arXiv:1604.06959v3 [cs.CR] 28 Feb 2017

Privacy, Discovery, and Authentication for the Internet of Things

(Extended Version)

David J. Wu[1], Ankur Taly[2], Asim Shankar[2], and Dan Boneh[1]

[1]Stanford University
[2]Google

Abstract

Automatic service discovery is essential to realizing the full potential of the Internet of Things (IoT). While discovery protocols like Multicast DNS, Apple AirDrop, and Bluetooth Low Energy have gained widespread adoption across both IoT and mobile devices, most of these protocols do not offer any form of privacy control for the service, and often leak sensitive information such as service type, device hostname, device owner's identity, and more in the clear.

To address the need for better privacy in both the IoT and the mobile landscape, we develop two protocols for private service discovery and private mutual authentication. Our protocols provide private and authentic service advertisements, zero round-trip (0-RTT) mutual authentication, and are provably secure in the Canetti-Krawczyk key-exchange model. In contrast to alternatives, our protocols are lightweight and require minimal modification to existing key-exchange protocols. We integrate our protocols into an existing open-source distributed applications framework, and provide benchmarks on multiple hardware platforms: Intel Edisons, Raspberry Pis, smartphones, laptops, and desktops. Finally, we discuss some privacy limitations of the Apple AirDrop protocol (a peer-to-peer file sharing mechanism) and show how to improve the privacy of Apple AirDrop using our private mutual authentication protocol.

1 Introduction

Consider a smart home with dozens of IoT devices: an alarm system, a nanny camera, health monitoring devices, house controls (e.g., lighting, heating), and electronics. Many of these devices need to be controlled by multiple people, including residents, guests, employees, and repairmen. The devices must be easily discoverable by all these people.

To provide a good experience, IoT devices advertise the services they offer using a service discovery mechanism. Examples include Mutlticast DNS (mDNS) [CK13a, CK13b], Apple Bonjour [Bon13], Bluetooth Low Energy (BLE) [BLE14], and Universal Plug-N-Play (UPnP) [UPn15]. These mechanisms require only a broadcast communication channel between the devices (unlike older discovery protocols [Jin13, CZH+99, ZMN05] that need a directory service). Moreover, these protocols adhere to the zero configuration networking charter (*Zeroconf*) [Zer04] and can operate with minimal user intervention.

Privacy is an important feature often missing in zero-configuration service discovery protocols (e.g., Zeroconf) [KW14a, KW14b, KBSW13, PGM+07]. Services broadcast extensive information about themselves in the clear to make it easy for clients to discover them. Advertisements often

include sensitive information such as service type, device hostname, and the device owner's identity. This poses a threat when the service is running on a private device (e.g., an alarm system or a smart watch). Identities obtained from personal devices can be used for user profiling, tracking, and launching social engineering attacks. A recent study [KBSW13] revealed that 59% of all devices advertise their owner's name in the clear, which is considered harmful by more than 90% of the device owners. Indeed, one would not want random visitors, or passerbys, to "discover" the alarm system in their home. Only authorized clients, such as the home owner and her family, a technician, or local police, should be able to discover this device.

In this work, we address this problem by building a new discovery and authentication mechanism that respects the privacy of both sides.

Private service discovery. Our goal is to ensure that services are only discoverable by an authorized set of clients. This problem is challenging as on one hand, services want to advertise themselves only after confirming that the client trying to discover them is authorized to see them. On the other hand, clients want to reveal their identity only after verifying that the service they are talking to is the desired one. In particular, a client device, such as a smartphone, should not simply identify itself to every device in the wild that requests it. This leads to a chicken-and-egg problem reminiscent of the settings addressed by secret handshakes and hidden credentials [BDS+03, LDB05, JKT06, AKB07, HBSO03, FAL04].

Private mutual authentication. A closely related privacy problem arises during authentication between mutually suspicious entities. Most existing mutual authentication protocols (SIGMA [CK02, Kra03], JFK [ABB+04], and TLS [DR08]) require one of the parties (typically the server) to reveal its identity to its peer before the other, effectively making that party's identity public to anyone who communicates with it. This is undesirable when the participants are personal end-user devices, where neither device is inclined to reveal its identity before learning that of its peer. Private mutual authentication is the problem of designing a mutual authentication protocol wherein each end learns the identity of its peer only if it satisfies the peer's authorization policy.[1]

An application. Our private discovery protocols apply broadly to many identification and key-exchange settings. Here we describe a common mobile-to-mobile example: peer-to-peer file sharing. Protocols such as AirDrop and Shoutr have become popular among mobile users for sharing photos and other content with their friends. These peer-to-peer protocols typically work by having a participant start a sharing service and making it publicly discoverable. The other device then discovers the service and connects to it to complete the file transfer. While this offers a seamless sharing experience, it compromises privacy for the device that makes itself discoverable—nearby devices on the same network can also listen to the advertisement and obtain identifiers from it. A private service discovery mechanism would make the service advertisement available only to the intended devices and no one else. The AirDrop protocol offers a "contacts-only" mode for additional privacy, but as we show in Section 2.1, this mechanism leaks significant private information. The private discovery protocols we develop in this paper provide an efficient solution to these problems.

[1] While protocols like SIGMA-I [Kra03, CK02] and TLS 1.3 [KW15, Res15] can ensure privacy against passive adversaries, they do *not* provide privacy against active attackers.

1.1 Our Contributions

This paper presents private mutual authentication and service discovery protocols for IoT and mobile settings. Given the network connectivity constraints implicit to these settings, our protocols do not require devices to maintain constant connectivity to an external proxy or directory service in the cloud. Furthermore, the protocols do not require the participants to have an out-of-band shared secret, thereby allowing devices with no pre-existing relationships to discover each other (in accordance with their respective privacy policies).

In Section 2, we motivate the desired features we seek in our protocols by presenting a case study of the Apple AirDrop protocol—specifically, its "contacts-only" mode for private file sharing. We describe several privacy vulnerabilities in the design of AirDrop, which we have disclosed to Apple. In light of these vulnerabilities, we define the robust privacy guarantees that we seek in our protocols.

Protocol construction. Our protocols are designed for distributed public-key infrastructures, such as the Simple Distributed Security Infrastructure (SDSI) [RL96]. Each principal has a public and private key-pair (for a signature scheme), and a hierarchical human-readable name bound to its public key by a certificate chain. The key primitive in our design is an encryption scheme that allows messages to be encrypted under an authorization policy so that it can be decrypted only by principals satisfying the policy. Using this primitive, we design a mutual authentication protocol where one party sends its identity (certificate chain) encrypted under its authorization policy. This protects the privacy of that party. The other party maintains its privacy by revealing its identity only *after* verifying the first party's identity. The same primitive is also used to construct a private service discovery protocol by having a service encrypt its advertisement under its authorization policy before broadcasting.

The service advertisements in our discovery protocol carry a signed semi-static Diffie-Hellman (DH) key. The signature provides authenticity for the advertisements and protects clients from connecting to an impostor service. The semi-static DH key enables clients to establish a secure session with the service using zero round-trips (0-RTT), similar to what is provided in TLS 1.3 [Res15, KW15].

The authorization policies considered in this work are based on name prefixes. For instance, a technician Bob from HomeSecurity Corp. may have the name `HomeSecurityCorp/Technician/Bob`, and a home security system might have a policy that only principals whose name starts with `HomeSecurityCorp/Technician` are allowed to discover it. Encrypting messages under a prefix-based authorization policy is possible using a prefix encryption scheme [LW14], which can be constructed using off-the-shelf identity-based encryption (IBE) schemes [BF01, BB04].

Protocol analysis. We give a full specification of our private mutual authentication and service discovery protocols in Sections 4 and 5. We also discuss a range of practical issues related to our protocol such as replay protection, ensuring perfect forward secrecy, and amortizing the overhead of the prefix encryption. In Appendices D and E, we provide a rigorous proof of the security and privacy of both protocols in the Canetti-Krawczyk key-exchange model [CK01, CK02, Kra03].

Implementation and evaluation. We implemented and deployed our protocols in the *Vanadium* open-source distributed application framework [Van]. We measured the end-to-end latency overhead for our private mutual authentication protocol on an Intel Edison, a Raspberry Pi, a

smartphone, a laptop, and a desktop. On the desktop, the protocol completes in 9.5 ms, which corresponds to a 1.8x slowdown over the SIGMA-I protocol that does *not* provide mutual privacy. On the Nexus 5X and the Raspberry Pi, the protocol completes in just over 300 ms (about a 3.8x slowdown over SIGMA-I), which makes it suitable for user-interactive services such as AirDrop and home security system controls that do not have high throughput requirements.

For the discovery protocol, a service's private discovery message consists of approximately 820 bytes of data. Since mDNS broadcasts support up to 1300 bytes of data, it is straightforward to deploy our discovery protocol over mDNS. In Section 6, we also discuss mechanisms for deploying the protocol over Bluetooth Low Energy and other protocols where the size of the advertisement packets are much more constrained.

Based on our benchmarks, our protocols are practical on a range of IoT devices, such as thermostats (e.g., Nest), security systems (e.g., Dropcam), and smart switches (e.g., Belkin Wemo). All of these devices have hardware comparable to a Pi or an Intel Edison. In fact, the Intel Edison is marketed primarily as a platform for building IoT applications. Moreover, as our AirDrop analysis demonstrates, many of the privacy issues we describe are not limited to only the IoT setting. Indeed, in Section 6.4, we show how our private mutual authentication and discovery protocols can be efficiently deployed to solve privacy problems in peer-to-peer interactions on smartphones. On more constrained processors such as the ARM Cortex M0, however, we expect the handshakes to take several seconds to complete. This makes our protocols less suitable in Cortex M0 applications that require fast session setup. Nonetheless, our protocols are sufficient for a wide range of existing IoT and mobile scenarios.

2 Desired Protocol Features

In this section, we define the privacy properties and features that we seek in our protocols. We begin with a case study of Apple's AirDrop protocol, and use it to motivate our privacy concerns and desired features.

2.1 Case Study: Apple AirDrop

AirDrop is a protocol for transferring files between two devices running OS X (Yosemite or later) or iOS (version 7 or later). It is designed to work whenever two AirDrop-enabled devices are close to each other and even when they do not have Internet access. AirDrop uses both Bluetooth Low Energy (BLE) and Apple's peer-to-peer WiFi technology (`awdl`) for device discovery and file transfer.

To receive files, devices make themselves discoverable by senders. AirDrop offers two modes for making devices discoverable: *everyone*, which makes the device discoverable by all nearby devices, and *contacts-only* (default), which makes the receiving device discoverable only by senders in its contacts. The contacts-only mode is meant to be a privacy mechanism and can be viewed as a solution to the private service discovery problem for the "contacts-only" policy.

Protocol overview. We analyzed the AirDrop protocol to understand its privacy properties and see how it solves the chicken-and-egg problem of private mutual authentication. Below, we present a high-level description of the AirDrop protocol in contacts-only mode, based on the iOS9 security

guide [Inc15] and our experiments with observing AirDrop flows between a MacBook Pro and an iPhone.

1. When a sender opens the sharing pane on her device, the device advertises a truncated hash of the sender's identity over BLE, and simultaneously queries for a service with label `_AirDrop._tcp.local` over mDNS.

2. A nearby receiving device matches the hash of the sender's identity against its contacts list. If a match is found, the receiving device starts a service and advertises its instance name, `awdl` IP address, and port number over mDNS, under the label `_AirDrop._tcp.local`.

3. The sending device obtains the receiving device's service advertisement, and initiates a TLS (version 1.2) connection to it over `awdl`. The TLS handshake uses client authentication, wherein each device sends its iCloud identity certificate[2] to the other in the clear. The handshake fails if either party receives a certificate for someone not in their contacts list.

4. Once the TLS connection is established, the receiver's description is displayed on the sending device's sharing pane. The sender selects the receiver as well as the files to be shared which are then sent over the established TLS channel.

Privacy weaknesses in Apple AirDrop. Our analysis indicates that AirDrop employs two main privacy checks in contacts-only mode. First, a receiving device responds only if the sender's identifier (received over BLE) matches one of its contacts, and second, a communication channel is established between a sender and receiver only if their respective certificates match a contact on their peer's device. While necessary, these checks are insufficient to protect the privacy of the sender and receiver. Below, we enumerate some of the privacy problems with the existing protocol.

- **Sender and receiver privacy and tracking.** The use of TLS 1.2 with client authentication causes both the sender and receiver to exchange certificates in the clear. This makes their identities, as specified by their certificates, visible to even a *passive* eavesdropper on the network. Moreover, the public keys in the certificates allow the eavesdropper to track the sender and receiver in the future. While using a key-exchange protocol like SIGMA-I [Kra03] or TLS 1.3 [Res15] would hide the certificate for one of the parties (the sender in this case), the certificate of the other party (the receiver) would still be revealed to an active attacker. Protecting the privacy of *both* parties against active attackers, requires *private* mutual authentication, as constructed in Section 4.

- **Sender impersonation.** Another privacy problem is that the sender's identifier advertised over BLE can be forged or replayed by an attacker to trick an honest receiver into matching it against its contacts. Based on the receiver's response, the attacker learns whether the receiver has the sender in their contacts, and moreover, could try to initiate a TLS session with the receiver to obtain its certificate. To protect against this kind of impersonation attack, discovery broadcasts must provide some kind of *authenticity*, as in Section 5.

[2] All AirDrop-enabled devices have an RSA public and private key pair and an iCloud certificate for the owner's identity.

2.2 Protocol Design Goals

The privacy properties of AirDrop are insufficient to solve the private service discovery problem. While our case study in Section 2.1 focuses exclusively on the AirDrop protocol, most existing key-exchange and service discovery protocols do not provide robust privacy and authenticity guarantees. We survey some of these alternative protocols in Section 8. In this section, we describe the strong privacy properties we seek in our protocols. We want these properties to hold even against an active network attacker (i.e., an attacker that can intercept, modify, replay and drop packets arbitrarily). We begin by describing two concrete privacy objectives:

- **Mutual privacy.** The protocols must ensure that the identities and any identifying attributes of the protocol participants are only revealed to authorized recipients. For service discovery, this applies to both the service being advertised and the clients trying to discover it.
- **Authentic advertisements.** Service advertisements should be unforgeable and authentic. Otherwise, an attacker may forge a service advertisement to determine if a client is interested in the service.

Finally, to ensure that our protocols are applicable in both IoT and peer-to-peer settings, we impose additional constraints on the protocol design:

- **No out-of-band pairing for participants.** The protocol should not require participants to exchange certain information or secrets out-of-band. This is especially important for the discovery protocol as the service may not know all the clients that might try to discover it in the future. Besides impacting feasibility, the pairing requirement also degrades the user experience. For instance, the main charm of AirDrop is that it "just works" without needing any *a priori* setup between the sender and recipient.
- **No cloud dependency during protocol execution.** The protocol should not rely on an external service in the cloud, such as a proxy or a directory service. Protocols that depend on cloud-based services assume that the participating devices maintain reliable Internet access. This assumption fails for many IoT devices, including devices that only communicate over Bluetooth, or ones present in spaces where Internet access is unreliable. Again, a nice feature of the AirDrop protocol is that it works even if neither device is connected to the Internet. Thus, we seek a protocol that maximizes peer-to-peer communication and avoids dependence on global services.

3 Preliminaries

In this section, we describe our identity and authorization model, as well as introduce the cryptographic primitives we use in our constructions.

Identity and authorization model. We define our protocols for a generic distributed public-key infrastructure, such as SDSI [RL96]. We assume each principal has a public and private key-pair for a signature scheme and one or more hierarchically-structured human-readable names bound to its public key via a certificate chain. For instance, a television set owned by Alice might have a certificate chain binding the name `Alice/Devices/TV` to it. The same television set may also

have a certificate chain with the name `PopularCorp/Products/TV123` from its manufacturer. Our protocols are agnostic to the specific format of certificates and how they are distributed.

Principals authenticate each other by exchanging certificate chains and providing a signature on a fresh (session-specific) nonce. During the authentication protocol, a principal validates its peer's certificate chain, and extracts the name bound to the certificate chain. Authorization decisions are based on this extracted name, and *not* the public key. For example, Alice may authorize all principals with names matching the prefix pattern `Alice/Devices/*` to access her television set. In this work, we consider prefix-based authorization policies.

Prefix-based policies can also be used for group-based access control. For instance, Alice may issue certificate chains with names `Alice/Family/Bob`, and `Alice/Family/Carol` to her family members Bob and Carol. She can then authorize her family to discover and access her home security system simply by setting the authorization policy to `Alice/Family/*`.

