise $|\mathcal{C}'| < t_L$. The set $\mathcal{C}'$ must include all corrupted clients that truly satisfy the condition (i.e., $P_i \in \mathcal{C}_{\text{corr}}$ and $|S_i \cap S_m| \geq t$); the remaining clients in $\mathcal{C}'$ are chosen randomly from the honest clients. For each client $P_i \in \mathcal{C}'$ that satisfies the condition, set $E_i = \text{Enc}(1)$; otherwise set $E_i = \text{Enc}(0)$. $\mathcal{S}_1$ invokes the SCP simulator to generate a simulated SCP execution such that the output is the corresponding E_i .

5. Label computation: For each client P_i , compute $pel_i = E_i + \text{Enc}(el_i^*)$ (homomorphic addition). Simulate partial decryptions: for corrupted clients $P_i \in \mathcal{C}_{\text{corr}}$, $\mathcal{S}_1$ can compute the real partial decryption (since it knows sk_i); for honest clients, generate random partial decryption values. $\mathcal{S}_1$ simulates the server's combination of decryptions: for $P_i \in \mathcal{C}'$, decrypt pel_i to obtain el_i^* , then decrypt el_i^* to obtain l_i^* ; for $P_i \notin \mathcal{C}'$, the decryption result is random. Finally, if $|\mathcal{C}'| \geq t_L$, the recovered label is L^* ; otherwise recovery fails.

Finally, $\mathcal{S}_1$ sends all messages generated during the simulation to the adversary $\mathcal{A}$.

The indistinguishability of the simulation follows from:

1. The random choice of hash functions is identical to the

real protocol.

2. For the EBF_i of honest clients, the IND-CPA security of threshold Paillier encryption ensures that the simulated EBF_i is computationally indistinguishable from the real one.

3. In the cardinality threshold test, the security of the SCP protocol guarantees that the simulated comparison results are indistinguishable from the real ones; the setting of E_i is consistent with the ideal functionality output, and the distribution of the ciphertexts is guaranteed by IND-CPA security.

4. In the label computation, the simulation of el_i^* relies on the IND-CPA security of symmetric encryption; the simulation of partial decryptions is guaranteed by the security of the threshold decryption algorithm. If the ideal functionality outputs L , the recovered label is L ; otherwise recovery fails or yields a random value, consistent with the real protocol.

Thus, for all PPT environments $\mathcal{Z}$,

$$\text{IDEAL}_{\mathcal{F}_{\text{MTLPSI}}, \mathcal{S}_1, \mathcal{Z}} \approx_c \text{REAL}_{\Pi_{\text{MTLPSI}}, \mathcal{A}_1, \mathcal{Z}},$$

where $\mathcal{S}_1$ is the simulator for Case 1 and $\mathcal{A}_1$ is the adversary in Case 1.

Case 2: adversary corrupts only a subset of clients. Let the corrupted set be $\mathcal{C}_{\text{corr}}$ with $|\mathcal{C}_{\text{corr}}| < t_h$ (the server is honest). The simulator $\mathcal{S}_2$ operates as follows:

Firstly, $\mathcal{S}_2$ receives from $\mathcal{Z}$ the inputs of the corrupted clients $\{S_i\}_{P_i \in \mathcal{C}_{\text{corr}}}$ and submits them to $\mathcal{F}_{\text{MTLPSI}}$.

Secondly, $\mathcal{S}_2$ simulates the protocol:

1. Initialization: Randomly select k hash functions and send them to all clients.

2. Encrypted label generation: If the ideal functionality outputs L , set $L^* = L$; otherwise generate a random L^* . Generate a random symmetric key key^* . Using Shamir's secret sharing, generate shares l_i^* and compute $el_i^* = \text{SymEnc}(l_i^*, \text{key}^*)$. Send el_i^* to each simulated client, and send (L^*, key^*) to the simulated server.

3. EBF generation: For each corrupted client $P_i \in \mathcal{C}_{\text{corr}}$, generate EBF_i using its real input S_i ; for each honest client, generate an all-zero Bloom filter and encrypt it to obtain a simulated EBF_i .

4. Cardinality threshold test: For each corrupted client $P_i \in \mathcal{C}_{\text{corr}}$, $\mathcal{S}_2$ invokes the SCP simulator to generate simulated server messages and randomly generates E_i (uniformly chosen as $\text{Enc}(0)$ or $\text{Enc}(1)$). Since $|\mathcal{C}_{\text{corr}}| < t_h$, the corrupted clients cannot decrypt E_i to verify its correctness.

5. Label computation: For each corrupted client P_i , $\mathcal{S}_2$ generates simulated pel_j for other clients $P_j (j \neq i)$ as random Paillier ciphertexts. Simulated partial decryption values are also random.

Finally, $\mathcal{S}_2$ sends the simulated messages to $\mathcal{A}$.

The indistinguishability of the simulation follows from:

1. The hash functions are chosen randomly as in the real protocol.

2. For the EBF_i of honest clients, IND-CPA security of threshold Paillier encryption makes the simulated EBF_i indistinguishable from the real one.

3. In the cardinality threshold test, E_i are ciphertexts and the corrupted clients cannot decrypt them (since $|\mathcal{C}_{\text{corr}}| < t_h$); the randomly generated E_i are indistinguishable from the real distribution. The SCP simulator ensures the indistinguishability of protocol messages.

4. In label computation, pel_j and partial decryption values are randomly generated; because corrupted clients cannot decrypt and due to the security of symmetric encryption and threshold decryption, the simulated view is computationally indistinguishable from the real view.

