REPHRAIN

Protecting citizens online

# Polynomial Representation Is Tricky:

## *Maliciously Secure Private Set Intersection Revisited*

Aydin Abadi, University College London

Steven J. Murdoch, University College London

Thomas Zacharias, University of Edinburgh

August 2021

UK Research and Innovation · University of Bristol · UCL · The University of Edinburgh · King's College London · University of Bath

# Polynomial Representation Is Tricky: Maliciously Secure Private Set Intersection Revisited

Aydin Abadi[⋆1] Steven J. Murdoch[⋆⋆1] Thomas Zacharias[⋆⋆⋆2]

[1] University College London
[2] University of Edinburgh

**Abstract.** Private Set Intersection protocols (PSIs) allow parties to compute the intersection of their private sets, such that nothing about the sets' elements beyond the intersection is revealed. PSIs have a variety of applications, primarily in efficiently supporting data sharing in a privacy-preserving manner. At Eurocrypt 2019, Ghosh and Nilges proposed three efficient PSIs based on the polynomial representation of sets and proved their security against active adversaries. In this work, we show that these three PSIs are susceptible to several serious attacks. The attacks let an adversary (1) learn the correct intersection while making its victim believe that the intersection is empty, (2) learn a certain element of its victim's set beyond the intersection, and (3) delete multiple elements of its victim's input set. We explain why the proofs did not identify these attacks and propose a set of mitigations.

## 1 Introduction

A Private Set Intersection protocol (PSI) lets mutually distrustful parties compute the intersection of their private sets such that nothing, about the sets' elements, beyond the result is revealed. PSIs have been studied extensively due to their numerous real-world applications to reduce online harm by preserving the Internet users' privacy, to some extent. For instance, they have been used in (a) contact tracing schemes that prevent the further spread of COVID-19 [16], (b) certain Google technologies that find target audiences for marketing campaigns [24] or check compromised credentials [35], (c) online gaming [10], and (d) remote diagnostics [9].

At Eurocrypt 2019, Ghosh and Nilges [20] proposed three PSIs (i.e., two-party, multi-party, and threshold multi-party) that are designed to remain secure against active adversaries. These protocols are efficient as they are primarily based on symmetric-key primitives and polynomial representation of sets, and avoid using zero-knowledge proofs usually utilised in the protocols that consider active adversaries. The three PSIs have been defined and proven secure in the

⋆ aydin.abadi@ucl.ac.uk
⋆⋆ s.murdoch@ucl.ac.uk
⋆⋆⋆ thomas.zacharias@ed.ac.uk

well-known Universal Composability (UC) paradigm [12]. To date, their multi-party protocol is the most efficient multi-party PSI designed to remain secure in the presence of active adversaries.

***Our Contributions.*** We identify *three attacks* that can be mounted on all the three "maliciously secure" PSIs in [20]. In particular, we show an adversary can successfully carry out the following attacks:

1. Attack 1: learning the result, i.e., sets' intersection, while making its honest counter-party believe that there is no element in the intersection.
2. Attack 2: learning a certain element (not necessarily in its set) of the honest party's set beyond the sets' intersection.
3. Attack 3: deleting multiple elements of its counter-party's input set.

Our attacks' analysis indicates that Attack 1 always succeeds (except with a negligible probability), also Attacks 2 and 3 succeed with a non-negligible probability (when the sets' universe size is polynomial or constant in the security parameter). We show that these attacks are feasible in terms of cost to the attacker. We identify several flaws in the protocols' *design* and *proofs* that led to the attacks remaining undetected. Accordingly, we propose a set of candidate *mitigations*. At a high level, most of the issues we identify are the result of a single case: *inappropriate use of polynomial representation*. This representation has been widely used in various cryptographic schemes beyond PSIs, such as in secret sharing [34], error-correcting codes [33], e-voting [27], or secure multi-party computation [25]. Nevertheless, our findings provide evidence that special care should be taken when polynomial representation is utilised in protocols that should remain secure against active adversaries. We hope our work will be used as a reference point by future researchers who need to integrate this representation into their protocols, to avoid (at least) the issues we highlight.

## 2 Related Work

PSIs were first introduced by Freedman *et al.* [17] that were mainly based on Paillier homomorphic encryption and polynomial representation of sets. Since then, numerous PSIs have been proposed. They can be broadly divided into *traditional* and *delegated* categories.

In *traditional* PSIs, e.g., protocols in [7, 13, 14, 17, 20–23, 29, 30, 37], data owners interactively compute the result using their local data. Currently, the protocol of Kolesnikov *et al.* in [29] is the fastest two-party PSI, which is secure against a semi-honest (or passive) adversary. It relies on an oblivious pseudorandom function and Cuckoo hashing. Recently, Pinkas *et al.* in [31] proposed an efficient PSI that is secure against a stronger (i.e., malicious/active) adversary. It is based on Cuckoo hashing, oblivious transfer, and a new data structure called probe-and-XOR of strings. Moreover, there have been efforts to improve the communication cost in PSIs, through fully homomorphic encryption and batching techniques [14] and additive homomorphic encryption, oblivious linear function evaluation, and polynomial representation [21]. Very recently, a new PSI has been proposed that achieves a better balance between communication and computation costs [13]. It relies on oblivious transfer, hashing, and symmetric-key primitives. Since the above schemes support only two parties, researchers proposed multi-party PSIs to let more than two parties efficiently compute the intersection. The multi-party PSIs in [22, 23, 30] have been designed to be secure against passive adversaries. To date, the protocol in [30] is the most efficient multi-party PSI secure against passive adversaries. Also, the multi-party PSIs in [7, 20, 37] have been designed to remain secure against active adversaries. There are two PSIs proposed in [37]. As their authors admit, one of them leaks (non-trivial) information and another one requires the involvement of two non-colluding servers, which is a strong assumption. Also, the PSI of Efraim *et al.* in [7] offers a weaker security guarantee and has a higher communication cost than the multi-party PSI in [20] does (as the authors admit). To date, the multi-party protocol in [20] is the most efficient multi-party PSI designed to be secure against active adversaries.

In *delegated* PSIs, e.g., in [1, 3–5, 26, 36, 38], an additional third party is involved to perform a part of the intersection computation and/or to store parties' encrypted sets. They can be divided into schemes that support (a) one-off delegation, e.g., in [26, 38], that requires parties to re-encode their data locally for each computation and (b) repeated delegation, e.g., in [1, 3–5, 36], that lets parties reuse their outsourced data without locally re-encoding it for each computation.

## 3 Background

In this section, we present the definitions and techniques used in the PSIs proposed by Ghosh and Nilges [20]. This work proposes three PSIs: (a) two-party, (b) multi-party, and (c) threshold multi-party. These PSIs use (a) polynomials to represent set elements, which lets parties compute the intersection in a privacy-preserving way, and (b) Oblivious Polynomial Addition (OPA) to let parties randomise each other's input polynomials. The OPA itself uses two primitives; namely, Oblivious Linear Function Evaluation (OLE) and enhanced OLE.

For the sake of simplicity, we will focus on and analyse the two-party PSI. In the following sections, we describe three attacks that can be mounted on it. The other two PSIs are susceptible to similar attacks. We use $\kappa$ as the security parameter. As in the original work, we consider finite fields $\mathbb{F}$ that are exponential in the size of the security parameter, $\kappa$. A function is negligible (in $\kappa$) if it is asymptotically smaller than any inverse polynomial function. By $[n]$ we denote the set $\{1, \ldots, n\}$. The size of a set $S$ is denoted by $|S|$ and set elements' universe is denoted by $\mathcal{U}$. We say the universe size, $|\mathcal{U}|$, is: (a) large, if $|\mathcal{U}|$ is exponential in $\kappa$, (b) medium, if $|\mathcal{U}|$ is polynomial in $\kappa$, and (c) small, if $|\mathcal{U}|$ is constant in $\kappa$.

### 3.1 Representing Sets by Polynomials

The idea of using a polynomial to represent a set's elements was proposed by Freedman *et al.* in [17]. Since then, the idea has been widely used, e.g., in [1, 3–5, 21, 28]. In this representation, set elements $S = \{s_1, ..., s_d\}$ are defined over

$\mathbb{F}$ and set $S$ is represented as a polynomial of form: $\mathbf{p}(x) = \prod_{i=1}^{d}(x - s_i)$, where $\mathbf{p}(x) \in \mathbb{F}[X]$ and $\mathbb{F}[X]$ is a polynomial ring. Often a polynomial, $\mathbf{p}(x)$, of degree $d$ is represented in the "coefficient form" as follows: $\mathbf{p}(x) = a_0 + a_1 \cdot x + ... + a_d \cdot x^d$. The form $\prod_{i=1}^{d}(x - s_i)$ is a special case of the coefficient form. As shown in [28, 8], for two sets $S^{(A)}$ and $S^{(B)}$ represented by polynomials $\mathbf{p}_A$ and $\mathbf{p}_B$ respectively, their product, which is polynomial $\mathbf{p}_A \cdot \mathbf{p}_B$, represents the set union, while their greatest common divisor, $gcd(\mathbf{p}_A, \mathbf{p}_B)$, represents the set intersection. For two degree-$d$ polynomials $\mathbf{p}_A$ and $\mathbf{p}_B$, and two degree-$d$ random polynomials $\boldsymbol{\gamma}_A$ and $\boldsymbol{\gamma}_B$ whose coefficients are picked uniformly at random from the field, it is proven in [8, 28] that: $\boldsymbol{\theta} = \boldsymbol{\gamma}_A \cdot \mathbf{p}_A + \boldsymbol{\gamma}_B \cdot \mathbf{p}_B = \boldsymbol{\mu} \cdot gcd(\mathbf{p}_A, \mathbf{p}_B)$, where $\boldsymbol{\mu}$ is a uniformly random polynomial, and polynomial $\boldsymbol{\theta}$ contains only information about the elements in $S^{(A)} \cap S^{(B)}$, and contains no information about other elements in $S^{(A)}$ or $S^{(B)}$.

