Privacy Preserving Collaborative Attack Campaign Detection

Master Thesis

Mrityunjaya Palanimurugan Vasanthakumari

Delft University of Technology

TU Delft

Privacy Preserving Collaborative Attack Campaign Detection

by

Mrityunjaya Palanimurugan Vasanthakumari

Student Number: 6227333

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on
Thursday June 18, 2026 at 10:00 AM

Supervisor:	Dr. Zekeriya Erkin
Committee Member:	Dr. Harm Griffioen
Committee Member:	Dr. Jeroen van der Ham-de Vos
Project Duration:	November, 2025 - June, 2026
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover: Content by Securus Communications, visual layout generated via Google Gemini [48, 22]

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Thesis contents were enhanced with the assistance of ChatGPT[43]

TU Delft

Preface

This thesis marks the end of my two years at TU Delft, a journey that has meant far more to me than completing a degree. It has been a period of learning, uncertainty, growth, and reflection. Along the way, I learned a great deal about research, but also about patience, discipline, friendship, and the kind of person I want to become. Most of all, I learned that no meaningful journey is completed alone. I was fortunate to be surrounded by people who made these years memorable.

I am especially grateful to my thesis supervisor, Prof. Dr. Zeki Erkin, for his guidance, encouragement, and trust throughout this project. His commitment to research that creates value beyond academia has been a great source of inspiration for me. I am grateful for the freedom he gave me to explore my ideas, the direction he provided when I needed clarity, and the confidence he helped build during this process. Working under his supervision has been one of the most valuable parts of my academic journey.

I also want to express my appreciation to my peers Siddharth, Maik, Milos, Roderick, Marco, Harvey, Mitchell, and Erin for their thoughtful feedback during the biweekly presentations. Their questions, suggestions, and discussions helped me improve both my research and the way I presented my work. More importantly, their presence made the process feel less solitary and more collaborative.

A special thanks goes to my friends Arjun, Chintan, Sampreet, and Karthikeyan. They were my support system through long days, late nights, difficult moments, and small victories. Thank you for the conversations, the laughter, the patience, and the constant encouragement. Many parts of this journey became easier because they were beside me and this thesis carries a part of their support as well.

Finally, I am deeply thankful to my family in India for their unconditional love and belief in me. Their support has always given me the strength to keep moving forward, especially in moments of doubt. Knowing that they were always cheering for me gave me confidence, comfort, and the courage to make my own decisions. I am deeply grateful for their presence in my life and for everything they have done for me.

Mrityunjaya Palanimurugan Vasanthakumari
Delft, June 2026

Abstract

The Internet connects organizations across finance, government, education, and other sectors, forming a global network of interdependent systems. Although this connectivity enables organizations to provide services, coordinate operations, and communicate with external users, it also increases their exposure to cyber attackers. Attackers exploit connected networks for financial gain, political objectives, or operational disruption, while often hiding their identity through distributed infrastructure such as botnets. These botnets are obtained either by renting existing infrastructure or by manually compromising large numbers of computers. To maximize the value of this infrastructure, attackers reuse the same botnets across multiple targets within a single campaign. Consequently, organizations in the same industry sector, which often expose similar network protocols and services, face related malicious activity from the same attacker-controlled infrastructure.

To defend themselves against such attackers, organizations deploy intrusion detection systems (IDSs). An IDS continuously monitors the incoming and outgoing network traffic, typically represented as packets, and inspects this traffic for signs of malicious behavior. IDSs commonly rely on two primary detection methods. Signature-based detection checks whether the packets match known attack signatures. Anomaly-based detection identifies traffic that deviates from expected or normal network behavior. Together, these methods help organizations detect known attacks, unusual behavior, and suspicious traffic patterns within their own networks.

Although individual IDS deployments provide network defenses, attackers exploit the limitations of local detection. For example, signature-based detection fails when attackers use evasion techniques such as packet fragmentation, where malicious traffic is split into multiple smaller packets that appear harmless when inspected separately. More broadly, a single IDS observes only the traffic directed at its own network. As a result, it sees suspicious activity as isolated local events and lacks the wider context needed to determine whether the same attacker infrastructure is targeting other organizations. Therefore, an IDS alone is not sufficient to identify broader attack campaigns that span multiple networks.

The inability of individual organizations to observe campaign-level activity creates a need for greater cross-organizational visibility. When the same attacker-controlled infrastructure targets organizations in the same sector, shared patterns appear across their network logs. Correlating these observations helps organizations determine whether suspicious activity is isolated or part of a coordinated campaign. However, obtaining such visibility requires multiple organizations to compare information derived from their logs. Directly sharing raw network logs is impractical because logs contain sensitive information, including credentials, internal IP addresses, authentication data, proprietary operational details, and other organization-sensitive information.

The hesitation in sharing raw network logs creates the need for a computational framework that enables secure correlation between organizations. Private Set Intersection (PSI) provides a cryptographic basis for this capability by allowing parties to compute common elements across private sets without revealing non-matching elements. PSI has been extensively studied and has been extended to Multi-party Private Set Intersection (MPSI), which supports n participants. A relevant variant for collaborative threat intelligence is Threshold MPSI, where an element is reported if it appears in at least t out of n private sets. In this thesis, the proposed TMPSI-based solution determines whether a suspicious IP address appears in a threshold number of organizations' network logs and alerts the participating organizations to possible coordinated attack activity. Organizations that have not yet observed the same activity can use the threshold result to proactively strengthen their defenses. In this way, the proposed framework transforms isolated IDS deployments into a collaborative threat intelligence system that preserves privacy, revealing attack campaigns that remain hidden from individual organizations while preserving the confidentiality of local network logs.

Contents

Preface i

Abstract ii

1 Introduction 1
- 1.1 Limitations of Local Intrusion Detection . . . 2
- 1.2 Need for Collaborative Threat Intelligence . . . 3
- 1.3 Privacy Challenges in Collaborative Threat Intelligence . . . 3
- 1.4 Private Set Intersection . . . 4
- 1.5 Research Challenge . . . 4
- 1.6 Key Contributions . . . 5
- 1.7 Thesis Outline . . . 5

2 Preliminaries 6
- 2.1 Security Models . . . 6
  - 2.1.1 Semi-Honest Security . . . 6
  - 2.1.2 Malicious Security . . . 6
- 2.2 Bloom Filter . . . 6
- 2.3 Threshold BFV Homomorphic Encryption . . . 9

3 Related Works 10
- 3.1 Existing literature . . . 10
- 3.2 Complexity comparison . . . 13

4 Protocol Description 15
- 4.1 Protocol Setup . . . 16
- 4.2 Circuit Evaluation . . . 18
- 4.3 Correctness of the Threshold Circuit . . . 23
- 4.4 Protocol Illustrative Example . . . 23
- 4.5 Complexity . . . 25
  - 4.5.1 Communication Complexity . . . 25
  - 4.5.2 Computational Complexity . . . 26
- 4.6 Security Analysis . . . 27
  - 4.6.1 Security Assumptions . . . 27
  - 4.6.2 Leakage . . . 27
  - 4.6.3 Simulation-Based Security Proof . . . 27

5 Evaluation 30
- 5.1 Experimental Setup . . . 30
- 5.2 Datasets . . . 30
- 5.3 Result Analysis . . . 31
  - 5.3.1 Timing Breakdown Across Protocol Phases . . . 31
  - 5.3.2 Effect of Scaling the Number of Parties and Threshold . . . 32
  - 5.3.3 Effect of Scaling Party Set Size . . . 33
  - 5.3.4 Effect of Scaling Intersection Elements . . . 34
  - 5.3.5 Effect of Reducing the False Positive Rate . . . 35
  - 5.3.6 Effect of Threshold on False Candidate Recovery . . . 35

6 Discussion & Future Work 37
- 6.1 Practical Deployment . . . 37
- 6.2 Limitations . . . 38
- 6.3 Future Work . . . 39

7 Conclusion 40

References 41

A Source Code Example 45

A.1 Bloom Filter Construction 45

A.2 Preparing Bloom Filters for Batched OpenFHE Evaluation 45

A.3 Packed BFV Encryption of Bloom Filter Chunks 45

A.4 Encrypted Threshold-Circuit Evaluation 46

1 Introduction

Modern organizations rely on interconnected digital systems to provide services, coordinate operations, and communicate with external users [40]. This dependence is not limited to conventional IT environments. In industrial and cyber-physical settings, communication networks also connect physical processes, sensors, actuators, and control systems to remote monitoring and management infrastructure [53, 26]. Although this connectivity enables more efficient and flexible operations, it also expands the exposure of organizational infrastructure, industrial control systems, and safety-critical environments to malicious threats [53, 26, 40].

This exposure turns connectivity into a security problem by broadening the attack surface. Increased exposure expands the set of points at which attackers enter, cause effects, or extract data [39]. Attackers use this larger attack surface to scan and probe exposed systems for weaknesses in software, configurations, network services, and operational practices [17, 47]. When attackers exploit such weaknesses, organizations risk the loss of sensitive information, operational disruption, financial and reputational damage, and loss of control over critical systems [53, 26, 40].

The same exposure that enables attackers to find weaknesses also allows them to exploit vulnerable systems at scale by relying on botnets. A botnet is a network of compromised machines controlled by an attacker, allowing malicious activity to be distributed across many seemingly unrelated systems [6, 50]. This distribution makes attacks harder to attribute, detect, and mitigate because the attacks do not originate from a single visible source [6, 50]. Beyond their technical value, botnets also have economic value in the cybercrime ecosystem: they are traded, rented, and used by different actors to support multiple malicious activities [33, 50]. Because building, maintaining, or renting such infrastructure requires resources, attackers have an incentive to reuse botnets for multiple attacks [33].

Botnet reuse allows the same distributed infrastructure to support campaigns with different scopes. In targeted campaigns, attackers concentrate resources on a specific victim and adapt the attack to the victim's environment [27]. In broader campaigns, distributed infrastructure supports scanning, probing, or exploitation across multiple organizations [17, 6, 50]. Across both types of campaigns, suspicious indicators provide observable traces of attacker activity. When those traces are relevant beyond the organization that first observes them, they create the basis for collaborative defense [3, 54].

Collaborative defense builds on this shared relevance by allowing organizations to compare suspicious indicators. Indicators such as suspicious IP addresses, scanning attempts, probing behavior, and failed connection attempts help other organizations recognize related activity in their own environments [3, 54]. When such indicators are correlated across network logs from multiple organizations, organizations gain additional context to distinguish isolated local observations from activity associated with a broader attack campaign [3, 54, 27].

However, this collaboration cannot be relying on unrestricted sharing of raw network logs. Such logs contain sensitive operational and personal information, including usernames, internal IP addresses, authentication data, network topology information, and other personally identifiable information [5, 51].

Sharing these logs directly risks exposing internal assets, services, and organizational practices, creating additional security and privacy risks for the contributing organizations [5, 51]. Legal and regulatory requirements further restrict the sharing of sensitive data subject to data-protection rules such as the General Data Protection Regulation [2, 41, 25].

These privacy and legal constraints define the central challenge addressed in this thesis: enabling organizations to compare suspicious indicators across network logs without revealing the sensitive local data from which those indicators are derived [5, 54]. This thesis addresses that challenge by proposing a privacy-preserving correlation of suspicious IP addresses across organizations. Suspicious IP addresses are a natural focus because they appear in network observations and, when associated with scanning or probing behavior, serve as useful indicators for collaborative defense [3, 54, 17].

For collaborative defense, an indicator becomes useful before it appears in every participating organization's logs. A single observation provides limited evidence because it may reflect an isolated local event, while requiring all organizations to observe the same IP address would make the result available only after the malicious activity had reached the entire group. Threshold-based correlation addresses this middle ground by identifying indicators observed by at least a predefined number of participants [36, 23]. To support this form of correlation, this thesis proposes a protocol for identifying IP addresses that appear in the logs of at least a predefined number of participating organizations, without revealing the underlying logs [4]. These reported IP addresses serve as shared warning indicators, allowing organizations that have not yet seen them to strengthen monitoring and defensive controls before related attack attempts reach their networks [3, 54].

1.1. Limitations of Local Intrusion Detection

Organizations commonly deploy Intrusion Detection Systems (IDSs) to detect malicious or policy-violating activity. An IDS supports this goal by monitoring network or system activity and identifying events that indicate attacks, policy violations, or other security-relevant incidents [47]. The monitoring location determines the main deployment category: a network-based IDS inspects traffic at strategic points in a network, while a host-based IDS observes activity on individual machines, including operating-system events, application behavior, file changes, audit logs, and user actions [47].

After collecting network or host activity, IDSs apply detection methodologies to determine whether observed behavior is suspicious. Signature-based detection compares observed activity against known patterns of malicious behavior, making it effective for detecting known attacks. Anomaly-based detection instead models normal behavior and raises alerts when observed activity deviates significantly from that baseline, allowing IDSs to detect activity that does not match known signatures [47]. The effectiveness of both approaches depends on deployment conditions, including the quality of configuration, detection rules, signatures, thresholds, and behavioral models [47, 52]. As a result, reliable intrusion detection remains a challenge in practice [47, 52].

One major source of this challenge is limited visibility in network intrusion detection, especially when encrypted traffic reduces the information available for inspection [47, 18]. Organizations encrypt network traffic to protect confidentiality and prevent unauthorized disclosure of sensitive information [18]. When an IDS does not have access to decrypted traffic, encryption prevents direct inspection of packet payloads [18]. As a result, detection in encrypted deployments relies on packet headers, flow features, timing information, and other metadata rather than full payload contents [18, 44, 42]. Reduced payload visibility weakens signature-based detection when signatures depend on recognizable content patterns within network traffic [47, 18, 44].

Even when packet contents are available for inspection, attackers can exploit differences between how an IDS interprets traffic and how the destination host processes the same traffic [46, 47]. Evasion techniques use these differences by modifying the format, timing, or structure of malicious traffic while preserving the attack's effect on the target system [46, 31]. Packet fragmentation is a representative example of this problem [46]. In a fragmentation-based attack, malicious content is divided across multiple smaller packets, so individual fragments appear incomplete or benign when inspected in isolation. A network IDS must therefore reassemble fragments and reconstruct the original traffic before analysis [46]. Reliable reassembly remains challenging because the IDS must group fragments correctly and interpret reconstructed traffic consistently with the destination host [46]. In high-volume networks,

fragment reassembly and deep inspection add processing requirements to network intrusion detection, while fragmentation remains a practical evasion technique against IDS/IPS systems [47, 31].

Beyond visibility and evasion, operational factors further limit IDS reliability by affecting how alerts are generated, tuned, and validated in practice [47, 52]. IDSs generate false positives when benign activity is incorrectly classified as malicious, and false negatives when malicious activity is not detected [47]. Reducing these errors requires careful configuration of rules, thresholds, signatures, baselines, and other deployment parameters [47]. Machine-learning and anomaly-based approaches have been studied for detecting complex or previously unseen activity, and the literature reports strong performance for many models under experimental conditions [45, 52]. However, performance in controlled experiments does not directly eliminate deployment challenges in real networks. Detection models still require careful feature selection, tuning, representative training data, and manual validation before they operate reliably in practice [52, 45]. Zero-day attacks remain challenging for IDSs because signature-based systems rely on known attack patterns, while anomaly-based and machine-learning systems depend on models, features, and training data that do not always generalize to new real-world behavior [47, 52, 45].

Taken together, these limitations show that an IDS deployed within a single organization provides an incomplete basis for understanding attacker activity [47, 52]. Even when local alerts are generated, the organization often lacks enough context to determine whether they represent isolated events, routine background activity, or part of a broader campaign. Cyber threat intelligence addresses this interpretive gap by relating local observations to wider patterns of attacker behavior.

1.2. Need for Collaborative Threat Intelligence

The limitations of local IDS deployments motivate a broader defensive approach based on cyber threat intelligence (CTI). CTI refers to the collection, analysis, and use of security-relevant information to understand existing and emerging cyber threats [3, 54]. Instead of treating IDS alerts as isolated technical events, CTI places local observations in a wider security context and helps organizations prioritize defensive actions [3, 54].

This wider context is especially important for observations that are weak or ambiguous within one network. Early-stage attack activity, such as reconnaissance, often appears in organizational logs as low-level interaction with public-facing systems rather than as a complete attack [27, 17, 47]. Within a single network, such activity is challenging to prioritize because it may resembles routine background traffic. [47, 52, 28].

Cross-organizational comparison gives these observations additional meaning [3, 54]. A suspicious source observed by one organization provides limited evidence on its own, but the same source appearing across multiple organizations indicates a pattern that is harder to dismiss as isolated noise. When the affected organizations expose similar services, repeated activity provides evidence of shared reconnaissance, reused attack infrastructure, or botnet-driven behavior [27, 6, 50]. However, performing this comparison requires participants to use data derived from their local security infrastructure, creating privacy and confidentiality concerns.

1.3. Privacy Challenges in Collaborative Threat Intelligence

Collaborative threat intelligence depends on observations that are usually extracted from an organization's own security infrastructure [3, 54]. These observations are useful because they are grounded in real operational activity, but that is also what makes them sensitive. A network log records security-relevant activity while also revealing details about the organization's infrastructure, exposed services, communication patterns, and defensive posture [5, 51].

This sensitivity creates a practical barrier to collaboration. Organizations want to know whether suspicious indicators observed in their own logs also appear in other organizations, but disclosing the underlying logs risks revealing more than the indicators themselves [54, 5]. For example, a shared log entry risks exposing which services are reachable, how internal systems communicate, or which assets are repeatedly targeted [5, 51]. As a result, organizations are reluctant to share raw security data even when doing so would improve detection of broader attack campaigns [54, 5].

Legal requirements add a further constraint. Cyber threat information that contains sensitive data must be protected under applicable data-protection rules. Regulations such as the General Data Protection Regulation therefore affect how organizations process and exchange security data in collaborative CTI workflows [2, 25, 41].

These constraints define the privacy requirement for collaborative threat intelligence: organizations need a way to securely compare suspicious indicators across datasets without directly revealing the local logs from which those indicators are derived [5, 54]. This requirement motivates the use of privacy-preserving cryptographic methods for secure correlation.