Therefore, for all PPT environments $\mathcal{Z}$,

$$\text{IDEAL}_{\mathcal{F}_{\text{MTLPSI}}, \mathcal{S}_2, \mathcal{Z}} \approx_c \text{REAL}_{\Pi_{\text{MTLPSI}}, \mathcal{A}_2, \mathcal{Z}},$$

where $\mathcal{S}_2$ is the simulator for Case 2 and $\mathcal{A}_2$ is the adversary in Case 2.

Combining both cases, the simulators constructed for the two corruption scenarios generate views that are computationally indistinguishable from the real protocol execution. Hence, protocol Π_{MTLPSI} securely realizes the ideal functionality $\mathcal{F}_{\text{MTLPSI}}$ in the semi-honest adversary model.

The above simulation establishes that an adversary controlling a subset of clients cannot distinguish the protocol execution from a simulated one, even if the corrupted clients collude. This holds because the adversary lacks sufficient shares to decrypt the threshold ciphertexts (since $|\mathcal{C}_{\text{corr}}| < t_h$) and cannot recover the label shares (since $|\mathcal{C}_{\text{corr}}| < t_L$). Moreover, the symmetric encryption of label shares ensures that even if the adversary obtains some el_i^* values, they remain indistinguishable from random without the key. Thus, the protocol provides passive collusion resistance in the semi-honest model.

5.4 Complexity analysis

5.4.1 Computational complexity

The computational complexity of our protocol primarily stems from cryptographic operations and set processing in the multi-party environment. The server side requires $O(n \cdot m \cdot k)$ basic operations, specifically including EBF position access computation for each client, homomorphic summation, and execution of secure comparison protocols. The computational overhead on the client side is $O(n \cdot k + \ell)$, where n is the client set size and ℓ is the Bloom filter length, mainly comprising BF construction, encryption operations, and partial decryption. The protocol also involves $O(m)$ Shamir secret sharing operations and symmetric encryption/decryption, as well as $O(m^2)$ share recovery computations (when using Lagrange interpolation). Overall, the computational complexity of the protocol is $O(n \cdot m \cdot k + m \cdot \ell + m^2)$.

5.4.2 Communication complexity

The communication complexity of the protocol is reflected in two dimensions: multi-party interactions and data transmission volume. In terms of interaction rounds, the protocol requires constant rounds for initialization, 1 round for EBF transmission, $O(n \cdot m)$ rounds for secure comparison protocol interactions, and $O(m)$ rounds for partial decryption and share exchange, resulting in an overall interaction round complexity of $O(n \cdot m)$. In terms of data transmission volume, the protocol needs to transmit $O(m \cdot \ell)$ Paillier ciphertexts, $O(n \cdot m)$ ciphertexts for secure comparison protocol execution, $O(m)$ partial decryption results, and $O(m)$ encrypted label shares. Considering that ℓ is typically proportional to the client set size n , the overall communication complexity is $O(n \cdot m)$.

6 Implementation and performance evaluation

We implemented the proposed protocol in C++, leveraging the GMP and NTL libraries for cryptographic operations and employing MurmurHash3 as the hash function for the Bloom filter.

6.1 Experimental setup

We conducted all experiments in a virtualized environment, where each guest machine was configured with 8 vCPUs and 8 GB of RAM, running Ubuntu 24.04.03 LTS. The host machine featured an AMD Ryzen 7 4800H processor with 16 GB of RAM. To evaluate performance under different network conditions, we performed experiments across two simulated network settings: a low-latency, high-bandwidth LAN with 10 Gbps bandwidth and 0.2 ms latency, and a higher-latency, lower-bandwidth WAN with 50 Mbps bandwidth and 80 ms latency.

The false positive rate p of the Bloom filter directly affects both the threshold test accuracy and the communication overhead. As derived in Section 4.2, the error probability is bounded by $P_{\text{err}}^{\max} \approx (n - t + 1) \cdot p$, indicating that the error decreases linearly with p . Meanwhile, the Bloom filter length required to achieve a given p is $\ell = -n \ln p / (\ln 2)^2$, and the communication cost is proportional to ℓ . Thus, reducing p from 10^{-2} to 10^{-3} reduces the error bound by one order of magnitude but increases the communication overhead by a factor of 1.5; further reducing p to 10^{-4} doubles the communication overhead relative to $p = 10^{-2}$. In our experiments, we fix $p \leq 1\%$ as a practical balance between accuracy and efficiency. To ensure the false positive rate of the Bloom filter remained below 1%, we fixed its number of hash functions to $k = 7$ and its length to $l = 7n / \ln 2$ bits.

6.2 Performance evaluation of TPSI protocol

We evaluate the performance of the proposed TPSI protocol through a series of experiments. First, we compare its efficiency against the current state-of-the-art protocol by Hu et al. (2024). Table 3 reports the results. Our protocol achieves substantial computational speedups, ranging from $24.6\times$ to $1965\times$ over Hu's scheme for set sizes from 10 to 1000. For example,

with 1000 elements, our total running time is only 29.73 s, whereas Hu's requires 58 426.4 s. This improvement stems from the use of lightweight additive homomorphic encryption and a secure comparison protocol, which avoids the prohibitive cost of fully homomorphic encryption employed in the reference design. Regarding communication, our total overhead varies between $0.62\times$ and $3.42\times$ that of Hu's protocol across the tested scales. Notably, at the largest comparable set size (1000), our communication drops to $0.62\times$ of Hu's, a reduction of 38%, which further highlights the practical appeal of our approach.