3.1 Cryptographic and Protocol Building Blocks

We write $\mathbb{Z}_p$ to denote the group of integers modulo p. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is drawn from $\mathcal{D}$. For a finite set S, we write $x \xleftarrow{R} S$ to denote that x is drawn uniformly at random from S. A function $f(\lambda)$ is negligible in a security parameter λ if $f = o(1/\lambda^c)$ for all $c \in \mathbb{N}$.

Identity-based encryption and prefix encryption. Identity-based encryption (IBE) [Sha84, BF01, Coc01, BB04] is a generalization of public-key encryption where public keys can be arbitrary strings, or *identities*. We give more details in Appendix A. Prefix encryption [LW14] is a generalization of IBE where the secret key $\mathrm{SK}_{\mathrm{ID}}$ for an identity ID can decrypt all ciphertexts encrypted to any identity ID′ that is a prefix of ID (in IBE, decryption succeeds only if $\mathrm{ID} = \mathrm{ID}'$).[3] Prefix encryption allows for messages to be encrypted under a prefix-based policy such that the resulting ciphertext can only be decrypted by principals satisfying the policy.

It is straightforward to construct prefix encryption from IBE. The following construction is adapted from the Lewko-Waters scheme [LW14]. The key for an identity $\mathrm{ID} = s_1/s_2/\cdots/s_n$ consists of n different IBE keys for the following sequence of identities: $(s_1), (s_1/s_2), \ldots, (s_1/s_2/\cdots/s_n)$. Encryption to an identity ID′ is just IBE encryption to the identity ID′. Given a secret key $\mathrm{SK}_{\mathrm{ID}}$ for ID, if ID′ is a prefix of ID, then $\mathrm{SK}_{\mathrm{ID}}$ contains an IBE identity key for ID′.

The syntax of a prefix encryption scheme is very similar to that of an IBE scheme. Secret keys are still associated with identities, but ciphertexts are now associated with prefix-constrained policies. In the following, we write $\mathsf{PE.Enc}(\mathrm{MPK}, \pi, m)$ to denote an encryption algorithm that takes as input the public key MPK, a message m, a prefix-constrained policy π, and outputs a ciphertext CT. When there is no ambiguity, we will treat MPK as an implicit parameter to $\mathsf{PE.Enc}$. We write $\mathsf{PE.Dec}(\mathrm{SK}_{\mathrm{ID}}, \mathrm{CT})$ for the decryption algorithm that takes in a ciphertext CT and a secret key $\mathrm{SK}_{\mathrm{ID}}$ (for an identity ID) and outputs a message if ID matches the ciphertext policy π, and a special symbol $\perp$ otherwise.

Other cryptographic primitives. We write $\{m\}_k$ to denote an authenticated encryption [BN00, Rog02, BRW03] of a message m under a key k, and $\mathsf{KDF}(\cdot)$ to denote a key-derivation func-

[3] Lewko and Waters [LW14] considered the reverse setting where decryption succeeds if ID is a prefix of ID′, but their construction is easily adaptable to our setting.

tion [DGH+04, Kra10]. We describe these additional primitives as well as the cryptographic assumptions (Hash Diffie-Hellman and Strong Diffie-Hellman [ABR01]) we use in our security analysis in Appendix A.

Key-exchange model. We analyze the security of our private mutual authentication and privacy service discovery protocols in the Canetti-Krawczyk [CK01, CK02, Kra03] key-exchange model, which models the capabilities of an active network adversary. We defer the formal specification of this model and our generalization of it to the service discovery setting to Appendix B.

4 Private Mutual Authentication Protocol

In this section, we describe our private mutual authentication protocol and discuss some of its features and limitations. We use the identity and authorization model described in Section 3.

Protocol execution environment. In our setting, each principal has a signing/verification keypair and a set of names (e.g., `Alice/Devices/TV`) bound to its public verification key via certificate chains. For each name, a principal possesses an identity secret key (for the prefix encryption scheme) extracted for that name. The secret key extraction is carried out by IBE root authorities (who possess the IBE master secret key MSK), which may coincide with certificate authorities. For example, Alice could be an IBE root that issued both an identity secret key and a certificate chain to its television set for the name `Alice/Device/TV`. Similarly, the television set may also have an identity secret key from its manufacturer Popular Corp. for the name `PopularCorp/Products/TV123`. Finally, each principal also has one or more prefix-constrained authorization policies.

In our protocol description, we refer to the initiator of the protocol as the *client* and the responder as the *server*. For a party P, we write ID_P to denote a certificate chain binding P's public key to one of its identities. For a message m, we write $\text{SIG}_P(m)$ to denote P's signature on m. We refer to each instantiation of the key-exchange protocol as a "session," and each session is identified by a unique session id, denoted sid.

Protocol specification. Our starting point is the 3-round SIGMA-I protocol [Kra03, CK02] which provides mutual authentication as well as privacy against passive adversaries. Similar to the SIGMA-I protocol, our protocol operates over a cyclic group $\mathbb{G}$ of prime order where the Hash-DH [ABR01] assumption holds. Let g be a generator of $\mathbb{G}$. We now describe our private mutual authentication protocol. The message flow is illustrated in Figure 1.

1. To initiate a session with id sid, the client C chooses $x \xleftarrow{\text{R}} \mathbb{Z}_p$, and sends (sid, g^x) to the server.
2. Upon receiving a start message (sid, g^x) from a client, the server S chooses $y \xleftarrow{\text{R}} \mathbb{Z}_p$, and does the following:
   (a) Encrypt its name ID_S using the prefix encryption scheme under its policy π_S to obtain an encrypted identity $\text{CT}_S \leftarrow \mathsf{PE.Enc}(\pi_S, \text{ID}_S)$.
   (b) Derive authenticated encryption keys $(\mathsf{htk}, \mathsf{atk}) = \mathsf{KDF}(g^x, g^y, g^{xy})$ for the handshake and application-layer messages, respectively.
   (c) Compute a signature $\sigma = \text{SIG}_S(\mathsf{sid}, \text{CT}_S, g^x, g^y)$ on its encrypted identity and the ephemeral session state, and encrypt (CT_S, σ) using htk to obtain a ciphertext c.

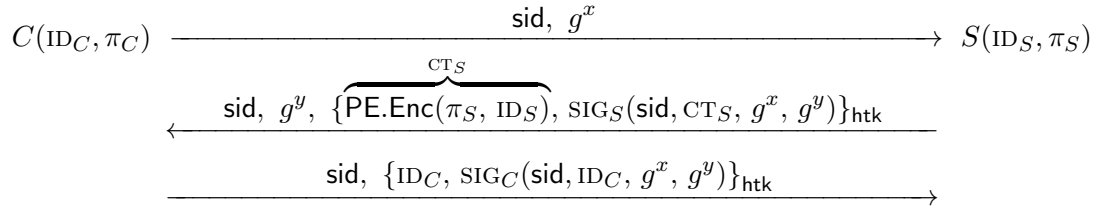

Figure 1: Message flow between the client C (with certificate ID_C and policy π_C) and the server S (with certificate ID_S and policy π_S) for the private mutual authentication protocol. Both the client and the server possess a secret signing key. The associated verification keys are bound to their identities via the certificates ID_C and ID_S, respectively. For a message m, $\text{SIG}_C(m)$ and $\text{SIG}_S(m)$ denote the client's and server's signature on m, respectively. Both the client and server know the master public key for the prefix-based encryption scheme, and the client possesses a secret key SK_C for the prefix-based encryption scheme for the identity associated with its certificate ID_C.

The server replies to the client with (sid, g^y, c).

3. When the client receives a response (sid, g^y, c), it derives the keys $(\mathsf{htk}, \mathsf{atk}) = \mathsf{KDF}(g^x, g^y, g^{xy})$. It tries to decrypt c with htk and aborts if decryption fails. It parses the decrypted value as (CT_S, σ_S) and checks whether its identity ID_C satisfies the server's policy π_S (revealed by CT_S). If the client satisfies the server's policy, it decrypts CT_S using its identity key SK_C to obtain the server's identity ID_S. If ID_S satisfies the client's policy π_C and σ_S is a valid signature on $(\mathsf{sid}, \text{CT}_S, g^x, g^y)$ under the public key identified by ID_S, the client replies to the server with the session id sid and an encryption c' of $(\text{ID}_C, \text{SIG}_C(\mathsf{sid}, \text{ID}_C, g^x, g^y))$ under htk. Otherwise, the client aborts.

4. Upon receiving the client's response (sid, c'), the server tries to decrypt c' using htk and aborts if decryption fails. It parses the decrypted value as (ID_C, σ_C) and verifies that ID_C satisfies its policy and that σ_C is a valid signature on $(\mathsf{sid}, \text{ID}_C, g^x, g^y)$ under the public key identified by ID_C. If so, the handshake completes with atk as the shared session key and where the client believes it is talking to ID_S and the server believes it is talking to ID_C. Otherwise, the server aborts.

4.1 Protocol Analysis

In this section, we highlight some properties of our private mutual authentication protocol.

Comparison with SIGMA-I. Our authentication protocol is very similar to the SIGMA-I key-exchange protocol [Kra03, §5.2], but with the following key difference: the server's certificate, ID_S, is sent encrypted under a prefix encryption scheme. Moreover, instead of deriving separate MAC and encryption keys from the shared DH key, we combine the two primitives by using an authenticated encryption scheme. Since we have only added an additional layer of prefix encryption to the certificates, each party's signature verification key is still bound to its identity as before. Thus, the proof that the SIGMA-I protocol is a secure key-exchange protocol [CK02, §5.3] (with perfect forward secrecy) translates to our setting.

Identity privacy. The identity of the server is sent encrypted under its prefix policy, so by security of the prefix encryption scheme, it is only revealed to clients that satisfy the policy. Conversely, an honest client only reveals its identity after it has verified that the server's identity satisfies its policy. We formally define our notion of mutual privacy in Appendix C. Then, in Appendix D, we show that the protocol in Figure 1 achieves our notion of mutual privacy. In contrast, the SIGMA-I protocol does not provide such a guarantee as the identity of the server is revealed to active adversaries.

Policy privacy. The protocol in Figure 1 ensures privacy for the client's policy, but not for the server's policy. This issue stems from the fact that the underlying prefix encryption scheme is not "policy-hiding," meaning an encryption of a message m to a prefix π is only guaranteed to hide m and not π. Since a passive adversary can observe whether a connection is successfully established between a client and a server, learning the server's policy allows the adversary to then infer information about the client's identity. We describe a solution based on anonymous IBE in Section 7.

Caching the encrypted certificate chains. By introducing prefix encryption, the protocol in Figure 1 is more computationally expensive for both the server and the client compared to the SIGMA-I protocol. However, for key-exchange security, it is not essential that the server re-encrypt its certificate chain using the prefix encryption scheme on each handshake. The server can instead reuse the *same* CT_S across multiple handshakes. Thus, the cost of the IBE encryptions needed to construct CT_S can be amortized across multiple handshakes. The client still needs to perform a single IBE decryption per handshake.

Unlinkability. One limitation of our private mutual authentication protocol is that an active network attacker can fingerprint a server. Certainly, if the server reuses the same encrypted certificate across multiple handshakes, an active attacker that does not satisfy the server's authorization policy can still use the encrypted certificate to identify the server across sessions. Even without the caching optimization, an active attacker can still track the server via its signature. This is because in many signature schemes such as ECDSA, the public verification key can effectively be recovered from a message-signature pair [Bro09]. Thus, as long as the server's signing key (and correspondingly, verification key) is long-lived, the server's public verification key can be used to identify and track the server across sessions. In fact, when deploying mutual authentication protocols, it is good practice to include the verification key in the signature in order to defend against Duplicate Signature Key Selection (DSKS) attacks [MS04, KM13].

Depending on the use case, this linkability issue can be problematic. A simple fix is to instead apply the prefix encryption to the entirety of the server's response message in the SIGMA-I protocol. Then, both the server's certificate chain as well as its signature on the ephemeral shares are encrypted under the prefix encryption scheme. As a result, the server's response is unique per session and can only be decrypted by clients whose identities satisfy the server's policy. Security of the resulting key-exchange protocol still reduces naturally to the SIGMA-I protocol and privacy still reduces to the security of the prefix encryption scheme. The main disadvantage of this protocol is that the server can no longer cache and reuse the prefix-encrypted ciphertext. Thus, the server must perform a prefix encryption during each handshake, increasing the computational load on the server. We give the message flow for this modified protocol in Figure 2. Using ProVerif [BSC16],

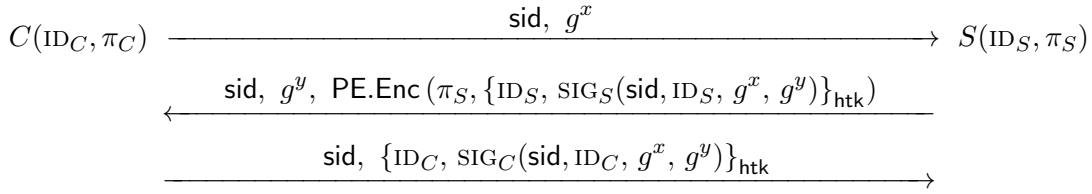

Figure 2: Message flow between the client C and the server S for the variant of the private mutual authentication protocol that provides unlinkability. The main difference between this protocol and the private mutual authentication protocol in Figure 1 is that here, the server encrypts its entire response using prefix-based encryption. In contrast, in the protocol in Figure 1, the server only encrypts its certificate chain using the prefix-based encryption scheme.

we are able to formally verify (in a Dolev-Yao model of protocol logic [DY81]) that this modified protocol satisfies the following notion of unlinkability: no client can distinguish between two servers that have different signing and verification keys but use the same policy. For the other properties (key-exchange security and mutual privacy), our analysis (in the standard computational model) in Appendix D applies.

Security theorem. We state the security theorem for our private mutual authentication protocol here, but defer the formal proof to Appendix D.

Theorem 4.1 (Private Mutual Authentication)**.** *The protocol in Figure 1 is a secure and private key-exchange protocol in the Canetti-Krawczyk key-exchange model assuming the Hash Diffie-Hellman assumption in* $\mathbb{G}$ *and the security of all underlying cryptographic primitives.*

5 Private Service Discovery Protocol

In this section, we describe our private service discovery protocol. The primary goal is to make a service discoverable only by parties that satisfy its authorization policy. Additionally, once a client has discovered a service, it should be able to authenticate to the server using zero round-trips (0-RTT), i.e., include application data on the first flow of the handshake. 0-RTT protocols are invaluable for IoT since devices are often constrained in both computation and bandwidth.

The key idea in our design is to have the service include a fresh DH share and a signature in its advertisement. The DH share allows 0-RTT client authentication, and the signature provides authenticity for the service advertisement. Next, the service encrypts its advertisement under its policy π_S before broadcasting to ensure that only authorized clients are able to discover it. A similar mechanism for (non-private) 0-RTT authentication is present in OPTLS and the TLS 1.3 specification [Res15, KW15], although OPTLS only provides server authentication.

Protocol specification. Our protocol works over a cyclic group $\mathbb{G}$ of prime order p with generator g where the Strong-DH and Hash-DH assumptions [ABR01] hold. The private discovery protocol can be separated into a broadcast protocol and a 0-RTT mutual authentication protocol. Each broadcast is associated with a unique broadcast identifier bid and each session with a unique session identifier sid. The protocol execution environment is the same as that described in Section 4. The basic message flow for the private discovery protocol is illustrated in Figure 3.

Service broadcast message. To setup a new broadcast with broadcast id bid, the server S chooses a fresh DH exponent $s \xleftarrow{R} \mathbb{Z}_p$, and encrypts $(\text{ID}_S, g^s, \text{SIG}_S(\mathsf{bid}, \text{ID}_S, g^s))$ using the prefix encryption scheme under its authorization policy π_S to obtain a broadcast ciphertext CT_S. The server broadcasts $(\mathsf{bid}, \text{CT}_S)$.

0-RTT mutual authentication. Upon receiving a broadcast $(\mathsf{bid}, \text{CT}_S)$, a client performs the following steps to establish a session sid with the server:

1. The client C checks that its identity ID_C satisfies the server's authorization policy π_S (included with CT_S). If so, it decrypts CT_S using its prefix encryption secret key and parses the decrypted value as $(\text{ID}_S, g^s, \sigma_S)$. It verifies that ID_S satisfies its policy π_C and that σ_S is a valid signature on $(\mathsf{bid}, \text{ID}_S, g^s)$ under the public key identified by ID_S. If any step fails, the client aborts.

