mathematics

MDPI

Article

Blockchain-Based Unbalanced PSI with Public Verification and Financial Security

Zhanshan Wang and Xiaofeng Ma *

Department of Control Science and Engineering, Tongji University, Shanghai 201804, China; 2211168@tongji.edu.cn
* Correspondence: xiaofengma@tongji.edu.cn

Abstract: Private set intersection (PSI) enables two parties to determine the intersection of their respective datasets without revealing any information beyond the intersection itself. This paper particularly focuses on the scenario of unbalanced PSI, where the sizes of datasets possessed by the parties can significantly differ. Current protocols for unbalanced PSI under the malicious security model exhibit low efficiency, rendering them impractical in real-world applications. By contrast, most efficient unbalanced PSI protocols fail to guarantee the correctness of the intersection against a malicious server and cannot even ensure the client's privacy. The present study proposes a blockchain-based unbalanced PSI protocol with public verification and financial security that enables the client to detect malicious behavior from the server (if any) and then generate an irrefutable and publicly verifiable proof without compromising its secret. The proof can be verified through smart contracts, and some economic incentive and penalty measures are executed automatically to achieve financial security. Furthermore, we implement the proposed protocol, and experimental results demonstrate that our scheme exhibits low online communication complexity and computational overhead for the client. At the same time, the size of the generated proof and its verification complexity are both $\mathcal{O}(logn)$, enabling cost-effective validation on the blockchain.

Keywords: blockchain; private set intersection; smart contract; RSA blind signature; public verification

MSC: 94A60

check for updates

Citation: Wang, Z.; Ma, X. Blockchain-Based Unbalanced PSI with Public Verification and Financial Security. *Mathematics* **2024**, *12*, 1544. https://doi.org/10.3390/math12101544

Academic Editor: Maurizio Naldi

Received: 4 April 2024
Revised: 8 May 2024
Accepted: 13 May 2024
Published: 15 May 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Private set intersection (PSI) can be regarded as a special case within secure multi-party computation (SMPC) wherein two parties each hold a set of private data and desire to compute the intersection of these sets without disclosing any information outside the intersection to each other. PSI has been widely adopted across various real-world applications, including private contact discovery [1], private location-based services in the Internet of Vehicles [2], privacy-aware social network relationship inference [3], and privacy-protected password checks [4].

PSI protocols can be broadly classified into two distinct categories, which are differentiated by the relative sizes of the datasets held by the participating parties. In scenarios where both parties have datasets of roughly equivalent magnitude, this configuration is termed as 'balanced'. Conversely, when there is a pronounced discrepancy in dataset sizes, with one party's dataset being considerably smaller than the other's, the scenario is characterized as 'unbalanced'. This paper mainly focuses on unbalanced PSI, where the client—typically the party with the smaller dataset—often operates with constrained device resources, including limitations in storage capacity and computational power when juxtaposed with the server, which conventionally holds a larger dataset. Additionally, the inter-party communication may be subject to bandwidth constraints, further complicating the PSI process.

While some PSI protocols in the literature [5–11] extended from an oblivious transfer (OT) extension [12] and oblivious key-value storage (OKVS) structure [11] have achieved high computation efficiency, these typically involve substantial data transmission between a server and client and might necessitate multiple rounds of communication, which is suboptimal in unbalanced settings characterized by bandwidth constraints. The protocol of Jarecki and Liu [13] has better communication efficiency but often incurs considerable computational workload, which is particularly challenging for clients with weak devices. In contrast, most of the efficient unbalanced PSI protocols in the literature [1,14–18] are fully simulatable only in the model of semi-honest (also known as passive) adversaries or only achieve security in the presence of malicious (also known as active) adversaries with one-sided simulatability [19]. The former, where the parties are required to follow the protocol, is unrealistic. In the latter case, although all parties' privacy is preserved, the correctness of the result cannot be guaranteed, which may cause losses to the parties receiving the intersection.

To get a trade-off between security and efficiency, several publicly verifiable covert (PVC) protocols have been proposed [20,21]. PVC protocols not only offer a heightened level of efficiency when compared to those designed for the malicious setting, but they also introduce a robust layer of security. They are capable of detecting an adversary's malicious behavior with a certain degree of probability, which is a significant enhancement over the semi-honest model. Furthermore, these protocols have the added advantage of generating proofs that are publicly verifiable. This feature serves as a powerful deterrent against rational adversaries as it introduces the risk of their malicious actions being exposed, thereby potentially dissuading them from engaging in such behavior in practical applications. However, current PVC protocols are designed mainly for SMPC tasks rather than PSI scenarios and exhibit low efficiency when directly applied to PSI, particularly in unbalanced settings. Additionally, a trusted third party is typically required to verify generated proofs in PVC protocols, which can entail considerable judicial costs. Although Zhu et al. [22] eliminated the need for a third party by leveraging smart contracts for proof verification and to achieve financial security, the large size and high verification complexity of proofs make validations on smart contracts costly in practice.

In this work, we propose a blockchain-based unbalanced PSI protocol that leverages the immutable nature of blockchain technology to enable public verification of the result and achieves financial security through smart contracts. In addition, our protocol can maintain high efficiency even under conditions of bandwidth constraints and weaker client-side device capabilities. Unlike conventional general-purpose PVC protocols, our design has lower size and verification complexities of proofs, which enables proofs to be verified on smart contracts at a low cost. To the best of our knowledge, our protocol is the first to introduce blockchain and smart contracts into the unbalanced PSI to obtain public verification and financial security against a malicious server. More specifically, the primary contributions of this work are as follows:

- On-chain anchoring of key data and generation of publicly verifiable proofs—in case of disputes over the final intersection, the client can readily generate publicly verifiable proofs based on tamper-proof Merkle roots of key data on the blockchain to accuse the server of dishonest behavior, thereby exerting a deterrent effect on the server.
- Smart contract-based automatic verification and financial security—proofs are validated over smart contracts, and corresponding economic rewards or penalties are automatically executed to achieve financial security based on the verification result. Meanwhile, all the verification results about a server in history have been permanently recorded on the blockchain and can be publicly accessed by anyone, which further enhances the deterrent effect on the server.
- Integration of Cuckoo filters—clients' storage overhead and final query time are reduced, and Cuckoo filters have inherent support for deletion, facilitating dynamic updates of data stored on the client to avoid redundant transmissions.

- Implementation and experimentation—the proposed unbalanced PSI protocol and associated smart contracts are realized, and experimental results demonstrate linear dependence of online communication based on clients' dataset size. Moreover, the transaction cost of executing verification on smart contracts is very low.

The paper is structured as follows: Section 2 presents an overview of the related work. Section 3 defines the required notation and introduces the fundamental concepts. Section 4 begins with the presentation of the basic protocol and details the enhancements we have implemented to improve both its efficiency and security. This is followed by a thorough description of the full protocol and a theoretical analysis. In Section 5, we present and analyze our experimental findings. Finally, Section 6 concludes, provides an insightful analysis of the challenges, and proposes potential avenues for future research.

2. Related Work

Freedman et al. [23] first formally defined PSI and introduced a PSI protocol based on oblivious polynomial evaluation. Since then, many efficient PSI protocols have been proposed, among which protocols based on OT extension present the most competitive performance [24]. Pinkas et al. [5] proposed the first OT-based PSI protocol under the semi-honest model, followed by a series of improvements in works [6–9]. In these protocols, parties first hash elements to a data structure and then evaluate an oblivious pseudorandom function (OPRF) for each bin through OT extension. Pinkas et al. [25] proposed a malicious PSI protocol by combining a data structure called PaXoS with the actively secure OOS protocol [7]. However, one inherent property of OT-based PSI is that communication is linear with the size of the larger set and can require multiple rounds of interactions, which is not suitable for unbalanced PSI settings: especially scenarios with limited bandwidth. Extended from the PaXoS structure in [25], the OKVS-based PSI protocol in [10,11] is also inherently encumbered by a significant communication overhead. This intrinsic limitation impedes the protocol's scalability and efficiency in unbalanced PSI settings.

With the advent of cloud computing, PSI protocols relying on a third party have been introduced in the literature [26,27]. In these protocols, participants encode their private data through a random function and send the encoded values to a third party, who computes the intersection and returns it to each party. Although these protocols are highly user-friendly and do not require all parties to be online simultaneously, their security largely depends on the trustworthiness of the third party.

As a specific case of SMPC, PSI can also be achieved by employing generic SMPC. Huang et al. [28] proposed the first PSI protocol based on garbled circuits, which was improved in work [29]. The advantage of such protocols lies in the ability to perform privacy-preserving computations on the obtained intersection, such as calculating the cardinality of the intersection or computing the sum of all elements in the intersection. Nonetheless, these methods require greater communication than OT-based PSI protocols, and clients may consume substantial memory when evaluating circuits. Therefore, PSI based on garbled circuits does not present significant advantages in scenarios where client devices have limited capabilities and no subsequent computations on the intersection are required.