For set sizes beyond 1000, we exclude Hu et al.'s protocol from benchmarking, as its running time already exceeds 58 000 s at 1000 elements, rendering larger-scale comparisons impractical. Instead, we solely evaluate our protocol under simulated LAN and WAN conditions, with set sizes ranging from 2^{11} to 2^{16} and varying intersection thresholds. As shown in Table 4, both runtime and communication are virtually insensitive to the required intersection ratio, confirming that the overhead is independent of the final intersection cardinality. The performance scales approximately linearly with the set size, in line with theoretical expectations.

In summary, the proposed TPSI protocol delivers a significant computational advantage over the state-of-the-art, with communication costs remaining within a reasonable and often favorable range. It exhibits threshold-insensitivity and good linear scalability, making it particularly suitable for resource-constrained IoT devices. For a medium-scale dataset ($n = 2^{12}$), the total protocol completes in roughly 129 s under WAN conditions with a communication cost of about 31.2 MB. This performance profile demonstrates its viability for privacy-preserving threshold set computation among edge devices with limited computing power and bandwidth, offering a practical cryptographic solution for collaborative analysis in IoT environments.

6.3 Performance evaluation of MTLPSI protocol

To evaluate the performance of our proposed MTLPSI protocol, we compare it with the T-MPSI protocol proposed by Bay et al. (2022), which shares the core cryptographic

Table 3 Communication cost (KB) and running time (s) comparisons of the proposed protocol with the protocol of Hu et al. (2024) in the balanced setting

Set size	Protocols	Communication cost			Running time		
		Total	Secret token generation	Standard PSI	Total	Secret token generation	Standard PSI
10	Hu's	28.1	25.3	2.8	11.83	11.74	0.09
	Ours	31.16	28.36	2.8	0.48	0.39	0.09
50	Hu's	52.9	38.8	14.1	59.73	59.59	0.14
	Ours	153.91	139.81	14.1	1.96	1.82	0.14
100	Hu's	106.9	78.8	28.1	234.44	234.25	0.19
	Ours	307.2	279.1	28.1	3.79	3.60	0.19
200	Hu's	179.5	123.3	56.2	824.76	824.48	0.28
	Ours	613.91	557.71	56.2	7.42	7.14	0.28
500	Hu's	462.25	346.85	115.4	4858.21	4857.43	0.78
	Ours	909.03	793.63	115.4	14.65	13.87	0.78
1000	Hu's	2746.1	2489.7	256.4	58 426.4	58 425.0	1.38
	Ours	1694.6	1438.2	256.4	29.73	28.45	1.38

Table 4 Running time (s) and communication cost (MB) of TPSI under different intersection threshold conditions

Set size	Ratio	Running time		Comm
		LAN	WAN	
2^{11}	0.7	59.94	65.45	15.60
	0.8	60.77	65.648	15.61
	0.9	61.03	64.98	15.61
2^{12}	0.7	121.36	129.31	31.22
	0.8	121.54	129.44	31.22
	0.9	121.55	129.48	31.23
2^{13}	0.7	239.93	250.97	62.40
	0.8	239.95	251.84	62.41
	0.9	239.88	251.98	62.42
2^{14}	0.7	484.30	512.52	124.72
	0.8	484.29	512.51	124.73
	0.9	484.31	512.51	124.73
2^{15}	0.7	976.43	1014.23	249.94
	0.8	976.24	1014.16	249.94
	0.9	975.99	1014.12	249.93
2^{16}	0.7	1949.23	2022.21	499.58
	0.8	1949.18	2022.53	499.63
	0.9	1949.43	2022.38	499.77

primitives with our MTLPSI (both use additive homomorphic encryption and secure comparison protocols) and represents the most closely related prior work in terms of multi-party threshold functionality. However, T-MPSI computes the actual intersection set rather than a label, whereas our protocol outputs only a label when the threshold condition is satisfied. Despite this functional difference, the shared primitives provide a meaningful basis for performance comparison. Crucially, both protocols provide security in the same semi-honest adversary model and rely on the same cryptographic hardness assumptions underlying additive homomorphic encryption. This ensures that the performance comparison is conducted between protocols with equivalent security guarantees, making the evaluation both fair and meaningful. We evaluate both protocols under identical network conditions and across the same set size range. The comparison focuses on two key metrics: computational efficiency and communication overhead.

The experimental results demonstrate that our protocol exhibits significant and consistent advantages in both computational efficiency and communication overhead. In terms of computational efficiency, as shown in Table 5, our protocol outperforms the baseline in both client-side and total running time under both network environments. This advantage becomes more pronounced as the set size increases. For example, in the LAN environment with $n = 2^{10}$ and $(m-1, t_h) = (10, 9)$, the total running time of our protocol is reduced by approximately 8.8% compared to the baseline. Regarding communication overhead, our protocol also shows superior performance. In all tested scenarios, both the client-side and total communication costs of our protocol are lower than those of the baseline. Taking the case of $n = 2^{10}$ and $(10, 9)$ as an example, our total communication cost is 13.23 MB, which is about 16.7% lower than the baseline's 15.88 MB. More importantly, our protocol places a lighter computational and communication burden on the more resource-constrained client side, which is particularly valuable in practical applications.

