

Policy-Enhanced Private Set Intersection: Sharing Information While Enforcing Privacy Policies

Emil Stefanov
UC Berkeley
emil@cs.berkeley.edu

Elaine Shi
UC Berkeley/PARC
elaines@cs.berkeley.edu

Dawn Song
UC Berkeley
dawnsong@cs.berkeley.edu

ABSTRACT

Companies, organizations, and individuals often wish to share information to realize valuable social and economic goals. Unfortunately, privacy concerns often stand in the way of such information sharing and exchange.

This paper proposes a novel cryptographic paradigm called Policy-Enhanced Private Set Intersection (PPSI), allowing two parties to share information while enforcing the desired privacy policies. Our constructions require minimal additional overhead over traditional Private Set Intersection (PSI) protocols, and yet we can handle rich policy semantics previously not possible with traditional PSI and Authorized Private Set Intersection (APSI) protocols. Our scheme involves running a standard PSI protocol over carefully crafted encodings of elements formed as part of a challenge-response mechanism. The structure of these encodings resemble techniques used for aggregating BLS signatures in bilinear groups. We prove that our scheme is secure in the malicious model, under the CBDH assumption, the random oracle model, and the assumption that the underlying PSI protocol is secure against malicious adversaries.

Keywords

authorized private set intersection; multiple authorities; rich policies; secure 2-party computation

1. INTRODUCTION

The need for two parties to exchange privacy-sensitive information arises in numerous application domains. Often, the two parties involved in the exchange are mutually distrustful and do not wish to reveal any additional information other than what is necessary. In particular, we consider the scenario where two parties each hold a set of elements and wish to find the intersection of their elements without revealing other elements that are not in the intersection. In such applications, it is important to ensure that *each data item being exchanged is properly authenticated or authorized*

by the owner(s) or creator(s) of that data item due to the following reasons:

- **Thwart dishonest behavior.** Unless some form of authentication is required, a malicious party can claim possession of fictitious data items, in an attempt to find out whether the other party possesses these data items. For example, if hospitals A and B are trying to find out their common patients, a malicious hospital A can fictitiously claim that Carol is their patient, in an attempt to find out whether Carol is a patient with hospital B .
- **Comply with privacy policies.** Sharing of privacy-sensitive information may be governed by certain privacy regulations, either made by the government or individual organizations. For example, two healthcare providers A and B may wish to exchange information about their common patients to improve service and facilitate diagnosis. However, due to privacy regulations such as the Health Insurance Portability and Accountability Act (HIPAA), they can only share a patient's record if both providers have obtained the patient's consent. The above is an example of a simple privacy policy. In other application scenarios, we may also desire the ability to support richer privacy policies. We demonstrate how to support rich privacy policies in Section 4 and give applications in the healthcare space in Section 6.

In this paper, we propose Policy-Enhanced Private Set Intersection (PPSI). In PPSI, each party has a set of elements, where each element may be authorized (signed) by a different authority or authorities. PPSI allows two parties to find the intersection of their sets, while enforcing rich privacy policies. The policies specify what authorizations each party must possess for its elements. Our scheme thwarts dishonest behavior by preventing a malicious party from using unauthorized elements during the set intersection to violate the privacy of the other party.

1.1 Results and Contributions

New problem definitions. One important contribution we make is the definition of a new problem, namely, Policy-Enhanced Private Set Intersection (PPSI). Existing Private Set Intersection (PSI) protocols and Authorized Private Set Intersection (APSI) protocols are not general enough and fail to adequately address the needs of above-mentioned application scenarios. To resolve this problem our PPSI scheme

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

offers the following rich capabilities not previously possible with existing PSI and APSI protocols:

- **Multiple authorities.** PPSI supports privacy policies where each element may be authorized by a different authority or different authorities. This makes PPSI particularly useful when each data item may not be owned or created by the same entity.
- **Rich privacy policies.** Many applications desire the ability to support expressive privacy policies. For example, we describe some useful policies in healthcare applications in Section 6. Our PPSI constructions can support rich policy semantics during the information sharing process, including *conjunctive and disjunctive policies, asymmetric policies, policies with attributes, and bundles of elements*.

Novel, provably secure constructions. We propose novel PPSI constructions that offer two main functionalities: 1) a signing functionality which allows an authority to authenticate or authorize an element for a party; 2) a set intersection protocol that allows two parties to find the intersection of their elements, while enforcing the desired privacy policy.

We prove the security of our scheme against *malicious* adversaries, assuming that the underlying PSI scheme is also secure in the malicious model. The proof also relies on the CBDH assumption and the random oracle model.

Efficiency. Our constructions are efficient in practice. Specifically, we require $O(n)$ communication bandwidth and $O(nm + n \log \log n)$ computation, where n is the maximum number of elements per party and m is the maximum number of authorities per element. Also, our protocol executes in $O(1)$ rounds.

Notably, our constructions require only minimal overhead over standard Private Set Intersection (PSI) protocols, but can support rich policies that are not possible with standard PSI. We need one additional round of communication over standard PSI, during which the parties exchange two group elements (elliptic points). In terms of computation, we incur an additional overhead of at most nm bilinear pairings over the traditional PSI protocols.

Table 1 summarizes the efficiency of our basic construction described in Section 3.3.

1.2 Technical Challenges and Highlight of Our Techniques

It turns out that the problem is non-trivial, even with relatively simple policies. A straightforward idea is to adopt an existing PSI protocol and require that each party demonstrate a zero-knowledge proof that each element encoded in a cryptographic commitment has the appropriate authorizations. However, *the complication is that when each element has a different authority or different authorities, one cannot reveal the identity of the authority when performing the zero-knowledge proofs, as the identity of the authority can leak information about the corresponding element*.

Special encodings. Our techniques may be of independent interest. Our scheme leverages a type of *special encoding* that allows us to circumvent the need for performing complicated and costly zero-knowledge proofs. Each party

computes the special encodings over their elements and then runs a standard PSI protocol over these encodings.

A party’s encoding for an element x is essentially a product of terms demonstrating its authorization on x and the anticipated terms demonstrating the other party’s authorization on x . The terms are cleverly crafted so that a party can compute its own terms by combining its own authorizations and a challenge sent by the other party. It can also compute the anticipated terms of the other party without having the other party’s authorizations.

If both parties satisfy their respective policies for an element, then both parties obtain the same encoding for that element, and this particular encoding appears in the set intersection. However, if a party does not possess the correct authorizations for an element, it is unable (computationally intractable) to compute the correct encoding for this element. As a result, this party is unable to learn whether the other party owns the element.

We point out that our encoding idea bears resemblance to techniques used for aggregating BLS signatures in bilinear groups [4].

1.3 Applications

Healthcare privacy. One application is to allow healthcare providers to exchange medical information in a way that complies with privacy regulations. For example, two hospitals wish to find their common patients. However, a hospital can only exchange information about a patient if they have the permission of that patient. Moreover, the two hospitals may be competitors, and may not wish to leak the identity of the patients who are not in the set intersection. In Section 6, we will describe more usage scenarios in the healthcare space, and demonstrate the rich policy semantics required by these applications.

Social networking. Consider a social network application where two users would like to find out their common friends without revealing other friends that are not in the intersection. To prevent a dishonest user from including fake friends in its input set, a friendship relation with any user U must be vouched for by the user U .