2. To initiate a session with id sid, the client first chooses an ephemeral DH exponent $x \xleftarrow{R} \mathbb{Z}_p$. It derives authenticated encryption keys $(\mathsf{htk}, \mathsf{htk}', \mathsf{eadk}) = \mathsf{KDF}(g^s, g^x, g^{sx})$, where htk and htk' are used to encrypt handshake messages, and eadk is used to encrypt any early application data the client wants to include with its connection request. The client encrypts the tuple $(\text{ID}_S, \text{ID}_C, \text{SIG}_C(\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_C, g^s, g^x))$ under htk to obtain a ciphertext c_1 and any early application data under eadk to obtain a ciphertext c_2. It sends $(\mathsf{bid}, \mathsf{sid}, g^x, c_1, c_2)$ to the server.

3. When the server receives a message from a client of the form $(\mathsf{bid}, \mathsf{sid}, g^x, c_1, c_2)$, it first derives the encryption keys $(\mathsf{htk}, \mathsf{htk}', \mathsf{eadk}) = \mathsf{KDF}(g^s, g^x, g^{sx})$, where s is the DH exponent it chose for broadcast bid. Then, it tries to decrypt c_1 with htk and c_2 with eadk. If either decryption fails, the server aborts the protocol. Otherwise, let $(\text{ID}_1, \text{ID}_2, \sigma)$ be the message obtained from decrypting c_1. The server verifies that $\text{ID}_1 = \text{ID}_S$ and that ID_2 satisfies its authorization policy π_S. Next, it checks that σ is a valid signature on $(\mathsf{bid}, \mathsf{sid}, \text{ID}_1, \text{ID}_2, g^s, g^x)$ under the public key identified by ID_2. If all these checks pass, the server chooses a new ephemeral DH exponent $y \xleftarrow{R} \mathbb{Z}_p$ and derives the session key $\mathsf{atk} = \mathsf{KDF}(g^s, g^x, g^{sx}, g^y, g^{xy})$.[4] The server encrypts the tuple $(\mathsf{bid}, \mathsf{sid}, \text{ID}_1, \text{ID}_2, g^s, g^x, g^y)$ under htk' to obtain a ciphertext c_1', and any application messages under atk to obtain a ciphertext c_2'. It replies to the client with $(\mathsf{bid}, \mathsf{sid}, g^y, c_1', c_2')$.

4. When the client receives a response message $(\mathsf{bid}, \mathsf{sid}, g^y, c_1', c_2')$, it first decrypts c_1' using htk' and verifies that c_1' decrypts to $(\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_C, g^s, g^x, g^y)$, where $s \in \mathbb{Z}_p$ is the server's semi-static DH share associated with broadcast bid, and $x \in \mathbb{Z}_p$ is the ephemeral DH share it used in session sid with broadcast bid. If so, it derives $\mathsf{atk} = \mathsf{KDF}(g^s, g^x, g^{sx}, g^y, g^{xy})$ and uses atk to decrypt c_2'. The handshake then concludes with atk as the shared session key.

5.1 Protocol Analysis

We now describe some of the properties of our private service discovery protocol in Figure 3.

[4] In this step, the server samples a fresh ephemeral DH share g^y that is used to derive the application-traffic key atk. This is essential for ensuring perfect forward secrecy for all subsequent application-layer messages (encrypted under atk). We discuss the perfect forward secrecy properties of this protocol in Section 5.1.

Server's Broadcast:

$$\mathsf{bid}, \mathsf{PE.Enc}(\pi_S, (\text{ID}_S, g^s, \text{SIG}_S(\mathsf{bid}, \text{ID}_S, g^s)))$$

0-RTT Mutual Authentication:

$$C(\text{ID}_C, \pi_C) \xrightarrow{\mathsf{bid}, \mathsf{sid}, g^x, \{\text{ID}_S, \text{ID}_C, \text{SIG}_C(\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_C, g^s, g^x)\}_{\mathsf{htk}}} S(\text{ID}_S, \pi_S)$$
$$\xleftarrow{\mathsf{bid}, \mathsf{sid}, g^y, \{(\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_C, g^s, g^x, g^y)\}_{\mathsf{htk}'}}$$

Figure 3: Basic message flow between the client C (with certificate ID_C and policy π_C) and the server S (with certificate ID_S and policy π_S) for the private discovery protocol. As in the private mutual authentication protocol (Figure 1), the client and server each possess a secret signing key, and the associated verification keys are bound to their identities via the certificates ID_C and ID_S, respectively. Both the client and server know the master public key for the prefix-based encryption scheme, and the client possesses a secret key SK_C for the prefix-based encryption scheme for the identity associated with its certificate ID_C. In this protocol, the client can also include early application data in the first flow of the 0-RTT mutual authentication protocol under a key derived from the client's ephemeral DH share g^x and the server's semi-static DH share g^s.

0-RTT security. The security analysis of the 0-RTT mutual authentication protocol in Figure 3 is similar to that of the OPTLS protocol in TLS 1.3 [KW15] and relies on the Strong-DH and Hash-DH assumptions [ABR01] in the random oracle model [BR93b]. Note that in contrast to the OPTLS protocol which only provides server authentication, our protocol provides *mutual authentication*.

Replay attacks. One limitation of the 0-RTT mode is that the early-application data is vulnerable to replay attacks. A typical replay-prevention technique (used by QUIC [FG14, LJBN15]) is to have the server maintain a list of client nonces in the 0-RTT messages and reject duplicates for the lifetime of the service advertisement. While this mechanism is potentially suitable if the service advertisements are short-lived and the service does not have to maintain large amounts of state, the general problem remains open.

Authenticity of broadcasts. Because the service broadcasts are signed, a client is assured of the authenticity of a broadcast before establishing a session with a service. This ensures that the client will not inadvertently send its credentials to an impostor service. However, an adversary that intercepts a service broadcast and recovers the associated semi-static DH exponent can replay the broadcast for an honest client. If the client then initiates a session using the DH share from the replayed advertisement, the adversary compromises the client's privacy. To protect against this kind of replay attack, the server should include an expiration time in its broadcasts, and more importantly, sign this expiration. Then, by having short-lived broadcasts, the adversary's window of opportunity in mounting a replay attack is greatly reduced. Note though that recovering the semi-static DH exponent s from g^s amounts to solving the discrete log problem in the underlying group, which is conjectured to be a difficult problem. Thus, it is unlikely that authenticity will be compromised even for long-lived broadcasts. However, perfect forward secrecy (discussed next) is compromised for all early-application data for the duration of the broadcast, and so, it is still preferable to have short-lived broadcasts.

Forward secrecy. Since the server's semi-static DH share persists across sessions, perfect forward secrecy (PFS) is lost for early-application data and handshake messages sent during the lifetime of each advertisement. Specifically, an adversary that compromises a server's semi-static DH share for a particular broadcast is able to break the security on all early-application data and handshake messages (which contain the client's identity) in any key-exchange session that occurred during the lifetime of the particular broadcast. To mitigate this risk in practical deployments, it is important to periodically refresh the DH-share in the server's broadcast (e.g., once every hour). The refresh interval corresponds to the window where forward secrecy may be compromised.

While PFS is not achievable for early-application and handshake messages for the lifetime of a service's broadcast, PFS is ensured for all application-layer messages (as long as the server's semi-static secret was not already compromised at the time of session negotiation). In particular, after processing a session initiation request, the server responds with a fresh ephemeral DH share that is used to derive the session key for all subsequent messages. In Appendix E, we show that the security of the session is preserved even if the server's semi-static secret is compromised (after the completion of the handshake) but the ephemeral secret is uncompromised. This method of combining a semi-static key with an ephemeral key is also used in the OPTLS [KW15] protocol.

Identity privacy. As was the case in our private mutual authentication protocol from Section 4, privacy for the server's identity is ensured by the prefix-based encryption scheme. Privacy for the client's identity is ensured since all handshake messages are encrypted under handshake traffic keys htk and htk'. We formally state and prove mutual privacy for the protocol in Appendix E.3.

Security theorem. We conclude by stating the security theorem for our private service discovery protocol. We give the formal proof in Appendix E.

Theorem 5.1 (Private Service Discovery)**.** *The protocol in Figure 3 is a secure and private service discovery protocol in a Canetti-Krawczyk-based model of key-exchange in the random oracle model, assuming the Hash Diffie-Hellman and Strong Diffie-Hellman assumptions in* $\mathbb{G}$*, and the security of the underlying cryptographic primitives.*

6 Protocol Evaluation and Deployment

In this section, we describe the implementation and deployment of our private mutual authentication and service discovery protocols in the Vanadium framework [Van]. We benchmark our protocols on a wide range of architectures: an Intel Edison (0.5GHz Intel Atom), a Raspberry Pi 2 (0.9GHz ARM Cortex-A7), a Nexus 5X smartphone (1.8GHz 64-bit ARM-v8A), a Macbook Pro (3.1GHz Intel Core i7), and a desktop (3.2GHz Intel Xeon).

Vanadium. We implement our private mutual authentication and service discovery protocols as part of the Vanadium framework for developing secure, distributed applications that can run on a multitude of devices ranging from small computers such as Raspberry Pis and Intel Edisons, to computers running Linux or Mac OS, to cloud services like Google Compute Engine. At the core of the framework is a remote procedure call (RPC) system that enables applications to expose services over the network. All RPCs are mutually authenticated, encrypted and bidirectional. The framework also offers an interface definition language (IDL) for defining services, a federated

naming system for addressing them, and a discovery API for advertising and scanning for services over a variety of protocols, including BLE and mDNS.

The Vanadium identity model is based on a distributed PKI. All principals in Vanadium possess an ECDSA P-256 signing and verification key-pair. Principals have a set of human-readable names bound to them via certificate chains, called *blessings*. Blessings can be extended locally and delegated from one principal to another. All RPCs are encrypted and mutually authenticated based on the blessings bound to each end. The access control model is based on blessing names, and is specified in [ABP+15].

We implement our protocols to enhance the privacy of the Vanadium RPC and discovery framework. Our entire implementation is in Go[5] (with wrappers for interfacing with third-party C libraries). Go is also the primary implementation language of Vanadium.

6.1 Identity-Based Encryption

The key primitive we require for our protocols is prefix-based encryption, which we can construct from any IBE scheme (Section 3.1). For our experiments, we implemented the Boneh-Boyen (BB_2) scheme [BB04, §5] over the 256-bit Barreto-Naehrig (`bn256`) [NNS10] pairings curve. We chose the BB_2 IBE scheme for its efficiency: it only requires a single pairing evaluation during decryption. Other IBE schemes either require pairing operations in both encryption and decryption [BF01] or require multiple pairing evaluations during decryption (BB_1) [BB04, §4.1]. We apply the Fujisaki-Okamoto transformation[6] [FO99] to obtain CCA-security. For the underlying symmetric encryption scheme in the Fujisaki-Okamoto transformation, we use the authenticated encryption scheme from NaCl [Ber09, BLS12]. All of our cryptographic primitives are chosen to provide at least 128 bits of security.

Benchmarks. We measured the performance overhead of the different IBE operations, as well as the pairing operation over the elliptic curve. Our results are summarized in Table 1. We use the off-the-shelf `dclxvi` implementation [NNS10] for the `bn256` pairing curve on all architectures other than the Nexus 5X. On the Nexus 5X, we use a Go implementation.[7] On the desktop, we were able to turn on assembly optimizations which led to a significant speedup in the benchmarks.

Deployment. The Vanadium ecosystem includes *identity providers*, which are principals that sign the root certificate of a blessing (certificate chain). While any principal can become an identity provider, Vanadium runs a prototype identity provider for granting blessings to users based on their Google account. These blessings begin with the name `dev.v.io`. We augment this service so that it also acts as an IBE root and issues prefix encryption secret keys. More specifically, we run an RPC service for exchanging any Vanadium blessing prefixed with `dev.v.io` (e.g., `dev.v.io/u/Alice/Devices/TV`) with a prefix encryption secret key for the blessing name.

[5] `https://golang.org`

[6] The Fujisaki-Okamoto transformation [FO99] provides a generic method of combining a semantically-secure public-key encryption scheme with a symmetric encryption scheme to obtain a CCA-secure hybrid encryption scheme in the random oracle model.

[7] We were not able to use the `dclxvi` implementation on the Android phone. A significant speedup should be possible with an optimized implementation.

	Intel Edison	Raspberry Pi 2	Nexus 5X⋆	Laptop	Desktop
Pairing	254.5 ms	101.2 ms	219.2 ms	5.0 ms	1.0 ms
Encrypt	406.7 ms	157.2 ms	161.3 ms	8.2 ms	3.5 ms
Decrypt	623.0 ms	235.2 ms	235.7 ms	11.6 ms	3.7 ms
Extract	107.5 ms	41.6 ms	47.2 ms	2.1 ms	0.5 ms

⋆ Unoptimized Go implementation of `bn256`.

Table 1: IBE micro-benchmarks

6.2 Private Mutual Authentication

We implemented the private mutual authentication protocol from Section 4 within the Vanadium RPC system as a means to offer a "private mode" for Vanadium services. We implemented the protocol from Figure 1 that allows caching of the encrypted server certificate chain. The implementation uses a prefix encryption primitive implemented on top of our IBE library.

Benchmarking. We measure the end-to-end connection setup time for our protocol on various platforms. To eliminate network latency, we instantiate a server and client in the same process. Since the encrypted server certificate chain can be reused across multiple handshakes, we precompute it before executing the protocol.[8] Both the client and the server use a prefix-based policy of length three. Note that the encryption and decryption times in our prefix encryption scheme are not affected by the length of the policy.

Results. We compare the performance of our protocol to the traditional SIGMA-I protocol in Table 2. The end-to-end latency on the desktop is only 9.5 ms, thanks to an assembly-optimized IBE implementation. The latency on smaller devices is typically around a third of a second, which is quite suitable for user-interactive applications like AirDrop. Even on the Intel Edisons (a processor marketed specifically for IoT), the handshake completes in just over 1.5 s, which is still reasonable for many applications. Moreover, with an optimized implementations of the IBE library (e.g., taking advantage of assembly optimizations like on the desktop), these latencies should be significantly reduced.

The memory and storage requirements of our protocol are very modest and well-suited for the computational constraints of IoT and mobile devices. Specifically, the pairing library is just 40 KB of code on the ARM processors (and 64 KB on x86). The public parameters for the IBE scheme are 512 bytes, and each IBE secret key is just 160 bytes. For comparison, a typical certificate chain (of length 3) is about 500 bytes in Vanadium. Also, our protocols are not memory-bound, and in particular, do not require much additional memory on top of the existing non-private SIGMA-I key-exchange protocol supported by Vanadium.

[8] If we use the variant of our private mutual authentication protocol that provides unlinkability (Figure 2), the server needs to perform a prefix-based encryption on every handshake. In our case, prefix-based encryption is just IBE encryption. Based on the micro-benchmarks from Table 1, the extra (per-policy) overhead is just a few hundred milliseconds on the small devices and under 10 milliseconds on the laptops and desktops.

	Intel Edison	Raspberry Pi 2	Nexus 5X	Laptop	Desktop
SIGMA-I	252.1 ms	88.0 ms	91.6 ms	6.3 ms	5.3 ms
Private Mutual Auth.	1694.3 ms	326.1 ms	360.4 ms	19.6 ms	9.5 ms
Slowdown	6.7x	3.7x	3.9x	3.1x	1.8x

Table 2: Private mutual authentication benchmarks.

Deployment. We implemented the private mutual authentication protocol from Section 4 within the Vanadium RPC system. Prior to our work, the RPC system only supported the SIGMA-I [Kra03, CK02] protocol for mutual authentication, which does not provide privacy for services. We implemented our private mutual authentication protocol to offer a "private mode" for Vanadium services.

6.3 Private Discovery

We also integrated the private discovery protocol from Section 5 into Vanadium.

Benchmarks. We benchmark the cryptographic overhead of processing service advertisements, and measure the size of the service advertisements. Processing service advertisements requires a single IBE decryption and one ECDSA signature verification. This cost can be estimated analytically from Table 1 and standard ECDSA benchmarks. For instance, on the Nexus 5X smartphone, which is a typical client for processing service advertisements, the cost is approximately 236 ms (IBE decryption) + 11 ms (ECDSA signature verification) = 247 ms.