Polynomials can also be represented in the "point-value form". In particular, a polynomial $\mathbf{p}(x)$ of degree $d$ can be represented as a set of $m$ ($m > d$) point-value pairs $\{(x_1, y_1), ..., (x_m, y_m)\}$ such that all $x_i$ are distinct non-zero points and $y_i = \mathbf{p}(x_i)$ for all $i$, $1 \leq i \leq m$. If $x_i$ are fixed, then we can represent polynomials as a vector $\vec{\boldsymbol{y}} = [y_1, ..., y_m]$. Polynomials in point-value form have been used previously in PSIs [1, 3–5, 21, 30]. A polynomial in this form can be converted into coefficient form via polynomial interpolation, e.g., using Lagrange interpolation [6]. Moreover, one can add or multiply two polynomials, in point-value form, by adding or multiplying their corresponding y-coordinates. In this case, the polynomial interpolated from the result would be the two polynomials' addition or product. Often PSIs that use this representation assume that all $x_i$ are picked from $\mathbb{F} \setminus \mathcal{U}$.

### 3.2 Oblivious Linear Function Evaluation

Oblivious Linear function Evaluation (OLE) is a two-party protocol that involves a sender and receiver. In OLE, the sender has two inputs $a, b \in \mathbb{F}$ and the receiver has input $c \in \mathbb{F}$, and the protocol allows the receiver to learn only $s = a \cdot c + b \in \mathbb{F}$, while the sender learns nothing. The PSIs in [20] that we analyse in this paper, sometimes invoke the OPA primitive, explained in the following section, which itself makes a black-box call to the OLE in [18]. Since the OLE has been proven secure in the UC framework, other caller protocols can make calls to OLE's ideal functionality, denoted by $\mathcal{F}_{\text{OLE}}$. The OPA also uses an enhanced version of the above OLE. The enhanced OLE and its ideal functionality are denoted by $\text{OLE}^+$ and $\mathcal{F}_{\text{OLE}^+}$, respectively. $\text{OLE}^+$ ensures that the receiver cannot learn anything about the sender's inputs, even if it sets its input to 0. We refers readers to the paper's full version [2], for $\mathcal{F}_{\text{OLE}}$, $\mathcal{F}_{\text{OLE}^+}$, and $\text{OLE}^+$.

### 3.3 Oblivious Polynomial Addition

Ghosh and Nilges [20] propose Oblivious Polynomial Addition (OPA) which can be seen as a variant of OLE, where parties' inputs are polynomials (instead of the field's elements). In particular, in this scheme two parties are involved, sender and receiver. The sender has two polynomials $\mathbf{r}$ and $\mathbf{u}$ and the receiver has a single polynomial, $\mathbf{p}$. The scheme allows the two parties to compute a linear combination of their inputs, i.e., $\mathbf{s} = \mathbf{p} \cdot \mathbf{r} + \mathbf{u}$, and lets the receiver learn the result, $\mathbf{s}$. The security of OPA requires that (a) nothing about the sender's input polynomials is leaked to the receiver (even if the receiver inserts a 0 polynomial), (b) nothing about the receiver's input polynomial and result is leaked to the sender, and (c) a malicious party who acts arbitrarily is detected by its counter-party, with a high probability. The OPA is presented in Figure 1.

- **Public parameters**: a vector of distinct non-zero elements: $\vec{\boldsymbol{x}} = [x_1, \ldots, x_{2d+1}]$

1. **Computing** $\mathbf{s}(x) = \mathbf{p}(x) \cdot \mathbf{r}(x) + \mathbf{u}(x)$, where the sender has $\mathbf{r}(x), \mathbf{u}(x)$ and the receiver has $\mathbf{p}(x)$ as inputs, $deg(\mathbf{u}) \leq 2d, deg(\mathbf{r}) = d, deg(\mathbf{p}) \leq d$.
   (a) Sender: $\forall j, 1 \leq j \leq 2d+1$, computes $r_j = \mathbf{r}(x_j)$ and $u_j = \mathbf{u}(x_j)$. Then, it inserts $(r_j, u_j)$ into $\mathcal{F}^{(j)}_{\text{OLE}^+}$.
   (b) Receiver: $\forall j, 1 \leq j \leq 2d+1$, computes $p_j = \mathbf{p}(x_j)$. Then, it inserts every $p_j$ into $\mathcal{F}^{(j)}_{\text{OLE}^+}$ and receives $s_j = p_j \cdot r_j + u_j$. It interpolates a polynomial $\mathbf{s}(x)$ using pairs $(x_j, s_j)$. Next, it checks if $deg(\mathbf{s}) \leq 2d$. Otherwise, it aborts.
2. **Consistency check**:
   (a) Sender: picks a random $x^* \stackrel{\$}{\leftarrow} \mathbb{F}$, and sends it to the receiver.
   (b) Receiver: picks random values $f, v \stackrel{\$}{\leftarrow} \mathbb{F}$ and inserts them into an instance of $\mathcal{F}_{\text{OLE}}$, denoted by $\mathcal{F}^1_{\text{OLE}}$. It inserts $(\mathbf{p}(x^*), -\mathbf{s}(x^*) + f)$ into another instance of $\mathcal{F}_{\text{OLE}}$, say $\mathcal{F}^2_{\text{OLE}}$.
   (c) Sender: picks a random value $t \stackrel{\$}{\leftarrow} \mathbb{F}$, and inserts it to $\mathcal{F}^1_{\text{OLE}}$ that sends $c = f \cdot t + v$ to the sender. It also inputs $\mathbf{r}(x^*)$ into $\mathcal{F}^2_{\text{OLE}}$ that sends $\bar{f} = \mathbf{r}(x^*) \cdot \mathbf{p}(x^*) - \mathbf{s}(x^*) + f$ to the sender which sums it with $\mathbf{u}(x^*)$. This yields $f' = \mathbf{r}(x^*) \cdot \mathbf{p}(x^*) - \mathbf{s}(x^*) + f + \mathbf{u}(x^*)$. The sender sends $f'$ to the receiver.
   (d) Receiver: It aborts if $f' \neq f$; otherwise, it sends $v$ to the sender.
   (e) Sender: It aborts if $f' \cdot t + v \neq c$.
3. Receiver: picks $x_r$ and runs similar consistency check with the sender.

Fig. 1: Oblivious Polynomial Addition (OPA) protocol [20]

### 3.4 Two-party PSI

In this section, we describe the two-party PSI of Ghosh and Nilges [20] that has been designed to be secure against an active adversary. The protocol mainly utilises polynomial representation of sets and OPA. At a high level, in this protocol, each party generates a polynomial that represents its set. After that, each party randomises its counter-party's polynomial. To do so, a party (as a sender) picks two random polynomials and inserts them into the OPA. The other party (as a receiver) inserts into the OPA its polynomial that represents its set; in return, it receives its polynomial in a randomised form. The parties switch their role and run the OPA again. Next, they exchange messages that allow them to find the result (intersection) polynomial whose roots contain the sets' intersection. Each party evaluates the result polynomial at every element of its set and considers the element in the intersection, if the evaluation's result is zero.

To check the result's correctness, the parties participate in an efficient "output verification" phase. In this phase, parties $A$ and $B$ pick random values $z$ and $q$ respectively. Then, a party evaluates its polynomials at its random element (say $z$) which yields a small set of values. It sends the result to its counter-party, which (a) combines the messages that the other party sent, (b) evaluates the result polynomial at $z$, (c) checks if the values generated in the previous two steps are equal, and (d) accepts the result if they are equal. The two-party PSI is presented in Figure 2. There is a minor difference between the two-party PSI presented in [20] and Figure 2. Namely, in Figure 2 we replaced the OPA's ideal functionality $\mathcal{F}_{\text{OPA}}$ with the actual protocol, OPA. This change (that does not affect the protocol at all) helps clarify the explanation of our attacks.

Each party $I \in \{A, B\}$ has a set $S^{(I)}$, where $m = max(|S^{(A)}|, |S^{(B)}|) + 1$.