In the above example, each element is authorized by a single party. Sometimes, each element may need to be certified by multiple authorities. For example, consider a variant of the above-mentioned social network application. Two users would like to find out the common Facebook groups they have joined without revealing all of their groups. Membership in a group should be authorized by its administrator(s). As each group may have multiple administrators, each element (group) in this case may have multiple authorities.

Inner joins for distributed databases. Suppose that two organizations A and B each have a database table with entries authorized by third-party authorities. They would like to run an INNER JOIN SQL query between their tables such that each party only learns the entries that appear in the query’s result and not the remaining entries of the other party’s table. Additionally, they require that only properly authorized elements appear in the result. In other words, our PPSI scheme enables A and B to run authorized

	Overall	Additional overhead over PSI
Computation	$O(nm + n \log \log n)$	at most nm pairings
Bandwidth	$O(n)$	2 group elements
# Rounds	$O(1)$	1 round

Table 1: Efficiency of our construction in Section 3.3. n is the maximum number of elements per user and m is the maximum number of authorities per element. The complexities are calculated assuming that we use [14] as the underlying PSI scheme.

privacy-preserving SQL queries of the form:

```
SELECT * FROM TableA INNER JOIN TableB ON
    TableA.Attributei = TableB.Attributei
```

1.4 Related Work

A Private Set Intersection (PSI) protocol [8–17] allows two parties to find the intersection of their respective sets such that neither party can infer elements in the other party’s set that are not in the intersection. However, PSI protocols allow each party to place any element in their own set. A dishonest party can therefore insert fabricated elements in its set that she suspects the other party might have. The intersection will reveal if the other party indeed has those elements in its set.

To address this issue, Authorized Private Set Intersection (APSI) and variants [5, 7, 9] ensure that each party can only use elements certified by a trusted authority in the intersection protocol. Existing APSI protocols assume that for each party, there exists a single authority responsible for certifying all of its elements. Therefore, these schemes do not support rich privacy policies coming from multiple authorities, such as the application scenarios mentioned earlier.

1.5 Organization

For ease of exposition, in Sections 2 and 3, we first present definitions and constructions supporting symmetric policies. Recall that symmetric policies are where each party must be authorized by the same authority or set of authorities for a given element x . In other words, the correct authority (or authorities) for an element depends only on the element and not the party. Then, in Section 4, we show how our basic construction can be naturally extended to support more expressive privacy policies, including asymmetric policies, attributes, bundles, and disjunctive policies. We provide security definitions and a proof overview in Sections 2.3 and 3.4 respectively. The full proofs are in Appendix A.

2. PROBLEM DEFINITIONS

2.1 Notations and Terminology

Let \mathcal{U} denote a (countable) universe of all possible elements. Let Λ denote the set of all authorities.

As mentioned earlier, two parties, P_A and P_B , wish to find the intersection of their sets in a way that complies with certain privacy policies, that is, only when both parties have the appropriate authorizations for an element should it appear in the intersection.

Policy. A privacy policy defines which authority or authorities must sign an element for a given party. For ease of exposition, we will first focus on *symmetric policies*, where

each element needs to be authorized by one or more authorities, and the set of authorities is determined by the element itself, but is not dependent on the parties. Below are a few examples of symmetric policies.

- Claimed friendship with Alice needs to be authorized by Alice.
- Claimed membership in a social group needs to be authorized by the administrator(s) of the group.

As each element’s authorities is determined by the element itself, we can use a function F to describe symmetric policies. Formally, let $F : \mathcal{U} \rightarrow 2^{|\Lambda|}$ denote a publicly-known policy function that maps each element to the set of authorities that must sign it. For example, let $x \in \mathcal{U}$, if $F(x) = \{\text{auth}_1, \text{auth}_3\}$, this means that element x has to be signed by authorities auth_1 and auth_3 . One simple policy function is the identity function, e.g., each patient’s record must be authorized by the patient herself, or claimed friendship with a user must be authorized by that user herself.

We say that $x \in \mathcal{U}$ is an *authorized element* for party P , if party P has received all the necessary signatures for x , i.e., P has received a signature σ_i for every $\text{auth}_i \in F(x)$.

2.2 Basic Problem Definitions

Apart from the necessary setup and key generation functionalities, a PPSI scheme should offer two main functionalities: 1) a signature scheme allowing an authority auth_i to authorize an element x for a party P ; 2) a set intersection protocol that allows two parties to find the intersection of their authorized elements.

We now present formal definitions for a basic PPSI scheme supporting symmetric policies. A Policy-Enhanced Private Set Intersection (PPSI) scheme (supporting symmetric policies) should provide the following algorithms or protocols:

- **Setup(λ):** The **Setup** algorithm is run only once at system initialization to generate public parameters **param**. The input λ represents the security parameter.
- **KeyGen(**param**):** Each authority auth_i runs the **KeyGen** algorithm to generate a signing and verification key pair $(\text{sk}_i, \text{vk}_i)$. auth_i then announces the public verification key vk_i but keeps the private signing key sk_i to itself.
- **Authorize(**param**, sk_i , x , P_j):** The **Authorize** algorithm allows an authority auth_i to grant a party P_j a signature on a specific element x .
- **Intersect(P_i , P_j , S_i , S_j):** Let $S_i, S_j \subseteq \mathcal{U}$. **Intersect** is an interactive protocol run by any two parties P_i and P_j on input sets S_i and S_j respectively. When both parties are honest, and assuming that P_i and P_j both

n	Max number of elements in each party's set.
m	Max number of authorities per element.
\mathcal{U}	The set of all possible elements.
$x \in \mathcal{U}$	An element.
Λ	The set of all authorities
$\text{auth}_i \in \Lambda$	An authority that signs elements.
P_i	A party participating in our protocol.
S_i	P_i 's set.
I	The resulting intersection.
vk_i, sk_i	auth_i 's public verification key and secret signing key.
σ or σ_i	A signature issued by an authority.
attr	An attribute attached to a signature.
$F(x)$	Authorities for element x (for symmetric policies).
$F(x, P_i)$	Authorities for element x and party P_i (for asymmetric policies).
g	A random generator for the bilinear group.
$R_i = g^{r_i}$	P_i 's challenge for the other party.

Table 2: Table of notations.

have the necessary signatures for elements in S_i and S_j respectively, then both parties would learn the intersection $S_i \cap S_j$ at the end of the protocol.

2.3 Security Definitions

We prove the security of our protocol against a *malicious adversary*, who may deviate arbitrarily from the specified protocol. We define security by comparing what a malicious adversary can do in the *real* protocol execution against what the adversary can do in an *ideal* world. In the ideal-world execution, both parties would submit their sets to an imaginary trusted third-party denoted as T . The trusted party T would make sure that both parties have the correct authorizations on the elements they submitted. If a party submits an element without the necessary authorizations, T simply ignores that element. T then computes the intersection of the elements satisfying the privacy policy and returns the intersection to both parties. In the real-world, we do not use T and the parties communicate directly to execute the real set intersection protocol. Roughly speaking, the security definition implies that any attack that a polynomial-time adversary can perform in the real world is also possible in the ideal world. Intuitively, this suggests that the real-world set intersection protocol is as secure as the protocol in the ideal world that relies on a trusted third-party.