At present, PSI protocols with high communication efficiency are mainly based on public key and homomorphic encryption. Meadow [30] and Huberman et al. [31] constructed PSI protocols using the Diffie–Hellman (DH) key exchange before PSI was formally defined in work [23]. Resende et al. [14] then reduced communication and improved computation efficiency by applying Cuckoo filters and using elliptic curve groups instead of prime-order groups. Nevertheless, the number of public key operations required by clients during the online phase is still linear with its set size. More importantly, zero-knowledge proof is usually required when extending these protocols to malicious security [13], which further increases clients' workload. Cristofaro et al. [15] proposed a PSI protocol based on RSA blind signatures, where complex public key operations on clients can be completed

offline independently of servers. However, this protocol similarly remains vulnerable to malicious servers.

Chen et al. [1] introduced a leveled fully homomorphic encryption (FHE) scheme into PSI by employing a series of optimization techniques, including batching, partitioning, and hashing. In this protocol, the communication is solely dependent on the set size of clients. Therefore, only a minimal amount of data need to be transmitted. Chen et al. [16] later extended this protocol to be secure against a malicious client by incorporating an OPRF preprocessing phase before FHE. However, their method cannot guarantee that the final intersection is correct. To address verification issues, Jiang et al. [32] introduced a homomorphic hash function to ensure output correctness. Due to the extensive use of pairing operations during the verification process, the client is burdened with a significant computation workload.

To the best of our knowledge, the protocol based on a hash proof system presented in [17] is currently the most communication-efficient unbalanced PSI protocol. However, the security of this protocol is only guaranteed under the semi-honest model. Similarly, the unbalanced PSI protocol proposed within [18], despite its commendable reduction in communication cost, presents a vulnerability in scenarios wherein the client operates with malicious intent.

On another front, Aumann et al. [33] introduced the notion of covert adversaries, who are allowed to behave maliciously but face a certain probability (deterrence factor) of being caught by the other party. They show that SMPC protocols designed against covert adversaries can achieve better efficiency than those designed against malicious adversaries. At the same time, this security model is meaningful in many real-world scenarios, such as business, finance, and politics, where entities might have an incentive to cheat yet cannot afford the loss of reputation or negative publicity from being caught cheating.

While the protocol of Aumann et al. [33] can ensure catching cheaters with a certain probability, it encounters significant challenges in persuading a third party (e.g., a court) to lend credence to such allegations. To address this issue, Asharov et al. [20] proposed publicly verifiable covert (PVC) security to enable the honest party to generate publicly verifiable proofs upon catching cheating, which can be verified by any third party without revealing the honest party's private information. Nonetheless, there exists the potential for collusion between one party and the designated third party responsible for verifying the proof. Zhu et al. [22] combined PVC with smart contracts, proposing a new notion called financial security. Although they optimized the size of proofs and verification algorithms, gas costs remain too high to be practical. Furthermore, the majority of existing PVC protocols are implemented based on garbled circuits. Although they can indeed be adapted for PSI, they suffer from the same drawbacks as PSI protocols based on garbled circuits [28,29] when it comes to unbalanced PSI scenarios, as discussed earlier.

3. Preliminaries

In this section, we formalize some notations and basic definitions to be used in this paper.

3.1. Summary of Notations

- $X, Y \subseteq \{0,1\}^\sigma$ are the server's and client's input sets, with sizes $v = |X|$ and $w = |Y|$, respectively.
- n_1 is the number of elements sampled and validated by the client.
- $r \leftarrow\$ S$ indicates r was sampled from S with uniform distribution.
- κ, λ are the computational and statistical security parameters, respectively.
- e, d, n denote an RSA key pair, where (e, n) constitutes the public key, and (d, n) constitutes the private key. It is required that $e > 1$, $gcd(e, \varphi(n)) = 1$, and $ed \equiv 1 \mod n$, where $\varphi(n)$ represents Euler's totient function of n.
- $H_1 : \{0,1\}^\sigma \to \mathbb{Z}_n^*, H_2 : \mathbb{Z}_n^* \to \{0,1\}^l$ are hash functions modeled as random oracles, where l is the length of H_2's output.

- M_1, M_2 correspond to Merkle roots that are separately uploaded to the blockchain by the client and the server, respectively.
- t_1, t_2 denote, respectively, deadlines for the server to upload M_2 and the client to submit the proof, subject to the condition that $t_1 < t_2$.

3.2. Blockchain and Smart Contract

The core concept of a blockchain was initially introduced by Satoshi Nakamoto [34]; it represents a technology solution that enables data storage, validation, and transmission without relying on a third party. It functions as a decentralized distributed ledger, embodying the principles of data integrity and trust. In a blockchain network, each node possesses equal status and rights, contributing to data consistency through a distributed network architecture and achieving consensus among nodes. This inherent decentralization is achieved by linking each block in the blockchain with the hash value of the previous block (except for the initial genesis block) [35]. Any malicious attempt to modify a transaction would require altering all subsequent blocks to gain acceptance from other nodes. The consensus mechanisms employed in blockchain systems, such as proof of work or proof of stake, impose significant costs on such manipulation attempts, thereby ensuring the immutability of the recorded data on the blockchain.

Currently, blockchains have evolved from basic distributed ledger databases into robust and reliable platforms. Ethereum, building upon the foundation laid by Bitcoin, introduced the concept of smart contracts. These smart contracts are executable through the Ethereum Virtual Machine (EVM), which supports Turing-complete computations. Unlike conventional programs, the execution outcome of a smart contract undergoes validation and requires consensus among all nodes before it is stored on the blockchain. By leveraging smart contracts, the traditional process of transaction endorsements can be transformed from manual legal agreements to automated code. Once the contract conditions are met, the contract's terms are automatically enforced, reducing administrative costs within conventional contract execution.

Although the Ethereum Virtual Machine (EVM) theoretically supports the execution of highly complex smart contracts, its stack-based RISC architecture inherently limits execution efficiency. Additionally, as mentioned earlier, each node must redundantly store the source code of smart contracts and execute computations to achieve consensus. Consequently, overly intricate smart contracts can impose a substantial burden on the Ethereum network. To address this issue, Ethereum implements strict pricing mechanisms for instructions and storage. For example, a single multiplication instruction consumes 5 gas, while executing an exponentiation instruction requires $10 + 50 * len_{exp}$ (where len_{exp} represents the number of bytes occupied by the exponent on the stack). The transaction fee is determined by the cumulative operations involved, meaning that the more data need to be stored or updated on the blockchain and the higher the computation complexity, the more expensive the transaction fee becomes. Therefore, in practical applications, it is crucial to minimize the complexity of smart contracts to avoid incurring substantial transaction costs for users and potential failed invocations of smart contracts due to Ethereum's block gas limit (currently set at 30 million gas per block).

3.3. RSA Blind Signature

In the standard signing process, signers are privy to the original message being signed. To safeguard user privacy, David Chaum first introduced the concept of blind signatures [36]. Blind signatures can be viewed as a distinctive variant of digital signatures for which the signer can endorse the original message without actually knowing its specific content. Presently, blind signatures are commonly employed in domains such as electronic cash and electronic voting to guarantee anonymity.

Definition 1 (Blind Signature Scheme)**.** *A blind signature scheme consists of the following probabilistic polynomial-time (PPT) algorithms:*

- The key-generation algorithm $KeyGen(1^\kappa)$ takes as input a security parameter 1^κ and outputs a pair of keys (sk, pk).
- The blind signing algorithm $< User(pk, m), Signer(sk) >$ is an interactive protocol between *User* and *Signer*. *User* is given a message m and a public key pk, and *Signer* is given a secret key sk. At the end of this protocol, *User* outputs either σ, a signature on m, or $\perp$ if the interaction is not successful.
- The verification algorithm $Verify(pk, m, \sigma)$ is deterministic and takes as input a public key pk, a message m, and a signature σ. It outputs 1 if σ is valid on m under pk and 0 otherwise.

We require that for every κ, every (sk, pk) output by $KeyGen(1^\kappa)$, and every message m in the appropriate underlying plaintext space, it holds that

$$Verify(pk, m, < User(pk, m), Signer(sk) >) = 1$$

Regarding security, a blind signature scheme must satisfy blindness and unforgeability. Let $BS = (KeyGen(1^\kappa), < User(pk, m), Signer(sk) >, Verify(pk, m, \sigma)$ be a blind signature scheme and $\mathcal{A}$ be an adversary. We consider the following two experiments:

The experiment $Blind_{\mathcal{A},BS}(\kappa)$:

1. $b \leftarrow_\$ \{0, 1\}, (pk, m_0, m_1, \text{st}) \leftarrow \mathcal{A}(1^k)$.
2. $\text{st} \leftarrow \mathcal{A}^{(User(pk,m_b),\cdot),(User(pk,m_{1-b}),\cdot)}(\text{st}), \sigma_b \leftarrow User(pk, m_b), \sigma_{1-b} \leftarrow User(pk, m_{1-b})$. If $\sigma_0 = \perp$ or $\sigma_1 = \perp$, then let $(\sigma_0, \sigma_1) = (\perp, \perp)$.
3. $b^* \leftarrow \mathcal{A}(\text{st}, \sigma_0, \sigma_1)$.