1. **PSI Computation**
   (a) Party $I \in \{A, B\}$: represents its set elements (i.e., all $s_j^{(I)} \in S^{(I)}$) as a degree-$m$ polynomial: $\mathbf{p}_I = \boldsymbol{\omega}_I(x) \cdot \prod_{j=1}^{ö}(x - s_j^{(I)})$, where $\boldsymbol{\omega}_I(x)$ is a random polynomial and $ö = |S^{(I)}|$. Each party $I \in \{A, B\}$ picks three random polynomials: $\mathbf{r}_I, \mathbf{u}_I$ and $\mathbf{r}'_I$, where the degree of $\mathbf{u}_I$ is $2m$ and the degree of $\mathbf{r}_I$ and $\mathbf{r}'_I$ is $m$.
   (b) The parties invoke OPA where party $A$ inserts $\mathbf{r}_A, \mathbf{u}_A$ and party $B$ inserts $\mathbf{p}_B$ to OPA, which outputs $\mathbf{s}_B = \mathbf{p}_B \cdot \mathbf{r}_A + \mathbf{u}_A$ to party $B$.
   (c) The parties again invoke OPA, this time party $A$ inserts $\mathbf{p}_A$ while party $B$ inserts $\mathbf{r}_B, \mathbf{u}_B$ to OPA that outputs $\mathbf{s}_A = \mathbf{p}_A \cdot \mathbf{r}_B + \mathbf{u}_B$ to party $A$.
   (d) Party $A$ sends $\mathbf{s}'_A = \mathbf{s}_A - \mathbf{u}_A + \mathbf{p}_A \cdot \mathbf{r}'_A$ to party $B$.
   (e) Party $B$ computes: $\mathbf{p}_\cap = \mathbf{s}'_A + \mathbf{s}_B + \mathbf{p}_B \cdot \mathbf{r}'_B - \mathbf{u}_B = \mathbf{p}_A \cdot \mathbf{r}'_A + \mathbf{p}_A \cdot \mathbf{r}_B + \mathbf{p}_B \cdot \mathbf{r}_A + \mathbf{p}_B \cdot \mathbf{r}'_B$. It sends the result polynomial, $\mathbf{p}_\cap$, to party $A$.
   (f) To find the intersection, party $I$ evaluates polynomial $\mathbf{p}_\cap$ at every element of its set, $s_j^{(I)}$, and considers the element in the intersection if $\mathbf{p}_\cap(s_j^{(I)}) = 0$.
2. **Output Verification**
   (a) Parties $A$ and $B$ pick random values $z, q \xleftarrow{\$} \mathbb{F}$ respectively and send them to their counter-party.
   (b) Party $B$ sends $\alpha_B = \mathbf{p}_B(z)$, $\beta_B = \mathbf{r}_B(z)$, and $\delta_B = \mathbf{r}'_B(z)$ to party $A$.
   (c) Party $A$ checks if: $\mathbf{p}_\cap(z) \stackrel{?}{=} \mathbf{p}_A(z) \cdot (\beta_B + \mathbf{r}'_A(z)) + \alpha_B \cdot (\mathbf{r}_A(z) + \delta_B)$.
   (d) Party $A$ sends $\alpha_A = \mathbf{p}_A(q)$, $\beta_A = \mathbf{r}_A(q)$, and $\delta_A = \mathbf{r}'_A(q)$ to party $B$.
   (e) Party $B$ checks if: $\mathbf{p}_\cap(q) \stackrel{?}{=} \mathbf{p}_B(q) \cdot (\beta_A + \mathbf{r}'_B(q)) + \alpha_A \cdot (\mathbf{r}_B(q) + \delta_A)$.

Fig. 2: Two-party PSI in [20]

## 4 Attack 1: Making Honest Party Learn Incorrect Result

In this section, we describe an attack scenario in which an adversary crafts certain messages in the PSI, that ultimately would allow that party to learn the actual result, i.e., the intersection, while (a) making its honest counter-party believe that there is no element in the intersection, and (b) not having misbehaviour detected. Thus, this attack allows the adversary to affect the PSI's correctness. The issue stems from a flaw in the protocol that lets a party include in the result a polynomial which is *not re-randomized* by its counter-party.

### 4.1 Attack Description

Without loss of generality, we let party $B$ be malicious. Our focus will be on the two-party PSI, presented in Figure 2. Both parties honestly perform steps 1a–1d. However, $B$ in step 1e, as part of computing polynomial $\mathbf{p}_\cap$, instead of summing $\mathbf{s}'_A$ with the product $\mathbf{p}_B \cdot \mathbf{r}'_B$, it sums $\mathbf{s}'_A$ with another random polynomial $\mathbf{r}''_B$ of degree $2m$ and then honestly adds the rest of the polynomials. So, now $\mathbf{p}_\cap$ is:

$$\begin{aligned}\tilde{\mathbf{p}}_\cap &= \mathbf{s}'_A + \mathbf{s}_B + \mathbf{r}''_B - \mathbf{u}_B \\ &= \mathbf{p}_A \cdot \mathbf{r}'_A + \mathbf{p}_A \cdot \mathbf{r}_B + \mathbf{p}_B \cdot \mathbf{r}_A + \mathbf{r}''_B\end{aligned} \tag{1}$$

Party $B$ sends $\tilde{\mathbf{p}}_\cap$ to $A$, in step 1e. In the "output verification" phase, both parties honestly take step 2a, to generate $z$ and $q$. In step 2b, $B$ honestly computes $\alpha_B = \mathbf{p}_B(z)$, $\beta_B = \mathbf{r}_B(z)$, but it sets $\delta_B = \mathbf{r}''_B(z) \cdot (\alpha_B)^{-1}$, instead of setting $\delta_B = \mathbf{r}'_B(z)$. It sends $\alpha_B, \beta_B$, and $\delta_B$ to $A$ which in step 2c:

1. evaluates the result polynomial, $\tilde{\mathbf{p}}_\cap$, at $z$ that yields:
   $$\tilde{\mathbf{p}}_\cap(z) = \mathbf{p}_A(z) \cdot \mathbf{r}'_A(z) + \mathbf{p}_A(z) \cdot \mathbf{r}_B(z) + \mathbf{p}_B(z) \cdot \mathbf{r}_A(z) + \mathbf{r}''_B(z)$$
2. generates value $\zeta$ as below (given messages $\alpha_B, \beta_B$, and $\delta_B$, sent by party $B$):
   $$\begin{aligned}\zeta &= \mathbf{p}_A(z) \cdot (\beta_B + \mathbf{r}'_A(z)) + \alpha_B \cdot (\mathbf{r}_A(z) + \delta_B) \\ &= \mathbf{p}_A(z) \cdot (\mathbf{r}_B(z) + \mathbf{r}'_A(z)) + \mathbf{p}_B(z) \cdot (\mathbf{r}_A(z) + \mathbf{r}''_B(z) \cdot (\alpha_B)^{-1}) \\ &= \mathbf{p}_A(z) \cdot \mathbf{r}_B(z) + \mathbf{p}_A(z) \cdot \mathbf{r}'_A(z) + \mathbf{p}_B(z) \cdot \mathbf{r}_A(z) + \mathbf{r}''_B(z)\end{aligned}$$
3. checks if $\tilde{\mathbf{p}}_\cap(z)$ equals $\zeta$, i.e., $\tilde{\mathbf{p}}_\cap(z) \stackrel{?}{=} \zeta$. If passed, then it accepts the result.

### 4.2 Attack Analysis

By using the above approach, malicious party $B$ can pass the verification in the PSI and convince $A$ to accept the manipulated result. Malicious party $B$ can generate the correct result (i.e., sets' intersection) *for itself*, by honestly computing $\mathbf{p}_\cap$ in step 1e and following the protocol in step 1f. However, given manipulated result $\tilde{\mathbf{p}}_\cap$, presented in Equation (1), honest party $A$ cannot learn the actual sets' intersection, for the following reason. Let us rewrite the manipulated result as $\tilde{\mathbf{p}}_\cap = \boldsymbol{\gamma} + \mathbf{r}''_B$, where $\boldsymbol{\gamma} = \mathbf{p}_A \cdot \mathbf{r}'_A + \mathbf{p}_A \cdot \mathbf{r}_B + \mathbf{p}_B \cdot \mathbf{r}_A$. Note that polynomial $\boldsymbol{\gamma}$ encodes the actual result, as its roots contain the intersection of the sets. But, $\mathbf{r}''_B$ is a random polynomial of degree $2m$, so the probability that its roots contain all elements in the intersection is negligible in $\kappa$. In particular, the said probability is $\frac{1}{|\mathbb{F}|^h}$, where $h$ is the intersection cardinality (for a formal analysis, we refer readers to our paper's full version [2]). This means that the set of roots of polynomial $\tilde{\mathbf{p}}_\cap = \boldsymbol{\gamma} + \mathbf{r}''_B$ does not contain all common roots of both polynomials $\boldsymbol{\gamma}$ and $\mathbf{r}'_B$, except with a negligible probability. Thus, the manipulated polynomial, $\tilde{\mathbf{p}}_\cap$, does not represent the intersection of the sets.

Accordingly, party $A$, which does not know $\mathbf{r}'_B$, cannot learn the correct result and the malicious party can succeed with a high probability, $Pr_1 = 1 - \frac{1}{|\mathbb{F}|^h}$. Attack 1 is efficient, as it requires the adversary to perform only $2m + 1$ extra modular additions and multiplications in total. We also examined the protocol's security proof. The inspection shows that the lack of analysis of the case where $\delta_B \neq \mathbf{r}'_B(z)$ in the proof, led to Attack 1. We refer readers to Appendix A.1 for a detailed analysis of the proof's flaw.

**Extension to Multi-party Protocol.** The security issue, identified in this section, is inherited by the multi-party PSI, presented in Figure 10 in [20], because it uses the same verification mechanism. Specifically, in the multi-party PSI, a malicious party (except the central party, $P_0$) in step 3 of phase 3, replaces $\mathbf{p}_i \cdot \mathbf{r}'_i$ with $\mathbf{r}'_i$. To pass the verification, in step 2 of phase 5, it sets $\delta_i = \mathbf{r}'_i(x^*) \cdot (\alpha_i)^{-1}$, instead of setting $\delta_i = \mathbf{r}'_i(x^*)$, where $x^*$ is a random value generated in step 1 of phase 5. For central party $P_0$ to mount a similar attack, it follows the instructions provided above for malicious party $B$. Since the threshold multi-party PSI makes a black-box call to the multi-party PSI, a similar attack we described in this section (and later sections) can be mounted to the threshold scheme too.

### 4.3 Candidate Mitigation

A closer look at the above attack reveals that the main source of the issue is the use of the polynomials' product $\mathbf{p}_I \cdot \mathbf{r}'_I$, in steps 1d and 1e, where the product is not re-randomized by the other party, and is a part of the result polynomial. Fortunately, the above issue can be efficiently addressed, for the two-party PSI, if the protocol is slightly adjusted. Nonetheless, addressing the issue for the multi-party PSI would require each party to interact with all other parties and so would add significant costs. The remedy for the two-party PSI relies on the idea that (1) each party randomizes its input polynomial, (2) each party re-randomizes its counter-party's input polynomial, and (3) the result polynomial consists of the sum of only the re-randomized input polynomials.