We now formally define the ideal functionality. The security definition involves multiple parties a subset of which is controlled by the adversary.

Authorize. T receives an authorization request from party P_i , requesting auth_j to authorize element x . T forwards the request to auth_j , who can either accept or reject the request. If auth_j accepts the authorization request, T replies *accept* to P_i and remembers that T has authorized P_i on element x . Otherwise, T replies *reject* to P_i .

SetIntersect. T receives a request from party P_i to perform set intersection with party P_j . T forwards the request to P_j . P_i and P_j now run an ideal set intersection protocol as below (unless P_j replies *abort*).

- i) P_i sends a set S_i to T , and T sends $|S_i|$ to P_j ; or ii) P_i sends *abort*.

- i) P_j sends a set S_j to T , and T sends $|S_j|$ to P_i ; or ii) P_j sends *abort*.
- T now checks whether each element in S_i and S_j has appropriate authorizations. Let $\hat{S}_i \subseteq S_i$ and $\hat{S}_j \subseteq S_j$ denote maximal subsets of S_i and S_j that have appropriate authorizations. T computes the intersection $I \leftarrow \hat{S}_i \cap \hat{S}_j$.
- T sends I to P_i , and P_i responds *ok* or *abort*.
- T sends I to P_j , and P_j responds *ok* or *abort*.

Definition 1 Let $E = (E_1, E_2, \dots, E_m)$ denote a sequence of events, where each E_i is of the form $(\text{Authorize}, P_i, \text{auth}_j)$ or $(\text{SetIntersect}, P_i, P_j)$. Let $\text{IDEAL}_{\mathcal{S}, E}$ denote the joint output distribution of all parties and the adversary \mathcal{S} in the ideal world under event sequence E . Let $\text{REAL}_{\mathcal{A}, E}$ denote the joint output distribution of all parties and the adversary \mathcal{A} in the real world under event sequence E .

We say that a PPSI scheme is secure, if for any polynomial-time adversary \mathcal{A} in the real world, there exists simulator \mathcal{S} in the ideal world, such that for any sequence of events E ,

$$\text{IDEAL}_{\mathcal{S}, E} \stackrel{c}{=} \text{REAL}_{\mathcal{A}, E}$$

where $\stackrel{c}{=}$ denotes computational indistinguishability.

Note that we cannot prevent an adversary from refusing to participate in the protocol or aborting in the middle of the protocol execution. As a result, our definition explicitly allows the ideal-world adversary to abort any time during the ideal-world protocol. Our definition also allows each party to use only a subset of their authorized elements as inputs the protocol.

Our protocol is not size-hiding, i.e., each party can learn the size of the other party's set. Therefore, in the ideal functionality, the trusted third-party reveals to each party the size of the other party's set. In particular, when both parties honestly use their authorized elements as inputs, i.e., $\hat{S}_A = S_A$ and $\hat{S}_B = S_B$, each party learns the size of the other party's authorized set. However, notice that a party can potentially fuzz the size of its set by padding the input set with random dummy elements for which it does not

possess appropriate authorizations. These dummy random elements will not appear in the final set intersection due to lack of authorizations; however, they can hide the number of authorized elements each party has.

3. CONSTRUCTION

3.1 Strawman Schemes

One strawman approach would be for the two parties to perform a regular Private Set Intersection (PSI) over the elements' signatures, thereby revealing the signed elements that they have in common. However, this requires that both parties have the exact same signature for the same element. This does not allow authorities to bind a signature to a specific party. The signature can thus be easily transferred to unauthorized parties.

It is conceivable that there are other solutions based on standard techniques for the problem than our construction. For example, we can imagine schemes based on secure multi-party computation, verifiable shuffles, and matching pairs of blinded elements. However, to the best of our knowledge, these approaches all tend to have much higher computational and bandwidth complexity than our construction which achieves $O(nm + n \log \log n)$ computational overhead and $O(n)$ bandwidth overhead – both almost linear in the number of elements n if we assume the number of authorities per element m to be a constant.

3.2 Preliminaries

Bilinear group. Our scheme utilizes a bilinear \mathbb{G} group of primary order p . There exists a non-degenerate bilinear mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ such that $\forall g_1, g_2 \in \mathbb{G}, \forall a, b \in \mathbb{Z}, e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. Our scheme relies on the following computational assumption.

Computational Bilinear Diffie-Hellman (CBDH) Assumption. Let $g \in \mathbb{G}$ denote a random generator of the group. The CBDH assumption posits the computational hardness of the following problem. Given randomly chosen $g^a, g^b, g^c \in \mathbb{G}$, compute $e(g, g)^{abc}$.

Private Set Intersection. A Private Set Intersection (PSI) protocol allows two parties to compute their set intersection without revealing other elements. Our protocol utilizes a standard PSI protocol (e.g., the scheme by Hazay and Nissim [14]) as a blackbox. We assume that the PSI protocol is secure in the malicious model, and refer the readers to [14] for a formal security definitions of PSI.

3.3 Main Construction

Our construction involves running a standard PSI protocol over special encodings formed as part of a challenge-response protocol. Below, we first describe our construction, including the key generation and authorization algorithm, as well as the intersection protocol. Then, in Section 3.4, we explain in detail how to construct the encodings used in the set intersection protocol, and the properties required for the encodings.

Setup. The Setup algorithm chooses a bilinear group \mathbb{G} of prime order p with pairing function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. It then chooses a random generator $g \in \mathbb{G}$. Next, it picks a hash function $H : \{0, 1\}^* \rightarrow \mathbb{G}$ which will be modelled as a random oracle. Finally, the Setup algorithm publishes a

description of the bilinear group, the generator g , as well as the hash function.

Key generation algorithm. To pick a signing and verification key pair, each authority auth_i randomly selects $\text{sk}_i \in_R \mathbb{Z}_p$. The verification key is $\text{vk}_i := g^{\text{sk}_i}$. Each authority i publishes its public verification key vk_i , but withholds its secret signing key sk_i .

Authorization algorithm. Let $H : \{0, 1\}^* \rightarrow \mathbb{G}$ denote a hash function modelled as a random oracle. We assume that each party P_i has a publicly-known unique name (e.g., an assigned name or a randomly generated identifier). Without risk of ambiguity, we overload the notation P_i to denote either the party or its name.

For authority auth_i to authorize element x for party P_j , auth_i computes the following BLS signature [4] and issues it to P_j .

$$\sigma \leftarrow H(x, P_j)^{\text{sk}_i}$$

In the above, the hash function is computed over the name of the element concatenated with the name of the receiving party. Notice that since the name of the party is incorporated into the signature, the signature cannot be transferred to another party.

Intersection protocol. Our protocol takes place in following four phases. The detailed construction is presented in Figure 1.

1. **Challenge Phase.** P_A sends P_B a random challenge R_A , and P_B sends P_A a random challenge R_B .
2. **Encoding Phase.** Each party computes an encoding for each element it possesses and with the appropriate authorizations. The encoding is dependent on the random challenges R_A and R_B . Figure 2 specifies the encoding function.
3. **Set Intersection Phase.** Both parties perform a standard Private Set Intersection (PSI) protocol using their respective encodings as the inputs. For the underlying PSI scheme we use the protocol described in [14].
4. **Recovery Phase.** At the end of the PSI protocol, each party learns the intersection of the encodings. Through the intersection of encodings, each party recovers the original elements in the intersection.