The advertisement size can also be estimated analytically. From Figure 3, our service advertisement has the following form:

$$\mathsf{bid}, \mathsf{PE.Enc}(\pi_S, (\text{ID}_S, g^s, \text{SIG}_S(\mathsf{bid}, \text{ID}_S, g^s)))$$

Our implementation of prefix encryption ($\mathsf{PE.Enc}$) has a ciphertext overhead[9] of 208 bytes on top of the plaintext. The Diffie-Hellman exponent (g^s) is 32 bytes, the broadcast id (bid) is 16 bytes, the ECDSA signature is 64 bytes, and a certificate chain (ID_S) of length three is approximately 500 bytes in size. The overall service advertisement is about 820 bytes.

Deployment. We deploy our service discovery protocol within the Vanadium discovery framework. The protocol allows services to advertise themselves while restricting visibility to an authorized set of clients. The Vanadium discovery API allows services to advertise over both mDNS and BLE. An mDNS `TXT` record has a maximum size of 1300 bytes [CK13a, CK13b], which suffices for service advertisements. Some care must be taken to ensure that the mDNS hostname and service label do no leak any private information about the service. For instance, we set the service label to `_vanadium_private._tcp.local` and hide the actual service type and instance name.

[9]This includes the size of the BB_2 ciphertext and the overhead of the Fujisaki-Okamoto transformation. The `dclxvi` library uses an optimized floating-based representation [NNS10] of points on the elliptic curve, which is not very space-efficient. By representing the points directly as one (using point compression [Jiv14]) or two field elements, it is possible to significantly reduce this overhead.

When the policy has multiple prefixes, our advertisements would no longer fit in a single mDNS `TXT` record. Furthermore, BLE advertisement payloads are restricted to 31 bytes [BLE14], which is far too small to fit a full service advertisement.

We can address this by running an auxiliary service (over GATT or peer-to-peer WiFi as in Apple AirDrop) to host the encrypted advertisement for the main service, and advertise the endpoint of this auxiliary service over BLE. The privacy of the main service is unaffected since the endpoint of the auxiliary service reveals nothing about it.

Encrypted service advertisements can only be interpreted by authorized clients. Thus, these advertisements can also be served by members of the network other than the service itself. A volunteering node may choose to cache the encrypted private service advertisements it sees on the network (for example, any advertisement within radio range) and serve them to clients. Clients can fetch advertisements for multiple services in the network in a single round-trip to the volunteer, without having to connect to each advertising device. While such volunteer caching nodes imply additional infrastructure, the key point is that their presence improves overall communication and power efficiency in the network without any loss of privacy to the advertising services or their clients.

6.4 Fixing AirDrop

Recall from Section 2.1 that during an AirDrop file exchange in contacts-only mode, a hash of the sender's identity is advertised over BLE and matched by potential receivers against their contacts. If there is a match, the receiver starts a service that the sender can connect to using TLS (version 1.2). In the TLS handshake, the sender and receiver exchange their certificates in the clear, which makes them visible to eavesdroppers on the network. This privacy vulnerability can be fixed using the private mutual authentication protocol from Section 4. In particular, once the receiver matches the sender's hash against one of its contacts, it uses the prefix encryption scheme to encrypt its identity under the name of the contact that matched the sender's hash. This ensures that the receiver's identity is only shared with the intended contact, even in the presence of active network attackers.[10] To deploy this protocol, Apple would provision all AirDrop-enabled devices with a secret key extracted for their identity in addition to their existing iCloud certificates. Apple would be the IBE root in this case.

7 Extensions

In this section, we describe several ways to extend our private mutual authentication and service discovery protocols to provide additional properties such as policy-hiding, delegation of decryption capabilities, and support for more complex authorization policies.

Policy-hiding authentication and discovery. A limitation of our private mutual authentication and service discovery protocols from Sections 4 and 5 is that the server's authorization policy is revealed to a network attacker. One solution to this problem is to use a "policy-hiding" prefix encryption scheme, where ciphertexts hide the prefix to which they are encrypted. This is

[10] The privacy guarantee for the sender is not as robust as information about its identity leaks via the hash advertised over BLE.

possible if we build the prefix encryption scheme from an *anonymous* IBE scheme such as Boneh-Franklin [BF01]. In anonymous IBE, ciphertexts do not reveal their associated identity, and thus, we achieve policy-hiding encryption. A drawback of the Boneh-Franklin scheme is that it requires evaluating a pairing during *both* encryption and decryption, thus increasing the computational cost over the BB_2 IBE scheme, where a pairing is only needed during decryption.

Delegating decryption abilities. In our current framework, only IBE roots can issue identity keys to principals. But there are many natural scenarios where non-root identities should also be able to issue identity keys. For example, an identity provider might issue Alice a certificate chain for the name `IDP/User/Alice`. While Alice can now issue certificates for child domains like `IDP/User/Alice/Phone`, she cannot extract an identity key for the child domain. As a result, delegation of decryption capabilities must still go through the IBE root for Alice's domain. To allow Alice to issue identity keys corresponding to derivatives (i.e., further extensions) of her name, we can use a hierarchical IBE scheme [GS02, HL02] to construct the underlying prefix encryption scheme. In hierarchical IBE, a principal holding a key for an identity `A/B/` is able to issue keys for any derivative identity `A/B/C`. This allows principals to independently manage their own subdomain without needing to go through a central identity provider.

Complex authorization policies. Our private mutual authentication and service discovery protocols support authorization policies that can be formulated as prefix constraints. While these are a very natural class of policies for the SDSI model, there are scenarios that may require a more diverse set of policies. For example, a server's authorization policy might require that the client's user ID be between 0 and 100. While such range queries can be expressed using a (large) number of prefixes, the size of the server's broadcast message grows with the number of prefixes in the policy. One way of supporting a more complex set of policies is to use a more general encryption scheme, such as an attribute-based encryption (ABE) [SW05, BSW07, GVW13, BGG$^+$14, GVW15]. For instance, if the policies can be expressed as Boolean formulas, then the Bethencourt et al. ABE scheme [BSW07] provides an efficient solution from pairings. Supporting more complicated policies is possible using lattice-based methods [GVW13, BGG$^+$14, GVW15], but this increased expressivity comes at the expense of performance.

8 Related Work

In this section, we survey some of the existing work on developing private mutual authentication and service discovery protocols.

Private mutual authentication. The term "private authentication" was first introduced by Abadi and Fournet [Aba03, AF04]. These papers define a privacy property for authentication protocols and the challenges associated with achieving it. Their definition of privacy is similar to the one we describe in Section 2.2 and states that the identity of each protocol participant is revealed to its peer only if the peer is authorized by the participant. In [AF04], Abadi and Fournet introduce two private authentication protocols and formally analyze them in the applied pi calculus. In addition to showing privacy, they also show that their protocol achieves unlinkability. While our private mutual authentication protocol shown in Figure 1 does not provide unlinkability, the variant described in Section 4.1 (shown in Figure 2) does achieve unlinkability.

A key difference between our work and [AF04] is the fact that we support prefix-based authorization policies. The protocols in [AF04] require the authorization policy to be specified by a set of public keys and do not scale when the set of public keys is very large. Moreover, in their protocol, clients and servers must either maintain (potentially long) lists of public keys for authorized peers, or look up a peer's public key in a directory service before each authentication request. In contrast, using IBE both enables support for more expressive authorization policies and eliminates the need to either maintain lists of public keys or run a public-key directory service.

Many other cryptographic primitives have also been developed for problems related to private mutual authentication, including secret handshakes [BDS+03, JKT06, AKB07], oblivious signature-based envelopes [LDB05], oblivious attribute certificates [LL06], hidden credentials [HBSO03, FAL04, BHS04], and more.

Secret handshakes and their extensions are protocols based on bilinear pairings that allow members of a group to identify each other privately. A key limitation of secret handshakes is that the parties can only authenticate using credentials issued by the same root authority. Oblivious signature-based envelopes [LDB05], oblivious attribute certificates [LL06] and hidden credentials [HBSO03, FAL04, BHS04] allow a sender to send an encrypted message that can be decrypted only by a recipient that satisfies some policy. Hidden credentials additionally hide the sender's policy. Closely related are the cryptographic primitives of attribute-based encryption [BSW07, GVW13] and predicate encryption [KSW08, GVW15], which allow more fine-grained control over decryption capabilities.

The protocols we have surveyed here are meant for *authentication*, and not authenticated key-exchange, which is usually the desired primitive. Integrating these protocols into existing key-exchange protocols such as SIGMA or TLS 1.3 is not always straightforward and can require non-trivial changes to existing protocols. In contrast, our work shows how IBE-based authentication can be very naturally integrated with existing secure key-exchange protocols (with minimal changes) to obtain private mutual authentication. Moreover, our techniques are equally applicable in the service discovery setting, and can be used to obtain 0-RTT private mutual authentication.

Service discovery. There is a large body of work on designing service discovery protocols for various environments—mobile, IoT, enterprise and more; we refer to [ZMN05] for a survey. Broadly, these protocols can be categorized into two groups: "directory-based" protocols and "directory-free" protocols.

In directory-based discovery protocols [CZH+99, ZMB+10, Jin13], there is a central directory that maintains service information and controls access to the services. Clients query directories to discover services while services register with the directory to announce their presence. While directory-based protocols allow for centralized management and tend to be computationally efficient, their main drawback is that they force dependence on an external service. If the directory service is unavailable then the protocol ceases to work. Even worse, if the directory service is compromised, then both server and client privacy is lost. Besides, mutually suspicious clients and servers may not be able to agree on a common directory service that they both trust. In light of these downsides, we designed decentralized, peer-to-peer protocols in this work.

Directory-free protocols, such as [KW14a, KW14b, ZMN04, ZMN06], typically rely on a shared key established between devices in a separate, out-of-band protocol. The shared key is then used to encrypt the private service advertisements so that only paired devices can decrypt. Other protocols like UPnP [Ell02] rely on public key encryption, where each device maintains a set of public keys

for the peers it is willing to talk to. In contrast, key-management in our IBE-based solution is greatly simplified—devices do not have to maintain long lists of symmetric or public keys. Our protocol is similar to the Tryst protocol [PGM+07], which proposes using an anonymous IBE scheme for encrypting under the peer's name (based on using a mutually agreed upon convention). A distinguishing feature of our protocol over Tryst is the support for prefix-based authorization policies.

9 Conclusion

Automatic service discovery is an integral component of the Internet of Things. While numerous service discovery protocols exist, few provide any notion of privacy. Motivated by the privacy shortcomings of the Apple AirDrop protocol, we introduce several important security and privacy goals for designing service discovery protocols for both IoT and mobile applications. We then show how to combine off-the-shelf identity-based encryption with existing key-exchange protocols to obtain both a private mutual authentication and a private service discovery protocol. Our benchmarks on the various devices show that our protocols are viable for a wide range of practical deployments and IoT scenarios.

Acknowledgments

We thank Martín Abadi, Mike Burrows, Felix Günther, and Adam Langley for many helpful comments and suggestions. We thank Bruno Blanchet for his help in verifying the unlinkability property of our modified private mutual authentication protocol in Section 4. We thank Felix Günther for pointing out an error in an earlier version of our private discovery protocol. This work was supported by NSF, DARPA, a grant from ONR, the Simons Foundation, and an NSF Graduate Research Fellowship. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[Aba03] Martín Abadi. Private authentication. In *PETS*, pages 27–40, 2003.

[ABB+04] William Aiello, Steven M. Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Angelos D. Keromytis, and Omer Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Trans. Inf. Syst. Secur.*, 7(2):242–273, 2004.

[ABP+15] Martín Abadi, Mike Burrows, Himabindu Pucha, Adam Sadovsky, Asim Shankar, and Ankur Taly. Distributed authorization with distributed grammars. In *Programming Languages with Applications to Biology and Security*, pages 10–26, 2015.

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In *CT-RSA*, pages 143–158, 2001.

[AF04] Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322:427–476, 2004.

[AKB07] Giuseppe Ateniese, Jonathan Kirsch, and Marina Blanton. Secret handshakes with dynamic and fuzzy matching. In *NDSS*, 2007.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *STOC*, pages 419–428, 1998.

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana K. Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *2003 IEEE S&P 2003*, pages 180–196, 2003.

[Ber09] Daniel J. Bernstein. Cryptography in NaCl, 2009.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

[BHS04] Robert W. Bradshaw, Jason E. Holt, and Kent E. Seamons. Concealing complex policies with hidden credentials. In *ACM CCS*, pages 146–157, 2004.

[BLE14] Bluetooth specification version 4.2, 2014.

[BLS12] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *LATINCRYPT*, pages 159–176, 2012.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *FOCS*, pages 112–117, 1982.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, pages 531–545, 2000.

[Bon13] Bonjour printing specification version 1.2, 2013.

[BR93a] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.

[BR93b] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.

[Bro09] Daniel R. L. Brown. SEC 1: Elliptic Curve Cryptography, 2009.

[BRW03] Mihir Bellare, Phillip Rogaway, and David Wagner. EAX: A conventional authenticated-encryption mode. *IACR Cryptology ePrint Archive*, 2003:69, 2003.

[BSC16] Bruno Blanchet, Ben Smyth, and Vincent Cheval. Proverif 1.93: Automatic cryptographic protocol verifier, user manual and tutorial, 2016.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE S&P*, pages 321–334, 2007.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001.

[CK02] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *CRYPTO*, pages 143–161, 2002.

[CK13a] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. RFC 6763 (Proposed Standard), February 2013.

[CK13b] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762 (Proposed Standard), February 2013.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, pages 360–363, 2001.

[CZH+99] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *MobiCom*, pages 24–35, 1999.

[DGH+04] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the cbc, cascade and HMAC modes. In *CRYPTO*, pages 494–510, 2004.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[DY81] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. In *FOCS*, pages 350–357, 1981.

[Ell02] Carl M. Ellison. Home network security. *Intel Technology Journal*, 6(4):37–48, 2002.

[FAL04] Keith B. Frikken, Mikhail J. Atallah, and Jiangtao Li. Hidden access control policies with hidden credentials. In *ACM WPES*, page 27, 2004.

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's QUIC protocol. In *ACM CCS*, pages 1193–1204, 2014.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, pages 503–523, 2015.

[HBSO03] Jason E. Holt, Robert W. Bradshaw, Kent E. Seamons, and Hilarie K. Orman. Hidden credentials. In *ACM WPES*, pages 1–8, 2003.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.

[Inc15] Apple Inc. The transport layer security (TLS) protocol version 1.3, September 2015.

[Jin13] Jini(TM) network technology specifications – Apache river version 2.2.0, 2013.

[Jiv14] A. Jivsov. Compact representation of an elliptic curve point, March 2014.

[JKT06] Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Authentication for paranoids: Multi-party secret handshakes. In *ACNS*, pages 325–339, 2006.

[KBSW13] Bastian Könings, Christoph Bachmaier, Florian Schaub, and Michael Weber. Device names in the wild: Investigating privacy risks of zero configuration networking. In *IEEE MDM*, pages 51–56, 2013.

[KE10] H. Krawczyk and P. Eronen. HMAC-based extract-and-expand key derivation function (HKDF), 2010.

[KM13] Neal Koblitz and Alfred Menezes. Another look at security definitions. *Adv. in Math. of Comm.*, 7(1):1–38, 2013.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO*, pages 429–448, 2013.

[Kra03] Hugo Krawczyk. SIGMA: the 'SIGn-and-MAc' approach to authenticated diffie-hellman and its use in the ike-protocols. In *CRYPTO*, pages 400–425, 2003.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO*, pages 631–648, 2010.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, 2008.

[KW14a] D. Kaiser and M. Waldvogel. Adding privacy to multicast dns service discovery. In *IEEE TrustCom*, pages 809–816, 2014.

[KW14b] D. Kaiser and M. Waldvogel. Efficient privacy preserving multicast dns service discovery. In *IEEE CSS*, 2014.

[KW15] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. *IACR Cryptology ePrint Archive*, 2015:978, 2015.

[LDB05] Ninghui Li, Wenliang Du, and Dan Boneh. Oblivious signature-based envelope. *Distributed Computing*, 17(4), 2005. Extended abstract in ACM PODC 2003.

[LJBN15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is quic? provable security and performance analyses. In *IEEE Symposium on Security and Privacy*, pages 214–231, 2015.

[LL06] Jiangtao Li and Ninghui Li. Oacerts: Oblivious attribute certificates. *IEEE Trans. Dependable Sec. Comput.*, 3(4):340–352, 2006.

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In *EUROCRYPT*, pages 58–76, 2014.

[MS04] Alfred Menezes and Nigel P. Smart. Security of signature schemes in a multi-user setting. *Des. Codes Cryptography*, 33(3):261–274, 2004.