Next, we present the modified two-party PSI. We first describe the "PSI computation" phase. In step (a) party $I \in \{A, B\}$: represents its set elements $s_j^{(I)} \in S^{(I)}$ as a degree-$m$ polynomial: $\mathbf{p}_I = \boldsymbol{\omega}_I(x) \cdot \prod_{j=1}^{\ddot{o}} (x - s_j^{(I)})$, where $\boldsymbol{\omega}_I(x)$ is a random polynomial and $\ddot{o} = |S^{(I)}|$. Each party $I$ picks three random polynomials: $\mathbf{r}_I, \mathbf{u}_I$ and $\mathbf{r}'_I$, where the degree of $\mathbf{u}_I$ is $3m$ and the degree of $\mathbf{r}_I$ and $\mathbf{r}'_I$ is $m$. It also computes $\bar{\mathbf{p}}_I = \mathbf{p}_I \cdot \mathbf{r}'_I$. In step (b) the parties invoke OPA where party $A$ inserts $\mathbf{r}_A, \mathbf{u}_A$ and party $B$ inserts $\bar{\mathbf{p}}_B$ to OPA, which outputs $\mathbf{s}_B = \bar{\mathbf{p}}_B \cdot \mathbf{r}_A + \mathbf{u}_A$ to $B$. In step (c) the parties invoke OPA again, this time $A$ inserts $\bar{\mathbf{p}}_A$ while $B$ inserts $\mathbf{r}_B, \mathbf{u}_B$ to OPA that outputs $\mathbf{s}_A = \bar{\mathbf{p}}_A \cdot \mathbf{r}_B + \mathbf{u}_B$ to $A$. In step (d) party $A$ sends $\mathbf{s}'_A = \mathbf{s}_A - \mathbf{u}_A$ to $B$. In step (e) party $B$ computes: $\mathbf{p}_\cap = \mathbf{s}'_A + \mathbf{s}_B - \mathbf{u}_B = \bar{\mathbf{p}}_A \cdot \mathbf{r}_B + \bar{\mathbf{p}}_B \cdot \mathbf{r}_A$. It sends $\mathbf{p}_\cap$ to $A$. In step (f) to find the intersection, party $I$ evaluates polynomial $\mathbf{p}_\cap$ at every element of its set, $s_j^{(I)}$, and considers the element in the intersection if $\mathbf{p}_\cap(s_j^{(I)}) = 0$. Now we move to the "output verification" phase. In step (a) parties $A$ and $B$ pick random values $z, q \stackrel{\$}{\leftarrow} \mathbb{F}$ respectively and send them to their counter-party. In step (b) party $B$ sends $\alpha_B = \bar{\mathbf{p}}_B(z)$ and $\beta_B = \mathbf{r}_B(z)$, to $A$. In step (c) party $A$ checks if: $\mathbf{p}_\cap(z) \stackrel{?}{=} \bar{\mathbf{p}}_A(z) \cdot \beta_B + \alpha_B \cdot \mathbf{r}_A(z)$. In step (d) party $A$ sends $\alpha_A = \bar{\mathbf{p}}_A(q)$ and $\beta_A = \mathbf{r}_A(q)$ to $B$. In step (e) party $B$ checks if: $\mathbf{p}_\cap(q) \stackrel{?}{=} \bar{\mathbf{p}}_B(q) \cdot \beta_A + \alpha_A \cdot \mathbf{r}_B(q)$. In short, the scheme is now secure because (1) $\mathbf{p}_\cap$ leaks nothing beyond the intersection, (2) neither party knows its counter-party's random polynomials $\mathbf{r}_I, \mathbf{r}'_I$ and $\boldsymbol{\omega}_I$, (3) the evaluation of random polynomial $\boldsymbol{\omega}_I$ at a random point yields a random value, and (4) the result polynomial is the sum of only re-randomized input polynomials. In our paper's full version [2], we outline how the solution can be used for the multi-party PSI.

## 5 Attack 2: Learning Honest Party's Element Beyond The Intersection

In this section, we describe an attack scenario in which a malicious party in the PSI exploits the OPA as a subroutine to check if a certain element (not necessarily an element of its set) exists or not in its honest counter-party's set.

The attack violates the protocol's privacy by allowing the adversary to (a) learn an element of the honest party's set beyond the sets' intersection or (b) efficiently establish the presence or absence of an element in the honest party's set without completing the PSI and without allowing the honest party to learn anything about the other party's set. The source of the issue is that, in the OPA, a sender is given the ability to *independently* pick a random value. This lets a malicious sender pick a value of its choice, $x'^*$, and check if that element is in its honest counter-party's set, i.e., if it is a root of the honest party's input polynomial. In the attack, if $x'^*$ is in the other party's set, then the adversary would *always* pass verifications; but, if $x'^*$ is not in that set, then it would be detected. In the latter case, the adversary still learns the additional information that $x'^*$ is not in its counter-party's set. For the sake of simplicity, in the attack's description below, we focus on a worst-case scenario where the adversary has no background knowledge of its counter-party's set, so it picks $x'^*$ uniformly at random from $\mathcal{U}$. As we will show later, the adversary can conclude that $x'^*$ is in the other party's set and escape from being detected with non-negligible probability, even if the element $x'^*$ is picked randomly from $\mathcal{U}$, when the universe size is medium or small.

### 5.1 Attack Description

Consider the case where malicious party $A$ guesses an element, $x'^* \stackrel{\$}{\leftarrow} \mathcal{U}$, of honest party $B$'s set, $S^{(B)}$. To evaluate its guess, $A$ participates in the PSI with $B$. $A$ follows steps 1a and 1b of Figure 2, and accordingly invokes the OPA. However, it deviates from some of the instructions in the OPA. In particular, both parties honestly take steps 1a and 1b of Figure 1, where $B$'s input, $\mathbf{p}$, is a polynomial that represents its set elements. But, in step 2a of Figure 1, $A$ instead of picking a uniformly random value, $x^* \stackrel{\$}{\leftarrow} \mathbb{F}$, uses $x'^*$, and sends that value to $B$ which (given $x'^*$) follows the protocol in step 2b of Figure 1. In step 2c of Figure 1, $A$ instead of inserting $\mathbf{r}(x'^*)$ to $\mathcal{F}^2_{\text{OLE}}$, it inserts an arbitrary value, $w'$, to $\mathcal{F}^2_{\text{OLE}}$, where $w' \neq \mathbf{r}(x'^*)$. In this case, $\mathcal{F}^2_{\text{OLE}}$ outputs $\bar{f} = w' \cdot \mathbf{p}(x'^*) - \mathbf{s}(x'^*) + f$ to $A$ which adds the output with $\mathbf{u}(x'^*)$, resulting in:

$$\begin{aligned} f' &= w' \cdot \mathbf{p}(x'^*) - \mathbf{s}(x'^*) + f + \mathbf{u}(x'^*) \\ &= w' \cdot \mathbf{p}(x'^*) - \mathbf{p}(x'^*) \cdot \mathbf{r}(x'^*) + f \end{aligned} \tag{2}$$

Both parties $A$ and $B$ honestly follow the rest of the OPA. If $A$ correctly guesses the set element, then it holds that $\mathbf{p}(x'^*) = 0$, because the element would be a root of polynomial $\mathbf{p}$ which represents the set. If $\mathbf{p}(x'^*) = 0$, then by Equation (2) it holds that $f' = f$. Therefore, the adversary can pass the check in step 2d of Figure 1 and at this point can conclude that $x'^*$ is in $B$'s set.

### 5.2 Attack Analysis

In the PSI, when the adversary concludes that $x'^*$ is in $B$'s set, it can (a) honestly take the rest of the steps or (b) avoid doing so. In the former case, the adversary learns the intersection and finds out the guessed element is in the receiver's set, while the honest party learns only the intersection. In the latter case, it learns a single element of the honest party's set without completing the PSI that saves it costs too, while the honest party learns nothing, not even the intersection. So, in either case, the successful adversary learns more than its counter-party does. Note that a malicious $B$ can also carry out the same attack as it is allowed to pick a (random) value of its choice in phase 3 of Figure 1.

Recall, $x'^*$ is picked uniformly at random from $\mathcal{U}$ and if $x'^*$ is in the receiver's set, then the adversary can always pass the OPA's verification. So, the probability that it can confirm $x'^*$ is in the other party's set and escape from being detected depends on the size of $\mathcal{U}$ and the set's cardinality. Specifically, the probability is $Pr_2 = \frac{|S^{(B)}|}{|\mathcal{U}|}$. The adversary can also find out $x'^*$ is *not* in the other party's set with probability $Pr'_2 = 1 - \frac{|S^{(B)}|}{|\mathcal{U}|}$. In the majority of PSIs, there is no assumption made on the size of $\mathcal{U}$, e.g., in [1, 3–5, 13, 14, 20, 26, 29]. The universe size can be large, medium, or even small; for instance, the universe size of temperature, salary, age, and medical treatment is small [11, 15]. Hence, the above adversary can confirm $x'^*$ is in the other party's set without being caught with non-negligible probability, when the universe size is medium or small, whereas that probability would be only negligible if the universe size is large. Also, when the adversary possesses background knowledge of its counter-party's set, it can increase the above probability. The background knowledge could be a small set of elements likely to be in the other party's set. In this case, the adversary picks $x'^*$ from this set to mount the attack; this is in principle akin to the well-known online dictionary attack. Interestingly, Attack 2 does not impose any additional cost to the adversary. This attack was not identified in the protocol's security proof because the proof does not analyse the case where an adversary in the "consistency check" deviates from the protocol and still passes the verification. We refer readers to Appendix A.2 for further discussion on the proof's flaw.

**Extension to Multi-party Protocol.** In the multi-party PSI, each party $P \in \{P_1, ..., P_{n-1}\}$ separately participates in the OPA along with the central party, $P_0$. This means a malicious party $P$ can use the above attack to check whether

$P_0$ has a certain element. Similarly, $P_0$ can carry out the attack. The central party's attack will have more severe repercussions than $P$'s attack, because in each run of the PSI, the central party can interact with and attack more parties (i.e., $n-1$ parties) and accordingly can learn more information.