Definition 2 (Blindness)**.** *A blind signature scheme BS is blind if for all PPT adversaries $\mathcal{A}$ with one-time access to two User oracles, there exists a negligible function negl such that*

$$\Pr(b^* = b) - \frac{1}{2} < negl(\kappa)$$

The experiment $Forge_{\mathcal{A},BS}(\kappa)$:

1. $(pk, sk) \leftarrow KeyGen(1^\kappa)$.
2. $(m_i, \sigma_i)_{i=1}^{k+1} \leftarrow \mathcal{A}^{(S(sk),\cdot)}(pk)$.
3. Let event *Success* be: $m_i \neq m_j \wedge Verify(m_i, \sigma_i, pk) = 1, \forall i, j \in [k+1], i \neq j$.

Definition 3 (Unforgeability)**.** *A blind signature scheme BS is unforgeable if for all PPT adversaries $\mathcal{A}$ with access to a Signer oracle, there exists a negligible function negl such that*

$$\Pr(Success) < negl(\kappa)$$

4. Protocol

We start with a succinct presentation of the basic protocol upon which our work relies, followed by the optimizations implemented. Subsequently, we outline the complete procedure of our protocol and proceed to conduct a theoretical analysis of its security and efficiency.

4.1. The Basic Protocol

Cristofaro et al. [37] proposed a secure PSI protocol under the one-more-RSA assumption [37]. This protocol is depicted in Figure 1 and works as follows: for each element $x_i \in X$, the server utilizes its private key (d, n) to sign the hash value of every element to obtain the signed value $sx_i = H_1(x_i)^d \mod n$. To prevent the client from reconstructing $H_1(x_i)$ from the signature, the server further applies a secondary hash function H_2 to the signed values to get $hx_i = H_2(sx_i)$. The client initiates by generating a random number r_j for each element $y_j \in Y$, which is employed to blind the original hash value $H_1(y_j)$ to get $b_j = H_1(y_j) \cdot r_j^e \mod n$. The blinded value b_j is then transmitted to the server.

Subsequently, the server signs each blinded value to derive $sb_j = b_j^d \mod n$ and sends both sb_j and hx_i back to the client. Upon receiving the server's signatures on the blinded values, the client first deblinds the signatures to recover the signature on the original element's hash value as $sy_j = sb_j / r_j \mod n$. Finally, the intersection between both sets can be determined by comparing all hx_i against hashed versions of the recovered signatures, i.e., $hy_j = H_2(sy_j)$.

Server		**Client**
..... Offline		
for $i \in [1, v]$ **do**		**for** $j \in [1, w]$ **do**
$sx_i = H_1(x_i)^d \mod n$		$r_j \leftarrow\!\!\$\ \mathbb{Z}_n^*$
$hx_i = H_2(sx_i)$		$b_j = H_1(y_j) * {r_j}^e \mod n$
..... Online		
	$\xleftarrow{B = \{b_1, b_2, \ldots, b_w\}}$	
for $j \in [1, w]$ **do**		
$sb_j = {b_j}^d \mod n$		
	$\xrightarrow[HX = \{hx_1, hx_2, \ldots, hx_v\}]{SB = \{sb_1, sb_2, \ldots, sb_w\}}$	
		$I = \emptyset$
		for $j \in [1, w]$ **do**
		$sy_j = sb_j / r_j \mod n$
		$hy_j = H_2(sy_j)$
		if $hy_j \in HX$ **then**
		add y_j to I
		output I

Figure 1. Basic protocol proposed in [37].

The correctness of the protocol is obvious. The signature associated with the client's element y_j can be derived by removing the corresponding random value r_j, as demonstrated by Equation (1).

$$sy_j = \frac{sb_j}{r_j} \mod n = \frac{(H_1(y_j) * {r_j}^e)^d}{r_j} \mod n = H_1(y_j)^d \mod n \quad (1)$$

Suppose there exist two elements $x_i^* \in X$ and $y_i^* \in Y$ such that $x_i^* = y_i^*$; the following Equation (2) holds true. Therefore, the elements in the intersection of sets $\{hx_1, hx_2, ..., hx_v\}$ and $\{hy_1, hy_2, ..., hy_w\}$ can be mapped back to the common elements in the intersection of sets X and Y.

$$hx_i^* = H_2(sx_i^*) = H_2\left(H_1(x_i^*)^d \mod n\right) = H_2\left(H_1(y_j^*)^d \mod n\right) = H_2(sy_j^*) = hy_j^* \quad (2)$$

In terms of security, this protocol is only secure against a malicious client. Informally, since the client only has access to some hash values of the server's signatures, the only way for the client to obtain the server's input is by first brute-forcing the value of sx_i from hx_i and then utilizing the public key (e, n) to compute $H_1(x_i) = (sx_i)^e \mod n$. At last, the client can retrieve the server's private input x_i by enumerating possible input domains of $H_1(x_i)$. However, given that sx_i ranges over $\mathbb{Z}_n^*$, the probability of the client brute-forcing sx_i from hx_i can be considered computationally negligible as long as the hash function H_2 is cryptographically secure. Regarding the client's privacy, since the data it transmits to the server consists solely of b_j, which contains a random number, the client's privacy is guaranteed statistically. A formal proof of the security properties can be found in reference [15].

4.2. Optimizations

In this section, we present optimizations for the basic protocol in Section 4.1 by incorporating Cuckoo filters to reduce the storage at the client side and leveraging the immutability of the blockchain to enhance the security against a malicious server.

4.2.1. Reduce Storage

In the basic protocol, the client is required to store the entire set HX for comparison. As a result, the storage space required increases with the size of the server's set. In the case of unbalanced PSI, the server's set is typically larger, which imposes significant storage overhead on the client. According to the birthday paradox, the approximate probability of experiencing a collision when mapping $(v + w)$ elements to a domain of size 2^l is $(v + w)^2/2^l$. Therefore, if the probability of a hash collision occurring is required to be no more than $2^{-\lambda}$, the output length of H_2 should be at least $l = 2\log_2(v + w) + \lambda - 1$. Assuming $\lambda = 40$, $v = 2^{30}$, and $w = 2^8$, approximately 11.25 GB of space is required to store set HX, which is highly prohibitive for clients with limited storage resources.

We introduce Cuckoo filters [38] to reduce the client's storage. A Cuckoo filter can be seen as a compact variant of Cuckoo hashing [39], but rather than storing a complete element, each entry holds the fingerprint of the element. The fingerprint generally refers to a segment of the bit string derived from hashing the original element. When inserting a new element x, two candidate bucket locations $i_1 = hash(x)$ and $i_2 = i_1 \oplus hash(f)$ are computed, where f represents the fingerprint of f. If one of these buckets has an empty entry, the fingerprint f is inserted into that vacant slot. Otherwise, one of the existing fingerprints f' in the ith bucket is replaced by the birthday paradox, where $i \leftarrow\$ \{i_1, i_2\}$. The displaced fingerprint f' is moved to a bucket indexed by $i' = i \oplus hash(f')$. If no empty entry is found within a threshold number of attempts, insertion fails. Experiments by Fan et al. [38] show that with bucket sizes of four and fingerprints that are six bits or longer, Cuckoo filters can achieve a load factor of 95% and accommodate up to 4 billion elements. In summary, Cuckoo filters manage to maintain high occupancy rates while storing smaller fingerprints, thereby resulting in low storage per element on average.

Following the fundamental configuration of Fan et al. [38], where each element has two candidate buckets and the bucket size is four, the average space occupied per element at maximum load is:

$$C \leq 1.05 \times (3 - \log_2 \epsilon) \tag{3}$$

where ϵ is the target false positive rate. Cuckoo filters can reduce the size of set HX from 11.25 GB to approximately 1.75 GB when ϵ is set to 0.001, thus substantially decreasing the storage demanded by the client.

Another significant advantage of the Cuckoo filter is its convenient lookup capability. When a client wishes to test whether a signature value hy_j is present in the set HX, it needs only check for the fingerprint of hy_j in the two candidate buckets. Therefore, the Cuckoo filter results in constant-time lookup complexity, which improves the client's computation efficiency at the same time.

Apart from supporting general insertion and lookup operations, Cuckoo filters allow for dynamic deletion. The deletion process is straightforward: locate the bucket containing the element required to be deleted through the lookup algorithm and remove the corresponding fingerprint from the bucket. Hence, the deletion complexity is also $\mathcal{O}(1)$. Leveraging this property, when the server's data undergo only minor changes, the client does not need to re-download the entire dataset. Instead, the server can send the updated data along with the corresponding instructions (whether to add or remove) to the client. The client can then execute the appropriate operation based on the instruction to obtain an updated filter reflecting the server's new set.