To further validate the scalability and robustness of our protocol on larger datasets, we extended the comparison to set sizes ranging from 2^{11} to 2^{16} , covering both LAN and WAN settings. The results are depicted in Fig. 6, which directly contrasts the performance of our protocol against the Bay's baseline at these larger scales. As observed, our protocol consistently outperforms the baseline across all evaluated metrics. For instance, at $n = 2^{16}$, our protocol achieves a significant reduction in total running time under both LAN and WAN conditions, and the communication overhead remains substantially lower than that of the baseline. Moreover, both protocols exhibit an approximately linear growth in runtime and communication with respect to the set size, which conforms to theoretical expectations. Notably, the performance gap between the two protocols widens as the scale increases, further underscoring the practical advantage of our design for large-scale deployments.

We evaluate the robustness of MTLPSI under client dropouts in Fig. 7, with parameters $m \in \{8, 10, 12, 14\}$, $d \in \{1, 2, 3, 4\}$, and $\max(t_h, t_L) = m/2$. The experimental results validate our design rationale: the protocol completes successfully with the correct label L in every tested scenario, as

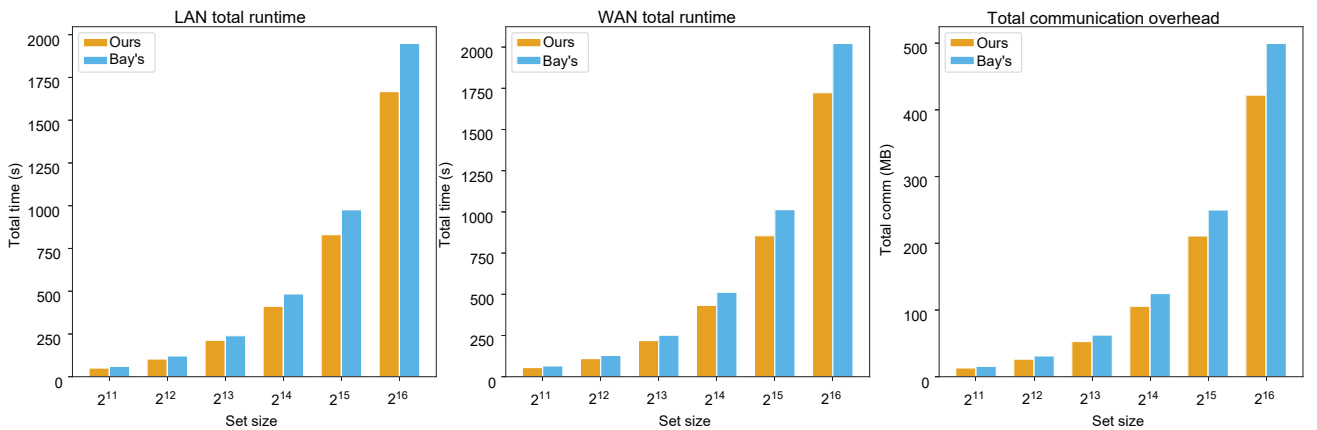
**Fig. 6 Performance comparison of our MTLPSI protocol with Bay et al.'s baseline**

Table 5 Running time (s) and communication cost (MB) of the MTLPSI protocol under different conditions