### 5.3 Candidate Mitigations

One may adjust the protocol such that once an honest receiver finds out $\mathbf{p}(x'^*) = 0$ it aborts, in step 2b of Figure 1. However, this behavior itself would reveal to the malicious sender that it has correctly guessed the element. The above issue can be tackled by letting the parties run a coin-tossing protocol (secure against active adversaries) to compute $x^*$, which would add a small cost.

## 6 Attack 3: Deleting Honest Party's Set Elements

In this section, we show how an adversary can delete certain elements of its counter-party's input set during the PSI computation, which affects the protocol's correctness and privacy. Briefly, the attack lets a successful adversary conclude that a certain *set of elements* exist in its victim's set without letting the victim find those elements in the intersection. The probability that the adversary succeeds without being detected is non-negligible when set elements' universe size is medium or small. The main source of the issue is the use of *point-value* (polynomial) representation of sets. Before we elaborate on the attack, we present the following theorem that is in the core of the adversary's strategy in order to successfully mount its attack. We refer readers to Appendix B for the theorem's formal statement and proof.

**Theorem 1 (informal).** *A set of y-coordinates of a polynomial can be multiplied by a set of non-zero values, such that the polynomial interpolated from the product misses a specific root of the original polynomial.*

### 6.1 Attack Description

We first focus on deleting *a single element*. Later, we will show that the malicious party can delete *multiple elements*. We split the attack into three phases (a) set manipulation, (b) passing OPA's verification, and (c) passing PSI's verification.

**Phase (a): Set Manipulation.** This phase involves both the PSI and OPA. Assume that malicious party $A$ guesses at least one of party $B$'s set elements, say $s_1^{(B)}$, and wants to delete it from $B$'s input. Similar to Attack 2, we assume $s_1^{(B)}$ is picked uniformly at random from $\mathcal{U}$. Loosely speaking, the idea behind the attack is that while the adversary takes steps of the OPA, as the PSI's subroutine, it also generates a *multiplicative inverse* of (y-coordinates of a polynomial representing) $s_1^{(B)}$ and delicately uses the inverse as part of its input. This ultimately cancels out the same element encoded in its counter-party's polynomial that is inserted into the same OPA. In particular, malicious party $A$ honestly follows the PSI in step 1a to generate polynomials $\mathbf{p}_A, \mathbf{u}_A$, and $\mathbf{r}'_A$, with an exception; namely, now it picks a random polynomial, $\bar{\mathbf{r}}_A$, of degree $m-1$ (instead of picking $\mathbf{r}_A$ of degree $m$). Then, in step 1b, party $A$ sends $\bar{\mathbf{r}}_A$ and $\mathbf{u}_A$ to the OPA. Next, $A$ performs as follows in step 1a of Figure 1.

(i) evaluates $\bar{\mathbf{r}}_A$ at every element $x_j \in \vec{\boldsymbol{x}} = [x_1, ..., x_{2d+1}]$. This results in a vector of y-coordinates: $\vec{\boldsymbol{q}}_1 = [\bar{\mathbf{r}}_A(x_1), ..., \bar{\mathbf{r}}_A(x_{2d+1})]$.

(ii) constructs another polynomial of the following form: $x - s_1^{(B)}$. Recall, $s_1^{(B)}$ is the element it guessed. It evaluates the polynomial at every element $x_j$. This results in a vector of y-coordinates: $[(x_1 - s_1^{(B)}), ..., (x_{2d+1} - s_1^{(B)})]$.

(iii) generates the multiplicative inverse of each y-coordinate, that was computed in step (ii). This yields $\vec{\boldsymbol{q}}_2 = [(x_1 - s_1^{(B)})^{-1}, ..., (x_{2d+1} - s_1^{(B)})^{-1}]$.

(iv) multiplies the elements of vectors $\vec{\boldsymbol{q}}_1$ and $\vec{\boldsymbol{q}}_2$, component-wise. This yields $\vec{\boldsymbol{q}}_3 = [\bar{\mathbf{r}}_A(x_1) \cdot (x_1 - s_1^{(B)})^{-1}, ..., \bar{\mathbf{r}}_A(x_{2d+1}) \cdot (x_{2d+1} - s_1^{(B)})^{-1}]$.

(v) evaluates random polynomial $\mathbf{u}_A$, generated honestly in step 1a, at every element $x_j$. This results in $\vec{\boldsymbol{q}}_4 = [\mathbf{u}_A(x_1), ..., \mathbf{u}_A(x_{2d+1})]$.

(vi) sends every pair $(q_{3,j}, q_{4,j})$ to $\mathcal{F}^{(j)}_{\text{OLE}^+}$, where $q_{3,j} \in \vec{\boldsymbol{q}}_3$ and $q_{4,j} \in \vec{\boldsymbol{q}}_4$.

This means that instead of sending $\mathbf{r}_A(x_j)$, malicious party $A$ now sends $q_{3,j} = \bar{\mathbf{r}}_A(x_j) \cdot (x_j - s_1^{(B)})^{-1}$ to $\mathcal{F}^{(j)}_{\text{OLE}^+}$. In this case, in step 1b of Figure 1, honest party $B$ (who inserted values $\mathbf{p}_B(x_j)$ into $\mathcal{F}^{(j)}_{\text{OLE}^+}$) receives the following values from $\mathcal{F}^{(j)}_{\text{OLE}^+}$. For every $j, 1 \leq j \leq 2d+1$ :

$$\begin{aligned} y_j &= \mathbf{p}_B(x_j) \cdot q_{3,j} + q_{4,j} \\ &= \underbrace{\Big(\boldsymbol{\omega}_B(x_j) \cdot (x_j - s_1^{(B)}) \cdot \prod_{i=2}^{ö}(x_j - s_i^{(B)})\Big)}_{\mathbf{p}_B(x_j)} \cdot \underbrace{\Big(\bar{\mathbf{r}}_A(x_j) \cdot (x_j - s_1^{(B)})^{-1}\Big)}_{q_{3,j}} + \underbrace{\mathbf{u}_A(x_j)}_{q_{4,j}} \\ &= \Big(\boldsymbol{\omega}_B(x_j) \cdot \prod_{i=2}^{ö}(x_j - s_i^{(B)})\Big) \cdot \Big(\bar{\mathbf{r}}_A(x_j)\Big) + \mathbf{u}_A(x_j) \end{aligned} \tag{3}$$

In the same step, party $B$ uses pairs $(x_j, y_j)$, $j \in [2d+1]$, to interpolate a polynomial, $\mathbf{s}'_B$, that has the following form.

$$\mathbf{s}'_B = \Big(\boldsymbol{\omega}_B(x) \cdot \prod_{i=2}^{ö}(x - s_i^{(B)})\Big) \cdot \Big(\bar{\mathbf{r}}_A(x)\Big) + \mathbf{u}_A(x) \tag{4}$$

Note that in the PSI, in step 1b of Figure 2, honest party $B$ will receive polynomial $\mathbf{s}'_B$ as the output of the OPA (if malicious party $A$ manages to pass the OPA's verification; we will show it does). Furthermore, each value $(x_j - s_1^{(B)}) \cdot \prod_{i=2}^{ö}(x_j - s_i^{(B)})$ in Equation (3) has the same structure as each $\mu_j$ has in Theorem 1 (in Appendix B). Hence, according to Equations (3) and (4) and Theorem 1, malicious party $A$ has managed to remove $s_1^{(B)}$ from roots of $\mathbf{p}_B$ used in the above step. This ultimately leads to the elimination of that element from the final result, i.e., the sets' intersection. To make that happen, $A$ follows the PSI in steps 1c and 1d of Figure 2 by honestly computing $\mathbf{s}'_A = \mathbf{s}_A - \mathbf{u}_A + \mathbf{p}_A \cdot \mathbf{r}'_A$, and sending $\mathbf{s}'_A$ to $B$. Given polynomials $\mathbf{s}'_B$ and $\mathbf{s}'_A$, party $B$, in step 1e of Figure 2, follows the protocol and computes the result polynomial (presented below) that is supposed to encode the sets' intersection.

$$\begin{aligned}\mathbf{p}_{\cap} &= \mathbf{s}'_{A} + \mathbf{s}'_{B} + \mathbf{p}_{B} \cdot \mathbf{r}'_{B} - \mathbf{u}_{B} \\ &= \mathbf{p}_{A} \cdot \mathbf{r}'_{A} + \mathbf{p}_{A} \cdot \mathbf{r}_{B} + \Big(\boldsymbol{\omega}_{B} \cdot \prod_{i=2}^{\ddot{o}} (x - s_i^{(B)})\Big) \cdot \bar{\mathbf{r}}_{A} + \mathbf{p}_{B} \cdot \mathbf{r}'_{B}\end{aligned} \tag{5}$$

Nevertheless, as it is evident in Equation (5), the result polynomial's roots do not include element $s_1^{(B)}$ with a high probability, even if both parties' sets contain it. Because the roots of polynomial $\Big(\boldsymbol{\omega}_{B} \cdot \prod_{i=2}^{\ddot{o}} (x - s_i^{(B)})\Big)$ lack that element, due to malicious party $A$'s manipulation described above.

For malicious party $A$ to fully succeed, it also needs to pass two verifications, one in the OPA and the other in the PSI. Below, we explain how it can do so.