1.4. Private Set Intersection

The privacy problem in collaborative threat intelligence is a private set comparison problem. Each organization holds a local set of suspicious indicators, such as IP addresses observed in IDS or network logs [3, 54, 47]. The goal is to learn which indicators are common across participants without exposing the underlying logs [5, 51].

Private Set Intersection (PSI) provides a cryptographic foundation for this type of comparison. In a PSI protocol, two parties compute the intersection of their private sets while revealing no additional information about elements outside the intersection. Freedman et al. proposed a foundational two-party PSI construction for this problem [20]. In the threat-intelligence setting, collaboration often involves more than two organizations [3, 54]. This motivates Multi-party Private Set Intersection (MPSI), which allows multiple participants to compute common elements across their private sets while limiting information leakage beyond the intended output [32, 8].

A standard multi-party private set intersection is too restrictive for collaborative threat detection because it reveals only indicators that appear in every participant's input set [8]. In threat intelligence, an indicator observed by a subset of organizations is already relevant when it helps identify activity beyond a single local environment [3, 54]. An IP address observed by several participants provides stronger evidence of shared malicious activity than an isolated local observation [27, 6, 50]. Threshold Multi-party Private Set Intersection (TMPSI) captures this requirement by identifying elements that occur in at least a threshold number of private input sets [36, 4].

1.5. Research Challenge

Although TMPSI provides the required privacy-preserving functionality, translating existing protocols into a deployable collaborative threat-intelligence system remains challenging. Existing TMPSI rely on assumptions that are convenient for cryptographic protocol design but challenging in operational environments. For example, they require direct interaction among participants, repeated communication rounds, or sufficient local computation and communication capacity at every organization [36, 12, 8].

These assumptions conflict with real collaborative security settings. Participating organizations operate under strict security policies, limited connectivity, and different resource constraints. They are also reluctant to maintain direct protocol interactions with every other participant, especially when the collaboration involves mutually distrustful organizations [3, 54, 5]. This creates a gap between TMPSI as a cryptographic primitive and its practical use as part of a collaborative threat-detection workflow.

This thesis addresses that gap for organizations that expose similar network protocols and services. Such organizations face related reconnaissance, scanning, or attack activity, making cross-organizational correlation valuable [17, 27, 6, 50]. At the same time, directly sharing network logs risks revealing sensitive operational information [5, 51]. A practical solution must therefore allow participants to correlate relevant observations across their logs while preserving the confidentiality of each organization's local data and reducing the deployment burden on the participants.

This leads to the following research question:

> *How can organizations that use similar network protocols and services securely correlate their network logs to detect broader attack campaigns?*

1.6. Key Contributions

This thesis considers n mutually distrustful organizations that want to collaboratively detect suspicious IP addresses without sharing their network logs. Each organization holds a private set of suspicious IP addresses extracted from local IDS logs, and the organizations agree on a threshold t. The objective is to reveal only those IP addresses that appear in at least t organizations' private sets.

The key contributions of this thesis are as follows:

- We propose a privacy-preserving collaborative threat-detection framework, called Circuit Threshold Multi-Party Private Set Intersection (Circuit-TMPSI), for correlating suspicious IP addresses across multiple organizations without exposing their complete local network logs.
- We design a Boolean-circuit-based construction for threshold multi-party private set intersection. To the best of our knowledge, Circuit-TMPSI is the first protocol to compute threshold intersections using Boolean circuits, making the construction conceptually simple and modular.
- We introduce a cloud-assisted architecture in which organizations outsource the threshold computation to a cloud server while preserving the privacy of their local inputs.
- Our protocol achieves computational complexity $O\left(m \cdot t \cdot \binom{n}{t}\right)$ and communication complexity $O(mn)$, where m is the Bloom filter length, n is the number of participating organizations, and t is the agreed threshold.
- We show that the protocol is most efficient when t is close to 1 or n, and least efficient when t is close to $n/2$, reflecting the combinatorial structure of the threshold computation.
- We show that the protocol is flexible across threshold choices: when $t = 1$, it computes the private set union, and when $t = n$, it computes the standard multi-party private set intersection.
- We made use a batching optimization in which 2048 bits are packed into a single ciphertext, reducing communication overhead by allowing each organization to transmit fewer ciphertexts.
- We provide a semi-honest security analysis showing that the protocol preserves the privacy of each organization's input against the cloud server and prevents any single party from decrypting the result independently.
- We implement Circuit-TMPSI in Python using the OpenFHE and PyProbables libraries. The implementation is open-source to support reproducibility and future deployment [38].

Overall, these contribution establishes the system model, define the threshold circuit construction, analyze its security and complexity, and provide an implementation that supports reproducible evaluation of the proposed approach.

1.7. Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 introduces the background concepts required to understand the proposed protocol. Chapter 3 reviews related work on threshold private set intersection, then compares the complexity of existing approaches. Chapter 4 presents the proposed Circuit-TMPSI protocol, including the system model, protocol design, circuit construction, complexity analysis, and security analysis. Chapter 5 evaluates the performance of the protocol by varying the main system parameters. Chapter 6 discusses practical deployment considerations, the limitations of the proposed approach, and possible directions for future work. Finally, Chapter 7 concludes the thesis.

2
Preliminaries

This section introduces the adversary models and cryptographic tools that form the foundation of the proposed protocol.

2.1. Security Models

We defines the adversarial models used to analyze the proposed protocol. A security model specifies how parties are expected to behave, what capabilities an adversary has, and what information the adversary is allowed to learn during protocol execution. Making these assumptions explicit is important because the security guarantee of a protocol depends on the adversarial setting in which the protocol is analyzed [21, 24, 34].

2.1.1. Semi-Honest Security

The semi-honest model, also called the honest-but-curious model, assumes that all parties follow the protocol specification correctly. However, parties are allowed to inspect their local views of the execution, including their inputs, randomness, received messages, intermediate values, and outputs [21, 24]. A protocol is secure in this model if no party obtains information about the other parties' private inputs beyond the intended output. This thesis analyzes the proposed protocol under the semi-honest security model.

2.1.2. Malicious Security

The malicious model, also called the active security model, considers a stronger adversary. In this model, corrupted parties are not required to follow the protocol specification and may deviate arbitrarily from it [21, 24]. A malicious party may send incorrect messages, choose malformed inputs, modify its behavior during execution, abort prematurely, or otherwise try to compromise the privacy or correctness of the protocol.

Because malicious adversaries have more freedom than semi-honest adversaries, malicious security requires additional safeguards. These safeguards include consistency checks, proofs of correct behavior, input validation, or mechanisms for detecting and handling aborts. Such protections increase the complexity and cost of the protocol. As a result, protocols that provide malicious security are generally more computationally expensive than protocols that provide security only against semi-honest adversaries [24, 34].

2.2. Bloom Filter

In our protocol, We require each party to hold a private set where each set contains distinct suspicious IP addresses from their own network logs. Each IP address is an element of the private set. Storing these sets directly can be memory-intensive, particularly when the number of IP addresses increases. Therefore, we represent each party's private set using a Bloom filter, a probabilistic data structure invented by Burton Howard Bloom [9]. Bloom filters provide a space-efficient representation of sets

with support for membership queries; however, this comes with the tradeoff that false positives may occur, while false negatives are impossible for inserted elements [9, 11].

A Bloom filter

$$BF = (BF[0], BF[1], \ldots, BF[m-1])$$

is a binary vector of length m that represents a set S with at most s elements. As shown in Algorithm 1, The m bits are initialized to 0 before adding elements of the set. The Bloom filter uses k hash functions

$$h_1, h_2, \ldots, h_k,$$

where each hash function maps an input element to an index in the Bloom filter:

$$h_i : \{0,1\}^* \to \{0, 1, \ldots, m-1\}.$$

Element insertion: To insert an element x into a Bloom filter, the element is hashed using all k hash functions:

$$h_1(x), h_2(x), \ldots, h_k(x).$$

As shown in Algorithm 2, Each hash function outputs an index in the Bloom filter, and the corresponding bit of the index is set to 1. If a bit is already set to 1, it remains unchanged. In this way, multiple elements can share common indices set to 1, making it prone to false positives.

Membership verification: To test whether an element x is present in the Bloom filter, the element is hashed using the k hash functions. If all corresponding Bloom filter indices are set to 1, then x is a member of the set with a known false positive probability. If at least one of the corresponding positions is set to 0, then x is definitely not present in the set. Membership verification is shown in Algorithm 3.

Since hash values of different elements may map to the same indices, Bloom filters can produce false positives. This occurs when the hash values of a non existing element map to bit positions that were already set to 1 by previous elements, causing hash collisions.

We use the false positive probability analysis presented by Bay et al. [8]. Let s denote the number of elements inserted into the Bloom filter, m the length of the Bloom filter, and k the number of hash functions. Let p denote the probability that a bit in the Bloom filter is set to 1, and let ε denote the false positive rate.

Initially, all m bits of the Bloom filter are set to 0. When one element is inserted, each hash function selects one bit position uniformly at random. The probability that a particular bit is not selected by one hash function is

$$1 - \frac{1}{m}.$$

Since k hash functions are applied for each element, and s elements are inserted, the total number of hash operations is ks. Therefore, the probability that a particular bit remains 0 after all insertions is

$$\left(1 - \frac{1}{m}\right)^{ks}.$$

Hence, the probability that a particular bit is set to 1 is

$$p = \Pr(BF[i] = 1) = 1 - \left(1 - \frac{1}{m}\right)^{ks}.$$

For large m, we use the approximation

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-ks/m}.$$

Thus,

$$p \approx 1 - e^{-ks/m}.$$

Following the false-positive-rate analysis of Bloom filters by Bose et al. [10], A false positive occurs when an element $x \notin S$ is queried and all k positions returned by the hash functions are already set to 1. Therefore, the false positive rate is

$$\varepsilon = \left(1 - \left(1 - \frac{1}{m}\right)^{ks}\right)^{k}.$$

Using the approximation above, we obtain

$$\varepsilon \approx \left(1 - e^{-ks/m}\right)^{k}.$$

The optimal number of hash functions is obtained by minimizing ε with respect to k. Equivalently, we minimize

$$\ln \varepsilon = k \ln\left(1 - e^{-ks/m}\right).$$

The false positive rate is minimized when the number of hash functions is

$$k = \frac{m}{s} \ln 2.$$

Substituting this value into the approximation

$$\varepsilon \approx \left(1 - e^{-ks/m}\right)^{k}$$

gives

$$e^{-ks/m} = e^{-\ln 2} = \frac{1}{2},$$

and hence

$$\varepsilon \approx \left(\frac{1}{2}\right)^{k} = \left(\frac{1}{2}\right)^{\frac{m}{s}\ln 2}.$$

Taking logarithms and solving for m gives

$$\ln \varepsilon = -\frac{m}{s}(\ln 2)^2, \qquad m = -\frac{s \ln \varepsilon}{(\ln 2)^2} = \frac{s \ln(1/\varepsilon)}{(\ln 2)^2}.$$

Equivalently,

$$m = s \log_2 e \cdot \log_2(1/\varepsilon).$$

Therefore, if m is chosen according to this bound, the optimal number of hash functions becomes

$$k = \log_2(1/\varepsilon).$$

Algorithm 1 Initialization

Require: BF of length m
1: **for** $i = 0$ to $m - 1$ **do**
2: $\quad BF[i] \leftarrow 0$
3: **end for**

Algorithm 2 Element Insertion

Require: Element x, Bloom filter BF
1: **for** $i = 0$ to $k - 1$ **do**
2: $\quad index \leftarrow H_i(x)$
3: $\quad$ **if** $BF[index] = 0$ **then**
4: $\quad\quad BF[index] \leftarrow 1$
5: $\quad$ **end if**
6: **end for**

Algorithm 3 Membership Verification

Require: Element x, Bloom filter BF
1: **for** $i = 0$ to $k - 1$ **do**
2: $index \leftarrow H_i(x)$
3: **if** $BF[index] = 0$ **then**
4: **output** "$x \notin S$"
5: **return**
6: **end if**
7: **end for**
8: **output** "$x \in S$"

Algorithm 4 Set Insertion

Require: Set S, Bloom filter BF of length m
1: **for** $i = 0$ to $m - 1$ **do**
2: $BF[i] \leftarrow 0$
3: **end for**
4: **for all** $x \in S$ **do**
5: **for** $i = 0$ to $k - 1$ **do**
6: $index \leftarrow H_i(x)$
7: **if** $BF[index] = 0$ **then**
8: $BF[index] \leftarrow 1$
9: **end if**
10: **end for**
11: **end for**
12: **output** BF

Figure 2.1: Bloom Filter Operations

2.3. Threshold BFV Homomorphic Encryption

This section explains the homomorphic encryption scheme used in our protocol. We use the threshold variant of the Brakerski–Fan–Vercauteren (BFV) scheme. BFV is a lattice-based homomorphic encryption scheme that supports homomorphic evaluation of arithmetic circuits over a plaintext ring [19]. In our implementation, we instantiate the BFV threshold scheme using OpenFHE, which provides practical implementations of BFV threshold variant [1].

In the threshold setting, the secret key is not held by a single party. Instead, the parties jointly generate a common public key, while each party keeps a corresponding secret key share. Plaintexts are encrypted using the public key whereas the decryption is then performed collaboratively, where each party computes a partial decryption shares that is combined to recover the plaintext. [37].

A threshold BFV scheme is a tuple of probabilistic polynomial-time algorithms

$$\mathsf{tBFV} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{EvalKeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{PartialDecrypt}, \mathsf{Combine})$$

defined as follows.

- $\mathsf{Setup}(1^\lambda, 1^d)$: The setup algorithm runs once before the protocol starts. It takes the security parameter λ and multiplicative depth bound d as input, and outputs public parameters pp. These parameters define the plaintext ring R_p, the ciphertext space, and the parameters for leveled homomorphic evaluation.
- $\mathsf{KeyGen}(\mathsf{pp}, n, t_{\mathsf{dec}})$: Using pp and the decryption threshold t_{dec}, the n parties generate a common public key pk and secret-key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$. Each party P_i keeps sk_i private.
- $\mathsf{EvalKeyGen}(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$: After key generation, the parties generate public evaluation keys ek. These keys allow the evaluator to apply arithmetic circuits to ciphertexts without learning the plaintexts.
- $\mathsf{Encrypt}(\mathsf{pk}, \mu)$: A party encrypts a plaintext value $\mu \in R_p$ using the common public key pk. The output is a ciphertext ct that hides μ while supporting homomorphic operations.
- $\mathsf{Eval}(\mathsf{ek}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$: Given the evaluation keys, an arithmetic circuit $C : R_p^k \to R_p$, and ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$, the evaluator computes the circuit on encrypted inputs. The output is an evaluated ciphertext $\widehat{\mathsf{ct}}$. If ct_i encrypts μ_i, then $\widehat{\mathsf{ct}}$ encrypts $C(\mu_1, \ldots, \mu_k)$.
- $\mathsf{PartialDecrypt}(\mathsf{sk}_i, \widehat{\mathsf{ct}})$: Each party P_i uses its secret-key share sk_i to compute a partial decryption share pd_i for the evaluated ciphertext $\widehat{\mathsf{ct}}$. A single share does not reveal the plaintext result.
- $\mathsf{Combine}(\mathsf{pd}_1, \ldots, \mathsf{pd}_n)$: The partial decryption shares are combined to recover the plaintext result $\widehat{\mu} \in R_p$, which is the output of the homomorphic computation.

3

Related Works

This section reviews prior work on Threshold Multi-party Private Set Intersection. The relevant literature studies protocols in which multiple parties identify elements whose occurrence reaches a specified threshold across private input sets. Existing work refers to this functionality using several terms, including quorum PSI, threshold MPSI, and over-threshold PSI. For consistency, this thesis uses the term Threshold MPSI to refer to this class of protocols.

3.1. Existing literature

One of the earliest general approaches to private threshold set operations was introduced by Kissner and Song. Their construction considers several set-operation variants, including set intersection, cardinality set intersection, threshold set intersection, and over-threshold set intersection [30]. In the over-threshold set-intersection setting, the parties learn only the elements whose total occurrence across the parties' private inputs reaches a specified threshold, together with the number of times each such element occurs [30].

The main idea behind Kissner and Song's construction is to represent set operations algebraically. Private sets are encoded as polynomials whose roots represent set elements, and over-threshold set-intersection is evaluated through homomorphic computation over encrypted polynomial representations [30]. The construction provides foundational results for over-threshold set intersection under both honest-but-curious and malicious adversary models. However, its algebraic formulation also introduces substantial cost: the protocols require direct interaction among the parties, encrypted polynomial computation, and additional zero-knowledge proofs in the malicious setting [30]. As a result, Kissner and Song's early construction is better viewed as a foundational technique for privacy-preserving threshold set operations than as a lightweight mechanism for large-scale cyber-threat-intelligence deployments.

The same polynomial-based direction was later generalized into a broader framework for privacy-preserving set and multiset operations [29]. The framework's main contribution is generality: it treats multisets through polynomial representations and uses polynomial operations to implement composable private operations, including multiset union, intersection, element reduction, cardinality set intersection, threshold set union, over-threshold set union, subset testing, and monotone multiset functions [29]. However, this added expressiveness does not remove the practical limitations of the polynomial-based approach. The protocols still rely on homomorphic computation over encrypted polynomial coefficients, direct party-to-party interaction, and, in several variants, shuffling or group decryption; malicious-security variants also require additional zero-knowledge proofs. Moreover, in the full framework's threshold and over-threshold set-union constructions, the threshold polynomial is formed by summing terms involving the polynomial and each of its successive formal derivatives up to the $(t-1)$-st derivative, rather than using only the single $(t-1)$-st derivative as in the earlier threshold set-intersection construction. This can increase the derivative-related polynomial work by approximately a factor of t [30, 29]. For large-scale cyber-threat-intelligence deployments, this line of work is therefore best viewed as foundational rather than as a lightweight deployment model.