Setting		Running time LAN setting				Running time WAN setting				Communication cost			
Protocol		Bay's		Ours		Bay's		Ours		Bay's		Ours	
n	$(m-1, t_h)$	Client	Total	Client	Total	Client	Total	Client	Total	Client	Total	Client	Total
2^6	(4,2)	1.338	1.339	1.253	1.254	1.754	1.835	1.898	1.978	0.099	0.389	0.084	0.337
	(4,3)	1.706	1.707	1.506	1.506	2.199	2.279	2.178	2.259	0.099	0.397	0.084	0.337
	(6,3)	2.297	2.298	2.126	2.127	2.789	2.871	2.843	2.924	0.099	0.578	0.085	0.508
	(6,5)	3.337	3.337	3.042	3.042	3.777	3.858	3.725	3.806	0.099	0.596	0.085	0.509
	(8,4)	3.555	3.555	3.357	3.357	4.055	4.136	4.093	4.174	0.099	0.767	0.085	0.682
	(8,7)	5.536	5.536	5.072	5.073	5.998	6.079	5.813	5.894	0.099	0.791	0.085	0.680
	(10,5)	5.147	5.146	4.887	4.888	5.589	5.669	5.601	5.682	0.099	0.959	0.086	0.857
	(10,9)	8.306	8.306	8.110	8.108	8.718	8.798	8.595	8.676	0.099	0.992	0.086	0.857
2^7	(4,2)	2.675	2.676	2.317	2.318	3.528	3.610	3.302	3.383	0.198	0.775	0.166	0.665
	(4,3)	3.473	3.474	2.910	2.910	4.170	4.251	3.942	4.022	0.198	0.792	0.166	0.664
	(6,3)	4.675	4.674	4.173	4.172	5.596	5.677	5.302	5.382	0.199	1.156	0.167	1.003
	(6,5)	6.565	6.565	5.851	5.850	7.597	7.677	7.057	7.137	0.198	1.189	0.167	1.000
	(8,4)	7.121	7.121	6.507	6.506	8.239	8.319	7.899	7.979	0.198	1.535	0.167	1.338
	(8,7)	11.093	11.093	9.842	9.841	12.108	12.188	11.280	11.360	0.198	1.588	0.167	1.340
	(10,5)	10.312	10.311	9.533	9.532	11.611	11.693	11.075	11.156	0.198	1.915	0.168	1.684
	(10,9)	16.623	16.623	15.443	15.443	17.827	17.907	16.874	16.955	0.198	1.983	0.168	1.683
2^8	(4,2)	5.356	5.356	4.518	4.518	6.781	6.862	6.219	6.299	0.396	1.550	0.332	1.327
	(4,3)	6.814	6.815	5.739	5.738	8.204	8.284	7.266	7.346	0.397	1.586	0.332	1.327
	(6,3)	9.155	9.155	8.078	8.077	11.259	11.339	10.448	10.529	0.397	2.314	0.332	1.991
	(6,5)	13.166	13.166	11.426	11.426	15.199	15.280	13.736	13.815	0.397	2.384	0.332	1.993
	(8,4)	14.434	14.434	12.912	12.910	17.054	17.135	15.541	15.621	0.397	3.074	0.333	2.662
	(8,7)	22.045	22.045	19.903	19.903	24.547	24.628	22.650	22.730	0.397	3.179	0.333	2.663
	(10,5)	20.707	20.707	19.029	19.028	24.060	24.140	22.173	22.254	0.397	3.836	0.332	3.323
	(10,9)	33.262	33.260	30.467	30.467	36.049	36.129	33.554	33.634	0.397	3.969	0.333	3.331
2^9	(4,2)	10.677	10.677	8.810	8.809	12.687	12.767	11.001	11.081	0.793	3.103	0.661	2.644
	(4,3)	13.617	13.617	11.113	11.113	15.641	15.722	13.345	13.426	0.794	3.176	0.662	2.646
	(6,3)	18.835	18.834	16.233	16.231	21.519	21.599	18.952	19.031	0.794	4.627	0.661	3.969
	(6,5)	26.175	26.175	23.261	23.259	29.100	29.179	25.918	25.997	0.794	4.762	0.662	3.970
	(8,4)	28.310	28.308	25.703	25.702	32.159	32.238	29.689	29.769	0.795	6.150	0.662	5.293
	(8,7)	43.829	43.829	39.250	39.248	47.605	47.683	42.929	43.009	0.794	6.355	0.660	5.282
	(10,5)	41.749	41.749	37.696	37.695	49.267	49.346	42.437	42.515	0.794	7.664	0.663	6.630
	(10,9)	66.596	66.596	60.891	60.887	71.491	71.569	65.618	65.696	0.793	7.927	0.663	6.634
2^{10}	(4,2)	21.242	21.243	17.534	17.532	24.268	24.346	20.366	20.445	1.588	6.214	1.320	5.281
	(4,3)	26.895	26.894	22.142	22.141	29.955	30.033	25.127	25.204	1.589	6.356	1.322	5.290
	(6,3)	36.415	36.415	31.679	31.676	41.010	41.088	35.889	35.966	1.588	9.249	1.323	7.937
	(6,5)	52.446	52.446	45.786	45.783	56.992	57.072	49.979	50.057	1.590	9.540	1.322	7.933
	(8,4)	57.302	57.301	50.486	50.482	62.204	62.282	55.944	56.022	1.589	12.295	1.322	10.576
	(8,7)	88.128	88.127	78.600	78.596	93.280	93.357	84.472	84.548	1.586	12.690	1.323	10.584
	(10,5)	84.826	84.825	75.267	75.263	90.567	90.643	82.304	82.381	1.588	15.322	1.324	13.236
	(10,9)	134.046	134.045	122.199	122.195	140.710	140.786	127.477	127.554	1.588	15.876	1.323	13.230

$m-d \geq \max(t_h, t_L)$ is always satisfied. While the runtime and communication overhead decrease steadily as more clients drop out (particularly for larger datasets), this is an expected consequence of having fewer active participants. We emphasize that the primary robustness contribution is not the performance reduction itself, but rather the protocol's ability to maintain correctness and security under participant unavailability. This characteristic holds significant practical value for IoT and edge computing scenarios, where edge devices are prone to going offline due to network instability, energy exhaustion, or hardware failures. The dropout tolerance of the MTLPSI protocol ensures that privacy-preserving computation can proceed efficiently even when some devices become unavailable, thereby avoiding the computation interruption or severe performance

degradation typical of traditional multi-party protocols in the face of participant dropouts.

To further evaluate the scalability of our MTLPSI protocol, we investigate its performance as the number of participants increases, with the set size fixed at $n = 2^7$. As summarized in Table 6, both the running time and communication overhead grow as the number of clients rises from 5 to 30.

Under LAN conditions, the total execution time increases from 2.28 s to 60.01 s, while under WAN conditions it rises from 3.43 s to 62.92 s. This growth is primarily attributable to the increased coordination and cryptographic operations required among more participants. Correspondingly, the total communication cost scales from about 0.67 MB to 4.14 MB, reflecting the linear expansion of protocol messages with the