**Phase (b): Passing OPA's Verification.** This phase involves only the OPA. Since we have already covered steps 1a and 1b in the OPA (in the previous phase description) we will focus only on the "consistency check" in this protocol. Parties $A$ and $B$ honestly follow the OPA in steps 2a and 2b. So, malicious party $A$ (as the sender) in step 2a honestly picks a random value $x^*$ and sends it to honest party $B$ (as the receiver). Then, $B$, in step 2b, picks random values $f, v$ and inserts them into $\mathcal{F}^1_{\text{OLE}}$. Party $B$ inserts $(\mathbf{p}_B(x^*), -\mathbf{s}'_B(x^*) + f)$ into $\mathcal{F}^2_{\text{OLE}}$. Recall, $\mathbf{s}'_B$ is the polynomial which was defined in Equation (4). Party $A$ in step 2c honestly picks a random value, $t$, and inserts it to $\mathcal{F}^1_{\text{OLE}}$ that sends $c = f \cdot t + v$ back to the same party. But, $A$ in the same step, sends $\bar{\mathbf{r}}_A(x^*) \cdot (x^* - s_1^{(B)})^{-1}$, instead of $\mathbf{r}_A(x^*)$, to $\mathcal{F}^2_{\text{OLE}}$ that returns the following value to $A$.

$$\begin{aligned}\bar{f} &= \bar{\mathbf{r}}_A(x^*) \cdot (x^* - s_1^{(B)})^{-1} \cdot \mathbf{p}_B(x^*) - \mathbf{s}'_B(x^*) + f \\ &= \bar{\mathbf{r}}_A(x^*) \cdot (x^* - s_1^{(B)})^{-1} \cdot \underbrace{\Big(\boldsymbol{\omega}_B(x^*) \cdot (x^* - s_1^{(B)}) \cdot \prod_{i=2}^{\ddot{o}} (x^* - s_i^{(B)})\Big)}_{\mathbf{p}_B(x^*)} - \mathbf{s}'_B(x^*) + f \\ &= \bar{\mathbf{r}}_A(x^*) \cdot \Big(\boldsymbol{\omega}_B(x^*) \cdot \prod_{i=2}^{\ddot{o}} (x^* - s_i^{(B)})\Big) - \mathbf{s}'_B(x^*) + f \\ &= -\mathbf{u}_A(x^*) + f\end{aligned} \tag{6}$$

Recall, polynomial $\mathbf{p}_B$, that was inserted by $B$, encodes all set elements of $B$, including $s_1^{(B)}$, whereas polynomial $\mathbf{s}'_B$ misses that specific element due to the party $A$'s manipulation in Equation (3). However, as it is indicated in Equation (6), party $A$ has managed to remove $x^* - s_1^{(B)}$ from $\mathbf{p}_B(x^*)$ too. This will let $A$ escape from being detected, because the result (i.e., $\bar{f} = -\mathbf{u}_A(x^*) + f$) is what an honest party $A$ would have computed. Malicious party $A$ completes step 2c honestly, by adding $\mathbf{u}_A(x^*)$ to $\bar{f}$ (i.e., it computes $f' = \bar{f} + \mathbf{u}_A(x^*)$) and sending $f'$ to $B$, which checks $f'$ equals the random value, $f$, it initially picked in step 2b. By Equation (6), $f' = \bar{f} + \mathbf{u}_A(x^*) = f$ holds; therefore, malicious party $A$ has managed to pass this verification.

**Phase (c): Passing PSI's Verification.** Next, we show how malicious party $A$ can also pass the verification in the PSI, i.e., in step 2d in Figure 2. Our focus will be on the "output verification" in the PSI. At a high level, to pass this verification, $A$ uses a similar trick that is used to pass the verification in the OPA. Specifically, in step 2a, both parties honestly agree on two values $z$ and $q$. Then, in step 2b, party $B$ honestly computes $\alpha_B$, $\beta_B$, and $\delta_B$ and sends them to $A$ which ignores the values and skips step 2c. Malicious party $A$, in step 2d, honestly generates $\alpha_A = \mathbf{p}_A(q)$ and $\delta_A = \mathbf{r}'_A(q)$; however, instead of setting $\beta_A = \mathbf{r}_A(q)$, it sets $\beta_A = \bar{\mathbf{r}}_A(q) \cdot (q - s_1^{(B)})^{-1}$. It sends $\alpha_A$, $\delta_A$, and $\beta_A$ to $B$ which acts honestly in step 2e. In particular, it:

1. evaluates the result polynomial, $\mathbf{p}_\cap$, at $q$ which yields:
$$\mathbf{p}_\cap(q) = \mathbf{p}_A(q)\cdot\mathbf{r}'_A(q)+\mathbf{p}_A(q)\cdot\mathbf{r}_B(q)+\Big(\boldsymbol{\omega}_B(q)\cdot\prod_{i=2}^{\ddot{o}}(q - s_i^{(B)})\Big)\cdot\bar{\mathbf{r}}_A(q)+\mathbf{p}_B(q)\cdot\mathbf{r}'_B(q)$$
2. generates value $\tau$ as below (given the three messages, sent by party $A$):
$$\begin{aligned}\tau &= \mathbf{p}_B(q)\cdot(\beta_A + \mathbf{r}'_B(q)) + \alpha_A\cdot(\mathbf{r}_B(q) + \delta_A)\\ &= \Big(\boldsymbol{\omega}_B(q)\cdot\prod_{i=2}^{\ddot{o}}(q - s_i^{(B)})\Big)\cdot\bar{\mathbf{r}}_A(q) + \mathbf{p}_B(q)\cdot\mathbf{r}'_B(q) + \alpha_A\cdot\mathbf{r}_B(q) + \alpha_A\cdot\delta_A\\ &= \Big(\boldsymbol{\omega}_B(q)\cdot\prod_{i=2}^{\ddot{o}}(q - s_i^{(B)})\Big)\cdot\bar{\mathbf{r}}_A(q) + \mathbf{p}_B(q)\cdot\mathbf{r}'_B(q) + \mathbf{p}_A(q)\cdot\mathbf{r}_B(q) + \mathbf{p}_A(q)\cdot\mathbf{r}'_A(q)\end{aligned}$$
3. checks if $\mathbf{p}_\cap(q)$ equals $\tau$ (i.e., $\mathbf{p}_\cap(q) \stackrel{?}{=} \tau$) and accepts the result, if the check passes.

As indicated above, it holds $\mathbf{p}_\cap(q) = \tau$. Hence, malicious party $A$ can pass the verification in the PSI and convince $B$ to accept the manipulated result.

**Deleting Multiple Elements.** Now we outline how malicious party $A$ can delete *multiple elements* from its counter-party's set during the PSI. Let $S' = \{s_1^{(B)}, ..., s_k^{(B)}\}$ be a set of elements that malicious party $A$ wants to delete from $B$'s set, where $k \leq m$, every element $s_i^{(B)} \in S'$ is picked uniformly at random from $\mathcal{U}$. In the "set manipulation" phase, in the PSI step 1a, party $A$ picks a random polynomial $\bar{\mathbf{r}}_A$ that now has a degree $m - k$. It performs as before in the rest of the same step. In step (ii), it constructs a polynomial that now has the form: $\prod_{i=1}^{k}(x - s_i^{(B)})$. In the same step, it evaluates the polynomial at every element $x_j$, which yields $[\prod_{i=1}^{k}(x_1 - s_i^{(B)}), ..., \prod_{i=1}^{k}(x_{2d+1} - s_i^{(B)})]$. It takes the rest of steps (iii)-(vi) as previously described in the set manipulation phase. The "passing OPA's verification" phase remains unchanged with the exception that, in (the OPA) step 2c, party $A$ now sends $\bar{\mathbf{r}}_A(x^*)\cdot\prod_{i=1}^{k}(x^* - s_i^{(B)})^{-1}$ to $\mathcal{F}^2_{\text{OLE}}$. Similarly, the "passing PSI's verification" phase remains the same as before, with a difference that, in (the PSI) step 2d, party $A$ now sets $\beta_A = \bar{\mathbf{r}}_A(q)\cdot\prod_{i=1}^{k}(q - s_i^{(B)})^{-1}$.

### 6.2 Attack Analysis

A trivial way for an adversary to delete certain elements from the intersection is to delete those elements from its own contributed set. However, there is a major difference between this trivial approach and Attack 3, in terms of the amount of information it learns. Specifically, if the adversary succeeds in Attack 3, it would conclude that its victim has all the deleted elements in its local set. On the contrary, it cannot learn such information by taking the above trivial approach. Moreover, there is a big difference between attacks 2 and 3, in terms of the amount of information the adversary learns. Namely, in the former it learns a *single* element while in the latter it learns *multiple* elements of its victim's set.

Recall, in Attack 3, the adversary always manages to pass the verifications in phases (b) and (c) if it correctly guesses $s_1^{(B)}$. So, its probability of success throughout Attack 3 boils down to correctly guessing that $s_1^{(B)}$ is in its counter-party's set. To compute that probability we can use the same analysis used for Attack 2 (in Section 5.2). As a result, the probability that the adversary successfully deletes a single element is $Pr_3 = \frac{|S^{(B)}|}{|\mathcal{U}|}$; in general, the probability that it can delete $k$ elements is $Pr'_3 = \frac{\prod_{i=0}^{k-1} |S^{(B)}|-i}{|\mathcal{U}|^k}$. The adversary can succeed to delete a constant number of elements, $k$, with a non-negligible probability when the universe is of medium or small size, while that probability is negligible when the universe is of large size. Background knowledge, about the other party's set, would benefit the adversary in this attack too. Attack 3 is efficient, as it only imposes $4m + 6$ extra modular additions and multiplications to the adversary when it deletes a single element. The main two flaws in the protocols' proofs that led to Attack 3 is that, in the OPA's proof, the definition of a malformed input has been limited to only two cases (i.e., polynomial of incorrect degree or zero-polynomial); also, in the PSI's proof, it is assumed the only way the adversary can change an original value is via the addition operation, so the multiplication is never analysed. We refer readers to Appendix A.3 for a detailed discussion on the above flaws.