A natural response to the cost of direct multi-party interaction is to outsource the main computation. Ma et al. follow this direction and propose over-threshold private set-operation protocols for lightweight clients, where most computation is shifted to two non-colluding cloud servers [35]. The approach uses hashing, polynomial encodings, and secret sharing so that clients send shares to the cloud servers rather than communicating directly with every other participant. This creates a client-server deployment pattern and reduces the computational and communication burden on clients [35].

The main scalability cost of Ma et al.'s approach shifts to the server side. The cloud servers construct cuckoo-hash structures over the element domain and evaluate secret-shared polynomial representations over candidate domain elements [35]. In a cyber-threat-intelligence setting where suspicious IP addresses are used as indicators, the IPv4 space contains 2^{32} possible values. Evaluating candidate elements over such a large domain can therefore create substantial server-side computation, especially when participating organizations hold large indicator sets. This cost is significant because operational threat-intelligence workflows commonly involve large and evolving collections of indicators of compromise and other cyber threat information [3, 54]. Thus, although the outsourced architecture is well suited to lightweight-client settings with two independent cloud servers, its domain-based evaluation step limits suitability for large-scale CTI deployments over very large indicator spaces.

Instead of evaluating over a large candidate universe, another practical direction relies on hash-bin reconstruction. Mahdavi et al. propose protocols for Over-Threshold Multi-Party Private Set Intersection (OT-MP-PSI), motivated by collaborative threat-intelligence scenarios in which indicators of compromise remain useful even when shared by only a subset of organizations [36]. Their construction follows a client-server style deployment: participants interact with a key holder to generate element-dependent shares, place those shares into hash bins, and send the padded hash-bin tables to a reconstructor, rather than interacting directly with every other participant [36].

Mahdavi et al.'s design introduces Oblivious Pseudo-Random Secret Sharing (OPR-SS), which combines the reconstruction properties of secret sharing with the obliviousness properties of oblivious pseudo-random functions [36, 49]. At a high level, participants place element-dependent shares into hash bins, and the reconstructor tests share combinations from corresponding bins through Lagrange interpolation to identify elements that satisfy the threshold condition [36]. This design improves practicality compared with encrypted polynomial computation, but the reconstruction phase remains the main bottleneck. The reconstructor must test combinations of shares from distinct participants within each bin, so the cost grows rapidly with the threshold , the number of parties, and the padded bin size [36]. Increasing the number of bins can reduce the number of combinations tested per bin, but it also increases the amount of hash table data sent to the reconstructor. This computation–communication tradeoff limits scalability in CTI deployments where organizations maintain large sets of indicators [36].

Bloom-filter-based protocols provide another path toward practical threshold multi-party private set intersection (T-MPSI). Bay et al. propose a T-MPSI protocol based on Bloom filters and additively homomorphic threshold public-key encryption [8, 9]. Their construction is designed for a star-topology setting in which one of the parties acts as the server and coordinates the threshold computation, while the remaining parties act as clients. [8].

The central idea is to encode client private sets as encrypted Bloom filters and perform threshold computation over encrypted Bloom-filter positions. In Bay et al.'s T-MPSI functionality, the output consists of elements from the server's set that appear in at least a threshold number of client sets [8]. This means that the protocol does not directly identify all elements that satisfy the threshold condition across the global union of all parties' inputs; threshold elements absent from the server's set are outside the reported output. The protocol uses threshold public-key encryption, decryption-to-zero, and a secure comparison subprotocol to perform these checks [8]. Although the star-topology communication pattern reduces direct party-to-party interaction and is practical for settings with many parties and relatively small sets, the design concentrates much of the threshold-testing workload at one coordinating party and relies on expensive homomorphic-encryption and secure-comparison operations. As a result, the approach is better suited to small-set T-MPSI settings than to large-scale CTI deployments involving high-volume indicator sets, such as large collections of IP-address [8].

While the preceding works target threshold-style outputs across multiple parties, quorum PSI studies a related but different functionality. Chandran et al. propose efficient protocols for multiparty PSI, circuit

PSI, and quorum PSI [12]. In the quorum PSI setting, a designated leader P_1 learns which elements of its own input set appear in at least k of the other parties' input sets. The functionality is therefore threshold-like but leader-centered rather than symmetric across all parties [12].

The quorum PSI construction uses hashing, weak private set-membership functionality instantiated with OPPRF-style techniques, equality testing, secret-share conversion, and lightweight multiparty computation [12, 32, 49]. At a high level, the protocol first compares the leader's candidate elements against the other parties' hashed representations and then uses an n-party computation to count how many parties contain each candidate and compare this count with the quorum threshold [12]. The design is efficient for its intended leader-centered setting, but it does not directly match the CTI setting considered in this thesis. Chandran et al.'s functionality helps a designated party identify elements from its own set that occur in enough other sets, whereas the goal here is to identify threshold indicators across all participating private sets and share the resulting indicators as collaborative threat intelligence [12].

The gap between generic threshold PSI and operational CTI is addressed more directly in recent work on collaborative network intrusion detection. Arpaci et al. propose an OT-MP-PSI scheme in which multiple institutions identify IP addresses that reach a required occurrence threshold in participants' network logs [4]. The work is closely related to this thesis through its focus on collaborative intrusion detection and privacy-preserving analysis of raw IP logs.

Arpaci et al.'s construction builds on OPR-SS and Shamir secret sharing, but uses a different hashing structure to reduce reconstruction work [4, 49]. Each participant stores element-dependent shares in 20 hash tables, where each table has $M \times t$ positions and M denotes the maximum input-set size [4]. Compared with the earlier binning-based OT-MP-PSI protocol of Mahdavi et al., the main improvement is that the aggregator tests combinations of participants rather than combinations of individual shares inside larger bins [36, 4]. This reduces the reconstruction search space from $O(M(N \log M/t)^{2t})$ to $O(t^2 M\binom{N}{t})$, making the scheme more practical for collaborative intrusion-detection data [4].

The practical focus of Arpaci et al.'s work is further reflected in two deployment variants: an interactive collusion-safe version with key holders and a non-interactive version that reduces communication by assuming a non-colluding aggregator [4]. The evaluation uses CANARIE IDS network logs, which makes the work especially relevant to operational CTI settings [4]. Nevertheless, the use of 20 hash tables per participant introduces storage and communication overhead. Since each table has $M \times t$ positions, increasing the threshold t or the set size$M$, increases table size, memory usage, and transmitted data. In addition, the reconstruction complexity still contains the combinatorial factor $\binom{N}{t}$, so reconstruction cost can grow quickly as the number of parties or the threshold changes [4]. These trade-offs motivate the exploration of alternative constructions for threshold-based correlation in collaborative intrusion detection.

Beyond efficiency and deployment model, another branch of work changes the information revealed by the threshold output. Yang et al. propose traceable over-threshold multi-party private set intersection, where a designated leader learns threshold elements from its own set together with the identities of the parties holding each reported element [56]. The functionality therefore differs from anonymous threshold PSI because holder information is intentionally included for each reported element. Yang et al. motivate this design through applications where provenance and accountability are needed to interpret the reported intersection, such as digital forensics, network anomaly detection, and suspicious-account analysis [56].

Yang et al. present two traceable OT-MP-PSI protocols based on Shamir secret sharing, OPPRF, hashing, secret-share updating, and, in the security-enhanced variant, oblivious linear evaluation [56, 49, 32]. The first construction is secure against collusion among up to $t-2$ semi-honest participants, while the security-enhanced variant strengthens security against up to $n-1$ semi-honest corruptions at the cost of additional computation and communication [56]. Traceability provides an important functional advantage when source attribution is required, but it also reveals more information than anonymous threshold MPSI. For CTI settings where organizations want to learn shared indicators without disclosing which participants contributed each indicator, this functionality may be less appropriate. In addition, the security-enhanced variant introduces extra OLE-based share-update operations, increasing communication and computation costs compared with the more efficient traceable construction [56].

Overall, prior work shows that threshold-style private set intersection can be realized through polynomial-

based, outsourced, hashing-based, Bloom-filter-based, and traceable approaches. However, these designs trade off privacy, trust assumptions, communication pattern, output functionality, and deployment cost. This motivates Threshold MPSI protocols that better balance privacy, efficiency, and deployability for collaborative CTI and intrusion-detection settings.

3.2. Complexity comparison

Tables 3.1 and 3.2 summarize the structural properties, communication complexity, and computation complexity of threshold MPSI and related protocols in the semi-honest setting. The notation follows the conventions of the cited works where possible, so some parameters have protocol-specific meanings. In particular, n generally denotes the number of parties, m the maximum input-set size, and t the threshold, while parameters such as k, u, b, λ, ρ, and σ are defined in the table notes.

Table 3.1: Structural comparison of threshold MPSI and related protocols.

Protocol	Topology	Rounds	Collusion resilience
Kissner and Song [30]	Mesh	$O(n)$	Up to $n-1$ collusions
Kissner and Song [29]	Mesh	$O(n)$	Up to $n-1$ collusions
Mahdavi et al. [36]	Star	$O(1)$	Up to k collusions
Ma et al. [35]	Star	$O(1)$	Two non-colluding servers
Bay et al. [8]	Star	$O(n)$	Up to $n-1$ colluding parties
Chandran et al. [12]	Mesh	$O(\log\sigma + \log n)$	Honest majority, $t < n/2$
Arpaci et al. non-interactive [4]	Star	$O(1)$	Non-colluding server
Arpaci et al. collusion-safe [4]	Star	$O(1)$	Up to k key-holder collusions
Yang et al. ET-OT-MP-PSI [56]	Mesh	Multi-round	$t-2$ semi-honest corruptions
Yang et al. ST-OT-MP-PSI [56]	Mesh	Multi-round	$n-1$ semi-honest corruptions

Note. In the table, n is the number of participating parties, t is the threshold, k is the number of tolerated collusions or key-holder collusions depending on the protocol, and σ is the input-size/security parameter used in the round complexity of Chandran et al. [12].

Table 3.2: Communication and computation complexity comparison of threshold MPSI and related protocols.

Protocol	Communication complexity	Computation complexity
Kissner and Song [30]	$O(n^3m)$	$O(n^3m^3)$
Kissner and Song [29]	$O(tn^3m)$	$O(tn^3m^3)$
Mahdavi et al. [36]	$O(tmnk)$	$O\Big(m\,(n\log(m/t))^{2t}\Big)$
Ma et al. [35]	$O(nu)$	$O(nu)$
Bay et al. [8]	$O(bnt)$	$O(bn)$
Chandran et al. [12]	$O(mn(\lambda\sigma + t'\log n))$	$O(mn(\lambda\sigma + \rho + \log n))$
Arpaci et al. non-interactive [4]	$O(tmn)$	$O\big(t^2m\binom{n}{t}\big)$
Arpaci et al. collusion-safe [4]	$O(tmnk)$	$O\big(t^2m\binom{n}{t}\big)$
Yang et al. ET-OT-MP-PSI [56]	$O(nm\rho)$	$O\left(m\max\left\{t^2\left(\frac{e(n-1)}{t-1}\right)^{t-1}, n\lambda\right\}\right)$
Yang et al. ST-OT-MP-PSI [56]	$O(n^2m\rho)$	$O\left(m\max\left\{t^2\left(\frac{e(n-1)}{t-1}\right)^{t-1}, n^2\rho, n\lambda\right\}\right)$

Note. In the table, n is the number of parties, m is the maximum set size, and t is the threshold. The parameter k is the number of key holders, u is the input-domain size, b is the Bloom-filter size, λ is the computational security parameter, and ρ is the statistical security parameter. The value σ is the bit length of encoded input elements, e is Euler's constant, and $t' = \min(t, \log n)$ is the auxiliary threshold parameter used by Chandran et al. [12].

The comparison shows that existing threshold MPSI protocols introduce different costs depending on their topology, set representation, and reconstruction method [30, 29, 36, 35, 8, 12, 4, 56]. Polynomial-based approaches incur high encrypted-polynomial computation, outsourced approaches shift cost or trust assumptions to servers, hash-bin approaches reduce some reconstruction work but can retain combinatorial factors, Bloom-filter-based approaches rely on expensive homomorphic and comparison operations, and leader- or traceability-oriented variants change the output functionality. These costs are especially important in cyber-threat-intelligence settings, where participating organizations may hold large indicator sets and repeated communication creates deployment challenges [3, 54, 4]. This motivates Threshold MPSI constructions that better balance privacy, efficiency, and deployability for high-volume CTI data.

4

Protocol Description

In this chapter, we first present a high-level overview of the protocol architecture and briefly discuss our Circuit-TMPSI protocol. We consider a setting with n parties $P_1, P_2, \ldots, P_n$, where each party P_i has a private set S_i containing suspicious IP addresses. The main objective is to securely compute the set of IP addresses that occur in at least t out of the n party's sets, where the threshold t is mutually agreed. The computation reveals only the final threshold intersection, and no additional information about the private sets of the parties is revealed.

Table 4.1: Notation used in the protocol.

Notation	Description
n	Total number of participating parties
t	Threshold value for the threshold intersection
t_{dec}	Decryption threshold for collaborative decryption
s	Length of each private set S_i
m	Length of the Bloom filter
pp	Public parameters for the FHE protocol
pk	Common public key jointly generated by the parties
sk_i	Secret-key share held by party P_i
ek	Evaluation keys used to perform homomorphic evaluation
P_i	The i-th party participating in the protocol
S_i	Private set held by party P_i
$S_i[j]$	The j-th element of the private set S_i
BF_i	Bloom filter corresponding to the private set S_i of party P_i
$BF_i[j]$	The j-th element of the Bloom filter BF_i
EBF_i	Encrypted Bloom filter of party P_i
$EBF_i[j]$	The j-th encrypted element of the encrypted Bloom filter EBF_i
BF^*	Bloom filter representing the threshold intersection after decryption
EBF^*	Encrypted Bloom filter representing the threshold intersection
C	Arithmetic circuit used for homomorphic threshold computation
$\wedge$	Logical AND operation
$\vee$	Logical OR operation
$\neg$	Logical NOT operation

4.1. Protocol Setup

At the beginning of the protocol, the parties agree on the public parameters pp, the number of participating parties n, the threshold value t, and the Bloom-filter false-positive rate ε. The decryption threshold is set to $t_{\mathsf{dec}} = n$, so all parties are required for collaborative decryption. Parties then jointly run the threshold BFV key-generation procedure, which produces a common public key pk for encryption and secret-key shares $sk_1, \ldots, sk_n$, held separately by the parties. After key generation, the parties run $\mathsf{EvalKeyGen}$ to generate the public evaluation keys ek. These keys allow the cloud server to evaluate arithmetic circuits over ciphertexts homomorphically. Party P_1 then sends the values n and t to the cloud server, which uses them to generate the arithmetic threshold circuit C.

As illustrated in Figure 4.1, each party P_i holds a private set S_i of IP addresses. The party encodes S_i as a Bloom filter BF_i using Algorithm 4, and then encrypts each Bloom-filter entry under the public key pk. This produces the encrypted Bloom filter EBF_i, which is sent to the cloud server. Since both Bloom-filter construction and encryption are performed locally, the cloud server does not learn the underlying IP addresses or the plaintext Bloom-filter entries.

After receiving the encrypted Bloom filters $EBF_1, \ldots, EBF_n$, the cloud server processes them position by position. For each Bloom-filter position $j \in \{0, \ldots, m-1\}$, it takes the encrypted entries $EBF_1[j], EBF_2[j], \ldots, EBF_n[j]$ and homomorphically evaluates the arithmetic threshold circuit C. The circuit determines, in encrypted form, whether at least t parties have a 1 at position j. This per-position threshold test is needed because an IP address belongs to the threshold intersection only when all of its corresponding Bloom-filter positions are set to 1 in at least t parties' Bloom filters. Thus, for each position j, the circuit outputs an encrypted $Enc(1)$ if at least t encrypted inputs correspond to 1, and an encrypted $Enc(0)$ otherwise. Applying this process to all Bloom-filter positions produces the encrypted threshold-intersection Bloom filter EBF^*, as shown in Algorithm 5.

The cloud server then returns EBF^* to the parties. To decrypt the result, each party computes a partial decryption share using its secret-key share and sends this partial decryption to P_1. Party P_1 combines the received partial decryption shares to recover the plaintext threshold-intersection Bloom filter BF^*, and then shares BF^* with all parties. Because no party holds the full secret key, this step preserves the threshold-decryption structure of the protocol. The decryption and recovery process is described in Algorithm 6.

Finally, the parties recover the threshold-intersection set I^* by membership verification, as shown in Algorithm 3. Each candidate IP address from the universe $\mathcal{U}$ is queried against the decrypted Bloom filter BF^*. If the address passes the Bloom-filter membership test, it is included in I^*. Because Bloom filters are probabilistic, I^* may contain false positives. However, Bloom filters do not produce false negatives for inserted elements. Therefore, I^* represents a candidate set of suspicious IP addresses with false positives that appear in at least t private sets.