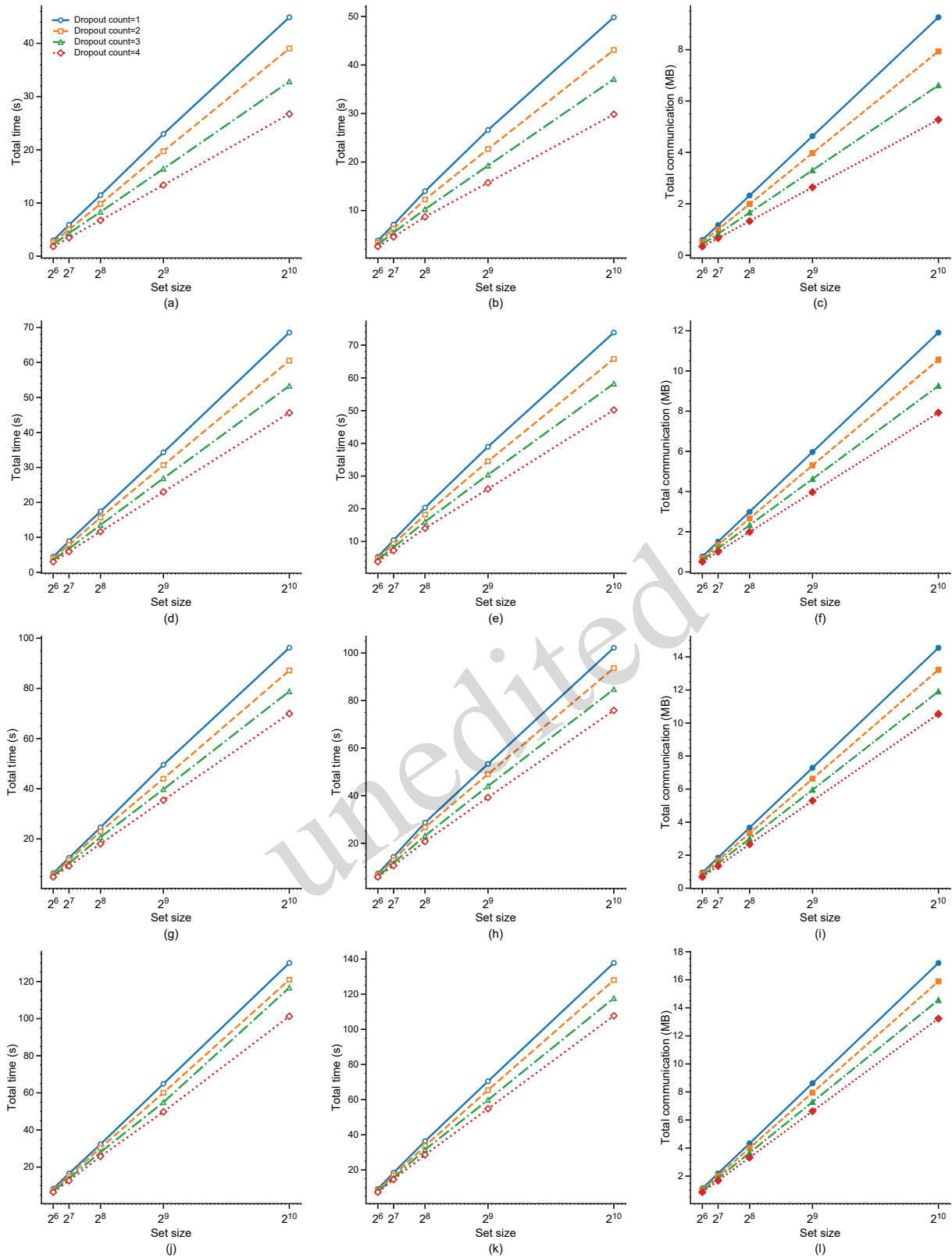


Fig. 7 Impact of dropout rate and set size on running time and communication cost: (a) LAN running time with 8 clients; (b) WAN running time with 8 clients; (c) total communication cost with 8 clients; (d) LAN running time with 10 clients; (e) WAN running time with 10 clients; (f) total communication cost with 10 clients; (g) LAN running time with 12 clients; (h) WAN running time with 12 clients; (i) total communication cost with 12 clients; (j) LAN running time with 14 clients; (k) WAN running time with 14 clients; (l) total communication cost with 14 clients

Table 6 Running time (s) and communication cost (MB) of MTLPSI under different numbers of clients

Setting	Running time				Communication cost	
	LAN		WAN			
Num	Client	Total	Client	Total	Client	Total
5	2.281	2.28	3.345	3.426	0.167	0.667
10	7.752	7.752	9.204	9.285	0.167	1.341
15	15.165	15.165	16.794	16.874	0.168	2.025
20	27.275	27.273	29.523	29.604	0.170	2.719
25	40.61	40.608	42.746	42.826	0.171	3.418
30	60.013	60.01	62.836	62.915	0.172	4.135

participant count. Notably, even with 25 participants, the total running time remains below 50 s in the WAN setting, demonstrating that the protocol can support moderately sized groups while maintaining practical latency.

The experiments in Table 6 evaluate MTLPSI with up to 30 clients, which is representative of typical IoT and edge-computing deployments such as collaborative threat detection or distributed sensor monitoring. For scenarios involving substantially larger participant sets (e.g., $m > 100$), we briefly discuss the expected performance bottlenecks.

The total running time and communication cost scale linearly with m , as analyzed in Section 5.4. The dominant cost factors are: 1. the secure comparison protocol, which requires $O(n \cdot m)$ operations between each client-server pair; 2. the communication rounds, which are latency-sensitive and may become a bottleneck in WAN settings; and 3. the partial decryption phase, where each client performs $O(m)$ operations. While the first two factors grow linearly with m , the partial decryption cost scales quadratically in the worst case if all clients decrypt all ciphertexts. In practice, this can be mitigated by restricting decryption to only the t_h clients that are actively participating, or by employing batching techniques.

We note that for deployments requiring more than 50 participants, several optimizations can be applied: 1. using a star-topology with a high-bandwidth server to reduce latency overhead, 2. parallelizing the secure comparison protocols across multiple threads or nodes, and 3. employing pre-computation techniques to reduce online communication.