**Extension to Multi-party Protocol.** The above attack can also be applied to the multi-party PSI, because it uses the same OPA and verification mechanisms as the two-party PSI uses. Therefore, each malicious party $P \in \{P_1, ..., P_{n-1}\}$ can delete an honest central party's set element(s) or a malicious centralised party can delete set element(s) of every honest $P$, without being detected.

### 6.3 Candidate Mitigation

The primary cause of the vulnerability discussed is the use of point-value polynomial representation in the OPA. Specifically, during polynomial multiplication in the OPA where polynomials are presented in point-value form, an adversary can craft its input polynomial such that when it is multiplied by an honest party's polynomial, the product polynomial (after interpolation) misses a certain root. Therefore, it is natural to ask: *can the issue be avoided if polynomials in the* *coefficient form are used in the OPA?* This is indeed the case. Specifically, if the OPA requires the input polynomials to be in coefficient form, then regardless of how the adversary constructs its input polynomial, the product of the two parties' polynomials, generated in the OPA, preserves both polynomials' roots. We refer readers to the paper's full version [2] for a formal statement and proof. The above adjustment imposes to the OPA additional computation cost $O(m^2)$ that stems from multiplying two polynomials in coefficient form.[3] Thus, the computation complexity of the two-party and multi-party PSIs would be higher. Specifically, it would be $O(m^2)$ for two-party and $O(n \cdot m^2)$ for multi-party PSIs, instead of $O(m \cdot \log m)$ and $O(n \cdot m \cdot \log m)$ as in the original protocol [20].

## 7 Conclusion and Future Work

Private set intersection (PSI) is a vital protocol with various real-world applications. At Eurocrypt 2019, Gosh and Nilges [20] proposed three PSIs: (a) two-party, (b) multi-party, and (c) threshold multi-party. To date, their multi-party protocol is the most efficient multi-party PSI designed to remain secure against active adversaries. In this work, we identified three attacks that can be mounted on all of these PSIs. The attacks let an adversary (1) learn the intersection while making its counter-party believe the intersection is empty, (2) learn a certain element of the honest party's set beyond the intersection, and (3) delete multiple elements of its counter-party's input set. We also identified various flaws in the protocols' design and security proofs and proposed a set of mitigations.

Our observation is that in all three attacks an adversary exploits two features of the protocols' design; namely, (a) the polynomial representation of sets and (b) polynomial-based consistency check. Our analysis indicated that the attacks could have been detected if, in the protocols' security proofs, there was a comprehensive study of (i) all checks, (ii) simulators' design, and (iii) malformed inputs' definition. We conclude that special care should be taken in the design and proof of PSIs that use the combination of the two aforementioned features.

Future research could investigate how the security of other protocols (e.g., noisy polynomial addition in [21]) that already used the schemes proposed in [20] could be affected by our findings. While our proposed mitigations add relatively low cost to the two-party PSI, they scale quadratically with the number of participants in the multi-party case. Designing efficient multi-party PSIs, secure against active adversaries, with linear costs is another interesting research line.

## Acknowledgments

Aydin Abadi is supported by REPHRAIN: The National Research Centre on Privacy, Harm Reduction and Adversarial Influence Online, under UKRI grant: EP/V011189/1. Steven J. Murdoch is supported by The Royal Society under grant UF160505.

[3] To lower the polynomial multiplication cost to $O(m \log_2 m)$, one may use Fast Fourier Transform (FFT). However, as FFT uses point-value polynomial representation, further security analysis is required to ensure the attack would not be enabled again.

## References


1. Abadi, A., Terzis, S., Metere, R., Dong, C.: Efficient delegated private set intersection on outsourced private datasets. IEEE TDSC (2018)
2. Abadi, A., Murdoch, S.J., Zacharias, T.: Polynomial representation is tricky: Maliciously secure private set intersection revisited (Full version) (2021), https://eprint.iacr.org/2021/1009.pdf
3. Abadi, A., Terzis, S., Dong, C.: O-PSI: delegated private set intersection on outsourced datasets. In: IFIP SEC (2015)
4. Abadi, A., Terzis, S., Dong, C.: VD-PSI: verifiable delegated private set intersection on outsourced private datasets. In: FC (2016)
5. Abadi, A., Terzis, S., Dong, C.: Feather: Lightweight multi-party updatable delegated private set intersection. IACR Cryptol. ePrint Arch. (2020)
6. Aho, A.V., Hopcroft, J.E.: The Design and Analysis of Computer Algorithms. Addison-Wesley Longman Publishing Co., Inc. (1974)
7. Ben-Efraim, A., Nissenbaum, O., Omri, E., Paskin-Cherniavsky, A.: Psimple: Practical multiparty maliciously-secure private set intersection, eprint arch. (2021)
8. Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. In: ACNS (2013)
9. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: CCS (2007)
10. Bursztein, E., Hamburg, M., Lagarenne, J., Boneh, D.: Openconflict: Preventing real time map hacks in online games. In: IEEE S&P (2011)
11. Camenisch, J., Zaverucha, G.M.: Private intersection of certified sets. In: FC (2009)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS (2001)
13. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: CRYPTO (2020)
14. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS (2017)
15. Cristofaro, E.D., Lu, Y., Tsudik, G.: Efficient techniques for privacy-preserving sharing of sensitive information. In: TRUST (2011)
16. Duong, T., Phan, D.H., Trieu, N.: Catalic: Delegated PSI cardinality with applications to contact tracing. In: ASIACRYPT (2020)
17. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: EUROCRYPT (2004)
18. Ghosh, S., Nielsen, J.B., Nilges, T.: Maliciously secure oblivious linear function evaluation with constant overhead. In: ASIACRYPT (2007)
19. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection (full version). eprint arch. (2017), https://eprint.iacr.org/2017/1064
20. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: EUROCRYPT (2019)
21. Ghosh, S., Simkin, M.: The communication complexity of threshold private set intersection. In: CRYPTO (2019)
22. Hazay, C., Venkitasubramaniam, M.: Scalable multi-party private set-intersection. In: PKC (2017)
23. Inbar, R., Omri, E., Pinkas, B.: Efficient scalable multiparty private set-intersection via garbled bloom filters. In: SCN (2018)
24. Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Saxena, S., Seth, K., Raykova, M., Shanahan, D., Yung, M.: On deploying secure computing: Private intersection-sum-with-cardinality. In: IEEE EuroS&P (2020)

25. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: FOCS (2000)
26. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: FC (2014)
27. Katz, J., Myers, S.A., Ostrovsky, R.: Cryptographic counters and applications to electronic voting. In: EUROCRYPT (2001)
28. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: CRYPTO (2005)
29. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS (2016)
30. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: CCS (2017)
31. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from paxos: Fast, malicious private set intersection. In: EUROCRYPT (2020)
32. Quarteroni, A., Sacco, R., Saleri, F.: Numerical mathematics, vol. 37. Springer Science & Business Media (2010)
33. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics (1960)
34. Shamir, A.: How to share a secret. Commun. ACM (1979)
35. Thomas, K., Pullman, J., Yeo, K., Raghunathan, A., Kelley, P.G., Invernizzi, L., Benko, B., Pietraszek, T., Patel, S., Boneh, D., Bursztein, E.: Protecting accounts from credential stuffing with password breach alerting. In: USENIX Security (2019)
36. Yang, X., Luo, X., Wang, X.A., Zhang, S.: Improved outsourced private set intersection protocol based on polynomial interpolation. Concurrency and Computation: Practice and Experience (2018)
37. Zhang, E., Liu, F., Lai, Q., Jin, G., Li, Y.: Efficient multi-party private set intersection against malicious adversaries. In: CCSW (2019)
38. Zhao, Y., Chow, S.S.M.: Can you find the one for me? privacy-preserving matchmaking via threshold PSI. ePrint Arch. (2018)

# A Identified Flaws In The Security Proofs

Below, we briefly explain a set of flaws we identified in the security proofs of the paper's conference [20] and full [19] versions. These flaws made the three attacks undetected. We categorise the flaws in three classes based on their relevance to each attack. For the sake of simplicity, we exclude the hat symbol, " ˆ ", used in the original proofs. See our paper's full version [2] for a more detailed analysis.

## A.1 Class 1: Not All Checks Have Been Included

In this section, we describe a flaw in the proof of two-party PSI (page 20 in [19]) that lets the environment use Attack 1 to distinguish the two worlds. Briefly, the flaw is that the proof does not consider the case where $\delta^*_B \neq \mathbf{r}'_B(z)$. Before we elaborate on it, we highlight two typos in "Hybrid" 1; namely, $\alpha^*_A \neq \mathbf{p}_A(z)$ and $\beta^*_A \neq \mathbf{r}_A(z)$ should have been $\alpha^*_B \neq \mathbf{p}_B(z)$ and $\beta^*_B \neq \mathbf{r}_B(z)$ respectively, as the proof is for corrupt party $B$. In Hybrid 2, it is stated that "an environment distinguishing Hybrid 1 and 2 must manage to send $\mathbf{p}^*_\cap$ such that $\mathbf{p}^*_\cap \neq \mathbf{p}_A \cdot (\mathbf{r}_B + \mathbf{r}'_A) + \mathbf{p}_B \cdot (\mathbf{r}'_B + \mathbf{r}_A)$ while passing the check in Step 5 [of figure 9] with non-negligible probability." The proof shows that the check fails *only* in the cases where $\alpha^*_B \neq \mathbf{p}_B(z)$ and $\beta^*_B \neq \mathbf{r}_B(z)$; therefore, $\delta^*_B \neq \mathbf{r}'_B(z)$ has been left out of the proof. The lack of such analysis leads to the following issue. As we have shown, the check does not fail for certain $\mathbf{p}_{\cap}$ and $\delta_B$ such that $\mathbf{p}_{\cap} \neq \mathbf{p}_A \cdot (\mathbf{r}_B + \mathbf{r}'_A) + \mathbf{p}_B \cdot (\mathbf{r}'_B + \mathbf{r}_A)$ and $\delta_B \neq \mathbf{r}'_B(z)$. So, the adversary can pass the check with a high probability in the real world (or Hybrid 0). The simulator, in Hybrid 2, detects this inconsistency (i.e., $\delta^*_B \neq \mathbf{r}'_B(z)$) according to Figure 11 in [19]. But, the simulator in Hybrid 1 cannot detect it, as it only aborts if $\alpha^*_B \neq \mathbf{p}_B(z)$ or $\beta^*_B \neq \mathbf{r}_B(z)$. Thus, Hybrids 1 and 2 (likewise Hybrids 0 and 2) are distinguishable by the environment.