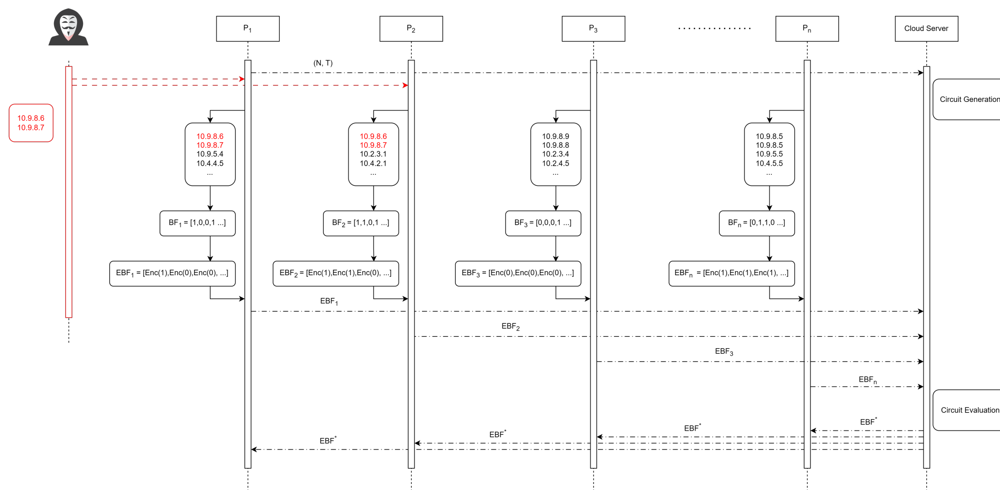


Figure 4.1: System architecture of the proposed Circuit-TMPSI protocol

Algorithm 5 Circuit-TMPSI Protocol

Require: Parties $P_1, \ldots, P_n$ with private sets $S_1, \ldots, S_n$ containing suspicious IP addresses, threshold t, Bloom filter length m, cloud server CS
Ensure: Encrypted Bloom filter EBF^* representing the threshold intersection
1: Party P_1 sends the parameters (n, t) to CS
2: CS constructs the arithmetic threshold circuit C for the given (n, t)
3: The parties jointly generate a public encryption key and secret-key shares $\mathsf{KeyGen}(\mathsf{pp}, n, t_{\mathsf{dec}=n}) \rightarrow (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$
4: The parties jointly generate public evaluation keys $\mathsf{EvalKeyGen}(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \rightarrow \mathsf{ek}$
5: Party P_1 send ek to CS
6: **for** $i = 1$ to n **do**
7: $BF_i \leftarrow$ SetInsertion(S_i) (Algorithm 4)
8: **end for**
9: **for** $i = 1$ to n **do**
10: **for** $j = 0$ to $m - 1$ **do**
11: $EBF_i[j] \leftarrow$ Encrypt$(pk, BF_i[j])$
12: **end for**
13: Party P_i sends EBF_i to CS
14: **end for**
15: CS initializes $EBF^* \leftarrow [\,]$
16: **for** $j = 0$ to $m - 1$ **do**
17: CS initializes $X_j \leftarrow [\,]$
18: **for** $i = 1$ to n **do**
19: CS sets $X_j \leftarrow X_j \cup \{EBF_i[j]\}$
20: **end for**
21: CS computes $EBF^*[j] \leftarrow$ Eval$(ek, C, X_j[1], X_j[2], \ldots, X_j[n])$
22: **end for**
23: CS returns EBF^* to the parties
24: **return** EBF^*

Algorithm 6 Collaborative Decryption and Intersection Recovery

Require: Encrypted threshold-intersection Bloom filter EBF^*, secret-key shares $sk_1, \ldots, sk_n$, IP address space $\mathcal{U}$
Ensure: Candidate threshold-intersection set I^*
1: $BF^* \leftarrow [\,]$
2: $I^* \leftarrow \emptyset$
3: **for** $j = 0$ to $m - 1$ **do**
4: **for** $i = 1$ to n **do**
5: Party P_i computes and sends to P_1:

$$d_i[j] \leftarrow \text{PartialDecrypt}(sk_i, EBF^*[j])$$

6: **end for**
7: Party P_1 combines the partial decryption shares:

$$BF^*[j] \leftarrow \text{Combine}\left(d_1[j], d_2[j], \ldots, d_n[j]\right)$$

8: **end for**
9: Party P_1 shares the recovered Bloom filter BF^* with all parties
10: **for all** $x \in \mathcal{U}$ **do**
11: **if** MembershipVerification(x, BF^*) returns "$x \in S$" **then**
12: $I^* \leftarrow I^* \cup \{x\}$
13: **end if**
14: **end for** (Algorithm 3)
15: **return** I^*

4.2. Circuit Evaluation

The cloud server is responsible for generating the threshold circuit c. The parties provide the public parameters n and t, where n is the number of participating parties and t is the threshold value. For each Bloom-filter position j, the cloud server receives the encrypted inputs

$$EBF_1[j], EBF_2[j], \dots, EBF_n[j]. \tag{4.1}$$

Conceptually, these ciphertexts encrypt the plaintext Bloom-filter values

$$BF_1[j], BF_2[j], \dots, BF_n[j] \in \{0,1\}. \tag{4.2}$$

The objective of the threshold circuit is to determine whether Bloom-filter position j is set to 1 by at least t parties. This condition is needed because an element belongs to the threshold intersection only if all of its corresponding Bloom-filter positions are set to 1 in at least t parties' Bloom filters. In plaintext form, the threshold circuit C_t is defined as

$$C_t(BF_1[j], \dots, BF_n[j]) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} BF_i[j] \geq t, \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

Thus, the circuit outputs 1 exactly when at least t of the n input bits are equal to 1, and outputs 0 otherwise.

Table 4.2: Truth table for the threshold circuit at Bloom filter position j

$BF_1[j]$	$BF_2[j]$	$BF_3[j]$	$\cdots$	$BF_n[j]$	Count	Output
0	0	0	$\cdots$	0	0	0
1	0	0	$\cdots$	0	1	0
0	1	0	$\cdots$	0	1	0
0	0	1	$\cdots$	0	1	0
1	1	0	$\cdots$	0	2	0
1	0	1	$\cdots$	0	2	0
0	1	1	$\cdots$	0	2	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
1	1	1	$\cdots$	0	$t-1$	0
1	1	1	$\cdots$	0	t	1
1	1	0	$\cdots$	1	t	1
1	0	1	$\cdots$	1	t	1
0	1	1	$\cdots$	1	t	1
1	1	1	$\cdots$	1	$t+1$	1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
1	1	1	$\cdots$	1	$n-1$	1
1	1	1	$\cdots$	1	n	1

A Boolean function can be represented in disjunctive normal form (DNF) by constructing one conjunction term for every truth-table row where the output is 1 [14, 55]. Each term describes one exact input row represented by a bit vector

$$b = (b_1, \dots, b_n) \in \{0,1\}^n. \tag{4.4}$$

The term includes $BF_i[j]$ when $b_i = 1$, and includes $\neg BF_i[j]$ when $b_i = 0$. Therefore, this term evaluates to 1 only for the input row it represents and evaluates to 0 for all other input rows.

Following the truth-table construction, the full DNF form of the threshold circuit is

$$C_t = \bigvee_{\substack{b \in \{0,1\}^n \\ \sum_{i=1}^{n} b_i \geq t}} \left(\bigwedge_{i:b_i=1} BF_i[j] \wedge \bigwedge_{i:b_i=0} \neg BF_i[j] \right). \tag{4.5}$$

The full DNF expression in Eq. (4.5) is correct [14, 55], but it is unnecessarily large. It contains terms that describe both the exact 0-values and the exact 1-values of every satisfying truth-table row. However, the threshold circuit does not need to preserve the exact 0-values. It only needs to check whether at least t Bloom-filter values are equal to 1. To simplify the DNF expression, we use the idempotent law

$$x \vee x = x. \tag{4.6}$$

This law allows a term that already appears in the DNF to be repeated without changing the Boolean function. In the threshold truth table, the last satisfying row is the row in which all Bloom-filter values are equal to 1. The corresponding term is

$$O = \bigwedge_{i=1}^{n} BF_i[j]. \tag{4.7}$$

Since O is already included in the full DNF expression (4.5), Eq. (4.6) allows us to reuse O during simplification:

$$C_t = C_t \vee O \vee O \vee \cdots . \tag{4.8}$$

The repeated copies of O can be used to combine DNF terms that contain negated Bloom-filter expressions. If a DNF term contains $\neg BF_r[j]$, then a copy of O, in which $BF_r[j]$ is 1, can be paired with it. The two terms have the form

$$(X \wedge \neg BF_r[j]) \vee (X \wedge BF_r[j]), \tag{4.9}$$

where X denotes their common part. These terms simplify as

$$(X \wedge \neg BF_r[j]) \vee (X \wedge BF_r[j]) = X \wedge (\neg BF_r[j] \vee BF_r[j]) = X. \tag{4.10}$$

Repeating this process removes the negated Bloom-filter expressions from the satisfying DNF terms.

After applying this simplification, the threshold circuit can be written using only conjunctions of Bloom-filter values:

$$C_t(BF_1[j], \ldots, BF_n[j]) = \bigvee_{\substack{S \subseteq \{1,\ldots,n\} \\ |S| \geq t}} \left(\bigwedge_{i \in S} BF_i[j] \right). \tag{4.11}$$

This expression outputs 1 if there exists a subset S of parties, with $|S| \geq t$, whose Bloom-filter values are all 1. It can be reduced further using the absorption law

$$x \vee (x \wedge y) = x. \tag{4.12}$$

Let

$$A \subseteq \{1, \ldots, n\}, \qquad |A| = t, \tag{4.13}$$

and let S be any larger subset such that

$$A \subseteq S, \qquad |S| > t. \tag{4.14}$$

Then the conjunction for S can be factored as

$$\bigwedge_{i \in S} BF_i[j] = \left(\bigwedge_{i \in A} BF_i[j] \right) \wedge \left(\bigwedge_{i \in S \setminus A} BF_i[j] \right). \tag{4.15}$$

Therefore, by Eq. (4.12),

$$\left(\bigwedge_{i \in A} BF_i[j] \right) \vee \left(\bigwedge_{i \in S} BF_i[j] \right) = \bigwedge_{i \in A} BF_i[j]. \tag{4.16}$$

Thus, every term with more than t Bloom-filter expressions is absorbed by one of its size-t subterms.

Therefore, only conjunctions containing exactly t Bloom-filter expressions must be kept. The optimized threshold circuit is

$$C_t(BF_1[j], \ldots, BF_n[j]) = \bigvee_{\substack{A \subseteq \{1,\ldots,n\} \\ |A|=t}} \left(\bigwedge_{i \in A} BF_i[j] \right). \tag{4.17}$$

After receiving the public parameters n and t, the cloud server generates the optimized circuit in Eq. (4.17). In this form, the circuit contains one AND term for every subset $A \subseteq \{1, \ldots, n\}$ of size t. Each AND term checks whether all parties in that subset have Bloom-filter value 1 at position j, and the final OR combines all such possibilities. Therefore, in plaintext, the circuit outputs 1 exactly when at least one subset of t parties has value 1, which is equivalent to at least t of the n input bits being equal to 1. The number of AND terms is

$$\binom{n}{t}, \tag{4.18}$$

one for each possible choice of t parties.

The optimized expression above gives the Boolean form of the threshold circuit. Since the circuit is evaluated homomorphically using BFV, it must be expressed as an arithmetic circuit over the plaintext ring. For Boolean values $a, b \in \{0, 1\}$, Boolean AND can be represented by multiplication:

$$a \wedge b = a \cdot b. \tag{4.19}$$

Boolean OR can be represented using the identity

$$a \vee b = 1 - (1 - a)(1 - b). \tag{4.20}$$

Therefore, for each subset

$$A \subseteq \{1, \ldots, n\}, \qquad |A| = t, \tag{4.21}$$

define

$$M_A = \prod_{i \in A} BF_i[j]. \tag{4.22}$$

Here, M_A is equal to 1 only when all parties in subset A have value 1 at position j. The arithmetic form of the threshold circuit is then

$$C_t(BF_1[j], \ldots, BF_n[j]) = 1 - \prod_{\substack{A \subseteq \{1,\ldots,n\} \\ |A|=t}} (1 - M_A). \tag{4.23}$$

The cloud server evaluates this arithmetic circuit homomorphically on the encrypted Bloom-filter bits:

$$EBF^*[j] = \text{Eval}\left(C_t, EBF_1[j], EBF_2[j], \ldots, EBF_n[j]\right). \tag{4.24}$$

Because this evaluation is performed over ciphertexts, the server does not obtain the plaintext circuit output. Instead, the result $EBF^*[j]$ is $\mathrm{Enc}(1)$ if at least t parties have a 1 at Bloom-filter position j, and $\mathrm{Enc}(0)$ otherwise. Applying this evaluation to all positions $j \in \{0, \ldots, m-1\}$ produces the encrypted threshold-intersection Bloom filter EBF^*.

Special case: $t = 1$. When $t = 1$, an IP address only needs to appear in at least one party's private set to be included in the output. Therefore, setting $t = 1$ computes the private set union of the parties' input sets [29, 30].

Using the threshold-circuit notation, when $t = 1$ we have

$$C_1(BF_1[j], \ldots, BF_n[j]) = \bigvee_{\substack{A \subseteq \{1,\ldots,n\} \\ |A|=1}} \left(\bigwedge_{i \in A} BF_i[j] \right). \tag{4.25}$$

Since every subset A of size 1 contains exactly one party, the inner AND term contains only one Bloom-filter value. Hence, the circuit simplifies to

$$C_1(BF_1[j], \ldots, BF_n[j]) = BF_1[j] \vee BF_2[j] \vee \cdots \vee BF_n[j]. \tag{4.26}$$

Thus, the output of the threshold circuit represents the union of the parties' private sets.

The following examples illustrate the simplification process described above.

Example: $n = 3$ **and** $t = 2$**.** Let

$$a = BF_1[j], \qquad b = BF_2[j], \qquad c = BF_3[j]. \tag{4.27}$$

For $n = 3$ and $t = 2$, the threshold circuit outputs 1 whenever at least two of the three inputs are equal to 1.

a	b	c	Count	$C_2(a,b,c)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	0
1	0	1	2	1
1	1	0	2	1
1	1	1	3	1

(4.28)

The threshold circuit expression obtained from the rows where the output is 1 is

$$C_2(a,b,c) = (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge b \wedge c). \tag{4.29}$$

This expression is correct, but it contains negated terms. Using the idempotent law in Eq. (4.6), the term $a \wedge b \wedge c$ can be reused during simplification:

$$(\neg a \wedge b \wedge c) \vee (a \wedge b \wedge c) = b \wedge c, \tag{4.30}$$

$$(a \wedge \neg b \wedge c) \vee (a \wedge b \wedge c) = a \wedge c, \tag{4.31}$$

$$(a \wedge b \wedge \neg c) \vee (a \wedge b \wedge c) = a \wedge b. \tag{4.32}$$

Therefore,

$$C_2(a,b,c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c) \vee (a \wedge b \wedge c). \tag{4.33}$$

The last term is redundant by the absorption law in Eq. (4.12):

$$(a \wedge b) \vee (a \wedge b \wedge c) = a \wedge b. \tag{4.34}$$

Hence, the optimized threshold circuit is

$$C_2(a,b,c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c). \tag{4.35}$$

Substituting back the Bloom filter notation gives

$$\begin{aligned} C_2(BF_1[j], BF_2[j], BF_3[j]) = (BF_1[j] \wedge BF_2[j]) \vee (BF_1[j] \wedge BF_3[j]) \\ \vee (BF_2[j] \wedge BF_3[j]). \end{aligned} \tag{4.36}$$

Example: $n = 4$ **and** $t = 3$**.** Let

$$a = BF_1[j], \qquad b = BF_2[j], \qquad c = BF_3[j], \qquad d = BF_4[j]. \tag{4.37}$$

For $n = 4$ and $t = 3$, the threshold circuit outputs 1 whenever at least three of the four inputs are equal to 1.

a	b	c	d	Count	$C_3(a,b,c,d)$
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	2	0
0	1	0	0	1	0
0	1	0	1	2	0
0	1	1	0	2	0
0	1	1	1	3	1
1	0	0	0	1	0
1	0	0	1	2	0
1	0	1	0	2	0
1	0	1	1	3	1
1	1	0	0	2	0
1	1	0	1	3	1
1	1	1	0	3	1
1	1	1	1	4	1

(4.38)

The threshold circuit expression obtained from the rows where the output is 1 is

$$\begin{aligned} C_3(a,b,c,d) = {} & (\neg a \wedge b \wedge c \wedge d) \vee (a \wedge \neg b \wedge c \wedge d) \\ & \vee (a \wedge b \wedge \neg c \wedge d) \vee (a \wedge b \wedge c \wedge \neg d) \vee (a \wedge b \wedge c \wedge d). \end{aligned} \tag{4.39}$$

Using the idempotent law in Eq. (4.6), the term $a \wedge b \wedge c \wedge d$ can be reused during simplification. The threshold circuit terms simplify as follows:

$$(\neg a \wedge b \wedge c \wedge d) \vee (a \wedge b \wedge c \wedge d) = b \wedge c \wedge d, \tag{4.40}$$

$$(a \wedge \neg b \wedge c \wedge d) \vee (a \wedge b \wedge c \wedge d) = a \wedge c \wedge d, \tag{4.41}$$

$$(a \wedge b \wedge \neg c \wedge d) \vee (a \wedge b \wedge c \wedge d) = a \wedge b \wedge d, \tag{4.42}$$

$$(a \wedge b \wedge c \wedge \neg d) \vee (a \wedge b \wedge c \wedge d) = a \wedge b \wedge c. \tag{4.43}$$

Therefore, the threshold circuit expression reduces to

$$\begin{aligned} C_3(a,b,c,d) = {} & (a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \\ & \vee (b \wedge c \wedge d) \vee (a \wedge b \wedge c \wedge d). \end{aligned} \tag{4.44}$$

The last term is redundant by the absorption law in Eq. (4.12). For example,

$$(a \wedge b \wedge c) \vee (a \wedge b \wedge c \wedge d) = a \wedge b \wedge c. \tag{4.45}$$

Hence, the optimized threshold circuit is

$$\begin{aligned} C_3(a,b,c,d) = {} & (a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \\ & \vee (b \wedge c \wedge d). \end{aligned} \tag{4.46}$$

Substituting back the Bloom filter notation gives

$$\begin{aligned} C_3(BF_1[j], BF_2[j], BF_3[j], BF_4[j]) = {} & (BF_1[j] \wedge BF_2[j] \wedge BF_3[j]) \\ & \vee (BF_1[j] \wedge BF_2[j] \wedge BF_4[j]) \\ & \vee (BF_1[j] \wedge BF_3[j] \wedge BF_4[j]) \\ & \vee (BF_2[j] \wedge BF_3[j] \wedge BF_4[j]). \end{aligned} \tag{4.47}$$

4.3. Correctness of the Threshold Circuit

We now show that the optimized threshold circuit correctly computes the threshold condition.