4.2.2. Improve Security

Although the server cannot obtain the client's private input in the basic protocol, it can cause the client's output to deviate from the correct result. For instance, during the signing on the client's blinded value b_i, the server might sign an arbitrary random value rather than the actual b_i sent by the client. The server could even economize its computational resources by simply responding with a random value instead of utilizing its private key for signing.

We organize critical data in the intersection process into a Merkle tree structure and upload the corresponding Merkle root to the blockchain for anchoring. This enables the client to produce a publicly verifiable and non-repudiable proof.

Specifically, after obtaining set $B = \{b_1, b_2, \ldots, b_w\}$, the client first computes the Merkle root M_1 of B. Next, the client sends each item in B according to its position in the Merkle tree to the server orderly and uploads M_1 to the blockchain. Upon receiving B and confirming that M_1 has been recorded on the blockchain, the server computes the Merkle root of B and verifies whether it is equal to M_1. If there is a match, the server proceeds with the subsequent signing; otherwise, it aborts the protocol. Given the inherent properties of the Merkle tree structure, any difference between the data received by the server and set B will result in different Merkle roots in the end.

In the same way, the server computes the Merkle root M_2 of set $S_B = \{sb_1, sb_2, \ldots, sb_w\}$ in the same order as B. It then sends the ordered set S_B to the client and uploads M_2 to the blockchain. Upon receipt of S_B, the client initially verifies whether the Merkle root of the set received equals M_2. If not, the client terminates the protocol; otherwise, the client samples a proportion of data from S_B based on its expected deterrence effect. Let $sb_t \in S_B$ denote one of the sampled elements: the client needs to verify whether Equation (4) holds true.

$$b_t = (sb_t)^e \mod n \tag{4}$$

If all sampled elements satisfy Equation (4), the result is considered correct. However, if there exists any element $sb_t \in S_B$ such that $b_t \neq (sb_t)^e \mod n$, it can be inferred that the server did not faithfully execute the protocol. In this case, the client can generate a publicly verifiable proof to accuse the server of malicious behavior during the intersection process. The proof consists of the following components:

- The elements $b_t \in B$ and $sb_t \in S_B$ that do not satisfy Equation (4);
- The index $t \in [1, w]$ of b_t in set B (or sb_t in set SB);
- The paths $road_{b_t}$ and $road_{sb_t}$ for b_t in the Merkle trees with roots M_1 and M_2, respectively.

The verification algorithm for $Proof = \{t, b_t, sb_t, road_{b_t}, road_{sb_t}\}$ consists of the following three steps.

1. Verify whether the element b_t is located at index t in the Merkle tree with root M_1 according to $road_{b_t}$. If the verification succeeds, proceed to the next step; otherwise, return False.
2. Verify whether the element sb_t is located at index t in the Merkle tree with root M_2 according to $road_{sb_t}$. If the verification succeeds, proceed to the next step; otherwise, return False.

3. Verify whether the equality $b_t = (sb_t)^e \mod n$ holds using the server's public key (e, n). If it does, return False; otherwise, return True.

A validation result of True indicates that the server did not adhere to the protocol's execution, whereas a result of False implies that the proof is invalid.

4.3. Full Protocol

In this section, we present our full protocol, which is based on the basic protocol with the optimizations proposed in Section 4.2. The overview of the system architecture and interaction logic is depicted in Figure 2.

The blockchain-based PSI system consists of four types of entities: server, client, blockchain, and IPFS (optional). The server and client are parties holding data. The blockchain is used to record the Merkle root, verify proofs submitted by the client, and execute corresponding economic measures according to verification results. IPFS is optional and is where the server uploads its encrypted data. In addition to IPFS, any cloud can be used to store the server's encrypted set. The client can also download data directly from the server when bandwidth permits. For the sake of simplicity, we assume that the server directly transmits its encrypted data to the client in the following protocol description.

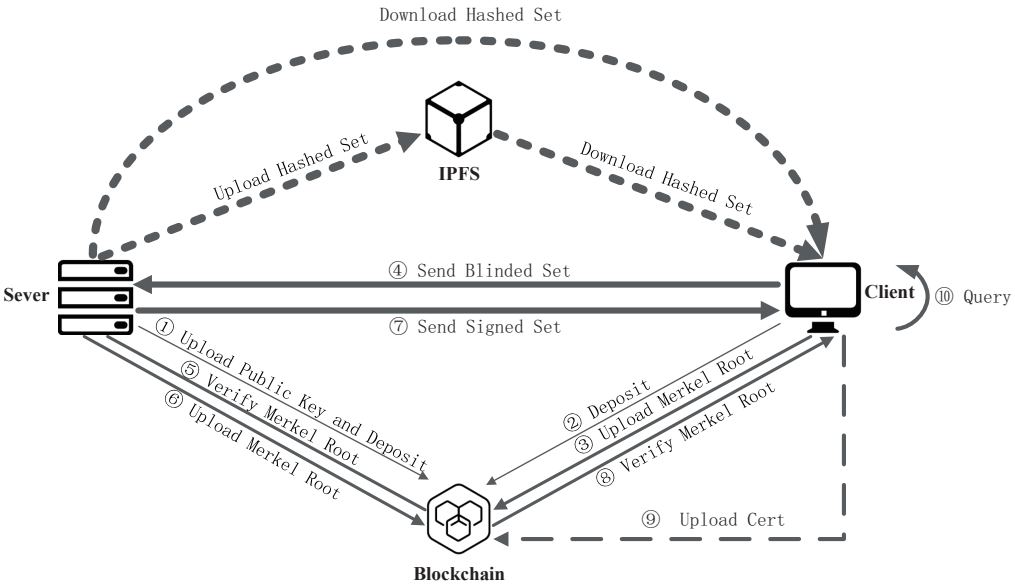

Figure 2. Overall system architecture diagram.

The description of our full protocol is illustrated in Figure 3 and consists of four main processes as follows:

1. **Setup** is used to perform some pre-processes and prepare for subsequent calculations.
   (a) The server publishes the public key (e, n) on the blockchain and stakes the required deposit.
   (b) For each $x_i \in X$, the server computes $sx_i = H_1(x_i)^d \mod n$ and $hx_i = H_2(s_{xi})$. It then generates a Cuckoo filter GF_X that inserts set $HX = \{hx_1, hx_2, \ldots, hx_v\}$.
   (c) The client generates w or more random numbers and encrypts them using the server's public key to obtain $R_i = r_j^e \mod n$.
   (d) The client pledges a certain amount of deposit to the smart contract under the contract requirements. Once the deposit meets the requirements, the blockchain emits a request event to the server.
   (e) Upon listening to the request event, the server can authorize the client to communicate with it.

2. **Computing** is the main process, whereby the client obtains blind signatures on its set through a single interaction with the server.
   (a) The client blinds set Y to obtain set $B = \{b_1, b_2, \ldots, b_w\}$, where $b_j = H_1(y_j) \cdot R_j \mod n$.
   (b) The client uploads the Merkle root M_1 of B to the blockchain and sends B to the server.
   (c) The server verifies whether the Merkle root of received data is consistent M_1 on the blockchain, and it exits the protocol if they do not match.
   (d) The server signs every element b_j in B to generate set $SB = \{sb_1, sb_2, \ldots, sb_w\}$ using the private key (d, n), where $sb_j = (b_j)^d \mod n$. It then uploads the Merkle root M_2 of SB to the blockchain and sends set SB and the Cuckoo filter GF_X to the client.
   (e) The client verifies whether the Merkle root of received data is consistent M_2 on the blockchain, and it exits the protocol if they do not match.
   (f) If the client fails to upload M_1 or the server fails to upload M_2 to the blockchain before time t_1, both parties' deposits will be unlocked and the protocol will be terminated.
3. **Verifying** aims to verify whether the blind signatures received from the server are valid.
   (a) The client randomly samples n_1 elements from SB, where the number of samples depends on the expected deterrence factor. It then checks whether the sample point b_t satisfies $b_t = (sb_t)^e \mod n$. If all samples satisfy this equation, it proceeds to the output phase; otherwise, it generates a proof $Proof = (t, b_t, sb_t, road_{b_t}, road_{sb_t})$.
   (b) The client submits $Proof$ to the blockchain and invokes the verification function within the smart contract to verify whether $Proof$ is valid. If the proof is valid, the smart contract automatically deducts the server's deposit as a penalty and refunds the client's deposit. Then, the protocol ends.
   (c) If the smart contract does not receive any valid proofs prior to time t_2, the server's deposit will be refunded. A portion of the client's deposit may be transferred to the server's account as a reward (subject to the specific business rules), while the remaining portion (if any) will be refunded to the client's account. Then, the protocol terminates.
4. **Output** is the last process and is responsible for calculating the final intersection.
   (a) The client recovers each element in SB using the random values used during blinding to obtain signatures $sy_j = sb_j / r_j \mod n$ of the original elements and hash values $hy_j = H_2(sy_j)$.
   (b) The client initializes the final intersection I as empty. It then looks up hy_j in the Cuckoo filter GF_X, and the corresponding original element y_j is added to I if hy_j exists in GF_X.
   (c) At last, the client outputs the final intersection I.