Our experiments use virtual machines (8 vCPUs, 8 GB RAM), which are more powerful than typical IoT edge devices. To assess practical deployability on constrained hardware, we estimate the client-side computational cost based on prior benchmarks. On a Raspberry Pi 4 Model B, a 2048-bit Paillier encryption takes approximately 35 ms (Qin et al., 2026). Using the Bloom filter parameters from our experiments ($k = 7$, $\ell = 7n/\ln 2$), for a client with $n = 2^8$, the filter length is $\ell \approx 2586$ bits, and the encryption cost per client is approximately $2586 \times 35 \text{ ms} \approx 90.5 \text{ s}$. For offline or low-frequency IoT applications such as periodic threat monitoring or daily health reporting, this remains feasible. For real-time or ultra-constrained deployments, edge gateway offloading or hardware acceleration would be required. The above results assume an available and uncompromised trusted setup. The TTP only performs offline label, key and share generation and distribution; it does not participate in online computation or access private sets. If the TTP is compromised, label confidentiality

may be affected, so practical deployments require a trusted authority or a distributed alternative.

In summary, our proposed MTLPSI protocol demonstrates efficient performance while delivering enhanced security features such as dropout tolerance and output privacy. It consistently outperforms the baseline protocol in both computational efficiency and communication overhead. Critically, its inherent dropout tolerance ensures stable operation in dynamic environments like IoT and edge computing, making it a practical solution for real-world applications with resource-constrained and intermittently connected devices.

7 Conclusions

In this paper, we present a practical TPSI protocol that achieves an efficient balance between communication and computational overhead. Building on this foundation, we further propose a Multi-party Threshold Labeled PSI protocol. The extended protocol not only improves performance efficiency but also incorporates resilience against client dropouts and passive collusion attacks. This enhancement provides stronger security and operational robustness in practical deployments, thereby demonstrating broader application potential in dynamic real-world scenarios such as IoT and edge computing.

While our protocols demonstrate practical efficiency and strong security in the semi-honest model, several important directions remain for future work. First, reducing or eliminating the reliance on the TTP in MTLPSI is a high priority; this could be achieved by replacing the TTP with a distributed key generation protocol or a decentralized random beacon. Second, extending the security model to malicious adversaries would significantly broaden the applicability of our protocols; this would require additional mechanisms such as zero-knowledge proofs to ensure correct behavior. Third, optimizing the protocol for large-scale deployments (e.g., million-element sets or hundreds of clients) is another promising direction, potentially leveraging batching techniques, GPU acceleration, or more efficient secure comparison protocols. Finally, we plan to explore post-quantum secure instantiations of the underlying primitives to ensure long-term security.

Acknowledgments

This work was supported by the Natural Science Foundation of Shaanxi Province (No. 2025JC-YBMS-664).

Author contributions

Shuai YANG designed the research, processed the data and drafted the paper. Yiliang HAN helped organize the paper. Shuai YANG and Yiliang HAN revised and finalized the paper.

Conflict of interest

Both authors declare that they have no conflict of interest.

Declaration on the use of generative AI tools

During the preparation of this work, the authors used DeepSeek to improve language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