Claim. For inputs $BF_1[j], \ldots, BF_n[j] \in \{0, 1\}$, the circuit

$$C_t(BF_1[j], \ldots, BF_n[j]) = \bigvee_{\substack{A \subseteq \{1,\ldots,n\} \\ |A|=t}} \left(\bigwedge_{i \in A} BF_i[j] \right) \tag{4.48}$$

outputs 1 if and only if

$$\sum_{i=1}^{n} BF_i[j] \geq t. \tag{4.49}$$

Proof. First, suppose that

$$\sum_{i=1}^{n} BF_i[j] \geq t. \tag{4.50}$$

Then at least t of the Bloom filter values at position j are equal to 1. Therefore, there exists a subset $A \subseteq \{1, \ldots, n\}$ such that $|A| = t$ and

$$BF_i[j] = 1 \quad \text{for all } i \in A. \tag{4.51}$$

For this subset A, we have

$$\bigwedge_{i \in A} BF_i[j] = 1. \tag{4.52}$$

Since the circuit in Eq. (4.48) takes the OR over all subsets A of size t, at least one term in the OR is equal to 1. Hence,

$$C_t(BF_1[j], \ldots, BF_n[j]) = 1. \tag{4.53}$$

Conversely, suppose that

$$C_t(BF_1[j], \ldots, BF_n[j]) = 1. \tag{4.54}$$

Then at least one term in the OR expression in Eq. (4.48) must be equal to 1. Hence, there exists a subset $A \subseteq \{1, \ldots, n\}$ with $|A| = t$ such that

$$\bigwedge_{i \in A} BF_i[j] = 1. \tag{4.55}$$

This implies that

$$BF_i[j] = 1 \quad \text{for all } i \in A. \tag{4.56}$$

Therefore, at least t Bloom filter values are equal to 1, and so

$$\sum_{i=1}^{n} BF_i[j] \geq t. \tag{4.57}$$

Thus, the circuit outputs 1 exactly when at least t parties have value 1 at the given Bloom filter position. This proves the correctness of the threshold circuit.

4.4. Protocol Illustrative Example

Consider three parties P_1, P_2, and P_3 with threshold $t = 3$. Each party holds a private set of IP addresses. In this example, the IP addresses

$$10.0.0.1, \quad 10.0.0.2, \quad 10.0.0.3 \tag{4.58}$$

appear in all three parties and therefore satisfy the threshold condition.

$$\begin{aligned} S_1 &= \{10.0.0.1, 10.0.0.2, 10.0.0.3, 10.0.0.4, 10.0.0.5,\\ &\quad 10.0.0.6, 10.0.0.7, 10.0.0.8, 10.0.0.9, 10.0.0.10\},\\ S_2 &= \{10.0.0.1, 10.0.0.2, 10.0.0.3, 10.0.0.11, 10.0.0.12,\\ &\quad 10.0.0.13, 10.0.0.14, 10.0.0.15, 10.0.0.16, 10.0.0.17\},\\ S_3 &= \{10.0.0.1, 10.0.0.2, 10.0.0.3, 10.0.0.18, 10.0.0.19,\\ &\quad 10.0.0.20, 10.0.0.21, 10.0.0.22, 10.0.0.23, 10.0.0.24\}. \end{aligned} \tag{4.59}$$

For illustration, assume a Bloom filter of length $m = 12$ is used. After applying the set insertion algorithm, the parties obtain the following Bloom filters:

$$BF_1 = [1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1] \tag{4.60}$$

$$BF_2 = [1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0] \tag{4.61}$$

$$BF_3 = [1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0] \tag{4.62}$$

Each party encrypts its Bloom filter bitwise using the joint public key pk:

$$\begin{aligned} EBF_1 = [&\mathsf{Enc}(1), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1),\\ &\mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(1)] \end{aligned} \tag{4.63}$$

$$\begin{aligned} EBF_2 = [&\mathsf{Enc}(1), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(1),\\ &\mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0)] \end{aligned} \tag{4.64}$$

$$\begin{aligned} EBF_3 = [&\mathsf{Enc}(1), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(1), \mathsf{Enc}(1),\\ &\mathsf{Enc}(0), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0)] \end{aligned} \tag{4.65}$$

Since $n = 3$ and $t = 3$, the threshold circuit is the conjunction of all three inputs:

$$C(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3. \tag{4.66}$$

In arithmetic form, this is evaluated as:

$$C(x_1, x_2, x_3) = x_1 \cdot x_2 \cdot x_3. \tag{4.67}$$

For each Bloom filter position j, the cloud server computes:

$$EBF^*[j] = \mathsf{Eval}\left(C, EBF_1[j], EBF_2[j], EBF_3[j]\right). \tag{4.68}$$

Equivalently,

$$EBF^*[j] = EBF_1[j] \cdot EBF_2[j] \cdot EBF_3[j]. \tag{4.69}$$

For example, for the first few positions:

$$EBF^*[0] = \mathsf{Enc}(1) \cdot \mathsf{Enc}(1) \cdot \mathsf{Enc}(1) = \mathsf{Enc}(1). \tag{4.70}$$

$$EBF^*[2] = \mathsf{Enc}(0) \cdot \mathsf{Enc}(0) \cdot \mathsf{Enc}(0) = \mathsf{Enc}(0). \tag{4.71}$$

$$EBF^*[3] = \mathsf{Enc}(1) \cdot \mathsf{Enc}(0) \cdot \mathsf{Enc}(1) = \mathsf{Enc}(0). \tag{4.72}$$

$$EBF^*[4] = \mathsf{Enc}(0) \cdot \mathsf{Enc}(1) \cdot \mathsf{Enc}(1) = \mathsf{Enc}(0). \tag{4.73}$$

Thus, the encrypted threshold-intersection Bloom filter is:

$$\begin{aligned} EBF^* = [&\mathsf{Enc}(1), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(0), \mathsf{Enc}(0), \mathsf{Enc}(1), \\ &\mathsf{Enc}(0), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0)]. \end{aligned} \tag{4.74}$$

After collaborative decryption, the parties obtain:

$$BF^* = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0]. \tag{4.75}$$

The parties then perform membership verification over the IP address space $\mathcal{U}$ using Algorithm 3. The IP addresses that pass membership verification are recovered as the threshold intersection:

$$I^* = \{10.0.0.1,\ 10.0.0.2,\ 10.0.0.3\}. \tag{4.76}$$

Therefore, the protocol outputs the IP addresses that appear in at least $t = 3$ out of the $n = 3$ parties.

4.5. Complexity

In this section, we analyze the communication and computational complexity of the proposed Circuit-TMPSI protocol. Let n denote the number of parties, m denote the Bloom filter length, and t denote the agreed threshold value.

4.5.1. Communication Complexity

Each party P_i constructs a Bloom filter BF_i of length m and encrypts each Bloom-filter entry to obtain the encrypted Bloom filter EBF_i. Therefore, each party sends m encrypted Bloom-filter entries to the cloud server. Since there are n parties, the total communication from the parties to the cloud server is

$$O(mn). \tag{4.77}$$

After evaluating the threshold circuit, the cloud server returns the encrypted threshold-intersection Bloom filter EBF^*, of length m, to all parties. Since EBF^* is sent to n parties, this requires

$$O(mn) \tag{4.78}$$

communication from the cloud server.

The parties then perform collaborative decryption. Each party computes one partial decryption share for each entry of EBF^* and sends these m partial decryption shares to P_1. Across all n parties, this requires

$$O(mn) \tag{4.79}$$

communication.

After combining the partial decryptions, P_1 shares the recovered Bloom filter BF^*, of length m, with all parties. Since BF^* is sent to n parties, this adds

$$O(mn) \tag{4.80}$$

communication.

Therefore, the total communication complexity of the protocol is

$$O(mn + mn + mn + mn). \tag{4.81}$$

Thus, the overall communication complexity is

$$O(mn). \tag{4.82}$$

The protocol requires two main communication rounds after setup. In the first round, the parties send their encrypted Bloom filters to the cloud server, and the cloud server returns the encrypted threshold-intersection Bloom filter EBF^* to all parties. In the second round, the parties send partial decryption shares to P_1, and P_1 shares the recovered Bloom filter BF^* with all parties.

The communication complexity per party is

$$O(m), \tag{4.83}$$

because each party sends one encrypted Bloom filter of length m, receives the encrypted output Bloom filter, sends m partial decryption shares, and receives the recovered Bloom filter.

4.5.2. Computational Complexity

The computational cost of the protocol comes from three main steps: Bloom-filter construction by the parties, homomorphic threshold-circuit evaluation by the cloud server, and collaborative decryption by the parties.

First, each party P_i constructs a Bloom filter BF_i from its private set S_i. If $|S_i| = s$ and the Bloom filter uses k hash functions, then inserting all elements of S_i requires

$$O(ks) \tag{4.84}$$

operations per party. Since the optimal number of Bloom-filter hash functions is

$$k = \frac{m}{s} \ln 2, \tag{4.85}$$

the Bloom-filter construction cost per party becomes

$$O(ks) = O(m \ln 2) = O(m). \tag{4.86}$$

The main computational cost occurs during the homomorphic evaluation of the threshold circuit by the cloud server. For each Bloom-filter position j, the server evaluates the threshold circuit over the encrypted values

$$EBF_1[j], EBF_2[j], \dots, EBF_n[j]. \tag{4.87}$$

The threshold circuit outputs 1 if at least t out of the n plaintext Bloom-filter entries are equal to 1. Since the circuit is evaluated homomorphically, the cloud server obtains an encrypted output rather than the plaintext value. As described in the circuit-generation section, the threshold circuit considers all subsets of t parties. The number of such subsets is

$$\binom{n}{t}. \tag{4.88}$$

For each subset of size t, the circuit computes the AND of the corresponding t Bloom-filter values. Therefore, the cost of evaluating all terms for a single Bloom-filter position is

$$O\left(t\binom{n}{t}\right). \tag{4.89}$$

Since the same threshold circuit is evaluated independently for each of the m Bloom-filter positions, the total computational complexity of the cloud server is

$$O\left(mt\binom{n}{t}\right). \tag{4.90}$$

After homomorphic evaluation, the parties collaboratively decrypt the encrypted threshold-intersection Bloom filter EBF^*. Each party computes one partial decryption share for each of the m encrypted Bloom-filter entries, which requires

$$O(m) \tag{4.91}$$

operations per party. Party P_1 then combines the partial decryption shares from all n parties for all m Bloom-filter entries. This reconstruction step requires

$$O(mn) \tag{4.92}$$

operations for P_1.

Therefore, the dominant computational cost for the cloud server is

$$O\left(mt\binom{n}{t}\right). \tag{4.93}$$

The leader party P_1 performs the additional decryption-combination step, so its computational complexity is

$$O(mn). \tag{4.94}$$

Each non-leader party performs Bloom-filter construction and partial decryption, giving per-party computational complexity

$$O(m). \tag{4.95}$$

4.6. Security Analysis

The proposed protocol is secure under semi-honest security model, where all parties and the cloud server follow the protocol but may try to infer additional information from the messages they observe.

4.6.1. Security Assumptions

The security of the protocol relies on the following assumptions:

- The underlying threshold BFV encryption is IND-CPA secure.
- No adversary obtains enough secret-key shares to decrypt ciphertexts outside the collaborative decryption procedure.
- The cloud server CS is semi-honest and performs the required homomorphic operations correctly.

The threshold parameter t denotes the threshold-intersection condition: an element is included in the output if it appears in at least t parties. This is independent of the decryption threshold of the threshold FHE scheme.

4.6.2. Leakage

The protocol reveals the public parameters n, t, m, the arithmetic circuit C, the public key pk, and the public evaluation keys ek. The cloud server observes the encrypted Bloom filters $EBF_1, \ldots, EBF_n$ and the encrypted output Bloom filter EBF^*. After collaborative decryption, the parties learn the output Bloom filter BF^* and the candidate threshold-intersection set I^*.The protocol also inherits the standard leakage of Bloom filters due to the false positive nature.

4.6.3. Simulation-Based Security Proof

We show that every semi-honest adversary's view can be simulated using only its allowed input and output.

Corrupted Cloud Server

Consider an adversary corrupting the cloud server CS. Its real view is

$$View_{CS} = (n, t, m, C, pk, ek, EBF_1, \ldots, EBF_n, EBF^*). \tag{4.96}$$

By IND-CPA security of the threshold FHE scheme, encryptions of the real Bloom filter bits are computationally indistinguishable from encryptions of dummy bits. Therefore, a simulator $\mathcal{S}_{CS}$ can generate a simulated view as follows:

1. Given (n, t, m, C), generate simulated public key material (pk, ek).
2. For every $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, m-1\}$, compute
$$\widetilde{EBF}_i[j] \leftarrow \mathsf{Enc}_{pk}(0). \tag{4.97}$$
3. Homomorphically evaluate C on the simulated ciphertexts to obtain $\widetilde{EBF}^*$.
4. Output
$$\widetilde{View}_{CS} = (n, t, m, C, pk, ek, \widetilde{EBF}_1, \ldots, \widetilde{EBF}_n, \widetilde{EBF}^*). \tag{4.98}$$

Since the simulated ciphertexts are computationally indistinguishable from the real ciphertexts,
$$View_{CS} \approx_c \widetilde{View}_{CS}. \tag{4.99}$$

Thus, the cloud server learns nothing about the parties' private sets.

Corrupted Party

Now consider an adversary corrupting party P_i. Its real view is
$$View_{P_i} = (S_i, BF_i, EBF_i, sk_i, pk, ek, EBF^*, BF^*, I^*), \tag{4.100}$$
where sk_i is the secret-key share of P_i. The party already knows S_i and BF_i. Since P_i holds only one secret-key share, it cannot decrypt other parties' encrypted Bloom filters or the encrypted output alone.

A simulator $\mathcal{S}_{P_i}$, given S_i and the expected output (BF^*, I^*), constructs a simulated view as follows:

1. Compute
$$BF_i \leftarrow \mathsf{SetInsertion}(S_i). \tag{4.101}$$
2. Generate simulated key material (pk, ek, sk_i).
3. Encrypt BF_i to obtain
$$\widetilde{EBF}_i[j] \leftarrow \mathsf{Enc}_{pk}(BF_i[j]). \tag{4.102}$$
4. Encrypt the prescribed output Bloom filter:
$$\widetilde{EBF}^*[j] \leftarrow \mathsf{Enc}_{pk}(BF^*[j]). \tag{4.103}$$
5. Output
$$\widetilde{View}_{P_i} = (S_i, BF_i, \widetilde{EBF}_i, sk_i, pk, ek, \widetilde{EBF}^*, BF^*, I^*). \tag{4.104}$$

By IND-CPA security, the simulated encrypted values are computationally indistinguishable from the real encrypted values. Hence,
$$View_{P_i} \approx_c \widetilde{View}_{P_i}. \tag{4.105}$$

Therefore, a corrupted semi-honest party learns only its own input and the final output.

Collaborative Decryption

During collaborative decryption, each party P_i computes a partial decryption share for every encrypted Bloom-filter entry:
$$d_i[j] \leftarrow \mathsf{PartialDecrypt}_{sk_i}(EBF^*[j]). \tag{4.106}$$
Party P_i then sends $d_i[j]$ to the designated reconstruction party P_1.

The plaintext Bloom-filter entry is recovered only after P_1 combines the required partial decryptions:
$$BF^*[j] = \mathsf{Combine}(d_1[j], d_2[j], \ldots, d_n[j]). \tag{4.107}$$

After recovering all entries, P_1 shares the plaintext threshold-intersection Bloom filter BF^* with the other parties.

By the security of threshold decryption, an insufficient set of partial decryption shares reveals neither the plaintext Bloom-filter entry nor any secret-key share.

Collusion Resistance

In the implemented protocol, the decryption threshold is set to $t_{\text{dec}} = n$, so recovering the plaintext output Bloom filter requires valid partial decryption shares from all n parties. This means that any coalition missing even one party does not have enough decryption shares to recover the plaintext output.

Now consider a coalition $\mathcal{A}$ consisting of the cloud server CS and at most $n - 1$ parties. Since at least one party remains outside the coalition, $\mathcal{A}$ is missing a required partial decryption share. As a result, the coalition cannot decrypt the encrypted output Bloom filter EBF^*. The same reasoning applies to the parties' encrypted Bloom filters $EBF_1, \ldots, EBF_n$, which also remain encrypted under the threshold FHE scheme.

Therefore, even if the coalition has access to the public key pk, the evaluation keys ek, the encrypted Bloom filters $EBF_1, \ldots, EBF_n$, and the encrypted output EBF^*, it cannot recover the corresponding plaintext Bloom filters without the missing decryption share. Thus, the protocol remains secure against collusion between CS and any subset of at most $n - 1$ parties.

5 Evaluation

We evaluated the Circuit-TMPSI protocol using a Python implementation that simulates multiple participating organizations and a cloud-assisted computation model on a single machine. Each organization is represented as a separate simulated party. During preprocessing, each party constructs a Bloom filter from its set of suspicious IP addresses and encrypts the resulting Bloom-filter entries. The encrypted Bloom filters are then passed to the main thread, which performs the homomorphic threshold-circuit evaluation. The main thread also performs threshold decryption of the encrypted output Bloom filter and searches the recovered Bloom filter to identify the IP addresses reported in the threshold intersection.

5.1. Experimental Setup

The experiments were conducted on DelftBlue 64-bit `compute-p1` nodes [16]. Each node is equipped with $2\times$ Intel Xeon Gold 6248R processors running at 3.0 GHz, with 48 CPU cores and 185 GB of memory. For each experimental run, we requested 8 CPU cores and 4 GB of RAM.

The implementation uses `PyProbables` to construct Bloom filters and the `OpenFHE` Python library to perform threshold BFV homomorphic encryption operations [7, 1, 38]. The BFV scheme is instantiated in Residue Number System form under the Ring Learning With Errors assumption [13] and configured for 128-bit classical security. We used a plaintext modulus of 65,537. The multiplicative depth was selected according to the threshold-circuit structure and was typically between 8 and 12 levels, depending on the sum-of-products representation of the circuit.

For all experiments, the Bloom-filter false-positive rate was set to 10^{-4}. Bloom-filter bits were processed in batches to reduce communication and ciphertext overhead, with each ciphertext packing up to 2048 Bloom-filter bits. If the Bloom filter contained fewer than 2048 bits, the batching size was set to the Bloom-filter length. Each experiment was executed five times, and the reported timing results are the average computation times across these runs.