Compared to the basic protocol, our protocol introduces an additional step of calculating the Merkle roots M_1 and M_2. However, the elements contained in B and SB remain unchanged, ensuring that this process does not impact the correctness of the results. Furthermore, in our protocol, the server transmits the Cuckoo filter GF_X, which inserts set HX instead of the original set HX. Nonetheless, due to the Cuckoo filter's low false positive rate when appropriately configured, the probability of the client making a false judgment on whether hy_j is in set HX does not exceed the false positive rate of the utilized filter. Therefore, given adherence to the protocol by both parties, the correctness of the basic protocol combined with the sufficiently low false positive rate of the Cuckoo filter ensures that our protocol yields the correct intersection result (with an error rate not exceeding the false positive rate of the Cuckoo filter).

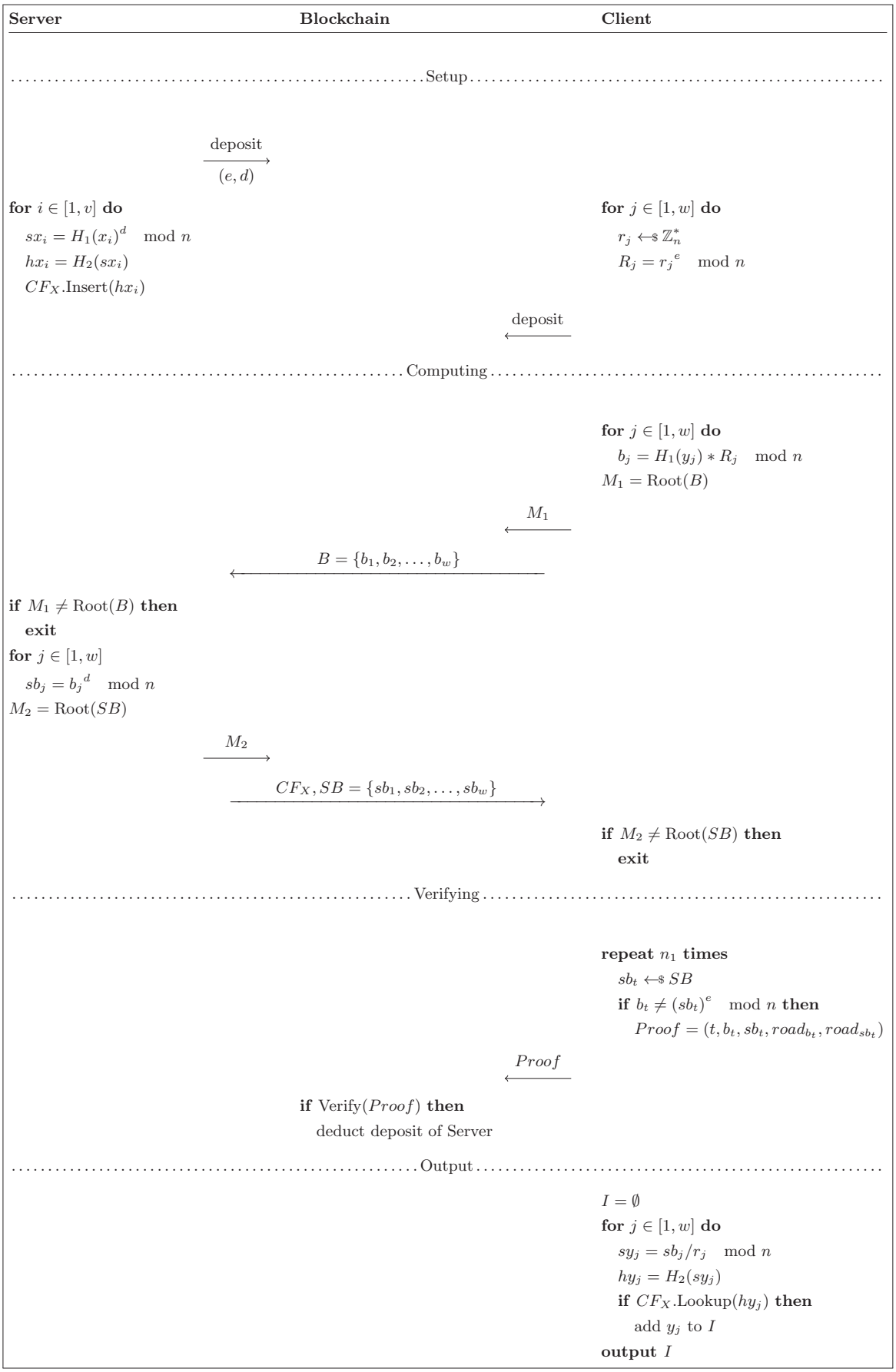

Figure 3. Our full protocol.

4.4. Theoretical Analysis

We conduct a theoretical analysis of our protocol from security and efficiency viewpoints in this section.

4.4.1. Security Analysis

In terms of security, our protocol does not compromise the security guarantees provided by the basic protocol. Informally, on the one hand, the uploaded Merkle roots M_1 and M_2 reveal no additional information about sets B and SB. On the other hand, encoding set HX with a Cuckoo filter does not provide the client with more information compared to directly transmitting set HX. Formally, if there exists a PPT algorithm $\mathcal{A}$ that can break our proposed protocol with a non-negligible advantage γ, we can treat this algorithm as a subroutine to construct an algorithm $\mathcal{A}'$ that breaks the basic protocol as follows: firstly, obtain sets HX and SB (or B); then, insert HX into a Cuckoo filter and compute the Merkle root of SB (or B) before invoking algorithm $\mathcal{A}$. The view of algorithm $\mathcal{A}$ running as a subroutine of algorithm $\mathcal{A}'$ is identical to its execution within our proposed protocol. Therefore, the advantage of algorithm $\mathcal{A}$ for breaking the basic protocol is also at least γ.

Additionally, our protocol can capture malicious behavior of the server with a certain probability. Specifically, let us assume that the proportion of the client's data correctly signed by the server is denoted as $p(0 \leq p \leq 1)$, meaning that only wp elements in set B are correctly signed. If the client randomly selects $n_1(1 \leq n_1 \leq w)$ data points from set SB for verification, the probability P_{cap} of capturing the server's malicious behavior is given by Equation (5).

$$P_{cap} = 1 - \frac{\binom{w(1-p)}{n_1} \cdot \binom{wp}{0}}{\binom{w}{n_1}} = 1 - \frac{A_{w(1-p)}^{n_1}}{A_w^{n_1}} \tag{5}$$

It can be observed that $P_{cap} = 1$ when $n_1 > w(1-p)$; otherwise, P_{cap} is shown in Equation (6).

$$P_{cap} \geq 1 - (1-p)^{n_1} \tag{6}$$

where $n_1 \leq w(1-p)$.

By Equations (5) and (6), we observe that when the server's malicious behavior remains constant, the higher the number of elements verified by the client during the validation phase, the greater the probability of detecting the server's malicious behavior. In an extreme case, if the client does not tolerate any error in the results, it can verify every element in set SB. However, this approach comes at the cost of increased workload. Therefore, our protocol allows the client to adaptively verify the results to achieve the desired level of deterrence even with limited computational resources.

Moreover, upon failure to validate any sampled element, the client can generate a publicly verifiable and irrefutable proof to accuse the server. Our protocol ensures accountability, defamation-free elements, and privacy of the proof as follows:

- **Accountability**: If Equation (4) does not hold, an honest client will always be able to produce a valid proof causing the output of the verification algorithm to be True. Given that both element $b_t \in B$ and its corresponding $sb_t \in SB$ reside in Merkle trees with roots M_1 and M_2, the client can provide effective paths $road_{b_t}$ and $road_{sb_t}$ to pass the first two steps of the verification algorithm. Therefore, as long as there exists an element $b_t \in B$ satisfying $b_t \neq (sb_t)^e \mod n$, the output of the verification algorithm will be True.
- **Defamation-Free**: If the server is honest, the probability that a client generates a proof such that the output of the verification algorithm is True is negligible. Since the server is honest, every $sb_t \in SB$ satisfies $b_t = (sb_t)^e \mod n$ except for a negligible probability. Additionally, the Merkle roots of B and SB have been recorded on the blockchain. Due to the tamper-evident nature of the blockchain, the client cannot modify M_1 and M_2. Consequently, if the client attempts to tamper with either b_t or sb_t

such that they do not satisfy Equation (4), the integrity of the Merkle tree renders it impossible for the client to provide valid Merkle paths for the altered elements.