3.4 Encodings for Symmetric Policies

As shown in Figure 2, the encoding is computed as a product of multiple terms, where each term is the result of a bilinear pairing. Intuitively, the encodings satisfy the following properties:

- **Conformity.** If both parties have an element $x \in \mathcal{U}$ and the appropriate authorizations, their respective encodings of the element x will be the same. Therefore, the encoding for element x will appear in the intersection at the end of the PSI protocol.
- **Unforgeability.** If a party does not have appropriate authorizations for the element x , it is unable to forge the correct encoding for x . In this way, a dishonest party who does not possess authorizations for element x cannot find out whether the other party has element x .

Inputs: P_A, P_B each has sets S_A and S_B , with the appropriate authorizations.
Outputs: P_A, P_B each obtains $I := S_A \cap S_B$

Protocol:

1. P_A : Select random $r_A \in_R \mathbb{Z}_p$, let $R_A \leftarrow g^{r_A}$
 P_B : Select random $r_B \in_R \mathbb{Z}_p$, let $R_B \leftarrow g^{r_B}$
 $P_A \rightarrow P_B$: R_A
 $P_B \rightarrow P_A$: R_B
2. P_A : $C_A \leftarrow \{\text{EncodeElem}(x, r_A, R_B, P_A, P_B) \mid x \in S_A\}$
 P_B : $C_B \leftarrow \{\text{EncodeElem}(x, r_B, R_A, P_B, P_A) \mid x \in S_B\}$
3. $P_A \Leftrightarrow P_B$: Engage in a PSI protocol with input sets C_A and C_B respectively.
Both parties obtain the set $C' := C_A \cap C_B$ of encodings at the end of the PSI protocol.
4. P_A, P_B : Recover the intersection I from their encodings C' .

Figure 1: Intersection protocol.

Figure 2 The EncodeElem function.

```

function EncodeElem( $x, r_{\text{self}}, R_{\text{other}}, P_{\text{self}}, P_{\text{other}}$ )
   $c \leftarrow 1 \in \mathbb{G}_T$ 
  for  $\text{auth}_i \in F(x)$  do
    Let  $\sigma_i$  denote  $P_{\text{self}}$ 's signature on  $x$  from  $\text{auth}_i$ .
     $c \leftarrow c \cdot \mathbf{e}(\sigma_i, R_{\text{other}}) \cdot \mathbf{e}(H(x, P_{\text{other}}), \mathbf{vk}_i)^{r_{\text{self}}}$ 
  end for
  return  $c$ 
end function

```

$x \in \mathcal{U}$ is the element to encode. $r_{\text{self}} \in \mathbb{Z}_p$ is the random exponent generated by the party itself. $R_{\text{other}} \in \mathbb{G}$ is the random challenge received from the other party. P_{self} and P_{other} represent the names of the party itself and the party it is communicating with.

The encoding contains two corresponding terms for each element-authority pair (x, auth_i) :

- *Response to the other party's challenge.* The first term, $\mathbf{e}(\sigma_i, R_{\text{other}})$, is a response to the other party's challenge R_{other} . Intuitively, if a party does not possess an authorization from auth_i , then it will not be able to generate this part of the encoding.
- *Anticipated response from the other party to one's own challenge.* The second term, $\mathbf{e}(H(x, P_{\text{other}}), \mathbf{vk}_i)^{r_{\text{self}}}$, is the anticipated response from the other party for one's own challenge R_{self} . Note that a party is always able to compute the anticipated response for its own challenge, even without knowing the other party's signature, since a party knows the exponent r_{self} of a challenge generated by itself. Let $\sigma'_i := H(x, P_{\text{other}})^{\text{sk}_i}$ denote the signature given to P_{other} from auth_i on element x . It is not hard to see that

$$\mathbf{e}(\sigma'_i, R_{\text{self}}) = \mathbf{e}(H(x, P_{\text{other}}), \mathbf{vk}_i)^{r_{\text{self}}}$$

In other words, if P_{other} has the correct signature from auth_i , its actual response to P_{self} 's challenge should match the response anticipated by P_{self} . In summary, this term enforces that the other party can only compute the encoding if it has a signature from the correct authority.

Theorem 1 *The PPSI scheme described in this section is secure against malicious adversaries, assuming 1) the un-*

derlying PSI protocol is simulatable in the malicious model; 2) the Computational Bilinear Diffie-Hellman (CBDH) assumption holds in the bilinear group \mathbb{G} ; and 3) the hash function H is a random oracle.

Proof overview. We now give an overview of our proof, and defer the detailed proof to Appendix A. We first define a hybrid protocol by replacing the PSI protocol with an ideal functionality for PSI. Due to the sequential modular composition theorems by Canetti [6], it suffices to prove that the hybrid protocol securely computes the ideal functionality defined in Section 2. We then construct a simulator which is given black-box access to a hybrid-world adversary \mathcal{A} . We show that if the encoding scheme is unforgeable in some sense, then the joint output distribution of all parties in the ideal world is indistinguishable from the joint output distribution in the hybrid world.

The description of our protocol in Figure 1 does not hide the number of authorized elements from the other party. If this number is also considered sensitive, a party can pad its set of encodings with random dummy encodings, and use the resulting set as inputs to the PSI protocol. Effectively, this reveals to the other party the total number of authorized elements and dummy elements.

Another possible method to hide the size of one's set is to use a Size-Hiding PSI protocol in place of the PSI protocol used in our construction. Our security proofs would still hold if the Size-Hiding PSI protocol is simulatable in the malicious model. Notably, Ateniese *et al.* recently propose a Size-Hiding PSI protocol secure under the semi-honest model [2]. Therefore, it is conceivable that a Size-Hiding PSI protocol in the malicious model will become available in the near future.

4. EXTENSIONS FOR RICHER POLICIES

In this section, we describe how to compute the encodings in the intersection protocol for different kinds of policies. Our main construction in Section 3.3 supports simple symmetric policies, and we now incrementally add support for asymmetric policies, attributes, bundles, and DNFs.

4.1 Asymmetric Policies

So far we have focused on symmetric policies, where the authorities associated with each element depend on the element itself.

Figure 3 The EncodeElem function for asymmetric policies.

```

function EncodeElem( $x, r_{\text{self}}, R_{\text{other}}, P_{\text{self}}, P_{\text{other}}$ )
   $c \leftarrow 1 \in \mathbb{G}_T$ 
  for  $\text{auth}_i \in F(x, P_{\text{self}})$  do
    Let  $\sigma_i$  denote  $\text{auth}_i$ 's signature for  $P_{\text{self}}$  on  $x$ .
     $c \leftarrow c \cdot e(\sigma_i, R_{\text{other}})$ 
  end for
  for  $\text{auth}_i \in F(x, P_{\text{other}})$  do
    Let  $\sigma_i$  denote  $\text{auth}_i$ 's signature for  $P_{\text{other}}$  on  $x$ .
     $c \leftarrow c \cdot e(H(x, P_{\text{other}}), \text{vk}_i)^{r_{\text{self}}}$ 
  end for
  return  $c$ 
end function

```

In other application scenarios (such as Use Case 3 in Section 6), the right authority may depend on both the element and the party performing set intersection.