5.2. Datasets

We use synthetic datasets to simulate the private sets of suspicious IP addresses held by the participating organizations. Each dataset is generated using four parameters: n, t, c, and s. Here, n denotes the number of parties, t denotes the threshold value, c denotes the number of IP addresses that are constructed to appear in the threshold intersection, and s denotes the number of IP addresses in each party's private set.

For each experiment, synthetic IP addresses are generated according to the selected parameter values and written to separate input files. Each file corresponds to one party and serves as the party's private input set S_i. This setup reflects the intended CTI deployment scenario, where each organization reads suspicious IP addresses from its own local network logs before participating in the protocol.

5.3. Result Analysis

This section evaluates the performance of the Circuit-TMPSI protocol under different parameter settings. We first break down the total computation time into the main protocol phases in order to identify where the dominant cost occurs. We then evaluate how the protocol behaves as key parameters are varied, including the number of parties n, the threshold value t, the number of shared IP addresses c, the private set size s, and the Bloom-filter false-positive rate ϵ.

5.3.1. Timing Breakdown Across Protocol Phases

To identify the main sources of computational cost, we measure the runtime of three protocol phases: preprocessing, circuit evaluation, and post-processing. Preprocessing corresponds to encrypting each party's Bloom filter, circuit evaluation corresponds to the cloud server's homomorphic evaluation of the threshold circuit, and post-processing includes collaborative decryption of the output Bloom filter and membership testing to recover the reported IP addresses.

Figure 5.1 shows the phase-level timing breakdown as the input set size increases. Since larger input sets require larger Bloom filters to maintain the selected false-positive rate, preprocessing cost increases because each party encrypts more Bloom-filter bits. Post-processing also increases because more output positions must be decrypted and searched. However, at larger input sizes, circuit evaluation accounts for the largest share of the total runtime.

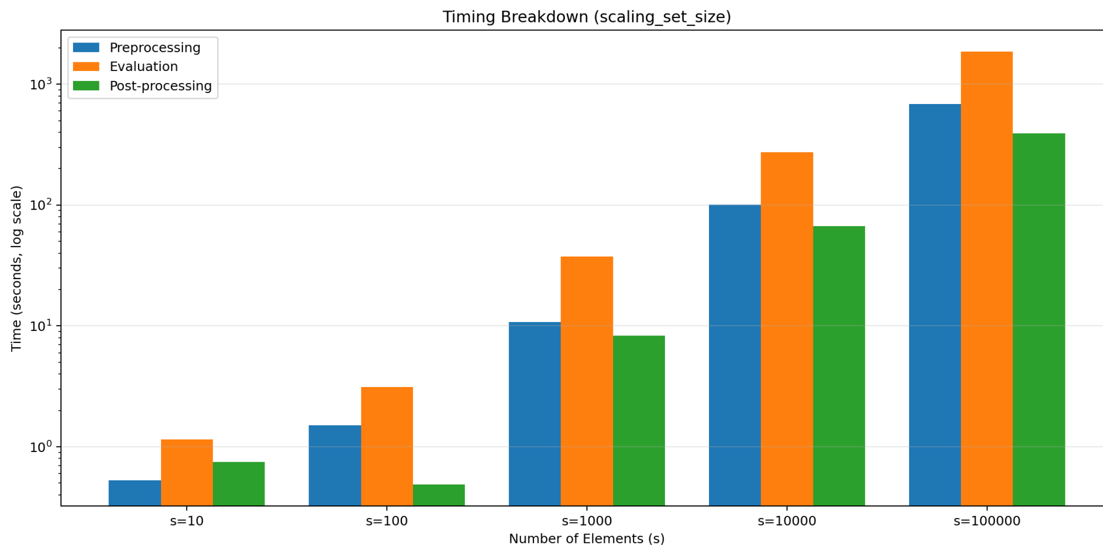


Figure 5.1: Timing breakdown of preprocessing, circuit evaluation, and post-processing as the input set size increases.

Figure 5.2 shows the timing breakdown as the threshold t varies from 1 to n. In this experiment, the input set size and Bloom-filter parameters are fixed. Therefore, preprocessing and post-processing change only slightly across threshold values because both phases process Bloom filters of the same length. In contrast, evaluation time varies substantially with t, since the threshold-circuit size depends on $\binom{n}{t}$.

Overall, preprocessing and post-processing are mainly determined by the Bloom filter length, while circuit evaluation is strongly affected by the threshold circuit size. The results show that homomorphic circuit evaluation is the dominant computational bottleneck, especially when the threshold produces a large number of circuit terms. Therefore, the remaining experiments focus on the circuit evaluation time and analyze how it changes as the protocol parameters are varied.

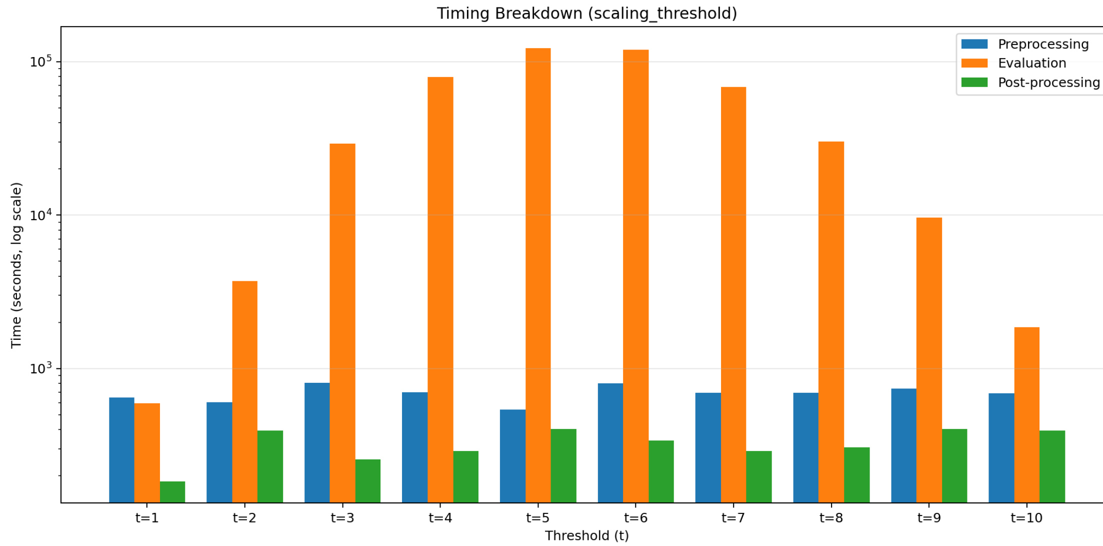


Figure 5.2: Timing breakdown of preprocessing, circuit evaluation, and post-processing as the threshold value varies.

5.3.2. Effect of Scaling the Number of Parties and Threshold

This experiment evaluates how the computation time of Circuit-TMPSI changes when both the number of parties n and the threshold value t are varied. We consider three party counts, $n = 6$, $n = 8$, and $n = 10$. For each value of n, the threshold t is varied from 1 to n. The number of IP addresses in the threshold intersection is fixed to $c = 5$, and the size of each party's private set is fixed to $s = 100000$. This setup allows us to observe how the protocol behaves as the collaboration size increases and as the threshold moves from low to high values.

Figure 5.3 shows two main trends. First, for a fixed number of parties, the computation time is lowest when the threshold is close to either end of the range, such as $t = 1$ or $t = n$, and highest when the threshold is close to the middle of the range. This behavior follows from the structure of the threshold circuit. For each Bloom-filter position, the circuit evaluates one term for every subset of t parties, so the number of terms is

$$\binom{n}{t}.$$

The binomial coefficients are symmetric, since

$$\binom{n}{t} = \binom{n}{n-t}.$$

This means that thresholds t and $n - t$ require the same number of circuit terms. The number of terms is smallest near the extremes and largest near the middle threshold values. This pattern can also be understood from Pascal's triangle. The entries in each row correspond to the binomial coefficients for a fixed value of n:

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array}$$

For example, the row of Pascal's triangle for $n = 4$ gives the coefficients

$$\binom{4}{0}, \binom{4}{1}, \binom{4}{2}, \binom{4}{3}, \binom{4}{4} = 1,\ 4,\ 6,\ 4,\ 1.$$

These values show the same symmetric pattern used by the threshold circuit: the number of terms increases toward the middle coefficient and then decreases again. Thus, thresholds near the middle of

the range require more circuit terms, whereas thresholds near the extremes require fewer terms. The same combinatorial structure explains the shape of the curves in Figure 5.3.

Second, the computation time increases as the number of parties increases. For the same threshold region, the curve for $n = 10$ is higher than the curves for $n = 8$ and $n = 6$. This occurs because adding more parties increases the number of encrypted Bloom filters involved in the computation and increases the number of possible party subsets considered by the threshold circuit. The effect is especially visible near the middle thresholds, where $\binom{n}{t}$ grows most rapidly. As a result, the middle-threshold cases for $n = 10$ require substantially more homomorphic computation than the corresponding cases for smaller values of n.

Overall, this experiment shows that the computation time depends on both the number of participating parties and the selected threshold. Increasing n raises the overall cost of the protocol, while the threshold value determines where the cost is concentrated. The most expensive cases occur when t is near the middle of the range, whereas thresholds close to 1 or n are more efficient.

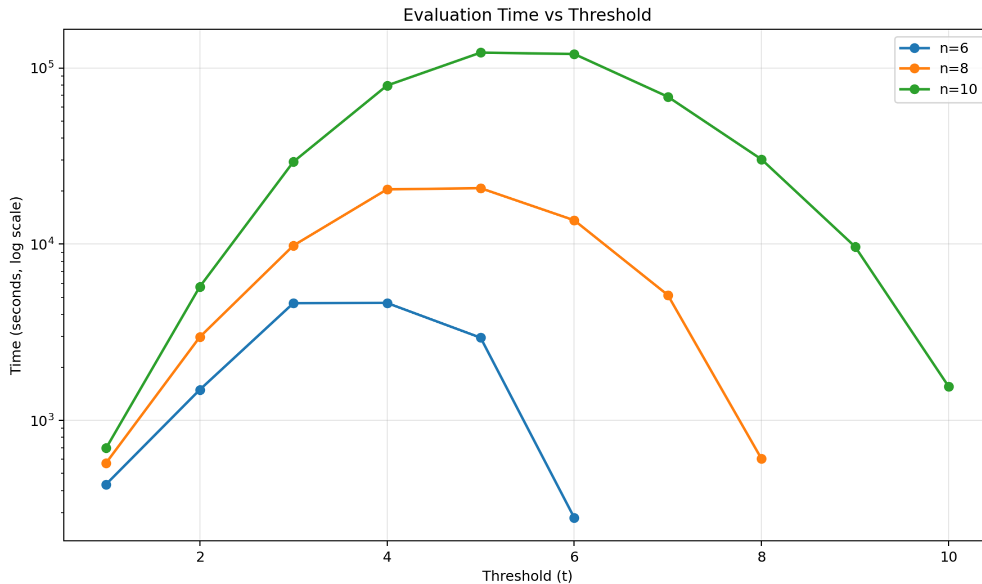


Figure 5.3: Computation time when varying the number of parties n and the threshold value t.

5.3.3. Effect of Scaling Party Set Size

This experiment evaluates how Circuit-TMPSI scales as the size of each party's private set increases. We fix the number of parties to $n = 10$ and vary the private set size s from 10 to 10^5. To confirm that the scaling trend is consistent across different threshold settings, we run the experiment for three threshold values: $t = 1$, $t = 5$, and $t = 10$.

Figure 5.4 shows that computation time increases as the private set size grows for all three threshold values. This increase is caused by the corresponding growth in the Bloom filter length. As discussed in Section 2, larger input sets require larger Bloom filters to maintain the selected false-positive rate. Since the threshold circuit is evaluated over each Bloom-filter position, a larger Bloom filter directly increases the number of encrypted positions processed by the cloud server.

The same increasing trend is observed for $t = 1$, $t = 5$, and $t = 10$, showing that set-size growth increases computation time regardless of the chosen threshold. The main observation in this experiment is that increasing the private set size enlarges the Bloom filter, which increases the number of encrypted positions processed by the cloud server and therefore increases the overall computation time.

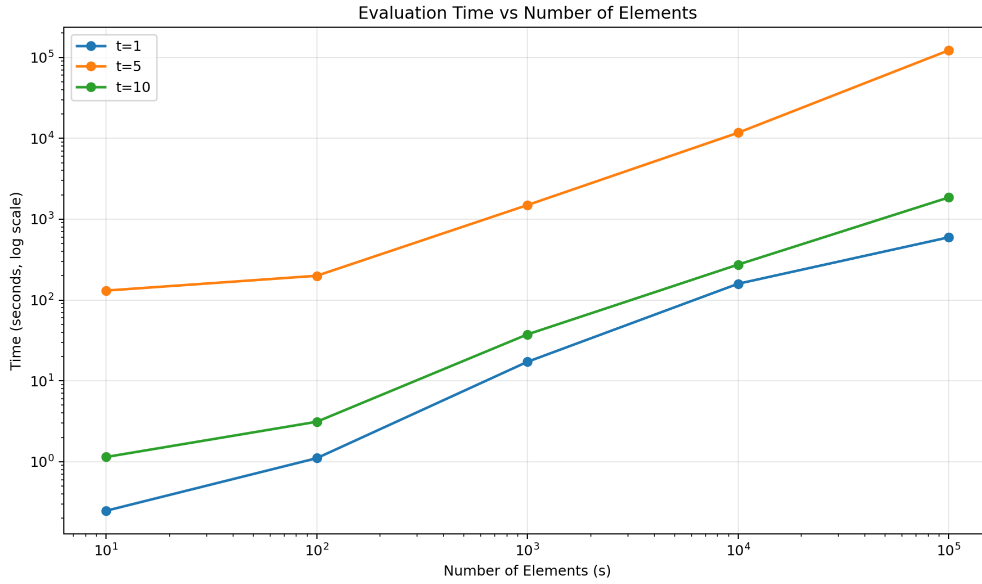


Figure 5.4: Computation time when varying the private set size s under different threshold values.

5.3.4. Effect of Scaling Intersection Elements

This experiment evaluates whether the number of IP addresses in the threshold intersection affects the runtime of Circuit-TMPSI. We consider three party counts, $n = 6$, $n = 8$, and $n = 10$. For each value of n, we set the threshold to $t = n$, which corresponds to the standard multi-party intersection case. Each party's private set contains $s = 10^5$ IP addresses, and the number of shared IP addresses c is varied from 10 to 10^5.

Figure 5.5 shows that changing c has little effect on the overall computation time. The runtimes vary slightly across different values of c, but there is no consistent increasing or decreasing trend as the number of shared IP addresses grows. This is expected because, after the private sets are encoded as Bloom filters, the homomorphic evaluation depends mainly on the Bloom filter length m, the number of parties n, and the threshold t. Since varying c does not change the number of Bloom-filter positions evaluated by the cloud server, it does not substantially affect the computation time.

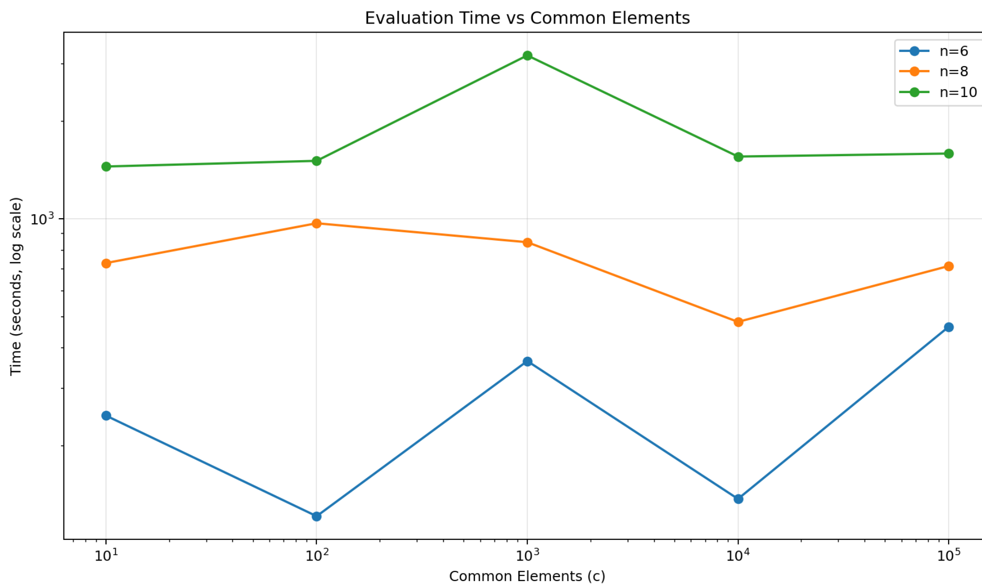


Figure 5.5: Computation time when varying the number of shared IP addresses c.

5.3.5. Effect of Reducing the False Positive Rate

Bloom filter introduces a trade-off between accuracy and size. Let s denote the number of inserted IP addresses per party, m denote the Bloom filter length, and k denote the number of hash functions. The false positive rate ϵ is given by

$$\epsilon = \left(1 - e^{-ks/m}\right)^k.$$

For a target false positive rate ϵ, the required Bloom filter length can be approximated as

$$m = -\frac{s \ln \epsilon}{(\ln 2)^2}.$$

Thus, lowering the false positive rate requires a larger Bloom filter.

Figure 5.6 shows how the Bloom filter length increases as the false positive rate decreases. Smaller values of ϵ require longer Bloom filters, which increases the number of encrypted batches processed by Circuit-TMPSI. In our implementation, each ciphertext packs up to 2048 Bloom-filter bits; therefore, a Bloom filter of length m is processed in approximately $\lceil m/2048 \rceil$ ciphertext batches. As a result, reducing the false positive rate increases communication and computation overhead, while improving output accuracy.