- **Privacy**: Apart from a negligible probability, the proof generated by the client does not disclose its private information. In the proof $Proof = (t, b_t, sb_t, road_{b_t}, road_{sb_t})$, t, b_t, and sb_t are known to the server during the intersection process. The Merkle paths $road_{b_t}$ and $road_{sb_t}$ can also be independently calculated by the server. Therefore, the proof $Proof$ does not reveal any additional information to the server, which preserves the client's privacy against the server and others.

Our protocol further enhances security by automatically executing economic incentives on smart contracts. For the client, once the server's malicious behavior is captured, it can be inferred from the above accountability that the client can generate a valid proof to pass the verification of the smart contract. Since both parties have already pledged a certain amount of deposit before the computation, the smart contract automatically deducts the server's deposit without the need for any third party for enforcement when the verification result is True. For the honest server, as implied by the defamation-free property, the client cannot generate a valid proof that passes the smart contract verification. Therefore, even if the client does not actively pay the server a reward, a portion of the client's deposit will automatically transfer to the server's account after time t_2. Furthermore, in our design, once a proof accusing the server is successfully validated by the smart contract, it is recorded on the blockchain through the contract's event mechanism. Similarly, the event of the smart contract not receiving a valid proof before time t_2 is also recorded on the blockchain. All these events contribute to the archive of the server providing the PSI service, which is publicly accessible and permanently stored on the blockchain. This, to some extent, enhances the deterrence against a malicious server.

Note that our protocol does not offer forward secrecy for the server. In practice, the server's data typically change minimally. If a leakage occurs at any given point, it would lead to the compromise of almost all the data. Therefore, forward security is meaningful for the server. On the other hand, the client can achieve forward security by employing different random numbers according to its specific requirements.

4.4.2. Efficiency Analysis

In this section, we conduct a theoretical analysis of the efficiency of the proposed protocol and compare it with the protocols of Chen et al. [16], Jiang et al. [32], and Pinkas et al. [25], as summarized in Table 1. Chen [16] introduced an OPRF phase to ensure security against a malicious client, resulting in two rounds of interaction built upon [1]. To address the issue of not being able to verify the intersection in [16], Jiang [32] incorporated publicly verifiable inner product computations, which increases the communication complexity to $\mathcal{O}(v)$ and requires a large number of bilinear pairing operations. The protocol of Pinkas [25] is computationally efficient, yet its communication is $\mathcal{O}(v + w)$. Furthermore, Paxos structures are employed to encode client's data, and extra checking operations are introduced during active OT extension, which further increases communication overhead.

As demonstrated in Section 4.4.1, our protocol ensures that clients can verify the correctness of the results with a certain success rate, which depends on the level of deterrence the client aims to achieve.

Our protocol is designed to be completed in a single round of communication, wherein the client sends B to the server and subsequently receives SB and GF_X in return from the server. Although it appears that the total communication overhead encompasses B, SB, and GF_X, the transmission of the Cuckoo filter GF_X can be accomplished without the requirement for both parties to be concurrently online. The server can store GF_X in IPFS or any third-party cloud storage, which allows the client to download GF_X at any moment before the output phase, mitigating potential bandwidth limitations during online intersection computation. Therefore, the actual online communication volume solely comprises B and SB, which results in the final communication complexity being $\mathcal{O}(w)$.

There may be slight changes on the server's local dataset after the first upload of GF_X, such as the addition of new data or the removal of existing data (the update operation can be split into removal and addition). Instead of generating a new Cuckoo filter and repeatedly uploading it, the server can directly upload changed elements and corresponding instructions (including "ADD" and "REMOVE"). Each update will result in a new version of GF_X. When the client later performs another PSI with the same server whose dataset has undergone minor changes, it can obtain the Cuckoo filter for a specified version by tracing each update between the version it currently has and the expected version. For example, the client follows the insertion algorithm to add new elements to GF_X when the instruction is "Add". For "REMOVE" instructions, the client uses the deletion algorithm to remove specified data from GF_X. However, the filter cannot exceed its maximum load (usually 95% when the bucket size is four and fingerprints are large enough). If the maximum load is exceeded, the server needs to extend the filter's size and updates a new Cuckoo filter.

Table 1. Comparison of related PSI protocols.

Protocol	Correctness	Number of Rounds	Communication Complexity	Computation of Server	Computation of Client
Chen [16]	✘	2	$\mathcal{O}(w \cdot logv)$	$\mathcal{O}(v^2)$	$\mathcal{O}(w \cdot logv)$
Jiang [32]	✔	2	$\mathcal{O}(v)$	$\mathcal{O}(v^2)$	$\mathcal{O}(v+w)$
Pinkas [25]	✔	3	$\mathcal{O}(v+w)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Ours	✔	1	$\mathcal{O}(w)$	$\mathcal{O}(w)$	$\mathcal{O}(1)$

In terms of computational efficiency, the server is required to sign each element in its own set and the client's blind set. Since the server's set size is usually large, this process involves a significant number of exponential operations. However, this only needs to be performed once. The cost of this process can be amortized over the subsequent PSI with other clients. Consequently, during the online computation phase, the server is only required to sign each element within the set B received from the client, yielding a computational complexity of $\mathcal{O}(w)$. Additionally, since the server is cognizant of the factorization of n, it can leverage the Chinese remainder theorem to dramatically improve the efficiency of the signing process.

The client's workload is predominantly composed of three main components: (1) raising random values r_j-s to the e-th power (mod n), (2) sampling to check the result's correctness, and (3) generating a proof when the check fails. The subsequent discussion demonstrates that, given a certain probability that the server is engaging in malicious behavior, the online computational complexity of the aforementioned three processes is independent of the sizes of the sets held by both parties and is computationally efficient.

Although the computational complexity of the first process appears to be $\mathcal{O}(w)$, this process can be completed offline as it does not depend on the server's and client's sets. On the other hand, as mentioned in the literature [15], unlike the typical requirement in RSA encryption, where the public exponent e should not be too small, we can use $e = 3$ due to the introduction of random values r_j, which can significantly reduce the complexity of exponential operations. In practical applications, the values R_j and $(r_j)^{-1}$ can be reused multiple times when the client's forward security is not required, further amortizing the client's computational overhead.

According to Equation (6), the number of samples required for spot checks is solely contingent upon the anticipated deterrence factor P_{cap} and the probability p of the server committing malicious acts, and sampling a subset of elements can achieve a high level of deterrence. For example, assuming a server's probability of incorrect computation is 0.001, the probability of detecting the server's malicious behavior can reach 72% when 2^7 elements are sampled for verification. Taking into account that the protocol of Pinkas et al. [25] also requires a certain amount of public key operations in the basic OT phase, we believe that, for the client, the computation complexity of our protocol is comparable to it.

Moreover, as mentioned above, the exponential operation during the checks is very simple (with an exponent of three).

The generation of the proof is trivial for the client. During the computing phase, the client has already computed the Merkle trees of sets B and SB separately, which allows it to easily retrieve paths $road_{b_t}$ and $road_{sb_t}$ from b_t and sb_t to the M_1 and M_2, respectively.

Furthermore, since the depth of a Merkle tree grows logarithmically with the size of the set, the proof's size is only $\mathcal{O}(logw)$. On the other hand, the verification algorithm only contains two Merkle paths' validation and a single exponentiation operation with an exponent of three, which can be accomplished at a very low cost on smart contracts by utilizing the built-in precompiled contracts on Ethereum.

5. Experiments

To evaluate the performance of the scheme proposed in this paper, two experiments are designed involving on-chain contract execution and off-chain computation. The off-chain computation is implemented using the C++ language with the cryptographic library GMP for large-integer arithmetic and OpenSSL for random number generation and hash function implementation. Both the server and client run on a Linux-based desktop equipped with a single-core Intel Xeon Gold 6278C CPU at a base frequency of 2.6 GHz and are executed in a single thread. The parameters are set as follows: $\kappa = 80, \lambda = 40, n_1 = 128$.

Table 2 lists the storage required by the client for different sizes of the server's set when the size of the client's set is fixed at 2^{12}. It can be seen that the Cuckoo filter significantly reduces the storage on the client, especially when the size of the server's set is large.

Table 2. The client's space cost of storing the server's hashed set in MB in the basic protocol and our protocol.

v	Basic Protocol	Our Protocol
2^{16}	0.56	0.19
2^{20}	9.88	3
2^{24}	174	48
2^{28}	3040	768

Special note: $w = 2^{12}$; the configuration of the Cuckoo filter used is as follows: bucket size $b = 4$, fingerprint length $f = 12$, and false positive rate ϵ is up to 0.0496% .