[NNS10] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *LATINCRYPT*, pages 109–123, 2010.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.

[PGM+07] Jeffrey Pang, Ben Greenstein, Damon McCoy, Srinivasan Seshan, and David Wetherall. Tryst: The case for confidential service discovery. In *HotNets*, 2007.

[Res15] E. Rescorla. The transport layer security (TLS) protocol version 1.3, July 2015.

[RL96] Ronald L. Rivest and Butler Lampson. SDSI - a simple distributed security infrastructure. Technical report, 1996.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *ACM CCS*, pages 98–107, 2002.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[Sho99] Victor Shoup. On formal models for secure key exchange. *IACR Cryptology ePrint Archive*, 1999:12, 1999.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[UPn15] UPnP(TM) device architecture 2.0, 2015.

[Van] Vanadium. `http://vanadium.github.io/`.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91, 1982.

[Zer04] IETF zero configuration networking (zeroconf). `https://datatracker.ietf.org/doc/charter-ietf` 2004.

[ZMB+10] Feng W. Zhu, Matt W. Mutka, Anish Bivalkar, Abdullah Demir, Yue Lu, and Chockalingam Chidambarm. Toward secure and private service discovery anywhere anytime. *Frontiers of Computer Science in China*, 4(3):311–323, 2010.

[ZMN04] Feng W. Zhu, Matt W. Mutka, and Lionel M. Ni. PrudentExposure: A private and user-centric service discovery protocol. In *IEEE PerCom*, pages 329–340, 2004.

[ZMN05] Feng W. Zhu, Matt W. Mutka, and Lionel M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 4(4):81–90, 2005.

[ZMN06] Feng W. Zhu, Matt W. Mutka, and Lionel M. Ni. A private, secure, and user-centric information exposure model for service discovery protocols. *IEEE Trans. Mob. Comput.*, 5(4):418–429, 2006.

A Additional Preliminaries

In this section, we review some additional preliminaries.

Identity-based encryption. Identity-based encryption (IBE) [Sha84, BF01, Coc01, BB04] is a generalization of public-key encryption where the public keys can be arbitrary strings, called *identities*. An IBE scheme consists of four algorithms:

- **Setup.** $\mathsf{Setup}(1^\lambda)$ takes the security parameter λ and outputs a master public key $\textsc{mpk}$ and a master secret key $\textsc{msk}$.
- **Extract.** $\mathsf{Extract}(\textsc{msk}, \textsc{id})$ takes $\textsc{msk}$ and an identity $\textsc{id}$ and outputs an identity secret key $\textsc{sk}_{\textsc{id}}$.
- **Encrypt.** $\mathsf{Encrypt}(\textsc{mpk}, \textsc{id}, m)$ takes $\textsc{mpk}$, an identity $\textsc{id}$, and a message m and outputs a ciphertext $\textsc{ct}$.
- **Decrypt.** $\mathsf{Decrypt}(\textsc{sk}_{\textsc{id}}, \textsc{ct})$ takes $\textsc{ct}$ and $\textsc{sk}_{\textsc{id}}$ and outputs a message m or a special symbol $\perp$.

The correctness requirement for an IBE scheme states that for all messages m, identities $\textsc{id}$ and keys $(\textsc{mpk}, \textsc{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, if $\textsc{sk} \leftarrow \mathsf{Extract}(\textsc{msk}, \textsc{id})$ and $\textsc{ct} \leftarrow \mathsf{Encrypt}(\textsc{mpk}, \textsc{id}, m)$, then $\mathsf{Decrypt}(\textsc{sk}, \textsc{ct}) = m$. In this work, we require IBE schemes that are secure against adaptive chosen-ciphertext attacks (CCA-secure).

IBE security. We now present the formal game-based definition for adaptive-CCA security in the context of an IBE scheme [BF01]. First, we define the $\mathsf{IND\text{-}ID\text{-}CCA}$ security game between an adversary $\mathcal{A}$ and a challenger. In the following, we let λ denote the security parameter. The game consists of several phases:

- **Setup phase:** The challenger samples parameters $(\textsc{mpk}, \textsc{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, and sends $\textsc{mpk}$ to $\mathcal{A}$.

- **Pre-challenge phase:** The adversary can adaptively make key extraction and decryption queries to the challenger. In a key-extraction query, the adversary submits an identity ID. The challenger replies with an identity key $\mathrm{SK}_{\mathrm{ID}} \leftarrow \mathsf{Extract}(\mathrm{MSK}, \mathrm{ID})$. In a decryption query, the adversary submits a ciphertext CT and an identity ID. The challenger then extracts an identity key $\mathrm{SK}_{\mathrm{ID}} \leftarrow \mathsf{Extract}(\mathrm{MSK}, \mathrm{ID})$ and replies with $\mathsf{Decrypt}(\mathrm{SK}_{\mathrm{ID}}, \mathrm{CT})$.
- **Challenge phase:** Once the adversary has finished making queries in the pre-challenge phase, it outputs two equal-length messages $\bar{m}_0$ and $\bar{m}_1$ along with a target identity $\overline{\mathrm{ID}}$ ($\overline{\mathrm{ID}}$ must not have appeared in any of the adversary's extraction queries in the pre-challenge phase). The challenger chooses a random bit $b \xleftarrow{\mathrm{R}} \{0,1\}$ and replies with a ciphertext $\overline{\mathrm{CT}} \leftarrow \mathsf{Encrypt}(\mathrm{MPK}, \overline{\mathrm{ID}}, \bar{m}_b)$.
- **Post-challenge phase:** In the post-challenge phase, the adversary can continue to make extraction and decryption queries as in the pre-challenge phase. However, it cannot request a key for the target identity $\overline{\mathrm{ID}}$ or request a decryption of the challenge ciphertext $\overline{\mathrm{CT}}$ under identity $\overline{\mathrm{ID}}$.
- **Output phase:** At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$. We say the adversary "wins" the IND-ID-CCA game if $b' = b$.

Definition A.1 (IND-ID-CCA Security [BF01])**.** An IBE scheme (Setup, Encrypt, Decrypt, Extract) is IND-ID-CCA-secure if no efficient adversary can win the IND-ID-CCA security game with probability that is non-negligibly greater than 1/2.

Additional cryptographic primitives. We also review some standard cryptographic definitions for pseudorandom generators (PRGs), pseudorandom functions (PRFs), strong randomness extractors, and authenticated encryption. Below, we write $\mathsf{negl}(\lambda)$ to denote a negligible function in the security parameter λ.

Definition A.2 (Pseudorandom Generator [Yao82, BM82])**.** A pseudorandom generator $G : \{0,1\}^\ell \to \{0,1\}^n$ where $\ell = \ell(\lambda)$, $n = n(\lambda)$, $\ell < n$ is secure if for all efficient adversaries $\mathcal{A}$,

$$\left|\Pr\left[s \xleftarrow{\mathrm{R}} \{0,1\}^\ell : \mathcal{A}(1^\lambda, G(s)) = 1\right] - \Pr\left[z \xleftarrow{\mathrm{R}} \{0,1\}^n : \mathcal{A}(1^\lambda, z) = 1\right]\right| = \mathsf{negl}(\lambda).$$

Definition A.3 (Pseudorandom Function [GGM84])**.** A pseudorandom function $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^m$ with key space $\mathcal{K}$, domain $\{0,1\}^n$ and range $\{0,1\}^m$ is secure if for all efficient adversaries $\mathcal{A}$,

$$\left|\Pr\left[k \xleftarrow{\mathrm{R}} \mathcal{K} : \mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[f \xleftarrow{\mathrm{R}} \mathsf{Funs}[\{0,1\}^n, \{0,1\}^m] : \mathcal{A}^{f(\cdot)}(1^\lambda) = 1\right]\right| = \mathsf{negl}(\lambda),$$

where $\mathsf{Funs}[\{0,1\}^n, \{0,1\}^m]$ denotes the set of all functions from $\{0,1\}^n$ to $\{0,1\}^m$.

Definition A.4 (Strong Randomness Extractor [NZ96, adapted])**.** A function $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^m$ is a strong randomness extractor if for all adversaries $\mathcal{A}$,

$$\Big|\Pr\left[k \xleftarrow{\mathrm{R}} \mathcal{K}, x \xleftarrow{\mathrm{R}} \{0,1\}^n : \mathcal{A}(1^\lambda, k, E(k,x)) = 1\right] - \Pr\left[k \xleftarrow{\mathrm{R}} \mathcal{K}, z \xleftarrow{\mathrm{R}} \{0,1\}^m : \mathcal{A}(1^\lambda, k, z) = 1\right]\Big| = \mathsf{negl}(\lambda).$$

Definition A.5 (Authenticated Encryption [BN00])**.** A symmetric-key encryption scheme ($\mathsf{Encrypt}$, $\mathsf{Decrypt}$) over a key space $\mathcal{K}$, a message space $\mathcal{M}$, and a ciphertext space $\mathcal{C}$ is an authenticated encryption scheme if it satisfies the following properties:

- **Correctness:** For all messages $m \in \mathcal{M}$,

$$\Pr\left[k \xleftarrow{R} \mathcal{K} : \mathsf{Decrypt}(k, \mathsf{Encrypt}(k, m)) = m\right] = 1 - \mathsf{negl}(\lambda).$$

- **Semantic Security (IND-CPA):** For all efficient adversaries $\mathcal{A}$,

$$\left|\Pr\left[k \xleftarrow{R} \mathcal{K}, b \xleftarrow{R} \{0,1\}; b' \leftarrow \mathcal{A}^{\mathbf{LoR}(k,b,\cdot,\cdot)}(1^\lambda) : b' = b\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda),$$

  where $\mathbf{LoR}(k, b, m_0, m_1)$ is the "left-or-right" encryption oracle which on input a key $k \in \mathcal{K}$, a bit $b \in \{0,1\}$, and two messages $m_0, m_1 \in \mathcal{M}$, outputs $\mathsf{Encrypt}(k, m_b)$.

- **Ciphertext Integrity:** For all efficient adversaries $\mathcal{A}$,

$$\Pr\left[k \xleftarrow{R} \mathcal{K}; \text{CT} \leftarrow \mathcal{A}^{\mathsf{Encrypt}(k,\cdot)}(1^\lambda) : \text{CT} \notin Q \wedge \mathsf{Decrypt}(k, \text{CT}) \neq \perp\right] = \mathsf{negl}(\lambda),$$

  where Q is the set of ciphertexts output by the $\mathsf{Encrypt}$ oracle.

Key derivation. A key derivation function (KDF) [DGH+04, Kra10] is an algorithm for extracting randomness from some non-uniform entropy source to obtain a uniformly random string suitable for use in other cryptographic primitives. For our private mutual authentication protocol in Figure 1, the key-derivation function is implemented by a hash function: $\mathsf{KDF}(g^x, g^y, g^{xy}) = H(g^x, g^y, g^{xy})$, and our security analysis relies on the hardness of the Hash Diffie-Hellman assumption [ABR01] (described below). For our private service discovery protocol, we defer an explicit description of our key-derivation procedure to Appendix E, where we provide a formal analysis of our protocol.

Cryptographic assumptions. We now state the Diffie-Hellman-type assumptions [ABR01] we use to show security of our protocols. Let $\mathbb{G}$ be a cyclic group of prime order p with generator g, and let $H : \mathbb{G} \to \mathcal{Z}$ be a hash function from $\mathbb{G}$ to an output space $\mathcal{Z}$. Then, the Hash Diffie-Hellman (Hash-DH) assumption in $\mathbb{G}$ states that the following distributions are computationally indistinguishable:

$$(g, g^x, g^y, H(g^x, g^y, g^{xy})) \quad \text{and} \quad (g, g^x, g^y, z),$$

where $x, y \xleftarrow{R} \mathbb{Z}_p$ and $z \xleftarrow{R} \mathcal{Z}$. The Strong Diffie-Hellman (Strong-DH) assumption in $\mathbb{G}$ states that computing g^{xy} is hard given (g, g^x, g^y) where $x, y \xleftarrow{R} \mathbb{Z}_p$, even if the adversary is given access to a verification oracle $\mathcal{O}(g, g^x, \cdot, \cdot)$. The verification oracle $\mathcal{O}$ outputs 1 if the input (g, g^x, g^y, g^z) is a decisional Diffie-Hellman (DDH) tuple ($z = xy \bmod p$) and 0 otherwise. In other words, the Strong-DH assumption states that the computational Diffie-Hellman (CDH) problem is hard even with access to a (restricted) DDH oracle.

B Key-Exchange Security

In this section, we describe the Canetti-Krawczyk model of key-exchange (KE) in the "post-specified" peer setting [CK02, Kra03]. In the post-specified peer setting, a peer's identity is possibly unavailable at the beginning of the KE protocol. Instead, the identity of the peer is revealed during the KE protocol. For example, at the beginning of the KE protocol, the client might only know the server's IP address and nothing more about the identity of the server. By participating in the KE protocol, the client learns the identity of the server. Earlier works on key exchange have also considered the "pre-specified" peer setting [BR93a, Sho99, CK01], where the peer's identity is assumed to be known at the beginning of the protocol. Our description in this section is largely taken from the description in [CK02].

Following earlier works [BR93a, BCK98], Canetti and Krawczyk [CK01, CK02, Kra03] model a KE protocol as a multi-party protocol where each party can run one or more instances of the protocol. At the beginning of the protocol execution experiment, the challenger sets up the long-term secrets for each party (as prescribed by the protocol specification). Then, the adversary is given control of the protocol execution. In particular, the adversary can activate parties to run an instance of the key-exchange protocol called a *session*. Each session is associated with a session identifier. Within a session, parties can be activated to either initiate a session, or to respond to an incoming message. The purpose of the session is to agree on a *session key* with another party in the network (called the *peer*) by exchanging a sequence of messages. Sessions can run concurrently, and messages are directed to the session with the associated session identifier.

In the post-specified model, a party can be activated to initiate a KE session by a tuple (P, sid, d), where P is the party at which the session is activated, sid is a unique session identifier, and d is a "destination address" used only for the delivery of messages for session sid. In particular, d has no bearing on the identity of the peer. A party can be activated as either an initiator or as a responder (e.g., to respond to an initialization message from another party). The output of a completed KE session at a party P consists of a public tuple (P, sid, Q) as well as a secret value atk, where sid is the session id, Q is the peer of the session, and atk is the application-traffic (or session) key. In addition, sessions can be *aborted* without producing a session key. We denote the output of an aborted session with a special symbol $\perp$. When the session produces an output (as a result of completing or aborting), its local state is erased. Only the tuple (P, sid, Q) and the application-traffic key atk persists at the conclusion of the session. Each party may also have additional *long-term* state such as a signature signing key. These long-term secrets are shared across multiple sessions and are not considered to be a part of a party's local session state. In the protocol analysis, we will uniquely identify sessions by a pair (P, sid), where P is the local party and sid is the session identifier.

Attacker model. The attacker is modeled to capture realistic attack capabilities in an open network. In particular, the attacker has full control over the network communication, and moreover, is allowed access to some of the secrets used or generated by the key-exchange protocol. More precisely, the attacker is modeled as a probabilistic polynomial-time machine with full control over the network. It is free to intercept, delay, drop, inject, or tamper with messages sent between the parties. In addition, the attacker can activate parties (as either an initiator or a responder) to begin an execution of the KE protocol. The attacker can also issue the following session exposure queries to learn ephemeral and long-term secrets held by the parties:

- StateReveal: The adversary can issue a StateReveal query against any incomplete session. It is then given the local state associated with the targeted session. Importantly, the local state does *not* contain long-term secrets such as signature signing keys which are shared across multiple sessions involving the same party.
- KeyReveal: The adversary can issue a KeyReveal query against any completed session. It is then given the application-traffic key atk associated with the targeted session.
- Corrupt: The adversary can issue a Corrupt query on any party P, at which point it learns all information in the memory of P, including any long-term secrets. After corrupting a party P, the adversary dictates all subsequent actions by P.

Note that the adversary in this model is fully adaptive.

Security definition. To define the security of a KE protocol, Canetti and Krawczyk give an indistinguishability-based definition that intuitively states that an adversary cannot distinguish the real session key output by an session from a uniformly random session key. However, since the adversary has the ability to make session-exposure queries, this definition is only meaningful if the power of the adversary is restricted. In particular, the adversary can only try to distinguish the session key for sessions (P, sid) on which it did not issue one of the three session-exposure queries defined above (otherwise, it can trivially distinguish the session key from a uniformly random key). Formally, if the adversary makes a session-exposure query on (P, sid), the session (P, sid) is considered exposed. Moreover, any sessions that "match" (P, sid) are also considered to be exposed. This restriction is necessary since the matching session also computes the same session key (in an honest execution of the protocol), thus allowing again a trivial distinguishing attack. In the post-specified model, a session (Q, sid) *matches* a completed session with public output (P, sid, Q) if the session (Q, sid) is incomplete or if the session (Q, sid) completes with public output (Q, sid, P).