### A.2 Class 2: Incomplete Simulator

In the proof of Lemma 4.1, i.e., OPA's security, in the paper's conference version [20], it is stated that "the only possibility for an environment to distinguish between the simulation and the real protocol is by succeeding in answering the check while using a malformed input, i.e. a polynomial of incorrect degree or 0-polynomials." We argue that this is not the only possible case. As we indicated in Attack 2, it is possible the adversary (in the real world) in the "consistency check" phase, deviates from the protocol and still passes the verification. This will ultimately let the environment distinguish the two worlds. Note, the proof should have included the simulation of the "consistency check" phase. Accordingly, the proof does not capture the case where $w'$ of the form $w' \neq \mathbf{r}(x'^*)$ is used by the adversary. In the simulation of the consistency check, the simulator can detect when it is given $w' \neq \mathbf{r}(x'^*)$, as it has already extracted polynomial $\mathbf{r}$ from the adversary. But, in the real world, as we have shown, the adversary can pass the check when $w' \neq \mathbf{r}(x'^*)$ and a certain value, $x'^*$, is used in this phase. Hence, the environment can distinguish the two worlds. This issue arises because the proof does not analyse the case where the check, in the consistency check phase, is passed but $w' \neq \mathbf{r}(x'^*)$ is used in this phase.

### A.3 Class 3: Incomplete Definition Of Malformed Input

Recall, the proof of Lemma 4.1 considers a malformed input if an input polynomial is (i) of incorrect degree or (ii) zero. The issue is that the proof shows only in these two cases the environment cannot distinguish the two worlds. We argue that an input can be malformed without satisfying conditions (i) or (ii). Similar to the description of Attack 3, let a corrupt sender (for all $j \in [2d+1]$) send $q_{3,j} = \bar{\mathbf{r}}_A(x_j) \cdot (x_j - s_1^{(B)})^{-1}$ to $\mathcal{F}^{(j)}_{\text{OLE}^+}$ in the ideal world. This lets the simulator obtain all $q_{3,j}$ and interpolate a polynomial, $\mathbf{q}$. There would be two cases: (1) $deg(\mathbf{q}) > d$, or (2) $deg(\mathbf{q}) \leq d$. In case (1), the simulator aborts. But in the real protocol (in step 1b of Figure 1) the honest party never aborts. Because, in general, polynomial $\mathbf{s}$ interpolated from $2d+1$ pairs $(x_j, s_j)$ always has degree at most $2d$ by Theorem 2. This issue lets the environment distinguish the two worlds. Now we move on to case (2). In the ideal world, in the consistency check phase, the simulator of the OPA is given random value $x^*$ and $w'' = \bar{\mathbf{r}}_A(x^*) \cdot (x^* - s_1^{(B)})^{-1}$ and wants to check $w'' \stackrel{?}{=} \mathbf{q}(x^*)$. Note, the equation may not always hold; because factors $(x_j - s_1^{(B)})^{-1}$ of y-coordinates $q_{3,j}$ from which $\mathbf{q}$ was interpolated, are not directly generated by evaluating a polynomial at $x_j$'s. The probability that $w'' = \mathbf{q}(x^*)$ depends on the choice of $x^*$. If the equation holds, then the simulator does not abort; also, the honest party does not abort as we showed in Attack 3. This is problematic, as the attack has been successfully mounted without being detected in both worlds. If $w'' \neq \mathbf{q}(x^*)$, the simulator aborts, but the honest party does not abort, as the adversary can pass the consistency check. So, the environment can distinguish the two worlds. This issue arises because, in the proof, the definition of a malformed input has been limited to only the above conditions (i) and (ii), and the proof never analyses the case where the check is passed while $w''$ (s.t., $w'' \neq \mathbf{r}(x'^*)$) is sent to $\mathcal{F}^2_{\text{OLE}}$.

The adversary in Attack 3, can pass the PSI's verification too. The issue is that in the PSI's proof (i.e., proof of Theorem 5.1 in [20]) when $A$ is corrupt, the case where $\beta_A$ is not the result of evaluating truly random polynomial $\mathbf{r}_A$ at $z$ (i.e., $\beta_A \neq \mathbf{r}_A(z)$) is never analysed in detail and also it is assumed that the only way the adversary changes the original value is via a modular addition (i.e., $\alpha_A + e$); so, a modular multiplication is never considered as a part of the attack. But, as we showed, the adversary can multiply its input y-coordinates by certain values to affect the result's correctness and pass the verification.

## B Attack 3 Theorems

We first restate Theorem 2 that will be used by the main one, i.e., Theorem 1.

**Theorem 2.** *(Uniqueness of interpolating polynomial [32]) Let* $\vec{\boldsymbol{x}} = [x_1, \ldots, x_v]$ *be a vector of non-zero distinct elements. For* $v$ *arbitrary values:* $y_1, \ldots, y_v$ *there is a unique polynomial:* $\boldsymbol{\tau}$*, of degree at most* $v-1$ *such that:* $\forall j, 1 \leq j \leq v :$ $\boldsymbol{\tau}(x_j) = y_j$*, where* $x_j, y_j \in \mathbb{F}$*.*

Informally, Theorem 1 states that a set of y-coordinates of a polynomial can be multiplied by a set of non-zero values, such that the polynomial interpolated from the product misses a specific root of the original polynomial.

**Theorem 1.** *Let* $\vec{\boldsymbol{x}} = [x_1, \ldots, x_v]$ *be a vector of non-zero distinct elements. Let* $\boldsymbol{\mu} = \prod_{i=1}^{\ddot{o}}(x - e_i) \in \mathbb{F}[X]$ *be a degree* $\ddot{o} < v$ *polynomial with* $\ddot{o}$ *distinct roots* $e_1, \ldots, e_{\ddot{o}}$*, and let* $\mu_j = \boldsymbol{\mu}(x_j)$*, where* $1 \leq j \leq v$*. For some* $c \in [\ddot{o}]$ *such that* $e_c \notin \{x_1, \ldots, x_v\}$*, let* $\boldsymbol{\mu}'$ *be a degree* $\ddot{o} - 1$ *polynomial interpolated from pairs* $(x_1, \mu_1 \cdot (x_1 - e_c)^{-1}), ..., (x_v, \mu_v \cdot (x_v - e_c)^{-1})$*. Then,* $\boldsymbol{\mu}'$ *will not have* $e_c$ *as root, i.e.* $\boldsymbol{\mu}'(e_c) \neq 0$*.*

*Proof.* For the sake of simplicity and without loss of generality, let $c = 1$. We can rewrite polynomial $\boldsymbol{\mu}$ as $\boldsymbol{\mu}(x) = (x - e_1) \cdot \prod_{i=2}^{\ddot{o}}(x - e_i)$. Then, every $\mu_j$ $(1 \leq j \leq v)$ can be written as: $\mu_j = (x_j - e_1) \cdot \prod_{i=2}^{\ddot{o}}(x_j - e_i)$. Accordingly, for every $j$, the product $\alpha_j := \mu_j \cdot (x_j - e_1)^{-1}$ has the form: $\alpha_j = \mu_j \cdot (x_j - e_1)^{-1} = \prod_{i=2}^{\ddot{o}}(x_j - e_i)$. Let $\boldsymbol{\mu}''$ be a degree $\ddot{o}-1$ polynomial with $\ddot{o}-1$ distinct roots identical to the roots of $\boldsymbol{\mu}$ excluding $e_1$, i.e., $\boldsymbol{\mu}''(e_1) \neq 0$. By the Polynomial Remainder Theorem, $\boldsymbol{\mu}''$ can be written as $\boldsymbol{\mu}''(x) = K \cdot \prod_{i=2}^{ö}(x - e_i)$, where $K \in \mathbb{F} \setminus \{0\}$. So, it holds that $\forall j \in [v] : \mu''(x_j) = K \cdot \prod_{i=2}^{ö}(x_j - e_i) = K \cdot \alpha_j$. This implies that $\boldsymbol{\mu}''$ is a degree $ö - 1$ polynomial interpolated from $(x_1, K \cdot \alpha_1), \ldots, (x_v, K \cdot \alpha_v)$. By its definition, the polynomial $\boldsymbol{\mu}'$ is interpolated from the pairs $(x_1, \alpha_1), \ldots, (x_v, \alpha_v)$. Thus, $K \cdot \boldsymbol{\mu}'$ is another degree $ö - 1$ polynomial interpolated from $(x_1, K \cdot \alpha_1), \ldots, (x_v, K \cdot \alpha_v)$. Due to Theorem 2, we have that $\boldsymbol{\mu}'' = K \cdot \boldsymbol{\mu}'$, so $\boldsymbol{\mu}''(e_1) = K \cdot \boldsymbol{\mu}'(e_1) \Rightarrow \boldsymbol{\mu}'(e_1) = K^{-1} \cdot \boldsymbol{\mu}''(e_1)$. We also know that $K^{-1} \neq 0$ and $\boldsymbol{\mu}''(e_1) \neq 0$. Since $\mathbb{F}$ is an integral domain, it follows that $\boldsymbol{\mu}'(e_1) = K^{-1} \cdot \boldsymbol{\mu}''(e_1) \neq 0$. $\square$