Let \mathcal{U} denote a countable universe of elements, let \mathcal{P} denote the set of all parties, and let Λ denote the set of authorities. We denote asymmetric policies using a publicly known policy function $F : \mathcal{U} \times \mathcal{P} \rightarrow 2^{|\Lambda|}$. F outputs the set of appropriate authorities given an element and a party. For example, if $F(x, P) = \{\text{auth}_1, \text{auth}_2\}$, this means that auth_1 and auth_2 must sign element x for party P .

Figure 3 describes how to modify the EncodeElem function to support asymmetric policies. The idea is essentially the same as the symmetric case. If auth_i must sign element x for party P_{self} , then P_{self} computes the term $e(\sigma_i, R_{\text{other}})$, which is a response to the challenge from P_{other} . If auth_j must sign element x for party P_{other} , then P_{self} computes the term $e(H(x, P_{\text{other}}), \text{vk}_j)^{r_{\text{self}}}$, which is the anticipated response from P_{other} to one's own challenge. The final encoding for an element is basically the product of all responses to the other party's challenge, and all anticipated responses from the other party.

4.2 Attributes

Authorities may wish to attach attributes to an element when making authorizations. For example, attributes may be used to determine the type or level of authorization given. Use Case 4 in Appendix 6 illustrates a possible healthcare application where attributes are used to specify the sensitivity of medical records. We show that our construction can be extended to support policy attributes.

Let \mathcal{V} denote the set of all possible attributes. Suppose the a public function $F : \mathcal{U} \times \mathcal{P} \rightarrow 2^{\Lambda \times \mathcal{V}}$ exists which outputs the necessary (authority, attribute) pairs given an element and a party. For example, if

$$F(x, P) = \{(\text{auth}_1, \text{attr}_1), (\text{auth}_1, \text{attr}_2), (\text{auth}_5, \text{attr}_3)\},$$

it means that for party P to be a rightful owner of element x , it is necessary that auth_1 has signed element x with attributes attr_1 and attr_2 for party P , and auth_5 has signed element x with attribute attr_3 for party P .

To support attributes, first, the authorities need to incorporate the attribute values in the hash when computing signatures. To authorize element x with attribute attr to party P , auth_i now computes the following signature:

$$\sigma \leftarrow H(x, \text{attr}, P)^{\text{sk}_i}$$

Second, the EncodeElem function needs to be modified to

Figure 4 The EncodeElem function supporting attributes and asymmetric policies.

```

function EncodeElem( $x, r_{\text{self}}, R_{\text{other}}, P_{\text{self}}, P_{\text{other}}$ )
   $c \leftarrow 1 \in \mathbb{G}_T$ 
  for  $(\text{auth}_i, \text{attr}) \in F(x, P_{\text{self}})$  do
    Let  $\sigma_i$  denote  $\text{auth}_i$ 's signature for  $P_{\text{self}}$  on  $x$  and attribute  $\text{attr}$ .
     $c \leftarrow c \cdot e(\sigma_i, R_{\text{other}})$ 
  end for
  for  $(\text{auth}_i, \text{attr}) \in F(x, P_{\text{other}})$  do
    Let  $\sigma_i$  denote  $\text{auth}_i$ 's signature for  $P_{\text{other}}$  on  $x$  and attribute  $\text{attr}$ .
     $c \leftarrow c \cdot e(H(x, \text{attr}, P_{\text{other}}), g^{\text{vk}_i})^{r_{\text{self}}}$ 
  end for
  return  $c$ 
end function

```

incorporate the attributes as in Figure 4.

4.3 Bundles

A group of elements may form a bundle. The bundle should appear in the intersection if both parties have all elements in the bundle, as well as the appropriate authorizations. Otherwise, the bundle should not appear in the intersection, and neither party should learn any partial information about elements in the bundle that the other party has.

Our scheme can be easily adapted to handle bundles by combining the encoding of each element of the bundle to produce a single encoding for the entire bundle. Specifically, the bundle's encoding is the product of the encodings of its elements.

4.4 Disjunctions and DNFs

So far, we have considered conjunctive policies. More generally, policies may also contain disjunctions, or Disjunctive Normal Forms (DNFs). For example, Section 6, Case 6 presents a scenario where for hospital A to share Carol's record with hospital B , either the record has low sensitivity, and hospital B has permissions to receive low sensitivity records from Carol, or the record is cardiology related, and hospital B has permissions to retrieve Carol's cardiology records.

As another example, imagine two online stores (e.g., Dell and Newegg) want to investigate a consumption pattern of their shared customers. Specifically, they want to determine which customers have bought both a computer from Dell and a monitor from Newegg. Therefore, they need to perform a set intersection operation on their sales datasets. Meanwhile, to prevent each company from inserting fictitious records, each sales record must be authorized by a recognized credit company, Mastercard or Visa.

In general, for parties P_A and P_B to share an element x , a DNF-style policy of the following form must be satisfied:

$$\text{policy} := C_1 \vee C_2 \vee \dots \vee C_k$$

where each C_i ($1 \leq i \leq k$) is a conjunctive clause of the form:

$$(\text{auth}_{i_1}, P_A, x, \text{attr}_1) \wedge \dots \wedge (\text{auth}_{i_\ell}, P_A, x, \text{attr}_\ell) \\ \wedge (\text{auth}_{j_1}, P_B, x, \text{attr}_1) \wedge \dots \wedge (\text{auth}_{j_{\ell'}}, P_B, x, \text{attr}_{\ell'})$$

In the above, each tuple $(\text{auth}_i, P, x, \text{attr})$ means that " auth_i gave authorizations to party P on element x with attribute

attr”. Specifically, each conjunctive clause specifies the policies for party P_A and P_B respectively.

Our basic construction can be extended to support DNFs, with the caveat that each party reveals to the other party which conjunctive clause is satisfied for an element. The idea is quite straightforward: for each conjunctive clause, each party uses the algorithm described in Figure 4 to compute an encoding. The encoding for an element is now the union of all encodings corresponding to all conjunctive clauses. Furthermore, each party will use the union of all encodings for all elements as inputs to the PSI protocol.

5. PERFORMANCE

In this section, we first explain a simple optimization for reducing the total number of bilinear pairing operations. Then, we present the asymptotic complexities and experimental performance of our protocol.

5.1 Optimization

When generating the encodings of the elements (see Figures 2, 3, and 4), we need to compute an expression of the form

$$e(\sigma_1, R_{\text{other}}) \cdot e(\sigma_2, R_{\text{other}}) \cdots e(\sigma_k, R_{\text{other}})$$

Because bilinear pairings are much more expensive than multiplication, we can compute the same quantity much more efficiently by instead calculating

$$e(\sigma_1 \cdot \sigma_2 \cdots \sigma_k, R_{\text{other}})$$

For our symmetric policy construction, this optimization reduces the number of pairings almost in half (as the trick does not apply to the terms representing anticipated responses from the other party).