- Bay A, Erkin Z, Hoepman JH, et al., 2022. Practical multi-party private set intersection protocols. *IEEE Trans Inform Forens Secur*, 17:1-15. <https://doi.org/10.1109/TIFS.2021.3118879>
- Bienstock A, Patel S, Seo JY, et al., 2024. Batch PIR and labeled PSI with oblivious ciphertext compression. *Proc 33rd USENIX Conf on Security Symposium*, p.5949-5966.
- Bloom BH, 1970. Space/time trade-offs in hash coding with allowable errors. *Commun ACM*, 13(7):422-426. <https://doi.org/10.1145/362686.362692>
- Branco P, Döttling N, Pu SH, 2021. Multiparty cardinality testing for threshold private intersection. *24th IACR Int Conf on Practice and Theory of Public Key Cryptography on Public-Key Cryptography*, p.32-60. https://doi.org/10.1007/978-3-030-75248-4_2
- Chase M, Miao PH, 2020. Private set intersection in the internet setting from lightweight oblivious PRF. *40th Annual Int Cryptology Conf on Advances in Cryptology*, p.34-63. https://doi.org/10.1007/978-3-030-56877-1_2
- Chen H, Huang ZC, Laine K, et al., 2018. Labeled psi from fully homomorphic encryption with malicious security. *Proc SIGSAC Conf on Computer and Communications Security*, p.1223-1237. <https://doi.org/10.1145/3243734.3243836>
- Chen YC, Huang KC, 2023. JEDI: joint and effective privacy preserving outsourced set intersection and data integration protocols. *IEEE Trans Inform Forens Secur*, 18:4504-4514. <https://doi.org/10.1109/TIFS.2023.3295941>
- Chielle E, Maniatakos M, 2025. Recurrent private set intersection for unbalanced databases with cuckoo hashing and leveled FHE. *Proc Network and Distributed System Security Symp*, p.1-17. <https://doi.org/10.14722/ndss.2025.240365>
- Cong KL, Moreno RC, Da Gama MB, et al., 2021. Labeled PSI from homomorphic encryption with reduced computation and communication. *Proc SIGSAC Conf on Computer and Communications Security*, p.1135-1150. <https://doi.org/10.1145/3460120.3484760>
- Davidson A, Cid C, 2017. An efficient toolkit for computing private set operations. *22nd Australasian Conf on Information Security and Privacy*, p.261-278. https://doi.org/10.1007/978-3-319-59870-3_15
- Feng LB, Liu YF, Hu K, et al., 2024. LPP-BPSI: a location privacy-preserving scheme using blockchain and private set intersection in spatial crowdsourcing. *Fut Gener Comp Syst*, 157:112-123. <https://doi.org/10.1016/j.future.2024.03.036>
- Fouque PA, Poupard G, Stern J, 2001. Sharing decryption in the context of voting or lotteries. *4th Int Conf on Financial Cryptography*, p.90-104. https://doi.org/10.1007/3-540-45472-1_7
- Gao Y, Luo YC, Wang LX, et al., 2024. Efficient scalable multi-party private set intersection(-variants) from bicentric zero-sharing. *Proc SIGSAC Conf on Computer and Communications Security*, p.4137-4151. <https://doi.org/10.1145/3658644.3690245>
- Gao Y, Qi L, Liu X, et al., 2025. Efficient fuzzy private set intersection from fuzzy mapping. *30th Int Conf on the Theory and Application of Cryptology and Information Security on Advances in Cryptology*, p.36-68. https://doi.org/10.1007/978-981-96-0938-3_2
- Garimella G, Pinkas B, Rosulek M, et al., 2021. Oblivious key-value stores and amplification for private set intersection. *41st Annual Int Cryptology Conf on Advances in Cryptology*, p.395-425. https://doi.org/10.1007/978-3-030-84245-1_14
- Ghosh S, Simkin M, 2019. The communication complexity of threshold private set intersection. *39th Annual Int Cryptology Conf on Advances in Cryptology*, p.3-29. https://doi.org/10.1007/978-3-030-26951-7_1
- Ghosh SW, Simkin M, 2023. Threshold private set intersection with better communication complexity. *26th IACR Int Conf on Practice and Theory of Public-Key Cryptography on Public-Key Cryptography*, p.251-272. https://doi.org/10.1007/978-3-031-31371-4_9
- Hu JW, Zhao YJ, Tan BHM, et al., 2024. Enabling threshold functionality for private set intersection protocols in cloud computing. *IEEE Trans Inform Forens Secur*, 19:6184-6196. <https://doi.org/10.1109/TIFS.2024.3402355>
- Kerschbaum F, Biswas D, de Hoogh S, 2009. Performance comparison of secure comparison protocols. *20th Int Workshop on Database and Expert Systems Applications*, p.133-136. <https://doi.org/10.1109/DEXA.2009.37>
- Kolesnikov V, Matania N, Pinkas B, et al., 2017. Practical multi-party private set intersection from symmetric-key techniques. *Proc SIGSAC Conf on Computer and Communications Security*, p.1257-1272. <https://doi.org/10.1145/3133956.3134065>
- Liu FH, Zhang E, Qin LY, 2023. Efficient multiparty probabilistic threshold private set intersection. *Proc SIGSAC Conf on Computer and Communications Security*, p.2188-2201. <https://doi.org/10.1145/3576915.3623158>
- Nevo O, Trieu N, Yanai A, 2021. Simple, fast malicious multiparty private set intersection. *Proc SIGSAC Conf on Computer and Communications Security*, p.1151-1165. <https://doi.org/10.1145/3460120.3484772>
- Qin Q, Yang YJ, Lin JX, et al., 2026. An integrated privacy preserving data aggregation framework for IoT networks using homomorphic encryption and secure computation. *Sci Rep*, 16(1):19663. <https://doi.org/10.1038/s41598-026-48831-6>
- Raghuraman S, Rindal P, 2022. Blazing fast PSI from improved OKVS and subfield VOLE. *Proc SIGSAC Conf on Computer and Communications Security*, p.2505-2517. <https://doi.org/10.1145/3548606.3560658>
- Rindal P, Schoppmann P, 2021. VOLE-PSI: fast OPRF and circuit-PSI from vector-ole. *40th Annual Int Conf on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, p.901-930. https://doi.org/10.1007/978-3-030-77886-6_31
- Rosulek M, Trieu N, 2021. Compact and malicious private set intersection for small sets. *Proc SIGSAC Conf on Computer and Communications Security*, p.1166-1181. <https://doi.org/10.1145/3460120.3484778>
- Uzun E, Chung SP, Kolesnikov V, et al., 2021. Fuzzy labeled private set intersection with applications to private real-time biometric search. *30th USENIX Security Symp*, p.911-928.
- van Baarsen A, Pu SH, 2024. Fuzzy private set intersection with large hyperballs. *43rd Annual Int Conf on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, p.340-369. https://doi.org/10.1007/978-3-031-58740-5_12
- Voigt P, von dem Bussche A, 2017. The EU General Data Protection Regulation (GDPR): a Practical Guide. Springer, Cham, Switzerland. <https://doi.org/10.1007/978-3-031-62328-8>
- Wang RC, Zhou J, Cao ZF, et al., 2024. Updatable private set intersection with forward privacy. *IEEE Trans Inform Forens Secur*, 19:8573-8586. <https://doi.org/10.1109/TIFS.2024.3461475>
- Wu AX, Yang YE, Wen JH, et al., 2025. Non-interactive set intersection for privacy-preserving contact tracing. *J Syst Archit*, 158:103307. <https://doi.org/10.1016/j.sysarc.2024.103307>
- Wu ML, Yuen TH, Chan KY, 2024. O-Ring and K-Star: efficient multiparty private set intersection. *Proc 33rd USENIX Conf on Security Symp*, p.6489-6506.
- Yang XL, Liu YS, He SY, 2024. Overview of privacy set intersection protocol based on heterogeneous network and social network user alignment. *IEEE Trans Netw Sci Eng*, 11(6):6692-6703. <https://doi.org/10.1109/TNSE.2024.3476497>