Figure 4 shows that online communication grows linearly with the size of the client's set. When $w = 2^{12}$, online communication is only 1 MB. From Figure 5, it is evident that the computation time for both parties is roughly linearly related to the size of the client's set. More specifically, the client's computation time is much shorter than the server's, and the online phase comprises only a small fraction of the total computation time. Therefore, in our protocol, the main computational workload is in the server's online computing phase, while the online computational workload for the client is minimal.

The on-chain contract is responsible for verifying proofs and automatically executing economic measures and is written in Solidity and tested on the Ethereum testnet Sepolia. To ensure compatibility with Ethereum's hash function, we utilize the keccak-256 hash function to compute Merkle roots during the off-chain computation. Keccak-256 is Ethereum's prevalent cryptographic hash function and is different from the standard sha3-256 only in padding mode, and it offers a high level of security. To reduce the transaction cost of verifying proofs on smart contracts, we employed Ethereum's precompiled contracts RIPEMD160 for Merkle paths' verification and ModExp for modular exponentiation arithmetic. Figure 6 presents the sizes of the generated publicly verifiable proofs along with the gas consumption required for their validation on smart contracts for various sizes of the client's set.

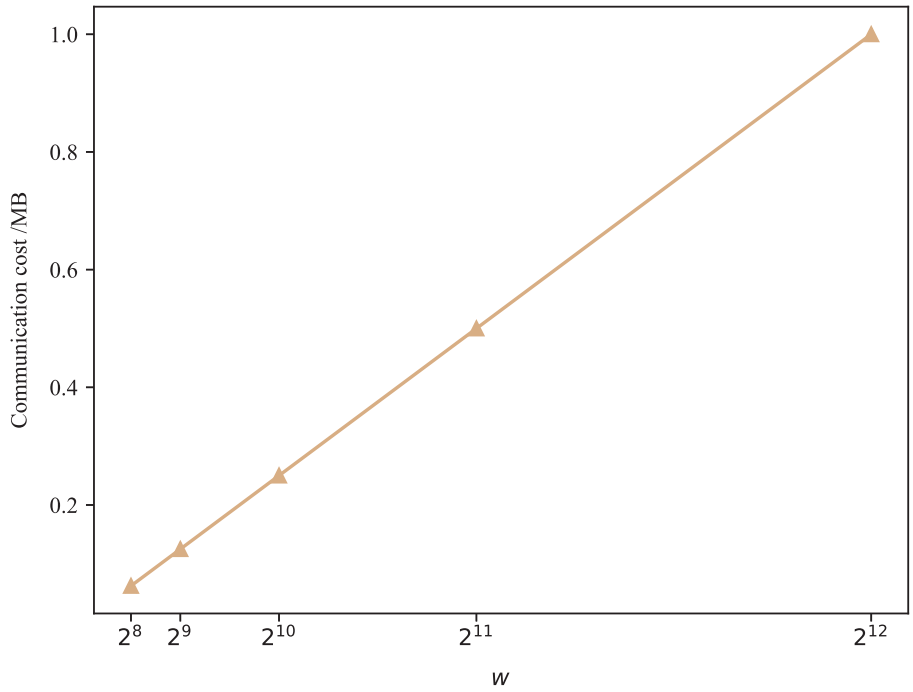

Figure 4. Online communication costs for different sizes of the client's set.

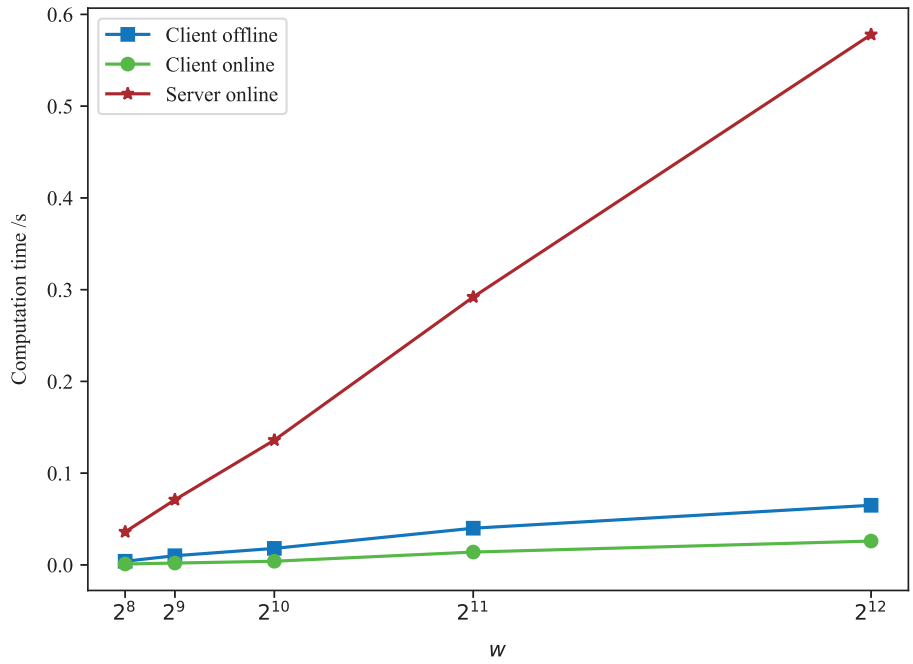

Figure 5. Computation time for different sizes of the client's set.

From Figure 6, it is apparent that both the size of the generated proofs and the gas required for their verification exhibit logarithmic growth with the increase in the client's set size. Currently, the contract call data in a single transaction is limited to 2048 bytes on Ethereum. Hence, our proofs are capable of supporting w up to 2^{28}, which sufficiently meets the demands for practical unbalanced PSI. It should be noted that even if this limit is exceeded, proofs can be split into two or more transactions for contract calls. Transaction costs for verifying proofs on our smart contract are much lower than the current block gas limit (30 million gas). Based on the current Ethereum mainnet gas price (20 Gwei) and ETH price (USD 3035.93) [40], when the size of the client's set is 2^{12}, the verification cost amounts to just USD 5.74.

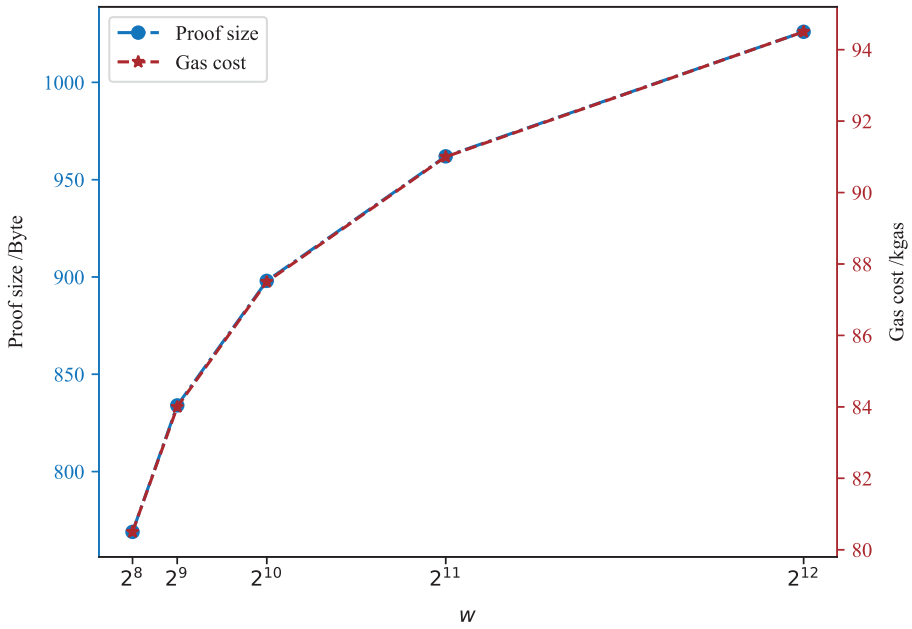

Figure 6. Size of proofs and gas cost for verification for different sizes of the client's set.

6. Conclusions

In this paper, we propose and implement a blockchain-based unbalanced PSI protocol with public verification and financial security. Our protocol allows the client to compute and verify the final intersection with efficient communication and client-side computation. The client can also generate publicly verifiable proofs to accuse the server of malicious behavior, which can be automatically validated by smart contracts at a low cost. By designing an appropriate economic incentive and penalty mechanism, our protocol provides financial security assurances on the foundation of public verification. We believe that our protocol enables clients to better protect their interests in unbalanced PSI scenarios without incurring significant costs.

Nevertheless, the server's online computational demands, in terms of public key operations, escalate linearly with the magnitude of the client's dataset. Consequently, the challenge of diminishing the computational burden on the server, without compromising the public verification of the result and the operational efficiency of the client, presents a significant avenue for subsequent scholarly inquiry.

Author Contributions: Methodology, Z.W.; Supervision, X.M. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was funded by the National Key R&D Program of China, grant number [2021YFC3340600].