5.2 Asymptotic Complexities

We first analyze the performance of our basic construction (described in Section 3.3) supporting symmetric policies. Later, in Section 5.4, we discuss the performance of the various extensions (described in Section 4).

The efficiency of our protocol depends on both the number of elements (n) and the number of authorities per element (m). We now present asymptotic bounds for the amount of computation, amount of bandwidth, and the number of communication rounds.

Computation: $O(nm + n \log \log n)$. The encoding phase performs a constant number of operations for each element-authority pair and is hence $O(nm)$. It computes a single encoding for each element resulting in $O(n)$ encodings. Those encodings are the input for the PSI phase, and by using the protocol by Hazay and Nissim [14], we can perform the PSI phase in $O(n \log \log n)$ time. The recovery phase is trivially $O(nm)$ and the challenge phase is $O(1)$. Summing up the above, the total computation is $O(nm + n \log \log n)$.

Bandwidth: $O(n)$. The communication between the two parties consists of the PSI protocol’s communication and the two group elements sent during the challenge phase. Since the input size for the PSI is n , using the PSI protocol by Hazay and Nissim [14], the bandwidth overhead for the PSI phase is $O(n)$. Therefore, the combined communication bandwidth for our scheme is $O(n)$.

m	1	2	3	4	5
average	1.70	3.10	4.45	5.65	7.07
std. dev.	0.06	0.17	0.22	0.04	0.27

Table 3: The time (in ms) for encoding an element given the number of authorities for that element. These results are for the symmetric policy construction with attributes.

Rounds: $O(1)$. The PSI protocol by Hazay and Nissim [14] consists of $O(1)$ communication rounds. We add one additional round for the challenge phase.

Note that our construction introduces only a small overhead on top of PSI, namely, a single round of extra communication where 2 group elements are exchanged, and at most nm bilinear pairings. And with this small additional overhead, we provide the ability to support rich privacy policies previously not possible with existing PSI and APSI schemes.

5.3 Empirical Performance

Our protocol can be broken down into two time consuming phases: (1) encoding elements, and (2) performing standard Private Set Intersection (PSI). There is a large body of existing work on building efficient PSI protocols [8–12, 14–17], and one can plug into our construction any existing PSI protocol that is fully simulatable under a malicious adversary model. Therefore, our experimental analysis below focuses on the additional overhead introduced by the encoding phase.

We generated 2,000 random elements with attributes and then computed the signatures for 2 parties by 5 authorities. We used different authorities for each element-authority pair, hence we have the total number of authorities $|\Lambda| = 10,000$. We set the maximum number of authorities per element to be $m = 5$. We then varied $m = 1, \dots, 5$ by choosing a random subsets of the corresponding authorities for each element, and computed all of the the element encodings in parallel.

After repeating this experiment 20 times, we calculated the average encoding time per element and standard deviation. The results are shown in Table 3.

Experiment Setup. Our experiment was implemented in C# and was run on 64-bit Windows 7 with an Intel Core i7 3.33 GHz CPU and 12GB of RAM (although the experiment used much less memory). For all of the pairing and elliptic curve operations, we used the Pairing Based Crypto Library [18].

5.4 Performance for Rich Policies

So far, we focused on the performance of the basic construction supporting symmetric policies. The performance of our protocols supporting richer policies can be analyzed in a similar fashion.

Asymmetric Policies. The performance for asymmetric policies is essentially the same as the performance for symmetric policies. Therefore, encoding n elements each having at most m authorities per element using an asymmetric policy is at least as fast as encoding using a symmetric policy for the same n and m .

Attributes. With attribute-enriched policies, the number of bilinear pairings is the number of (element, authority,

attribute) tuples for each party.

Bundles. The cost of encoding a bundle scales linearly with the number of elements. For example, the cost of encoding a bundle of b elements is b times the cost of encoding a single element. This is due to the fact that the elements of the bundle have to be first encoded individually. Combining them incurs a series of elliptic point multiplications, but their cost is significantly outweighed by the pairing function that is applied to each element.

DNF policies. Each DNF policy consists of multiple conjunctive clauses. The cost of encoding an element under a DNF policy is simply the sum of the cost of encoding each conjunctive clause, where the cost for encoding a conjunctive clause has been discussed earlier – depending on whether the conjunctive clause is symmetric, asymmetric, attribute-enriched, etc.

With a DNF policy consisting of k conjunctive clauses, the encoding for an element will consist of k group elements instead of one.

To summarize, suppose the maximum number of conjunctive clauses for a DNF policy is k , and the maximum number of literals for a conjunctive clause is m . Then, the communication overhead of our protocol will be $O(nk)$, and the computational overhead will be $O(nmk + nk \log(nk))$.

6. USE CASES IN HEALTHCARE

In this section, we describe some motivating use cases in the healthcare space. These use cases demonstrate the rich policy semantics that we may wish to support. We choose the medical space, as privacy is a particularly sensitive issue when it comes to sharing health records. Furthermore, there are various government and organizational regulations that seek to protect the privacy of medical records. Beyond the healthcare space, our constructions are also valuable in a wide range of application domains, as mentioned earlier in Section 1.

6.1 Symmetric Policies

We consider the following type of policies to be symmetric. For two parties A and B to share an element x , both A and B must have authorizations from the *same* authority or set of authorities.

Below we describe two use cases where symmetric policies are used. In Case 1, each element must be authorized by a single authority depending on the element. In Case 2, each element must be authorized by multiple authorities depending on the element.

Case 1: Policy-compliant medical information sharing. Imagine that two hospitals A and B wish to share information about their common patients. This can enhance doctors’ understanding about a patient’s medical history and therefore improve diagnosis accuracy. Due to privacy regulations such as HIPAA, the two hospitals can only share information about Carol if both have Carol’s consent.

In this scenario, if we consider each patient to be an element in the set, then each element must be authorized by the patient herself in order for the element to appear in the set intersection.

Case 2: Detecting duplicate claims. Imagine that two medical insurance companies wish to detect insurance fraud. Each insurance company has a database of claim records,

where each record is a tuple (PATIENT, DOCTOR, DATE OF VISIT). The two insurance companies wish to detect duplicate claims. To protect the privacy of honest patients and to prevent each insurance company from using fictitious records, we require that each claim record must be authenticated by a patient and the corresponding doctor’s office.

In this scenario, each claim record can be considered as an element, and an element must be authenticated by two authorities, namely, the patient and the doctor’s office.

6.2 Asymmetric Policies

Asymmetric policies refer to cases where the policy requirements are asymmetric for the two parties involved in the protocol.

Case 3: Policy-compliant medical information exchange. A hospital A wishes to retrieve medical information about its patients from a cloud service provider such as Google Health. However, the cloud service provider can only share Carol’s record with hospital A , if 1) hospital A has permission from Carol to retrieve her records; and 2) the cloud service provider has permissions from Carol and her primary doctor to share Carol’s medical records.

In this example, the policy requirements for the cloud service provider and the hospital are asymmetric.

6.3 Attributes, Bundles, and Disjunctions

Attributes may be attached to authorizations made to a party, such as the following example.

Case 4: Sensitivity level of medical records. Some medical records may be more privacy sensitive than others. For example, a medical record related to HIV is generally more privacy sensitive than a record related to the common cold.