To be more precise, at any point in the indistinguishability game, the adversary can issue a Test query on any completed session (P, sid). Let (P, sid, Q) be the public output and atk be the secret output by this session. The challenger then sets $k_0 = \mathsf{atk}$ and $k_1 \xleftarrow{\mathrm{R}} \mathcal{K}$, where $\mathcal{K}$ is the key-space for the session key. The challenger then chooses a bit $b \xleftarrow{\mathrm{R}} \{0,1\}$ and gives the adversary k_b. The adversary can continue making session exposure queries as well as activate sessions and interact with the parties after making the Test query. At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$. The adversary wins the distinguishing game if $b' = b$. An adversary is *admissible* if the session (P, sid) and all sessions matching (P, sid, Q) are unexposed. We now state the formal security definition from [CK02, Kra03].

Definition B.1 (Key-exchange security [CK02, Kra03])**.** A key-exchange protocol π is secure if for all efficient and admissible adversaries $\mathcal{A}$, the following properties hold:

1. Suppose a session (P, sid) completes at an uncorrupted party P with public output (P, sid, Q) and secret output atk. Then, with overwhelming probability, if Q completes session (Q, sid) while P and Q are uncorrupted, the public output of (Q, sid) is (Q, sid, P), and the secret output is atk.
2. The probability that $\mathcal{A}$ wins the test-session distinguishing game is negligibly close to $1/2$.

B.1 Service Discovery Model

In Section 5, we introduced a private service discovery protocol that consisted of two sub-protocols: a broadcast protocol where the server announced its identity and other relevant endpoint information, and a 0-RTT mutual authentication protocol. In this section, we describe how we can extend the Canetti-Krawczyk key-exchange framework to also reason about the security of the service discovery protocol.

One major difference in the service discovery setting is that multiple clients can legitimately respond to the server's broadcast message and initiate handshakes with the server. In contrast, in the mutual authentication setting, all messages sent between clients and servers are intended for a single peer. Thus, the server's broadcast message might include "semi-static" secrets that persist for the lifetime of the broadcast message (which might include more than one session). In our security model, the servers can maintain three kinds of state: local session state that persists for a single session, semi-static state that persists for the lifetime of a broadcast, and long-term state that persists across multiple broadcasts.

In the service discovery setting, the adversary is allowed to activate servers to produce a broadcast message. Specifically, a server can be activated to construct a broadcast message by a tuple (P, bid) where P is the party being activated, and bid is a unique broadcast identifier. Next, the adversary can activate parties to respond to a broadcast message by providing a tuple $(P, \mathsf{bid}, \mathsf{sid})$ along with a broadcast message. Here, P identifies the party that is being activated to initiate a key-exchange protocol, bid is the broadcast identifier, and sid is a session specific identifier. In contrast to the basic key-exchange model, the session activation tuple does not include a destination address. It is assumed that the server's discovery broadcast already includes the relevant endpoint information. In the broadcast setting, the pair $(\mathsf{bid}, \mathsf{sid})$ is taken to be the unique session identifier. The output of a successful key-exchange session at a party P consists of a public tuple $(P, \mathsf{bid}, \mathsf{sid}, Q)$, where $(\mathsf{bid}, \mathsf{sid})$ identifies the session and Q is the peer of the session, as well as a secret application-traffic key atk. As in the key-exchange setting, sessions can also abort without producing any output. When a local session completes (either successfully or due to aborting), the local session state is erased. Moreover, whenever a server is activated to initialize a new broadcast, its previous broadcast state is also erased. In the subsequent protocol analysis, we will uniquely identify broadcasts by a pair (P, bid) and sessions by a triple $(P, \mathsf{bid}, \mathsf{sid})$.

In the service discovery setting, the adversary's goal is still to distinguish the session key output by a completed session from a session key that is chosen uniformly at random. In addition, we allow the adversary an additional query that allows it to learn any semi-static secrets that persist for the lifetime of a particular broadcast:

- $\mathsf{BroadcastReveal}$: The adversary can issue a $\mathsf{BroadcastReveal}$ query against any server that has initiated a broadcast. The adversary is given any semi-static state the server is maintaining for the lifetime of its current broadcast. However, the adversary is not given any long-term secrets such as signing keys which persist between broadcasts.

Admissibility requirements. In the key-exchange security game, it was necessary to impose restrictions on the adversary's power to prevent it from trivially winning the security game. The same holds in the discovery setting. Specifically, we say a session $(P, \mathsf{bid}, \mathsf{sid})$ is "exposed" if the adversary makes the following queries:

- The adversary makes a $\mathsf{KeyReveal}$ query on $(P, \mathsf{bid}, \mathsf{sid})$.

- The adversary makes a $\mathsf{Corrupt}$ query on P before the session $(P, \mathsf{bid}, \mathsf{sid})$ has expired.
- P is the initiator of the key-exchange protocol (the client), and the adversary had made a $\mathsf{StateReveal}$ query on $(P, \mathsf{bid}, \mathsf{sid})$.
- P is the responder in the key-exchange protocol (the server), and the adversary has made both a $\mathsf{StateReveal}$ query on $(P, \mathsf{bid}, \mathsf{sid})$, and either a $\mathsf{BroadcastReveal}$ query on (P, bid) or a $\mathsf{Corrupt}$ query on P before (P, bid) has expired.

Moreover, when the adversary makes one of the above queries and exposes a session, all "matching" sessions are also marked as exposed. We adapt the definition from the basic key-exchange setting: a session $(Q, \mathsf{bid}, \mathsf{sid})$ matches a completed session with public output $(P, \mathsf{bid}, \mathsf{sid}, Q)$ if $(Q, \mathsf{bid}, \mathsf{sid})$ is incomplete or if $(Q, \mathsf{bid}, \mathsf{sid})$ completes with public output $(Q, \mathsf{bid}, \mathsf{sid}, P)$.

We give some intuition about the exposure conditions. The last requirement effectively states that in the case of server compromise, as long as one of the semi-static secret (from the broadcast) and the ephemeral session-specific secret is not compromised, the adversary cannot learn anything about the negotiated key. The above definition also captures perfect forward secrecy since the adversary is allowed to corrupt parties after the expiration of the test session.

We can now define security for the service discovery protocol in the same manner as we defined key-exchange security for a mutual authentication protocol (Definition B.1):

Definition B.2 (Service discovery security)**.** A service discovery protocol π is secure if for all efficient and admissible adversaries $\mathcal{A}$, the following properties hold:

1. Suppose a session $(P, \mathsf{bid}, \mathsf{sid})$ completes at an uncorrupted party P with public output $(P, \mathsf{bid}, \mathsf{sid}, Q)$ and secret output atk. Then, with overwhelming probability, if Q completes session $(Q, \mathsf{bid}, \mathsf{sid})$ while P and Q are uncorrupted, the public output of $(Q, \mathsf{bid}, \mathsf{sid})$ is $(Q, \mathsf{bid}, \mathsf{sid}, P)$, and the secret output is atk.
2. The probability that $\mathcal{A}$ wins the test-session distinguishing game is negligibly close to $1/2$.

C Key-Exchange Privacy

In this section, we formally define our notion of privacy for a key-exchange protocol. Intuitively, we say a protocol is private if no efficient client or server is able to learn anything about another party's identity unless they satisfy that party's authorization policy. This property should hold even if the adversary corrupts multiple parties; as long as none of the corrupted parties satisfy the target's authorization policy, then they should not learn anything about the target's identity. More formally, we operate in the Canetti-Krawczyk key-exchange model, but make several modifications to the adversary's capabilities and its objectives. We now describe our model formally.

Formal definition. Let n be the number of parties participating in the protocol execution experiment. We denote the parties $P_1, \ldots, P_n$. In addition to the n parties, we also introduce a special test party P_T whose identity will be hidden from the adversary at the beginning of the protocol execution experiment. We now define two experiments Expt_0 and Expt_1. For $b \in \{0, 1\}$, the experiment Expt_b proceeds as follows:

- **Setup phase:** At the beginning of the experiment, the adversary submits a tuple of distinct identities $(\text{ID}_1, \ldots, \text{ID}_n)$ and two challenge identities $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ (distinct from $\text{ID}_1, \ldots, \text{ID}_n$). For each $i \in [n]$, the challenger sets up the long-term secrets for party P_i. For instance, depending on the protocol specifications, the challenger might issue a certificate binding the credentials for P_i to its identity ID_i. The challenger associates the identity $\text{ID}_T^{(b)}$ with the special party P_T, and performs the setup procedure for P_T according to the protocol specification.

- **Protocol execution:** In this phase, the adversary is free to activate any party ($P_1, \ldots, P_n$ and P_T) to initiate sessions and respond to session messages. When the adversary activates a party P to initiate a session (P, sid), it can also specify a policy $\pi_{(P,\mathsf{sid})}$ associated with the session (P, sid). Similarly, when the adversary activates a party Q as a responder, it can specify a policy $\pi_{(Q,\mathsf{sid})}$ associated with the session (Q, sid). In addition, when the adversary activates a party Q to respond to a session message for session (Q, sid), it also specifies the sender's policy. Party Q answers the query if and only if it satisfies the sender's policy. If the adversary does not specify a policy for a session (P, sid), the party P accepts all connections. The adversary can also issue $\mathsf{StateReveal}$, $\mathsf{KeyReveal}$ and $\mathsf{Corrupt}$ queries on any party participating in the protocol execution (with the same semantics as in the key-exchange security game described in Appendix B).

- **Output phase:** At the end of the protocol execution, the adversary outputs a bit $b' \in \{0, 1\}$.

Intuitively, we will say that a key-exchange protocol is *private* if no efficient adversary can distinguish Expt_0 from Expt_1 with probability that is non-negligibly greater than $1/2$. This captures the property that no active network adversary is able to distinguish interactions with a party $\text{ID}_T^{(0)}$ from those with a party $\text{ID}_T^{(1)}$. Of course, as defined currently, an adversary can trivially win the security game. Since we have not placed any restrictions on the adversary's queries, it can, for instance, corrupt the target party P_T and learn its identity.

Admissibility requirements. To preclude trivial ways of learning the identity of the test party, we enumerate a series of constraints on the adversary's power. We say an adversary is "admissible" for the privacy game if the following conditions hold:

- The adversary does not issue a $\mathsf{Corrupt}$ query on P_T. In addition, the adversary does not make a $\mathsf{StateReveal}$ query on any session (P_T, sid) that completes.

- Whenever a session (P_T, sid) completes with public output Q, then the adversary has not made a $\mathsf{StateReveal}$ query on the session (Q, sid), and moreover, Q is not corrupt before the completion of (P_T, sid).

- Let Π_T denote the set of policies the adversary has associated with the test party P_T. Let $I \subseteq [n]$ be the indices of the parties the adversary has corrupted in the course of the protocol execution. Then, for all policies $\pi \in \Pi_T$ and indices $i \in I$, it should be the case that ID_i does not satisfy π.

- Whenever the adversary associated a policy π with a particular session (P, sid), it must be the case that either $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ both satisfy π or neither satisfy π.

These admissibility requirements are designed to rule out several "trivial" ways of breaking the privacy of the test party P_T in the protocol execution experiment. Certainly, if the adversary corrupted the test party, or exposed (via a StateReveal or a Corrupt query) a completed session (P_T, sid), it can easily learn the identity of P_T. These scenarios are captured by the first two admissibility requirements. The third admissibility requirements states that the adversary never corrupts a party who is authorized to talk with the test party. This is unavoidable because a party is always willing to reveal its identity to any other party that satisfies its policy. Thus, if the adversary obtained the credentials of a party that satisfied the test party's policy, it should be able to learn the test party's identity. The final admissibility requirement captures the fact that we work in the model where a network adversary can always tell whether a handshake completes or aborts. Thus, if the adversary is allowed to choose the policies the protocol participants use, then it must be constrained so it cannot choose a policy that allows it to trivially distinguish the two test identities.

With these admissibility requirements, we now state our privacy definition for key-exchange protocols.

Definition C.1 (Key-exchange privacy)**.** A key-exchange protocol π is private if for all efficient and admissible adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ outputs 1 in Expt_0 is negligibly close to the probability that it outputs 1 in Expt_1.

The SIGMA-I protocol. To provide some additional intuition on the nature of our privacy definition, we first show that the SIGMA-I protocol [Kra03, CK02] does not satisfy our notion of privacy against *active* network attackers. Consider the case where we have two parties: P_1 and the test party P_T. The adversary chooses an arbitrary identity ID_1 for P_1 and two distinct identities $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ for P_T. The adversary activates P_1 to initiate a session (P, sid), and forwards the message (a DH share g^x) to P_T. The test party P_T responds with its DH share g^y and an encryption of its identity (along with other components) under an encryption key derived from g^x and g^y. At this point in the protocol execution, none of the sessions have completed. The adversary now performs a StateReveal query on P_1 to learn x, from which it can derive the key P_T used to encrypt its message. Now, the adversary simply decrypts the response message from P_T and recovers its identity. Thus, we conclude that the original SIGMA-I protocol does not provide privacy in the presence of an active network adversary.

D Analysis of Private Mutual Authentication Protocol

In this section, we show that the protocol in Figure 1 from Section 4 is a private mutual authentication protocol (satisfies both Definition B.1 and Definition C.1). In the following description, we will often refer to the first message from the client to the server as the "initialization" message, the server's response as the "response" message, and the final message from the client as the "finish" message.

Security. Security of our protocol follows directly from the security of the SIGMA-I protocol. Specifically, we have the following theorem.

Theorem D.1. *The protocol in Figure 1 is a secure key-exchange protocol (Definition B.1) assuming the Hash-DH assumption holds in $\mathbb{G}$ and the security of the underlying cryptographic primitives*

(the signature scheme and the authenticated encryption scheme).

Proof. As noted in Section 4, the two differences between our protocol and the SIGMA-I protocol from [CK02] is that we use authenticated encryption to encrypt and authenticate the handshake messages rather than separate encryption and MAC schemes. The other difference is that in our protocol, the server substitutes a prefix-based encryption of its certificate chain for its actual certificate chain in the response message to the client.

In the original SIGMA-I protocol, the requirement on the certificate chain is that it provides an authenticated binding between an identity and its public verification key. An encrypted certificate chain provides the exact same level of assurance. After all, an honest client only accepts the server's credential if the client successfully decrypts the encrypted certificate chain and verifies the underlying certificate chain. Therefore, we can substitute encrypted certificate chain into the SIGMA-I protocol with no loss in security. Security of our resulting protocol then follows directly from security of the original SIGMA-I protocol [CK02, §5.3]. □

Privacy. We now show that our key-exchange protocol is private. In our analysis, we construct the encryption scheme used to encrypt certificates from an IBE scheme as described in Section 3.1. Privacy for the client's identity then reduces to the security of the underlying key-exchange (since the client encrypts its identity under a handshake secret key), while privacy for the server's identity reduces to CCA-security of the underlying IBE scheme (since the server encrypts its identity using a prefix encryption scheme constructed from IBE).

Theorem D.2. *The protocol in Figure 1 is private (Definition C.1) assuming the IBE scheme used to construct the prefix encryption scheme is* IND-ID-CCA*-secure, the Hash-DH assumption holds in* $\mathbb{G}$*, and the underlying cryptographic primitives (the signature scheme and the authenticated encryption scheme) are secure.*

Proof. We proceed using a hybrid argument. First, we define a simulator that will simulate the role of the challenger for the adversary $\mathcal{A}$ in the protocol execution environment. Then, we specify a sequence of hybrid experiments where we modify the behavior of the simulator.

Specifically, the simulator $\mathcal{S}$ takes as input the number of parties n, the security parameter λ, and the adversary $\mathcal{A}$, and plays the role of the challenger in the protocol execution experiment with $\mathcal{A}$. At the beginning of the simulation, $\mathcal{A}$ submits a tuple of identities $(\text{ID}_1, \ldots, \text{ID}_n)$ and two test identities $\text{ID}_T^{(0)}$, $\text{ID}_T^{(1)}$. During the protocol execution, the simulator chooses the parameters for each of the parties, and handles the responses to the adversary's queries according to the specifications of the particular hybrid experiment. We now define our sequence of hybrid experiments:

- **Hybrid** H_0**:** This is the real experiment Expt_0 (i.e., the simulator responds to the adversary's queries as described in Expt_0).
- **Hybrid** H_1**:** Same as H_0, except whenever the test party P_T is activated to process a server's response message, if all of the validation checks pass, the simulator uses the identity $\text{ID}_T^{(1)}$ in place of $\text{ID}_T^{(0)}$ when constructing P_T's finish message.
- **Hybrid** H_2**:** This is the real experiment Expt_1.