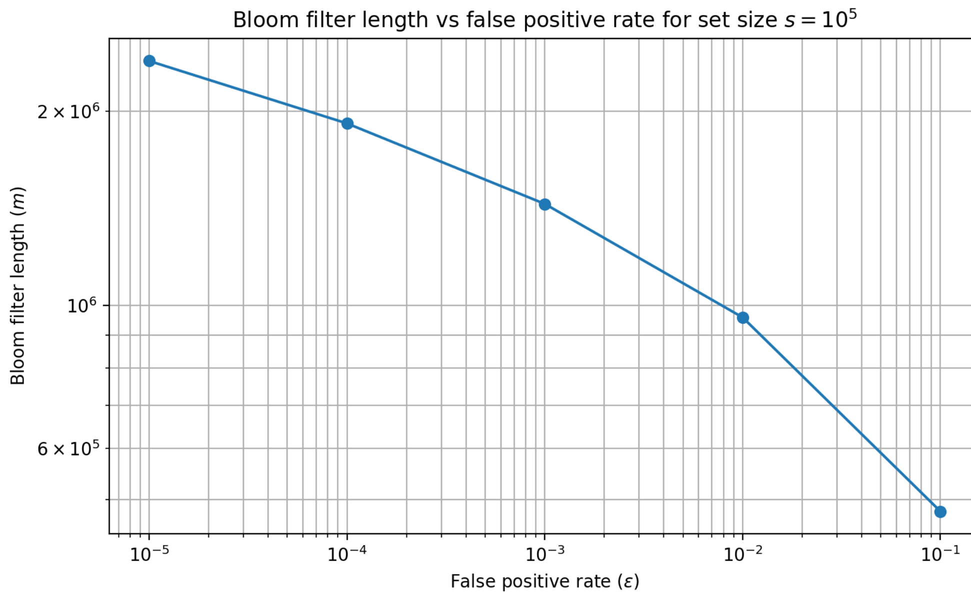


Figure 5.6: Bloom filter length required for different false positive rates.

5.3.6. Effect of Threshold on False Candidate Recovery

This experiment examines how the threshold value affects the number of false candidate IP addresses recovered from the decrypted threshold Bloom filter. The party set size is fixed at $s = 10^5$, the Bloom-filter false-positive rate is set to $\epsilon = 10^{-4}$, and the number of seeded threshold-intersection IP addresses is fixed at $c = 5$. We evaluate three party counts, $n \in \{6, 8, 10\}$. For each value of n, the threshold t is varied from 2 to n.

After homomorphic evaluation and collaborative decryption, the parties obtain the threshold Bloom filter BF^*. To recover IP addresses from BF^*, the implementation first forms the union of all IP addresses that appear in the generated party input sets. It then checks each IP address in this union for membership in BF^*. This recovery strategy reflects the experimental setting, where the generated input IP addresses are known, and avoids enumerating the full IPv4 address space of 2^{32} possible addresses. Consequently, the reported false candidate counts represent false positives observed within the generated input union only. They should not be interpreted as the total number of possible false positives over the entire IPv4 address space.

The case $t = 1$ is excluded from this experiment because it corresponds to the union of the parties' private sets. In other words, any IP address appearing in at least one party's input set is a valid threshold output. Since the recovery procedure only tests IP addresses from the generated input union, every tested IP address is a true output when $t = 1$. Therefore, this experiment cannot observe false candidates for $t = 1$ without testing additional IP addresses outside the generated union. Such a test would require the full IPv4 address space which is time consuming. For this reason, the analysis focuses on thresholds $t \geq 2$, where false candidates can be identified within the generated input union.

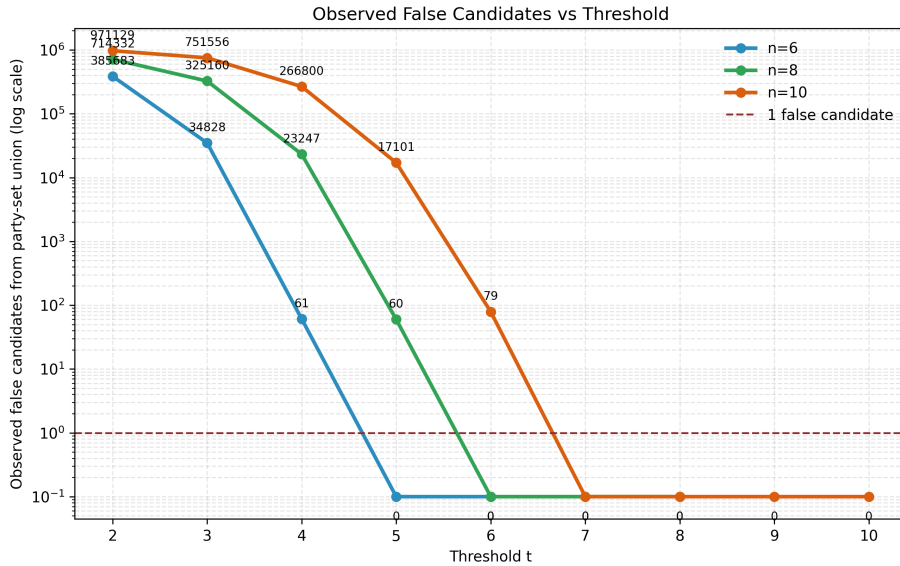


Figure 5.7: Observed false candidate counts recovered from the decrypted threshold Bloom filter for $n \in \{6, 8, 10\}$, party set size $s = 10^5$, seeded intersection size $c = 5$, and Bloom-filter false-positive rate $\epsilon = 10^{-4}$. The thresholds range from $t = 2$ to $t = n$, and the y-axis is shown on a logarithmic scale.

Figure 5.7 shows that the number of observed false candidates decreases as the threshold approaches the total number of parties. At lower and intermediate thresholds, an IP address from the generated input union that does not satisfy the threshold condition may still pass membership verification because its Bloom-filter positions can satisfy the threshold condition by chance. As t increases, more parties must contribute to each Bloom-filter position before it is set in BF^*. This makes the threshold Bloom filter more selective and reduces the probability that an IP address outside the exact threshold intersection is accepted as a candidate.

In the observed runs, the number of false candidates drops to zero when the threshold becomes sufficiently close to n. This shows that higher threshold values make the recovered candidate set more selective and reduce accidental candidate recovery. Although the experiment tests only IP addresses from the generated input union, rather than all 2^{32} possible IPv4 addresses, the same qualitative trend is expected for a larger search space. As t increases, more parties must support each Bloom-filter position before it is set in BF^*, making false positives less likely. Therefore, the fewest false positives are expected when $t = n$, where the threshold condition is most restrictive.

6

Discussion & Future Work

This chapter examines the practical deployment of the proposed Circuit-TMPSI protocol, discusses its main limitations, and outlines directions for future work.

6.1. Practical Deployment

The experimental results in Chapter 5 demonstrate that the proposed protocol can support collaborative threat intelligence workflows when its deployment parameters are selected appropriately. In particular, the threshold value plays a central role in determining both the efficiency and the accuracy of the protocol output. For an input size of 10^5 IP addresses per party and 10 participating organizations, the protocol completed evaluation in just under 2000 seconds when the threshold was close to either end of the range, namely $t = 1$ or $t = 10$, as shown in Figure 5.3. This corresponds to a runtime of approximately 30 minutes, which makes the protocol better suited to periodic CTI correlation tasks than to real-time detection.

The choice of threshold has two main deployment implications. First, as shown by the threshold-scaling experiment in Figure 5.3, the threshold directly affects computation time. Second, as shown by the false-positive analysis in Figure 5.7, it also affects output accuracy. Lower thresholds are more permissive because fewer parties need to contain the corresponding Bloom-filter positions before an IP address is considered a match. This can increase coverage, but it may also produce more false positives. In contrast, higher thresholds are stricter because the Bloom-filter positions associated with an IP address must be supported by more parties before the indicator is represented in the final intersection Bloom filter BF^*. Therefore, increasing the threshold, especially when it is chosen closer to n, substantially improves output accuracy by reducing false positives.

In practical CTI deployments, the threshold should therefore be selected according to the intended operational meaning of the output. A low threshold is suitable when organizations want to identify IP addresses that may have been observed by only a small number of participants. For example, in a deployment with 10 organizations, setting $t = 2$ reports indicators whose Bloom-filter positions are supported by at least two organizations. This setting can be useful for broad exploration or early warning, but it may include more false positives because the required level of agreement is low. A higher threshold is more appropriate when organizations want to prioritize indicators that are repeatedly observed across many participants. For example, setting $t = 6$ in the same deployment reports indicators supported by a majority of the participating organizations. Such indicators provide stronger evidence that the activity is not an isolated local event, but part of a broader reconnaissance or attack campaign [3, 54]. In this setting, choosing a threshold greater than $n/2$ emphasizes indicators with stronger cross-organizational support and reduces false positives from isolated local observations.

Practical deployment also depends on whether the protocol can be executed with manageable communication overhead. The implementation reduces communication cost through batching, where up to 2048 Bloom-filter bits are packed into a single ciphertext. This reduces the number of ciphertexts exchanged between the participating organizations and the cloud server, allowing large Bloom filters

to be processed without transmitting or evaluating each bit separately. Batching also improves privacy at the representation level because the cloud server observes packed ciphertexts rather than separate ciphertexts for individual Bloom-filter bits. This reduces the bit-level information available from the encrypted representation during evaluation.

This communication efficiency is supported by the protocol's cloud-assisted deployment model, which avoids full pairwise communication between organizations. Participating organizations do not need to modify their internal network infrastructure or establish direct computation channels with every other party. Instead, most computation is outsourced to the cloud server, while the organizations mainly submit encrypted Bloom filters and participate in threshold decryption. To recover the final intersection Bloom filter, each party sends its decryption share to the designated leader P_1. Consequently, each organization only needs to communicate with the cloud server and the leader party, rather than maintaining pairwise communication with all other participants. This model is well suited to operational environments in which organizations follow strict security policies and avoid direct communication channels with external parties [3, 54].

Overall, the proposed protocol is practical for collaborative CTI deployments in which organizations aim to correlate indicators under a clearly defined threshold policy. The results show that deployment feasibility depends on selecting a threshold that balances runtime and accuracy. With batching and a cloud-assisted communication model, the protocol supports large-scale CTI correlation while reducing communication overhead and limiting what the cloud server can infer from the encrypted Bloom-filter representation.

6.2. Limitations

The proposed Circuit-TMPSI protocol has several limitations that affect its performance, accuracy, and leakage profile. The first limitation is the cost of evaluating the threshold circuit. In the optimized circuit, the cloud server evaluates one AND term for every subset of t parties, so the number of AND terms is $\binom{n}{t}$. This value is relatively small when t is close to the extremes, such as $t = 1$ or $t = n$. However, it grows significantly when t is close to $n/2$, where the number of possible subsets is largest. Since each AND term requires homomorphic operations, the evaluation time increases for middle threshold values. This explains the trend observed in the threshold-scaling experiment in Figure 5.3, where computation time is highest around middle thresholds and lower when the threshold is close to either end of the range.

A second limitation comes from the use of Bloom filters, which may produce false positives [9, 11]. In the proposed protocol, a false positive occurs when an IP address is reported as satisfying the threshold condition even though it does not appear in at least t parties' private sets. This can happen because different IP addresses may map to overlapping Bloom-filter positions. In practical CTI deployments, such false positives can have operational consequences. For example, if the reported threshold intersection is used to block malicious IP addresses, a false positive may cause a legitimate IP address to be incorrectly blocked. This can affect legitimate users and negatively impact an organization's services or business operations.

Reducing false positives requires larger Bloom filters. As shown in Figure 5.6, lowering the desired false positive rate ϵ increases the required Bloom filter length m. This directly affects the cost of the Circuit-TMPSI protocol, because a larger Bloom filter requires each party to encrypt and transmit more entries. It also requires the cloud server to evaluate the threshold circuit at more Bloom-filter positions. Therefore, the protocol involves a trade-off between accuracy and performance: reducing the false positive rate improves output reliability, but it increases both communication cost and homomorphic-evaluation time.

A final limitation is that some protocol parameters remain visible to the cloud server. In particular, the cloud server learns the number of participating parties n, the threshold value t, and the structure of the arithmetic circuit C. These parameters do not reveal the parties' private set elements, because the Bloom-filter entries are encrypted before being sent to the cloud server. However, they still reveal metadata about the computation, including the size of the collaboration and the threshold condition being evaluated. This leakage does not expose the underlying IP addresses, but it shows that the protocol does not fully hide all information about the collaborative setting. Hiding such circuit-related

and deployment-level metadata is therefore an important direction for future work.

6.3. Future Work

Several directions remain for improving the proposed Circuit-TMPSI protocol. Building on the leakage discussed above, a first direction is to reduce the computation metadata revealed to the cloud server. In the current protocol, the cloud server learns n, t, and the structure of the threshold circuit C. Future work should investigate secure mechanisms for generating or evaluating this circuit without directly revealing these parameters. Hiding such metadata would strengthen the privacy guarantees of the protocol beyond the confidentiality of the encrypted Bloom-filter entries.

A second direction is to study how the threshold parameter should be selected in collaborative threat intelligence deployments. In the current protocol, the threshold t determines how many organizations must observe the same indicator before it is reported in the output. However, the best threshold value is not only a cryptographic parameter. It also depends on the operational meaning of the indicator and on the level of cross-organizational agreement required before organizations treat the observation as significant.

The appropriate threshold may vary depending on the indicator type and the decision context. For example, a distinctive payload pattern associated with a known exploit may be meaningful even if it is observed by only one organization, because the indicator itself provides strong evidence of malicious activity. In such cases, a lower threshold can support early detection and faster investigation. In contrast, a common signal, such as scanning against an exposed port, may be less conclusive when observed by a single organization. For such noisy indicators, a higher threshold is more appropriate because repeated observation across several organizations provides stronger evidence of coordinated activity. Future work should therefore evaluate threshold selection for different types of correlated indicators, including IP addresses, port numbers, exposed services, payload features, and other CTI artifacts. Studying different indicator types and organization groups would help identify threshold values that produce reliable and actionable CTI output.

A third direction is to develop more efficient methods for evaluating the threshold function itself. The current approach evaluates all $\binom{n}{t}$ combinations of t parties. This guarantees correctness, but the number of terms grows quickly when t is close to $n/2$. In a plaintext setting, an OR expression can stop as soon as one term evaluates to 1. However, such early stopping is not directly applicable in the proposed protocol. If the cloud server learns when evaluation stops, it may also learn that the corresponding Bloom-filter position satisfies the threshold condition. This stopping signal would leak information about the encrypted threshold-intersection Bloom filter EBF^*.

For this reason, any optimization based on early stopping must hide intermediate results from the cloud server. One possible direction is to study privacy-preserving comparison or control mechanisms that can be evaluated under homomorphic encryption. Secure comparison techniques under homomorphic encryption are relevant for this purpose [15]. However, such mechanisms must be designed carefully so that they do not reveal which Bloom-filter positions satisfy the threshold condition or when the evaluation of a particular position stops. Optimizing threshold evaluation while preserving this privacy therefore remains a promising but nontrivial direction for future work.

7 Conclusion

In this work, we presented Circuit-TMPSI, a privacy-preserving threshold multi-party private set intersection protocol for collaborative cyber-threat intelligence. The protocol enables multiple organizations to identify suspicious IP addresses that appear in at least t private sets, while keeping each organization's individual network logs private.

Circuit-TMPSI represents each party's private set using a Bloom filter and evaluates a threshold circuit over encrypted Bloom-filter bits. The optimized threshold circuit checks, for each Bloom-filter position, whether at least t parties have value 1 at that position. Since the circuit is evaluated homomorphically, the cloud server obtains only encrypted outputs, $\mathrm{Enc}(1)$ or $\mathrm{Enc}(0)$, and does not learn the plaintext Bloom-filter entries or the threshold result during computation.

After collaborative decryption, the encrypted result EBF^* is recovered as a plaintext Bloom filter BF^*. This Bloom filter represents the threshold result at the Bloom-filter level: if an IP address truly appears in at least t private sets, then all of its corresponding Bloom-filter positions are set in BF^*. Therefore, membership verification over BF^* does not remove true threshold elements. However, because Bloom filters are probabilistic, other IP addresses may also pass the membership test even though they do not satisfy the threshold condition. Thus, the protocol outputs a candidate threshold-intersection set rather than an exact set. The probability of such false positives can be reduced by choosing suitable Bloom-filter parameters, including the Bloom-filter length m, the number of hash functions k, and the target false-positive rate ε.

The security analysis shows that Circuit-TMPSI protects the parties' private sets in the semi-honest model. The cloud server observes only encrypted Bloom-filter entries, public parameters, and the public threshold circuit. The privacy of the encrypted values follows from the semantic security of the threshold BFV scheme, while the final result is recovered only through collaborative decryption. As a result, no single party holds the complete secret key, and the cloud server cannot decrypt either the parties' encrypted inputs or the encrypted threshold-intersection Bloom filter.

The evaluation shows that the main performance costs are determined by the Bloom-filter length m, the number of parties n, and the threshold circuit size $\binom{n}{t}$. The protocol is most expensive when t is close to $n/2$, where the number of threshold subsets is largest, and more efficient when t is closer to 1 or n. The results also show the expected Bloom-filter trade-off: lowering the false-positive rate improves output accuracy but requires a larger Bloom filter, which increases communication and homomorphic evaluation cost.

Overall, Circuit-TMPSI provides a practical approach for privacy-preserving collaborative attack-campaign detection. It allows organizations to identify commonly observed suspicious IP addresses without directly exposing their private logs. Future work can build on this construction by exploring more compact threshold-circuit designs, improving the accuracy–efficiency trade-off of the Bloom-filter representation, and developing mechanisms that reduce the information revealed by the public circuit structure.

References

[1] Ahmad Al Badawi et al. "OpenFHE: Open-Source Fully Homomorphic Encryption Library". In: *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. WAHC'22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 53–63. doi: 10.1145/3560827.3563379. url: https://doi.org/10.1145/3560827.3563379.