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, H.; Laine, K.; Rindal, P. Fast Private Set Intersection from Homomorphic Encryption. In Proceedings of the 24th ACM-SIGSAC Conference on Computer and Communications Security (ACM CCS), Dallas, TX, USA, 30 October–3 November 2017; pp. 1243–1255. [CrossRef]
2. Zhou, Q.; Zeng, Z.; Wang, K.; Chen, M. Privacy Protection Scheme for the Internet of Vehicles Based on Private Set Intersection. *Cryptography* **2022**, *6*, 64. [CrossRef]
3. Mezzour, G.; Perrig, A.; Gligor, V.; Papadimitratos, P. Privacy-Preserving Relationship Path Discovery in Social Networks. In Proceedings of the 8th International Conference on Cryptology and Network Security, Kanazawa, Japan, 12–14 December 2009; Volume 5888, pp. 189–208.

4. Li, J.; Liu, Y.M.; Wu, S. Pipa: Privacy-preserving Password Checkup via Homomorphic Encryption. In Proceedings of the 16th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS), Virtual Event, Hong Kong, 7–11 June 2021; pp. 242–251. [CrossRef]
5. Pinkas, B.; Schneider, T.; Zohner, M.; Assoc, U. Faster Private Set Intersection based on OT Extension. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014; pp. 797–812.
6. Kolesnikov, V.; Kumaresan, R.; Rosulek, M.; Trieu, N. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS), Vienna, Austria, 24–28 October 2016; pp. 818–829. [CrossRef]
7. Orrù, M.; Orsini, E.; Scholl, P. Actively Secure 1-out-of-*N* OT Extension with Application to Private Set Intersection. In Proceedings of the RSA Conference on Cryptographer's Track (CT-RSA), San Francisco, CA, USA, 14–17 February 2017; Volume 10159, pp. 381–396. [CrossRef]
8. Pinkas, B.; Schneider, T.; Zohner, M. Scalable Private Set Intersection Based on OT Extension. *Acm Trans. Priv. Secur.* **2018**, *21*, 7. [CrossRef]
9. Pinkas, B.; Schneider, T.; Segev, G.; Zohner, M.; Assoc, U. Phasing: Private Set Intersection using Permutation-based Hashing. In Proceedings of the 24th USENIX Security Symposium, Washington, DC, USA, 12–14 August 2015; pp. 515–530.
10. Jiang, Z.; Guo, X.; Yu, T.; Zhou, H.; Wen, J.; Wu, Z. Private Set Intersection Based on Lightweight Oblivious Key-Value Storage Structure. *Symmetry* **2023**, *15*, 2083. [CrossRef]
11. Raghuraman, S.; Rindal, P. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In Proceedings of the Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 2505–2517. [CrossRef]
12. Ishai, Y.; Kilian, J.; Nissim, K.; Petrank, E. Extending oblivious transfers efficiently. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2003; pp. 145–161.
13. Jarecki, S.; Liu, X.M. Fast Secure Computation of Set Intersection. In Proceedings of the 7th Conference on Security and Cryptography for Networks, Amalfi, Italy, 13–15 September 2010; Volume 6280, pp. 418–435.
14. Resende, A.C.D.; Aranha, D.F. Faster Unbalanced Private Set Intersection. In Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC), Nieuwpoort, Curaçao, 26 February–2 March 2018; Volume 10957, pp. 203–221. [CrossRef]
15. Cristofaro, E.D.; Tsudik, G. Practical private set intersection protocols with linear complexity. In Proceedings of the 14th Practical Private Set Intersection Protocols with Linear Complexity, Tenerife, Canary Islands, 25–28 January 2010. [CrossRef]
16. Chen, H.; Huang, Z.C.; Laine, K.; Rindal, P. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Toronto, ON, Canada, 15–19 October 2018; pp. 1223–1237. [CrossRef]
17. Zhao, Q.; Jiang, B.; Zhang, Y.; Wang, H.; Mao, Y.; Zhong, S. Unbalanced private set intersection with linear communication complexity. *Sci. China Inf. Sci.* **2024**, *67*, 132105. [CrossRef]
18. Ning, J.; Tan, Z.; Zhang, K.; Ye, W. Low Communication-Cost PSI Protocol for Unbalanced Two-Party Private Sets. *IET Inf. Secur.* **2024**, *2024*, 6052651. [CrossRef]
19. Hazay, C.; Lindell, Y. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *J. Cryptol.* **2010**, *23*, 422–456. [CrossRef]
20. Asharov, G.; Orlandi, C. Calling Out Cheaters: Covert Security with Public Verifiability. In Proceedings of the 18th International Conference on Theory and Application of Cryptology and Information Security (ASIACRYPT), Beijing, China, 2–6 December 2012; Volume 7658, pp. 681–698.
21. Hong, C.; Katz, J.; Kolesnikov, V.; Lu, W.j.; Wang, X. Covert Security with Public Verifiability: Faster, Leaner, and Simpler. In Proceedings of the Advances in Cryptology—EUROCRYPT 2019, Darmstadt, Germany, 19–23 May 2019; pp. 97–121.
22. Zhu, R.Y.; Ding, C.C.; Huang, Y. Efficient Publicly Verifiable 2PC over a Blockchain with Applications to Financially-Secure Computations. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), London, UK, 11–15 November 2019; pp. 633–650. [CrossRef]
23. Freedman, M.J.; Nissim, K.; Pinkas, B. Efficient Private Matching and Set Intersection. In Proceedings of the Advances in Cryptology—EUROCRYPT 2004, Interlaken, Switzerland, 2–6 May 2004; pp. 1–19.
24. Morales, D.; Agudo, I.; Lopez, J. Private set intersection: A systematic literature review. *Comput. Sci. Rev.* **2023**, *49*, 100567. [CrossRef]
25. Pinkas, B.; Rosulek, M.; Trieu, N.; Yanai, A. PSI from PaXoS: Fast, Malicious Private Set Intersection. In Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Zagreb, Croatia, 10–14 May 2020; Volume 12106, pp. 739–767. [CrossRef]
26. Fan, C.; Jia, P.; Lin, M.; Wei, L.; Guo, P.; Zhao, X.; Liu, X. Cloud-Assisted Private Set Intersection via Multi-Key Fully Homomorphic Encryption. *Mathematics* **2023**, *11*, 1784. [CrossRef]
27. Abadi, A.; Dong, C.; Murdoch, S.J.; Terzis, S. Multi-party Updatable Delegated Private Set Intersection. In Proceedings of the 26th International Conference on Financial Cryptography and Data Security, Grenada, 2–6 May 2022. [CrossRef]
28. Huang, Y.; Evans, D.; Katz, J.; Malka, L. Faster secure two-party computation using garbled circuits. In Proceedings of the 20th USENIX Conference on Security, San Francisco, CA, USA, 8–12 August 2011.

29. Ciampi, M.; Orlandi, C. Combining Private Set-Intersection with Secure Two-Party Computation. In Proceedings of the 11th International Conference on Security and Cryptography for Networks (SCN), Amalfi, Italy, 5–7 September 2018; Volume 11035, pp. 464–482. [CrossRef]
30. Meadows, C. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 7–9 April 1986; p. 134. [CrossRef]
31. Huberman, B.A.; Franklin, M.; Hogg, T. Enhancing privacy and trust in electronic communities. In Proceedings of the 1st ACM Conference on Electronic Commerce, Denver, CO, USA, 3–5 November 1999. [CrossRef]
32. Jiang, Y.; Wei, J.; Pan, J. Publicly Verifiable Private Set Intersection from Homomorphic Encryption. In Proceedings of the Security and Privacy in Social Networks and Big Data, Xi'an, China, 16–18 October 2022; pp. 117–137.
33. Aumann, Y.; Lindell, Y. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *J. Cryptol.* **2010**, *23*, 281–343. [CrossRef]
34. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 18 February2024).
35. Martínez, V.G.; Hernández-Álvarez, L.; Encinas, L.H. Analysis of the Cryptographic Tools for Blockchain and Bitcoin. *Mathematics* **2020**, *8*, 131. [CrossRef]
36. Chaum, D. Blind Signature System. In *Advances in Cryptology: Proceedings of Crypto 83*; Springer: Boston, MA, USA, 1984; p. 153. [CrossRef]
37. Bellare, M.; Namprempre, C.; Pointcheval, D.; Semanko, M. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *J. Cryptol.* **2003**, *16*, 185–215. [CrossRef]
38. Fan, B.; Andersen, D.G.; Kaminsky, M.; Mitzenrnacher, M.D. Cuckoo Filter: Practically Better Than Bloom. In Proceedings of the 10th ACM International Conference on Emerging Networking Experiments and Technologies (ACM CoNEXT), Sydney, Australia, 2–5 December 2014; pp. 75–87. [CrossRef]
39. Pagh, R.; Rodler, F.F. Cuckoo hashing. *J. Algorithms* **2004**, *51*, 122–144. [CrossRef]
40. Etherscan. Available online: https://etherscan.io/ (accessed on 25 March 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.