We now show that each consecutive pair of hybrid experiments is computationally indistinguishable. This suffices to show that Expt_0 and Expt_1 are computationally indistinguishable, and correspondingly, that the protocol in Figure 1 provides privacy.

Claim D.3. *Hybrids* H_0 *and* H_1 *are computationally indistinguishable if the Hash-DH assumption holds in* $\mathbb{G}$ *and the underlying cryptographic primitives are secure.*

Proof. Observe that H_0 and H_1 are identical experiments, except in H_1, the finish messages sent by the test party P_T contain an encryption of the identity $\text{ID}_T^{(1)}$ under the handshake encryption key htk instead of $\text{ID}_T^{(0)}$. At a high level then, the claim follows from the fact that the SIGMA-I protocol ensures confidentiality of the client's finish message against active attackers [CK02, §5.3].

Specifically, Canetti and Krawczyk show that for any complete session (P, s, Q) that is not exposed by the adversary (that is, neither this session nor its matching session (Q, sid) has been corrupted by a $\mathsf{StateReveal}$ or a $\mathsf{Corrupt}$ query), breaking the semantic security of the information encrypted under htk in the finish message of session (P, sid) implies a distinguisher between htk and a random encryption key. This then implies an attack either on the handshake security of the protocol or one of its underlying cryptographic primitives.

In our privacy model, the admissibility requirement stipulates that the test party P_T is never corrupted during the protocol execution. Suppose that in the protocol execution experiment, the adversary activates P_T to respond to a server's message in a session (P_T, sid) and P_T responds with a finish message (a ciphertext under the handshake key htk). Since P_T is honest, if it sends the finish message, then it also completes the session setup by outputting the tuple (P_T, sid, Q). Next, we use the fact that if (P_T, sid) completes with peer Q, by the admissibility requirement, the adversary must not have made a $\mathsf{StateReveal}$ query on (Q, sid) nor was Q corrupt before the completion of (P_T, sid). Thus, we can directly invoke the security properties of the SIGMA-I protocol and argue that the handshake key htk used by P_T to encrypt the finish message in session (P_T, sid) is computationally indistinguishable from a uniformly random key. The claim then follows by semantic security of the authenticated encryption scheme (strictly speaking, we require a hybrid argument over each session where P_T sends a finish message). □

Claim D.4. *Hybrid* H_1 *and* H_2 *are computationally indistinguishable if the IBE scheme used to construct the prefix encryption scheme is* $\mathsf{IND\text{-}ID\text{-}CCA}$*-secure.*

Proof. Let q be an upper bound on the number of sessions where $\mathcal{A}$ activates the test party P_T as the responder (the server) in the protocol execution experiment. We define a sequence of $q + 1$ intermediate hybrids $\mathsf{H}_{1,0}, \ldots, \mathsf{H}_{1,q}$ where hybrid experiment $\mathsf{H}_{1,i}$ is defined as follows:

- **Hybrid** $\mathsf{H}_{1,i}$**:** Same as H_1, except the first i times that P_T is activated as a responder, P_T substitutes the identity $\text{ID}_T^{(1)}$ for $\text{ID}_T^{(0)}$ in its response message. In all subsequent sessions where P_T is activated as a responder, it uses the identity $\text{ID}_T^{(0)}$.

By construction, hybrid experiment H_1 is identical to $\mathsf{H}_{1,0}$ and H_2 is identical to $\mathsf{H}_{1,q}$. We now show that for all $i \in [q]$, hybrid $\mathsf{H}_{1,i-1}$ is computationally indistinguishable from hybrid $\mathsf{H}_{1,i}$, assuming that the IBE scheme is $\mathsf{IND\text{-}ID\text{-}CCA}$-secure.

Claim D.5. *For all* $i \in [q]$*, hybrids* $\mathsf{H}_{1,i-1}$ *and* $\mathsf{H}_{1,i}$ *are computationally indistinguishable assuming that the IBE scheme is* $\mathsf{IND\text{-}ID\text{-}CCA}$*-secure.*

Proof. Suppose $\mathcal{A}$ is an adversary that is able to distinguish $\mathsf{H}_{1,i-1}$ from $\mathsf{H}_{1,i}$. We show how to use $\mathcal{A}$ to build an adversary $\mathcal{B}$ for the $\mathsf{IND\text{-}ID\text{-}CCA}$-security. At the beginning of the $\mathsf{IND\text{-}ID\text{-}CCA}$ game, algorithm $\mathcal{B}$ is given the public parameters MPK for the IBE scheme. It starts running adversary

$\mathcal{A}$ and obtains a tuple of identities $(\text{ID}_1, \ldots, \text{ID}_n)$ and test identities $\text{ID}_T^{(0)}$, $\text{ID}_T^{(1)}$. Algorithm $\mathcal{B}$ then simulates the setup procedure in H_1. In particular, for each $i \in [n]$, it chooses a signing and verification key for each party P_i. In addition, it also issues a certificate binding the identity ID_i to its associated signature verification key. Next, it prepares two certificates, one binding $\text{ID}_T^{(0)}$ to the verification key for ID_T, and another binding $\text{ID}_T^{(1)}$ to the verification key for ID_T. Note that $\mathcal{B}$ does not make any queries to the IBE extraction oracle, and thus, does not associate any IBE identity keys with each party. We will describe how algorithm $\mathcal{B}$ is able to answer the queries consistently in the simulation in spite of this fact. Finally, algorithm $\mathcal{B}$ gives MPK to $\mathcal{A}$ and the protocol execution phase begins. Algorithm $\mathcal{B}$ simulates the response to each of adversary $\mathcal{A}$'s queries as follows:

- **Client initialization queries.** These are handled exactly as in H_1 and H_2.
- **Server response queries.** When adversary $\mathcal{A}$ activates a party P to respond to a client initialization query with some policy π, if $P \neq P_T$, algorithm $\mathcal{B}$ simulates the response message as in H_1. This is possible because $\mathcal{B}$ chooses all of the parameters that appears in the response message and also has the master public key MPK of the IBE scheme. If $P = P_T$, then let ℓ be the number of times $\mathcal{A}$ has already activated P_T to respond to a client initialization message in the protocol execution thus far. Then, algorithm $\mathcal{B}$ proceeds as follows:
  - If $\ell < i - 1$, then $\mathcal{B}$ constructs the response message as described in H_2, that is, using the identity $\text{ID}_T^{(1)}$ in the response message.
  - If $\ell \geq i$, then $\mathcal{B}$ constructs the response message as described in H_1, that is, using the identity $\text{ID}_T^{(0)}$ in the response message.
  - If $\ell = i - 1$, then $\mathcal{B}$ submits the tuple $(\text{ID}_T^{(0)}, \text{ID}_T^{(1)})$ with the policy π as its challenge identity in the IBE security game. It receives a ciphertext $\overline{\text{CT}}_S$ from the challenger. Algorithm $\mathcal{B}$ constructs the rest of P_T's response message as in H_1 and H_2 using $\overline{\text{CT}}_S$ as its encrypted certificate chain.
- **Client response queries.** When the adversary $\mathcal{A}$ delivers a message to a client session (P, sid), $\mathcal{B}$ responds as follows:
  1. Let π_S be the sender's policy associated with the message. If $P \neq P_T$ and ID_P does not satisfy π_S, $\mathcal{B}$ aborts the session. If $P = P_T$ and $\text{ID}_T^{(0)}$ does not satisfy π_S, $\mathcal{B}$ aborts the session. Recall that our admissibility requirement specifies that either $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ both satisfy π_S, or neither satisfy π_S.
  2. $\mathcal{B}$ parses the adversary's message as $(\mathsf{sid}, g^y, \text{CT})$. It checks that there is a local session (P, sid), and aborts the protocol if not. Next, it derives the keys $(\mathsf{htk}, \mathsf{atk}) = \mathsf{KDF}(g^x, g^y, g^{xy})$ where x is the ephemeral DH exponent it chose for session (P, sid). It decrypts CT with htk and parses the result as a tuple (CT_S, σ_S), again aborting the protocol if decryption fails or the resulting message has the wrong form. Finally, $\mathcal{B}$ checks that π_S is the actual policy associated with CT_S, aborting the protocol if not.
  3. If $\mathcal{B}$ is still in the pre-challenge phase or if $\mathcal{B}$ is in the post-challenge phase and either $\text{CT}_S \neq \overline{\text{CT}}_S$ or $\pi_S \neq \bar{\pi}$ (where $\bar{\pi}$ is the identity $\mathcal{B}$ submitted to the IBE challenger and $\overline{\text{CT}}_S$ is the ciphertext $\mathcal{B}$ received from the IBE challenger), then $\mathcal{B}$ queries the IBE decryption oracle on CT_S with identity π_S to obtain a decrypted identity ID_Q. $\mathcal{B}$ performs

the remaining checks as would be done in hybrids H_1 and H_2, and constructs the client's finish message in the same manner.

If $\mathcal{B}$ is in the post-challenge phase, $\text{CT}_S = \overline{\text{CT}}_S$, and $\pi_S = \bar{\pi}$, then $\mathcal{B}$ aborts the session if the identity $\text{ID}_T^{(0)}$ does not satisfies the policy associated with session (P, sid). Recall again that under our admissibility requirement, either $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ both satisfy the policy associated with session (P, sid), or neither satisfy the policy. If $\mathcal{B}$ does not abort, then $\mathcal{B}$ verifies that the signature σ_S is a valid signature on $(\mathsf{sid}, \text{CT}_S, g^x, g^y)$ where g^x, g^y are the session id and ephemeral DH shares associated with session (P, sid). If the signature verifies, then $\mathcal{B}$ constructs P's finish message as in H_1 and H_2.

- **Server finish queries.** These are handled exactly as in H_1 and H_2.
- StateReveal **and** KeyReveal **queries.** These are handled exactly as in H_1 and H_2.
- Corrupt **queries.** If $\mathcal{A}$ asks to corrupt a party $P \neq P_T$ (since $\mathcal{A}$ is admissible), $\mathcal{B}$ queries the IBE extraction oracle for the secret keys for ID_P and each prefix of ID_P. It gives these secret keys to $\mathcal{A}$, the long-term signing key associated with ID_P, and any ephemeral secrets for incomplete sessions currently in the local storage of P.

At the end of the game, $\mathcal{A}$ outputs a guess b'. Algorithm $\mathcal{B}$ echoes this guess.

To complete the proof, we first show that $\mathcal{B}$ is an admissible IBE adversary in the IND-ID-CCA-security game. By construction, $\mathcal{B}$ never requests the challenger to decrypt the challenge ciphertext under the challenge identity. It thus suffices to argue that $\mathcal{B}$ never queries the extraction oracle on the challenge identity $\bar{\pi}$. This follows by admissibility of $\mathcal{A}$. In the above specification, algorithm $\mathcal{B}$ only makes extraction queries when the adversary corrupts a party. By the admissibility requirement, none of the corrupted parties can satisfy any of the policies the adversary associated with P_T, which in particular, includes $\bar{\pi}$. Thus, in the simulation, $\mathcal{B}$ never needs to issue an extraction query for the identity $\bar{\pi}$, and so, $\mathcal{B}$ is admissible.

Next, we show that if $\mathcal{B}$ receives an encryption of $\text{ID}_T^{(0)}$ from the IBE challenger in the challenge phase, then it has perfectly simulated hybrid $\mathsf{H}_{1,i-1}$ for $\mathcal{A}$ and if it receives an encryption of $\text{ID}_T^{(1)}$ from the IBE challenger, it has perfectly simulated $\mathsf{H}_{1,i}$ for $\mathcal{A}$. It is easy to verify that $\mathcal{B}$ correctly simulates the client initialization, server finish, and exposure queries. For the server response queries, $\mathcal{B}$ constructs the encrypted identity as prescribed by $\mathsf{H}_{1,i-1}$ when the IBE challenger replies with an encryption of $\text{ID}_T^{(0)}$. The encrypted identity is constructed as prescribed by $\mathsf{H}_{1,i}$ when the IBE challenger replies with an encryption of $\text{ID}_T^{(1)}$.

Finally, we argue that $\mathcal{B}$ correctly simulates the client response queries. The only non-trivial case is when the adversary submits $\text{CT}_S = \overline{\text{CT}}_S$ and $\pi_S = \bar{\pi}$. In all other cases, the behavior of $\mathcal{B}$ is identical to that in the real scheme (unchanged between $\mathsf{H}_{1,i-1}$ and $\mathsf{H}_{1,i}$). In the case where $\text{CT}_S = \overline{\text{CT}}_S$ and $\pi_S = \bar{\pi}$, then by construction, CT_S is either a valid encryption of the identity $\text{ID}_T^{(0)}$ or of $\text{ID}_T^{(1)}$. By admissibility, the client's policy in session (P, sid) either accepts both $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$, or rejects both. Thus, in the real experiment, the client's decision to abort the session or continue the handshake is *independent* of whether the server's identity was $\text{ID}_T^{(0)}$ or $\text{ID}_T^{(1)}$. The remainder of the response processing is identical to that in the real scheme, so we conclude that $\mathcal{B}$ correctly simulates the client's behavior. The claim follows. □

Since each pair of hybrids $\mathsf{H}_{1,i-1}$ and $\mathsf{H}_{1,i}$ for all $i \in [q]$ are computationally indistinguishable under the IND-ID-CCA-security of the IBE scheme, we conclude that H_1 and H_2 are computationally indistinguishable. □

Theorem D.2 now follows from Claims D.3 and D.4. □

E Analysis of Private Service Discovery Protocol

In this section, we formally argue the security of our private service discovery protocol from Section 5 under Definition B.2. First, we give a more formal specification of the protocol in Figure 4. In the subsequent sections, we demonstrate our discovery protocol is secure in our extended Canetti-Krawczyk key-exchange model (Theorem E.1, Appendix E.1), that it provides 0-RTT security (Theorem E.9, Appendix E.2), and that it provides privacy for both the client and the server (Theorem E.10, Appendix E.3).

E.1 Key-Exchange Security

First, we show that the protocol in Figure 4 is a secure key-exchange protocol in the service-discovery extension to the Canetti-Krawczyk key-exchange model from Appendix B.1. Specifically, we prove the following theorem:

Theorem E.1. *The protocol in Figure 4 is a secure service discovery protocol (Definition B.2) in the random oracle model, assuming the Strong-DH assumption and the Hash-DH assumption hold in $\mathbb{G}$, as well as the security of the underlying cryptographic primitives (the signature scheme, the PRG, the authenticated encryption scheme, and the extraction algorithm).*

In the following, we show that the two properties in Definition B.2 hold for the key-exchange protocol in Figure 4.

Proof of Property 1. Our proof of Property 2 (Appendix E.1.1) below will also show that our service discovery protocol satisfies Property 1. In particular, we refer to hybrid arguments H_0 through H_3 in the proof of Property 2.

E.1.1 Proof of Property 2.

Following similar analysis of key-exchange protocols in [KPW13, KW15], we assume selective security in the adversary's choice of the test session. In our case, we require that at the beginning of the security game, the adversary commits to the following:

- The test session $(P, \mathsf{bid}, \mathsf{sid})$.
- The peer's identity Q in the test session.
- Whether P is the initializer or the responder in the test session.

Note that selective security implies full adaptive security at a security loss that grows polynomially in the number of parties and the number of sessions the adversary initiates (where the simulator guesses the adversary's test session at the beginning of the protocol execution experiment).

Let $\mathbb{G}$ be a group where the Hash-DH and Strong-DH assumptions hold and let g be a generator for $\mathbb{G}$. Let H_1, H_2 be hash functions. Let PRG be a secure pseudorandom generator (PRG) with seed-space $\mathcal{K}$. Let $\mathsf{Extract}$ be a key-extraction algorithm with output space $\mathcal{K}$. The key-extraction algorithm can be instantiated with the HMAC-based key derivation function [KE10] like that used in the OPTLS protocol [KW15].