[2] Adham Albakri, Eerke Boiten, and Rogério De Lemos. "Sharing Cyber Threat Intelligence Under the General Data Protection Regulation". In: *Privacy Technologies and Policy*. Ed. by Maurizio Naldi et al. Cham: Springer International Publishing, 2019, pp. 28–41. isbn: 978-3-030-21752-5.

[3] N.I.O.F.S. AND. *Guide to Cyber Threat Information Sharing: NiST SP 800-150*. CreateSpace Independent Publishing Platform, 2016. isbn: 9781548712853. url: https://books.google.nl/books?id=U0eGswEACAAJ.

[4] Onur Eren Arpaci, Raouf Boutaba, and Florian Kerschbaum. *Over-Threshold Multiparty Private Set Intersection for Collaborative Network Intrusion Detection*. 2025. arXiv: 2510.12045 [cs.CR]. url: https://arxiv.org/abs/2510.12045.

[5] Hayretdin Bahsi and Albert Levi. "Preserving organizational privacy in intrusion detection log sharing". In: *2011 3rd International Conference on Cyber Conflict*. 2011, pp. 1–14.

[6] Michael Bailey et al. "A Survey of Botnet Technology and Defenses". In: *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. 2009, pp. 299–304. doi: 10.1109/CATCH.2009.40.

[7] Tyler Barrus. *PyProbables*. Version 0.7.0. Feb. 2026. url: https://github.com/barrust/pyprobables.

[8] Aslı Bay et al. "Practical Multi-Party Private Set Intersection Protocols". In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 1–15. doi: 10.1109/TIFS.2021.3118879.

[9] Burton H. Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. issn: 0001-0782. doi: 10.1145/362686.362692.

[10] Prosenjit Bose et al. "On the false-positive rate of Bloom filters". In: *Information Processing Letters* 108.4 (2008), pp. 210–213. issn: 0020-0190. doi: https://doi.org/10.1016/j.ipl.2008.05.018. url: https://www.sciencedirect.com/science/article/pii/S0020019008001579.

[11] Andrei Broder and Michael Mitzenmacher. "Network Applications of Bloom Filters: A Survey". In: *Internet Mathematics* 1.4 (2003), pp. 485–509.

[12] Nishanth Chandran et al. *Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI*. Cryptology ePrint Archive, Paper 2021/172. 2021. url: https://eprint.iacr.org/2021/172.

[13] Jung Hee Cheon et al. *Homomorphic Encryption for Arithmetic of Approximate Numbers*. Cryptology ePrint Archive, Paper 2016/421. 2016. url: https://eprint.iacr.org/2016/421.

[14] Y. Crama and P.L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011. isbn: 9781139498630. url: https://books.google.nl/books?id=3KmyKpw_pbUC.

[15] Ivan Damgard, Martin Geisler, and Mikkel Kroigard. "Homomorphic encryption and secure comparison". In: *Int. J. Appl. Cryptol.* 1.1 (Feb. 2008), pp. 22–31. issn: 1753-0563. doi: 10.1504/IJACT.2008.017048.

[16] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2. 2024.

[17] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "ZMap: fast internet-wide scanning and its security applications". In: *Proceedings of the 22nd USENIX Conference on Security*. SEC'13. Washington, D.C.: USENIX Association, 2013, pp. 605–620. isbn: 9781931971034.

[18] European Union Agency for Cybersecurity (ENISA). *Encrypted Traffic Analysis*. Tech. rep. ENISA, Apr. 2020. url: https://www.enisa.europa.eu/publications/encrypted-traffic-analysis.

[19] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. 2012. url: https://eprint.iacr.org/2012/144.

[20] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. “Efficient Private Matching and Set Intersection”. In: *Advances in Cryptology – EUROCRYPT 2004*. Springer, 2004, pp. 1–19. doi: 10.1007/978-3-540-24676-3_1.

[21] Oded Goldreich. *Foundations of Cryptography, Volume 2: Basic Applications*. Cambridge University Press, 2004.

[22] Google. *Gemini*. Large language model. Version interacted with on June 5, 2026. Generated layout and text assistance for “Anatomy of a Cyber Attack”. 2026. url: https://gemini.google.com/.

[23] Chenghong Guan, J. S. van Assen, and Zekeriya Erkin. “Collective Threshold Multiparty Private Set Intersection Protocols for Cyber Threat Intelligence”. In: *Proceedings of the 16th IEEE International Workshop on Information Forensics and Security*. IEEE, 2024. url: https://research.tudelft.nl/en/publications/collective-threshold-multiparty-private-set-intersection-protocol/.

[24] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-14303-8.

[25] Martin Horák, Václav Stupka, and Martin Husák. “GDPR Compliance in Cybersecurity Software: A Case Study of DPIA in Information Sharing Platform”. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. ARES ’19. Canterbury, CA, United Kingdom: Association for Computing Machinery, 2019. isbn: 9781450371643.

[26] Abdulmalik Humayed et al. “Cyber-Physical Systems Security—A Survey”. In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1802–1831. doi: 10.1109/JIOT.2017.2703172.

[27] Eric Hutchins, Michael Cloppert, and Rohan Amin. “Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains”. In: vol. 1. Jan. 2011.

[28] Klaus Julisch. “Mining Alarm Clusters to Improve Alarm Handling Efficiency”. In: *Proceedings of the 17th Annual Computer Security Applications Conference*. IEEE Computer Society, 2001, pp. 12–21. doi: 10.1109/ACSAC.2001.991526.

[29] Lea Kissner and Dawn Song. “Privacy-Preserving Set Operations”. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Springer Berlin Heidelberg, 2005. isbn: 978-3-540-31870-5.

[30] Lea Kissner and Dawn Song. *Private and Threshold Set-Intersection*. Tech. rep. CMU-CS-04-182. School of Computer Science, Carnegie Mellon University, Nov. 2004.

[31] Hakan Kılıç, Neşet Sertaç Katal, and Ali Aydın Selçuk. “Evasion Techniques Efficiency Over The IPS/IDS Technology”. In: *2019 4th International Conference on Computer Science and Engineering (UBMK)*. 2019, pp. 542–547. doi: 10.1109/UBMK.2019.8907177.

[32] Vladimir Kolesnikov et al. “Practical Multi-party Private Set Intersection from Symmetric-Key Techniques”. In: 2017. url: https://eprint.iacr.org/2017/799.

[33] Zhen Li, Qi Liao, and Aaron Striegel. “Botnet Economics: Uncertainty Matters”. In: *Managing Information Risk and the Economics of Security*. Ed. by M. Eric Johnson. Boston, MA: Springer US, 2009, pp. 245–267. isbn: 978-0-387-09762-6. doi: 10.1007/978-0-387-09762-6_12. url: https://doi.org/10.1007/978-0-387-09762-6_12.

[34] Yehuda Lindell. “How to Simulate It – A Tutorial on the Simulation Proof Technique”. In: *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Ed. by Yehuda Lindell. Cham: Springer International Publishing, 2017, pp. 277–346. isbn: 978-3-319-57048-8. doi: 10.1007/978-3-319-57048-8_6. url: https://doi.org/10.1007/978-3-319-57048-8_6.

[35] Liju Ma et al. “Over-threshold multi-party private set operation protocols for lightweight clients”. In: *Computer Standards & Interfaces* 88 (2024), p. 103781. issn: 0920-5489. doi: https://doi.org/10.1016/j.csi.2023.103781. url: https://www.sciencedirect.com/science/article/pii/S0920548923000624.

[36] Rasoul Akhavan Mahdavi et al. "Practical Over-Threshold Multi-Party Private Set Intersection". In: *Proceedings of the 36th Annual Computer Security Applications Conference*. ACSAC '20. Austin, USA: Association for Computing Machinery, 2020, pp. 772–783. isbn: 9781450388580. doi: 10.1145/3427228.3427267.

[37] Christian Mouchet. "Multiparty Homomorphic Encryption: From Theory to Practice". PhD thesis. École Polytechnique Fédérale de Lausanne, 2023. url: https://infoscience.epfl.ch/entities/publication/bdab454b-9076-4bf2-b0f7-e40382f6c5c0.

[38] Mrityunjaya. *Circuit - Threshold MPSI*. Version 1.2.0. 2026. url: https://github.com/Martigo69/Threshold-PSI-Circuit.

[39] National Institute of Standards and Technology. *Attack Surface*. NIST Computer Security Resource Center Glossary. Accessed: 2026-06-01. url: https://csrc.nist.gov/glossary/term/attack_surface.

[40] National Institute of Standards and Technology. *The NIST Cybersecurity Framework (CSF) 2.0*. NIST Cybersecurity White Paper NIST CSWP 29. National Institute of Standards and Technology, 2024. doi: 10.6028/NIST.CSWP.29.

[41] Livinus Obiora Nweke and Stephen Wolthusen. "Legal Issues Related to Cyber Threat Information Sharing Among Private Entities for Critical Infrastructure Protection". In: *2020 12th International Conference on Cyber Conflict (CyCon)*. Vol. 1300. 2020, pp. 63–78. doi: 10.23919/CyCon49761.2020.9131721.

[42] Chaeyeon Oh, Joonseo Ha, and Heejun Roh. "A Survey on TLS-Encrypted Malware Network Traffic Analysis Applicable to Security Operations Centers". In: *Applied Sciences* 12.1 (2022). issn: 2076-3417. doi: 10.3390/app12010155. url: https://www.mdpi.com/2076-3417/12/1/155.

[43] OpenAI. *ChatGPT*. https://chat.openai.com/. Accessed: 2026-05-07. 2026.

[44] Eva Papadogiannaki and Sotiris Ioannidis. "Acceleration of Intrusion Detection in Encrypted Network Traffic Using Heterogeneous Hardware". In: *Sensors* 21.4 (2021). issn: 1424-8220. doi: 10.3390/s21041140. url: https://www.mdpi.com/1424-8220/21/4/1140.

[45] Andrea Pinto et al. "Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure". In: *Sensors* 23.5 (2023). issn: 1424-8220. doi: 10.3390/s23052415. url: https://www.mdpi.com/1424-8220/23/5/2415.

[46] Thomas Henry Ptacek and Timothy Nakula Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". In: 1998. url: https://api.semanticscholar.org/CorpusID:16418229.

[47] Karen Scarfone and Peter Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST Special Publication 800-94. National Institute of Standards and Technology, 2007. doi: 10.6028/NIST.SP.800-94. url: https://csrc.nist.gov/pubs/sp/800/94/final.

[48] Securus Communications. *Anatomy of a Cyber Attack*. Report content by Securus Communications; cover page and alternate text layout generated via Google Gemini. AI-assisted version available in the Gemini Archive platform. 2026. url: https://securuscomms.co.uk/anatomy-of-a-cyber-attack/ (visited on 06/05/2026).

[49] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613. doi: 10.1145/359168.359176.

[50] Sérgio S.C. Silva et al. "Botnets: A survey". In: *Computer Networks* 57.2 (2013). Botnet Activity: Analysis, Detection and Shutdown, pp. 378–403. issn: 1389-1286. doi: https://doi.org/10.1016/j.comnet.2012.07.021. url: https://www.sciencedirect.com/science/article/pii/S1389128612003568.

[51] Adam J. Slagell and William Yurcik. *Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization*. 2004. arXiv: cs/0409005 [cs.CR]. url: https://arxiv.org/abs/cs/0409005.

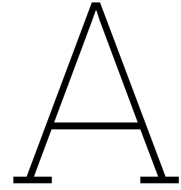
[52] Robin Sommer and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *2010 IEEE Symposium on Security and Privacy*. 2010, pp. 305–316. doi: 10.1109/SP.2010.25.

[53] Keith Stouffer et al. *Guide to Industrial Control Systems (ICS) Security*. Tech. rep. NIST Special Publication 800-82 Revision 2. National Institute of Standards and Technology, 2015. doi: 10.6028/NIST.SP.800-82r2.

[54] Thomas D. Wagner et al. "Cyber threat intelligence sharing: Survey and research directions". In: *Computers & Security* 87 (2019), p. 101589. issn: 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.101589. url: https://www.sciencedirect.com/science/article/pii/S016740481830467X.

[55] Ingo Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons and B. G. Teubner, 1987.

[56] Le Yang et al. "Practical Traceable Over-Threshold Multi-Party Private Set Intersection". In: *arXiv preprint arXiv:2512.24652* (2025).



Source Code Example

This appendix shows selected source-code fragments from the batched OpenFHE implementation. The snippets illustrate the main stages of the implementation: Bloom filter construction, conversion to batched OpenFHE inputs, packed BFV encryption, and encrypted threshold-circuit evaluation.

A.1. Bloom Filter Construction

```
def build_bloom_filter(self, party_set: list) -> list:
    """Build and unpack a party Bloom filter into a flat bit vector."""
    bloom_filter = BloomFilter(
        est_elements=self.party_set_size,
        false_positive_rate=self.false_positive_rate,
    )

    for ip in party_set:
        bloom_filter.add(ip)

    result_bits = []
    for chunk in bloom_filter.bloom:
        for bit_pos in range(self._bits_per_chunk):
            result_bits.append((chunk >> bit_pos) & 1)

    return result_bits[: self.num_bloom_bits]
```

A.2. Preparing Bloom Filters for Batched OpenFHE Evaluation

```
def prepare_bloom_filter_for_openfhe(self, bloom_filter_bits: list) -> np.ndarray:
    """Convert Bloom bits to int64; OpenFHE encryption happens during evaluation."""
    return np.array(bloom_filter_bits, dtype=np.int64)


def prepare_bloom_filters_for_openfhe(self, bloom_filters: list) -> list:
    """Prepare all party Bloom filters for packed OpenFHE evaluation."""
    workers = self._get_party_preprocess_workers()

    if workers <= 1 or len(bloom_filters) <= 1:
        return [
            self.prepare_bloom_filter_for_openfhe(bits)
            for bits in bloom_filters
        ]

    with ProcessPoolExecutor(max_workers=min(workers, len(bloom_filters))) as executor:
        return list(executor.map(_prepare_bloom_filter_bits_worker, bloom_filters))
```

A.3. Packed BFV Encryption of Bloom Filter Chunks

```
def encrypt_bloom_filter_chunk(
```

```
    self,
    bloom_filter_bits,
    context: dict,
    batch_lanes: int,
    start: int,
    bit_count: int,
):
    """Encrypt␣one␣packed␣chunk␣of␣a␣party␣Bloom␣filter."""
    cc = context["cc"]
    public_key = context["public_key"]

    stop = min(start + batch_lanes, bit_count)
    span = stop - start

    packed_values = [0] * batch_lanes
    packed_values[:span] = [
        int(value) for value in bloom_filter_bits[start:stop]
    ]

    plaintext = cc.MakePackedPlaintext(packed_values)
    return cc.Encrypt(public_key, plaintext)


def encrypt_bloom_filter(
    self,
    bloom_filter_bits,
    context: dict,
    batch_lanes: int,
    bit_count: int,
) -> list:
    """Encrypt␣one␣Bloom␣filter␣into␣packed␣BFV␣ciphertext␣chunks."""
    encrypted_chunks = []

    for start in range(0, bit_count, batch_lanes):
        encrypted_chunks.append(
            self.encrypt_bloom_filter_chunk(
                bloom_filter_bits,
                context,
                batch_lanes,
                start,
                bit_count,
            )
        )

    return encrypted_chunks
```

A.4. Encrypted Threshold-Circuit Evaluation

```
def _build_packed_threshold_evaluator(self, parsed_terms: list):
    """Create␣a␣slotwise␣OpenFHE␣evaluator␣for␣the␣threshold␣SOP␣circuit."""

    def evaluate_threshold(cc, encrypted_party_bits):
        term_values = []

        for term in parsed_terms:
            term_value = encrypted_party_bits[term[0]]

            for idx in term[1:]:
                term_value = cc.EvalMult(term_value, encrypted_party_bits[idx])

            term_values.append(term_value)

        if not term_values:
            return encrypted_party_bits[0]

        # OR over binary slots: a OR b = a + b - ab.
        while len(term_values) > 1:
            next_level = []

            for left_idx in range(0, len(term_values), 2):
```

```
23                if left_idx + 1 >= len(term_values):
24                    next_level.append(term_values[left_idx])
25                    continue
26
27                left = term_values[left_idx]
28                right = term_values[left_idx + 1]
29                next_level.append(
30                    cc.EvalSub(
31                        cc.EvalAdd(left, right),
32                        cc.EvalMult(left, right),
33                    )
34                )
35
36            term_values = next_level
37
38        return term_values[0]
39
40    return evaluate_threshold
41
42
43 def evaluate_encrypted_circuit(self, circuit: str, encrypted_bloom_filters: list) -> list:
44    """Pack␣Bloom␣positions␣into␣BFV␣slots␣and␣evaluate␣the␣threshold␣circuit."""
45    bit_count = len(encrypted_bloom_filters[0])
46
47    batch_lane_candidates = self._resolve_batch_lane_candidates(bit_count)
48    context = None
49    batch_lanes = None
50
51    for lanes in batch_lane_candidates:
52        self._ensure_openfhe_context(circuit, lanes)
53        context = self._openfhe_contexts[(circuit, lanes)]
54        batch_lanes = lanes
55        break
56
57    self._active_openfhe_context = context
58    self._active_encrypted_bit_count = bit_count
59    self._active_batch_lanes = batch_lanes
60
61    cc = context["cc"]
62    parsed_terms = context["parsed_terms"]
63    evaluate_threshold = self._build_packed_threshold_evaluator(parsed_terms)
64
65    chunk_count = (bit_count + batch_lanes - 1) // batch_lanes
66    encrypted_result_chunks = []
67
68    for chunk_idx in range(chunk_count):
69        chunk_start = chunk_idx * batch_lanes
70
71        encrypted_party_bits = [
72            self.encrypt_bloom_filter_chunk(
73                encrypted_bloom_filters[party_idx],
74                context,
75                batch_lanes,
76                chunk_start,
77                bit_count,
78            )
79            for party_idx in range(self.num_parties)
80        ]
81
82        encrypted_result_chunk = evaluate_threshold(cc, encrypted_party_bits)
83        encrypted_result_chunks.append(encrypted_result_chunk)
84
85    return encrypted_result_chunks
```