The patient or hospital might attach a sensitivity level *low*, *medium*, or *high* to a medical record. The patient can then sign-off on the record by authorizing it. A patient Carol may authorize a hospital B to obtain records marked with only “low” or “medium” sensitivity level.

Case 5: Bundles. Sometimes multiple elements may form a bundle. A bundle appears in the set intersection if and only if both parties have all elements in a bundle.

Two hospitals A and B may be interested in sharing patient’s prescription records for patients satisfying a certain criterion. Each prescription record contains metadata of the form (PATIENT, DRUG). Hospitals A and B wish to share prescription records for patients who have been treated by both drug 1 and drug 2. As before, each prescription record must be authorized by the corresponding patient.

In this case, the records “(Carol, Drug 1)” and “(Carol, Drug 2)” form a bundle. The bundle should appear in the set intersection if and only if A and B each possesses both records in the bundle. Otherwise, they should not be included in the set intersection.

Case 6: Disjunctions. The previous cases are examples of conjunctive policies. In other words, for an element to appear in the set intersection, a set of requirements has to be satisfied simultaneously.

We may also have disjunctive policies, or more generally, Disjunctive Normal Forms (DNFs). For example, for hospital A to share Carol’s record with hospital B , the following disjunctive policy must be satisfied:

- Either Carol’s medical record is marked by Carol with low sensitivity, and Carol has given permission to hospital B to retrieve her low-sensitivity medical records;
- Or Carol’s medical record has been classified as “cardiology” by an RHIO, and meanwhile, hospital B has Carol’s permission to her cardiology related records.

7. CONCLUSION

We introduced a new cryptographic paradigm for private set intersection with rich policies, allowing two parties to selectively share data while satisfying privacy policies. Our protocol ensures that only properly authorized elements which satisfy certain privacy policies appear in the set intersection. Our protocols support rich policies, including conjunctive and disjunctive policies, attribute-enriched policies, asymmetric policies, and bundles of elements. We prove that our scheme is secure under the malicious model, given the CBDH assumption, the security of the underlying PSI protocol, and assuming the random oracle model.

8. REFERENCES

- [1] A. I. Anton, J. B. Eart, M. W. Vail, N. Jain, C. M. Gheen, and J. M. Frink. HIPAA’s effect on web site privacy policies. *IEEE Security and Privacy*, 5, January 2007.
- [2] G. Ateniese, E. D. Cristofaro, and G. Tsudik. Size-hiding private set intersection. Cryptology ePrint Archive, Report 2010/220, 2010. <http://eprint.iacr.org/>.
- [3] E. Bertino, B. C. Ooi, Y. Yang, and R. H. Deng. Privacy and ownership preserving of outsourced medical data. In *ICDE*, 2005.
- [4] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. pages 514–532. Springer-Verlag, 2001.
- [5] J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *Financial Cryptography*, 2009.
- [6] R. Canetti. Security and composition of multi-party cryptographic protocols. *JOURNAL OF CRYPTOLOGY*, 1998.
- [7] E. Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-preserving policy-based information transfer. In *PETS*, 2009.
- [8] E. D. Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Asiacrypt*, 2010.
- [9] E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, 2010.
- [10] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *ACNS*, 2009.
- [11] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, 2005.
- [12] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt*, 2004.
- [13] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, 2008.
- [14] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography*, 2010.
- [15] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, 2009.
- [16] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, 2010.
- [17] L. Kissner and D. Song. Private and threshold set-intersection. In *CRYPTO*, 2005.
- [18] B. Lynn. Pairing-based cryptography library. <http://crypto.stanford.edu/abc/>.

APPENDIX

A. PROOFS OF SECURITY

Suppose the PSI protocol we use in the protocol is fully simulatable under the malicious model. Due to the sequential modular composition theorems by Canetti [6], we can replace the PSI module in our protocol with the ideal functionality for PSI. We refer to the resulting protocol as the hybrid protocol. We formally describe the hybrid protocol below. Although not explicitly stated, parties P_A and P_B may abort the protocol at any message boundary.

- P_A picks random $r_A \in \mathbb{Z}_p$, and sends to P_B the value $R_A := g^{r_A} \in \mathbb{G}$.
- P_B picks random $r_B \in \mathbb{Z}_p$, and sends to P_A the value $R_B := g^{r_B} \in \mathbb{G}$.
- P_A computes

$$C_A \leftarrow \{\text{EncodeElem}(x, r_A, R_B, P_A, P_B) \mid x \in S_A\},$$
 and sends C_A to T_{PSI} . T_{PSI} now sends $|C_A|$ to P_B .
- P_B computes

$$C_B \leftarrow \{\text{EncodeElem}(x, r_B, R_A, P_B, P_A) \mid x \in S_B\},$$
 and sends C_B to T_{PSI} . T_{PSI} now sends $|C_B|$ to P_A .
- T_{PSI} computes $C' := C_A \cap C_B$, and sends C' to P_A .
- T_{PSI} sends C' to P_B .

Due to the sequential modular composition theorems by Canetti [6], it suffices to show that the hybrid protocol is secure as stated by Lemma 1.

Definition 2 Let E denote an event sequence. Let $\text{IDEAL}_{S,E}$ denote the joint output distribution of all parties and the adversary \mathcal{S} in the ideal world, under event sequence E . Let $\text{HYBRID}_{A,E}$ denote the joint output distribution of all parties and the adversary \mathcal{A} in the hybrid world, under event sequence E . We say that the hybrid protocol securely computes the ideal functionality defined in Section 2.3, if for any polynomial-time adversary \mathcal{A} in the hybrid world, there exists simulator \mathcal{S} in the ideal world, such that for any sequence of events E ,

$$\text{IDEAL}_{S,E} \stackrel{c}{=} \text{HYBRID}_{A,E}$$

where $\stackrel{c}{=}$ denotes computational indistinguishability.

Lemma 1 *Assume that the CBDH assumption holds in the bilinear group \mathbb{G} , and the hash function H is a random oracle. Then, the hybrid protocol described earlier securely computes the ideal functionality defined in Section 2.3.*

In the following sections, we prove Lemma 1.

A.1 Simulator Construction

We build an ideal-world simulator \mathcal{S} who is given black-box access to a hybrid-world adversary \mathcal{A} .

\mathcal{S} starts out by choosing a bilinear group \mathbb{G} of prime order p , with a pairing function denoted $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. It also picks a random generator $g \in \mathbb{G}$. \mathcal{S} gives these system parameters to \mathcal{A} .

We assume that all authorities are honest. \mathcal{S} now chooses the signing and verification key pair $(\text{sk}_i, \text{vk}_i)$ for every authority, and gives the verification keys vk_i to \mathcal{A} . \mathcal{S} interacts with \mathcal{A} as below:

- *Hash query.* Upon receiving a hash query on input (x, P) , \mathcal{S} checks to see if that query has been made before. If so, return the same answer. If not, \mathcal{S} generates a random element in $g' \in \mathbb{G}$ and returns g' . \mathcal{S} also remembers the query and its answer.
- *Authorize.* Whenever \mathcal{A} asks auth_i to sign element x for a corrupted party P_A , \mathcal{S} first makes a hash query on input (x, P_A) . After determining the hash, \mathcal{S} computes the signature and returns it to the adversary. As \mathcal{S} knows the signing keys of all parties, \mathcal{S} can compute a signature for any requested element.
- *Set intersection.* Whenever \mathcal{A} asks to perform the set intersection protocol between a corrupted party P_A (controlled by the adversary), and an honest party P_B , \mathcal{S} performs the following simulation. \mathcal{A} may abort any time during the following execution, in which case \mathcal{S} also sends **abort** to the trusted third-party T .

\mathcal{S} chooses some $R_B \in \mathbb{G}$ and sends it to \mathcal{A} . \mathcal{A} chooses some $R_A \in \mathbb{G}$ and sends it to \mathcal{S} . \mathcal{A} now outputs a set of encodings. From these encodings, \mathcal{S} can recover which elements they correspond to. To do this, \mathcal{S} basically iterates over the hash queries that \mathcal{A} has made, computes the expected encodings for these elements, and checks to see if they match any encodings submitted by the adversary. It is not hard to see that \mathcal{S} can compute all encodings, as it knows the secret signing keys sk_i for all authorities. \mathcal{S} then sends these elements (recovered from encodings) to the trusted third-party T . T replies with $|S_B|$ the length of P_B 's set, followed by elements in the intersection represented by the set I . \mathcal{S} sends \mathcal{A} the length $|S_B|$. Furthermore, \mathcal{S} computes the encodings for elements in I , and returns these encodings to \mathcal{A} .

Clearly, unless \mathcal{A} is able to forge an encoding for an element for which it does not possess an appropriate signature, then the joint outputs of all parties in the ideal world are identically distributed as those in the hybrid protocol. Notice that if \mathcal{A} did forge an encoding for some element z for which it had not made an authorization query, then the ideal-world trusted third-party T would filter out that element from the resulting set intersection, causing the output distribution to be different in the ideal protocol from the hybrid protocol.

A.2 Unforgeability of Encodings

Lemma 2 *Assume that the CBDH assumption holds in the group \mathbb{G} , and the hash function H is a random oracle. Let \mathcal{A} denote polynomial-time adversary in the hybrid protocol, who has full control of all corrupted parties. Let P_A denote a corrupted party, and assume that P_A has not received auth_i 's signature on element x . Then, during a set intersection protocol between P_A and any honest party P_B , \mathcal{A} is unable to compute the correct encoding $\text{EncodeElem}(x, r_A, R_B, P_A, P_B)$ except with negligible probability. In the above, $R_B \in_R \mathbb{G}$ is chosen at random by P_B , and $r_A \in \mathbb{Z}_p$ is chosen arbitrarily by the adversary \mathcal{A} .*

We now prove Lemma 2 formally. Imagine a simulator \mathcal{S}' who receives a CBDH instance (g, g^a, g^b, g^c) . We will show that if there exists a polynomial-time adversary \mathcal{A} who can successfully forge an encoding with non-negligible probability, then \mathcal{S}' can leverage \mathcal{A} to break the CBDH assumption, and compute $e(g, g)^{abc}$ with non-negligible probability.

\mathcal{S}' gives to \mathcal{A} a description of the bilinear group \mathbb{G} , the pairing function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and the generator g . All these parameters are inherited from the CBDH instance. \mathcal{S}' first chooses a random authority auth_{i^*} , and sets its public verification key $\text{vk}_{i^*} := g^a$, where g^a is from the CBDH instance. Note that \mathcal{S}' does not know the secret signing key sk_{i^*} . Nevertheless, the \mathcal{S}' is able to perform the simulation. For all other authorities, the simulator \mathcal{S}' picks their signing and verification key pairs as normal, and gives all the verification keys to the adversary \mathcal{A} .

- *Hash query.* Except for one hash query (z^*, P^*) chosen at random by \mathcal{S}' , \mathcal{S}' picks a fresh random exponent $\rho \in \mathbb{Z}_p$, and return g^ρ to \mathcal{A} . The simulator remembers the query and ρ .

For the randomly chosen hash query (z^*, P^*) the simulator \mathcal{S}' returns g^c from the CBDH instance to \mathcal{A} .

- *Authorize.* If \mathcal{A} asks for a signature on element z^* for a corrupted party P^* , the \mathcal{S}' aborts. The probability of not aborting is at least $\frac{1}{q}$, where q is the total number of hash queries and authorize queries made by \mathcal{A} .

If the authorization request is for $(z, P) \neq (z^*, P^*)$, the simulator \mathcal{S}' first makes a hash query to obtain the hash value $H(z, P)$. Next the simulator \mathcal{S}' computes the signature. Note that in this case, the simulator is always able to compute the correct signature even when the signer is auth_{i^*} for whom \mathcal{S}' does not possess the secret signing key. This is due to the fact that the simulator \mathcal{S}' picked the exponent for $H(z, P) = g^\rho$ in a hash query. With knowledge of ρ , the simulator can compute the signature as $\text{vk}_{i^*}^\rho$.

- *Set intersection.* \mathcal{A} forges an encoding at least once during a set intersection protocol. Suppose that the adversary performs set intersection protocols at most q' times. The simulator guesses at random that \mathcal{A} will make the first forgery during the j^* -th protocol run, where $1 \leq j^* \leq q'$. The guess is correct with probability at least $\frac{1}{q'}$.

We will assume that the j^* -th protocol run is between an honest party P_B and corrupted party P^* (which the simulator tries to guess in the hash queries) – if this is not the case, \mathcal{S}' simply aborts. Now during the j^* -th

protocol run, \mathcal{S}' sends $R_B := g^b$ to \mathcal{A} , where g^b is from the CBDH instance. \mathcal{A} sends R_A to \mathcal{S}' . \mathcal{A} now outputs m encodings, and the simulator \mathcal{S}' randomly guesses one of them to be the forged encoding, denoted as c^* . We henceforth assume that c^* encodes the element z^* – and this happens with at least $\frac{1}{q}$ probability, where q is the total number of hash queries and authorization requests made by \mathcal{A} . Note that c^* will contain two terms for every authority associated with the element z^* – one term for each party. Except for the term $e(H(z^*, P^*)^{\text{sk}_{i^*}}, R_B) = e(g, g)^{abc}$, \mathcal{S}' can compute and divide out all other terms. Therefore, what remains will be the simulator's guess of $e(g, g)^{abc}$.

In the above simulation, \mathcal{S}' will succeed in guessing $e(g, g)^{abc}$ if and only if:

- \mathcal{A} is successful in forgery.
- \mathcal{S}' correctly guesses that \mathcal{A} 's first successful forgery is an encoding for corrupted party P^* , and element z^* .
- \mathcal{S}' correctly guesses that \mathcal{A} 's first successful forgery happens during the j^* -th protocol interaction, and for the k^* -th element in that protocol interaction.

The probability that the above happens is at least $\frac{\epsilon}{qq'm}$, where q is the total number of hash queries and authorization requests made by the adversary, q' is the total number of set intersection protocols the adversary engages in, m is the maximum number of encodings produced by the adversary in every protocol interaction, and ϵ is the success probability of \mathcal{A} . As the adversary \mathcal{A} is polynomially bounded, this means that the simulator \mathcal{S}' succeeds with probability at least $\frac{\epsilon}{\text{poly}(\lambda)}$.