Server's broadcast message:

$$\mathsf{bid}, \text{ID}_S, g^s, \text{SIG}_S(\mathsf{bid}, \text{ID}_S, g^s)$$

Protocol messages:

Client — Server

$$\xrightarrow{\quad \mathsf{bid}, \mathsf{sid}, g^x, \{\text{ID}_S, \text{ID}_C, \text{SIG}_C(\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_C, g^s, g^x)\}_{\mathsf{htk}} \quad}$$

$$\xleftarrow{\quad \mathsf{bid}, \mathsf{sid}, g^y, \{\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_C, g^s, g^x, g^y\}_{\mathsf{htk}'} \quad}$$

Broadcast description: When a server S is activated to broadcast a discovery message with broadcast id bid, it erases any existing semi-static state (from a previous broadcast). It chooses $s \xleftarrow{R} \mathbb{Z}_p$ and outputs the message $(\mathsf{bid}, \text{ID}_S, g^s, \text{SIG}_S(\mathsf{bid}, \text{ID}_S, g^s))$. The server stores the constant s together with its current broadcast id bid. Without loss of generality, we assume that S never reuses a broadcast id (e.g., the broadcast ids must be in ascending order).

Protocol actions:

1. When a party C is activated to initialize a session $(\mathsf{bid}, \mathsf{sid})$ with broadcast message $(\mathsf{bid}', \text{ID}_{S'}, g^{s'}, \sigma')$, it first checks that $\mathsf{bid} = \mathsf{bid}'$ and that no previous session $(C, \mathsf{bid}, \mathsf{sid})$ has been initialized (aborting the session if the checks fail). If the checks succeed, it looks up the verification key for the party S' identified by $\text{ID}_{S'}$ and checks that σ' is a valid signature on the tuple $(\mathsf{bid}, \text{ID}_{S'}, g^{s'})$ under the public key of $\text{ID}_{S'}$. If any of the checks fail, C aborts the session. Otherwise, C chooses a random exponent $x \xleftarrow{R} \mathbb{Z}_p$, and computes $k = H_1(g^{s'}, g^x, g^{s'x})$, $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k)$. It replies to S' with the message $(\mathsf{bid}, \mathsf{sid}, g^x, \{\text{ID}_{S'}, \text{ID}_C, \text{SIG}_C(\mathsf{bid}, \mathsf{sid}, \text{ID}_{S'}, \text{ID}_C, g^{s'}, g^x)\}_{\mathsf{htk}})$. In addition, it stores its ephemeral exponent x, the server's identity $\text{ID}_{S'}$ and the server's DH share $g^{s'}$ in the local state of session $(C, \mathsf{bid}, \mathsf{sid})$.
2. When a party S is activated as a responder with session id sid and a message of the form $(\mathsf{bid}', \mathsf{sid}', g^{x'}, \text{CT}')$, it looks up its current broadcast id bid and semi-static exponent s. Then, it checks that $\mathsf{bid}' = \mathsf{bid}$, $\mathsf{sid}' = \mathsf{sid}$, and no previous session $(S, \mathsf{bid}, \mathsf{sid})$ has been initialized (aborting the session if the checks fail or there is no broadcast state). It then computes $k = H_1(g^s, g^{x'}, g^{sx'})$, $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k)$, and tries to decrypt CT' with htk. If decryption succeeds (does not output $\perp$), S parses the output as $(\text{ID}_{S'}, \text{ID}_{C'}, \sigma')$. It checks that $\text{ID}_{S'} = \text{ID}_S$, and looks up the verification key for the party C' identified by $\text{ID}_{C'}$. It verifies that σ' is a valid signature on $(\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_{C'}, g^s, g^{x'})$ under the public key of C'. If any of these checks fail, S aborts the protocol. Otherwise, it chooses an ephemeral exponent $y \xleftarrow{R} \mathbb{Z}_p$ and computes $\mathsf{atk} = \mathsf{Extract}(\mathsf{exk}, H_2(g^{x'}, g^y, g^{x'y}))$. It replies to C' with the message $(\mathsf{bid}, \mathsf{sid}, g^y, \{\mathsf{bid}, \mathsf{sid}, \text{ID}_S, \text{ID}_{C'}, g^s, g^x, g^y\}_{\mathsf{htk}'})$. It also outputs the public tuple $(S, \mathsf{bid}, \mathsf{sid}, C')$ and the secret session key atk.
3. Upon receiving a response message of the form $(\mathsf{bid}', \mathsf{sid}', g^{y'}, \text{CT}')$, the party C checks whether there is a session $(C, \mathsf{bid}', \mathsf{sid}')$. If not, C ignores the message. Otherwise, it loads the ephemeral constant x, the server's identity $\text{ID}_{S'}$, and the server's DH share $g^{s'}$ from its local storage, computes $k = H_1(g^{s'}, g^x, g^{s'x})$, $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k)$. It tries to decrypt CT' with htk'. If CT' successfully decrypts to the tuple $(\mathsf{bid}', \mathsf{sid}', \text{ID}_{S'}, \text{ID}_C, g^{s'}, g^x, g^{y'})$, where x is the ephemeral exponent in the local state of session $(C, \mathsf{bid}, \mathsf{sid})$, then C outputs the public tuple $(C, \mathsf{bid}', \mathsf{sid}', S')$ and the secret session key $\mathsf{atk} = \mathsf{Extract}(\mathsf{exk}, H_2(g^x, g^{y'}, g^{xy'}))$. Otherwise, C aborts the protocol.

Figure 4: Formal specification of the private discovery protocol. We present the "non-private" version of the service discovery protocol without the prefix-based encryption. In the private version, the server's broadcast is encrypted under a prefix encryption scheme to its policy.

We begin by defining a simulator $\mathcal{S} = \mathcal{S}(\mathcal{A})$. On input n (the number of parties), λ (a security parameter) and an adversary $\mathcal{A}$, the simulator $\mathcal{S}$ simulates a run of the security game for the mutual authentication protocol. In the selective security setting, the adversary begins by committing to a test session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the peer $\bar{Q}$ in the test session, and whether $\bar{P}$ was the initiator or the responder in the test session. Then, the simulator $\mathcal{S}$ initializes the n parties by choosing a private signing key and a public verification key for each of the n parties. When $\mathcal{A}$ activates a party, the simulator $\mathcal{S}$ performs the protocol actions (as described in Figure 4) on behalf of the parties, and gives $\mathcal{A}$ the outgoing messages as well as the public output of each session.

Description of simulator. We now introduce several variants of the simulator $\mathcal{S}$, which we generically denote $\bar{\mathcal{S}}$. The simulator $\bar{\mathcal{S}}$ behaves very similarly to $\mathcal{S}$, except for the following differences.

1. At the beginning of the simulation, the simulator chooses three exponents $\bar{s}, \bar{x}, \bar{y} \xleftarrow{R} \mathbb{Z}_p$ and four keys $\overline{\mathsf{htk}}, \overline{\mathsf{htk}}', \overline{\mathsf{exk}}, \overline{\mathsf{atk}} \in \mathcal{K}$. The specification of $\overline{\mathsf{htk}}, \overline{\mathsf{htk}}', \overline{\mathsf{exk}}, \overline{\mathsf{atk}}$ will determine the different variants of the simulator $\bar{\mathcal{S}}$.

2. In the selective security model, the adversary commits to a test session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the peer $\bar{Q}$, and the role of $\bar{P}$ in the test session at the beginning of the experiment. For notational convenience, let $\bar{S} \in \{\bar{P}, \bar{Q}\}$ denote the server the adversary commits to for the test session and similarly, let $\bar{C} \in \{\bar{P}, \bar{Q}\}$ denote the client to which it commits.

   The simulator $\bar{\mathcal{S}}$ simulates the execution of the key-exchange security game exactly as $\mathcal{S}$, except for the following differences.

   - If the adversary activates $\bar{S}$ to initiate the broadcast $(\bar{S}, \overline{\mathsf{bid}})$, the simulator uses $\bar{s}$ as the semi-static DH exponent in the broadcast.
   - If the adversary activates $\bar{C}$ to initiate the session $(\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the simulator uses $\bar{x}$ as the ephemeral DH exponent in the start message.
   - If the adversary activates $\bar{S}$ as a responder to the session $(\bar{S}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the simulator uses $\bar{y}$ as the ephemeral DH exponent in the response message.
   - It substitutes the keys $\overline{\mathsf{htk}}, \overline{\mathsf{htk}}', \overline{\mathsf{exk}}$ for the keys $\mathsf{htk}, \mathsf{htk}', \mathsf{exk}$ whenever the DH shares $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$ are used to derive the keys during the simulation (that is, when the simulator needs to compute the quantity $\mathsf{PRG}(H_1(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}}))$). Similarly, it uses $\overline{\mathsf{atk}}$ in place of atk whenever the shares $(g^{\bar{x}}, g^{\bar{s}}, g^{\bar{y}}, g^{\bar{s}\bar{x}}, g^{\bar{x}\bar{y}})$ are used to derive the session key (that is, when the simulator needs to compute the quantity $\mathsf{Extract}(\overline{\mathsf{exk}}, H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}}))$).

3. At the end of the protocol, $\mathcal{A}$ outputs a bit. The simulator $\bar{\mathcal{S}}$ outputs the same bit.

Our security analysis consists of two main cases, depending on whether the adversary compromises the server's semi-static broadcast secret or not. Recall from our admissibility requirement that as long as it is not the case that both the server's ephemeral DH secret and the server's semi-static broadcast secret are compromised, we say the adversary is admissible. This is very similar to the case analysis in [KW15]. More concretely, our two cases are given as follows:

- **Case 1:** The adversary neither issues a $\mathsf{BroadcastReveal}$ query on $(\bar{S}, \overline{\mathsf{bid}})$ nor corrupts $\bar{S}$ before $(\bar{S}, \overline{\mathsf{bid}})$ expires.

- **Case 2:** The adversary does not issue a $\mathsf{StateReveal}$ query on $(\bar{S}, \overline{\mathsf{bid}})$.

For each case, we define a sequence of hybrid experiments, and then show that each consecutive pair of hybrid experiments are computationally indistinguishable. Before we begin the case analysis, we prove the following lemma.

Lemma E.2. *Suppose a session* $(P, \mathsf{bid}, \mathsf{sid})$ *completes with a peer* Q*. Assume moreover that neither* P *nor* Q *has been corrupted before the completion of* $(P, \mathsf{bid}, \mathsf{sid})$*. Then, assuming that* SIG *is a secure signature scheme (and that the certificates are authenticated), the following hold:*

- *If* P *is the client and* Q *is the server, then the adversary initiated a broadcast* (Q, bid) *and* P *must have been activated to initiate a session* $(P, \mathsf{bid}, \mathsf{sid})$ *with the broadcast message output by* (Q, bid)*.*
- *If* P *is the server and* Q *is the client, the session* $(Q, \mathsf{bid}, \mathsf{sid})$ *cannot complete with a peer* P' *where* $P' \neq P$*.*

Proof. We prove each claim separately:

- When P is activated to initialize a session $(P, \mathsf{bid}, \mathsf{sid})$ with a broadcast message $(\mathsf{bid}', \text{ID}_{S'}, g^{s'}, \sigma')$, it first checks that $\mathsf{bid}' = \mathsf{bid}$ and that σ' is a valid signature on $(\mathsf{bid}, \text{ID}_{S'}, g^{s'})$ under the public key identified by $\text{ID}_{S'}$. Since $(P, \mathsf{bid}, \mathsf{sid})$ completes with peer Q, it must be the case that $S' = Q$. Since Q has not been corrupted before the completion of $(P, \mathsf{bid}, \mathsf{sid})$, it signs at most one broadcast message containing bid. Thus, if $(P, \mathsf{bid}, \mathsf{sid})$ completes with Q, it must have been initialized with the broadcast message output by (Q, bid) since this is the only message that contains a valid signature from Q with the broadcast id bid. Otherwise, $\mathcal{A}$ can be used to break the security of the signature scheme SIG.
- Since $(P, \mathsf{bid}, \mathsf{sid})$ completes with peer Q and P is the server, P must have received a message containing a signature from Q on a tuple containing $(\mathsf{bid}, \mathsf{sid}, \text{ID}_P, \text{ID}_Q)$. By assumption, Q has not been corrupted before the completion of $(P, \mathsf{bid}, \mathsf{sid})$, so it would only sign a message of this form if it was activated to initialize a session $(Q, \mathsf{bid}, \mathsf{sid})$. Otherwise, the adversary must have forged a signature under Q's signing key.

  But if Q was activated to initiate the session $(Q, \mathsf{bid}, \mathsf{sid})$ and the session completes with a peer $P' \neq P$, then the only signature that an honest Q would have produced that contains the session identifier $(\mathsf{bid}, \mathsf{sid})$ is a signature on a tuple containing $(\mathsf{bid}, \mathsf{sid}, \text{ID}_{P'}, \text{ID}_Q) \neq (\mathsf{bid}, \mathsf{sid}, \text{ID}_P, \text{ID}_Q)$. In this case, an honest Q would never have signed any tuple containing $(\mathsf{bid}, \mathsf{sid}, \text{ID}_P, \text{ID}_Q)$. Thus, any adversary that can cause $(P, \mathsf{bid}, \mathsf{sid})$ to complete with peer Q and have $(Q, \mathsf{bid}, \mathsf{sid})$ complete with peer P' can be used to forge signatures for SIG (in particular, under Q's signing key). □

We now consider the two possible cases, and argue that in each case, the adversary's advantage in distinguishing the session key of an unexposed session from random is negligible.

Case 1: s **is not compromised.** In this case, we rely on the security of the server's broadcast secret for the privacy of the session key. We now define our sequence of hybrid experiments:

- **Hybrid** H_0**:** This is the real protocol execution experiment. Specifically, the simulator $\bar{\mathcal{S}}$ where $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}}) = \mathsf{PRG}(H_1(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}}))$, and $\overline{\mathsf{atk}} = \mathsf{Extract}(\overline{\mathsf{exk}}, H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}}))$.

- **Hybrid H_1:** Same as hybrid H_0, except $\bar{\mathcal{S}}$ also aborts if $\mathcal{A}$ queries H_1 on the input $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$.
- **Hybrid H_2:** Same as hybrid H_1, except $\overline{\mathsf{htk}}, \overline{\mathsf{htk}}', \overline{\mathsf{exk}}$ are uniformly random over $\mathcal{K}$.
- **Hybrid H_3:** Same as hybrid H_2, except $\bar{S}$ also aborts if the session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ does not match $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. In addition, $\bar{S}$ also aborts if the session key output by the test session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ is not $\overline{\mathsf{atk}}$.
- **Hybrid H_4:** Same as hybrid H_3, except $\overline{\mathsf{atk}}$ is uniformly random over $\mathcal{K}$.

We now show that each consecutive pair of hybrid experiments described above is computationally indistinguishable.

Claim E.3. *Hybrids H_0 and H_1 are computationally indistinguishable if the Strong-DH assumption holds in $\mathbb{G}$ and H_1 is modeled as a random oracle.*

Proof. Let $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ be the session, and $\bar{Q}$ be the peer that the adversary commits to at the beginning of the experiment. By definition, this means that $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \bar{Q})$ is the public output of the test session.

Let $\mathcal{A}$ be a distinguisher between H_0 and H_1. We use $\mathcal{A}$ to build a Strong-DH adversary $\mathcal{B}$ as follows. Algorithm $\mathcal{B}$ is given a Strong-DH challenge $(g, g^{\bar{s}}, g^{\bar{x}})$, as well as a DDH oracle $\mathcal{O}(g^{\bar{s}}, \cdot, \cdot)$ where $\mathcal{O}(g^{\alpha}, g^{\beta}, g^{\gamma}) = 1$ if $\gamma = \alpha\beta \pmod p$, and 0 otherwise. We now describe the operation of $\mathcal{B}$. In the security reduction, $\mathcal{B}$ will need to program the outputs of H_1.

At the beginning of the simulation, algorithm $\mathcal{B}$ generates a private signing key along with a public verification key (in the same manner as $\bar{\mathcal{S}}$) for each of the n parties. Algorithm $\mathcal{B}$ also initializes two empty tables T_1 and T_2. The first table T_1 is used to maintain the mapping from tuples $(g^{\alpha}, g^{\beta}, g^{\gamma})$ to random oracle outputs $H_1(g^{\alpha}, g^{\beta}, g^{\gamma})$, and will be used for answering oracle queries to H_1. The second table T_2 maps between group elements g^{β} and random oracle outputs $H_1(g^{\bar{s}}, g^{\beta}, g^{\gamma})$ where $\gamma = \bar{s}\beta \pmod p$. In other words, T_2 maps between DDH tuples with first component $g^{\bar{s}}$ and is used to answer $\mathsf{KeyReveal}$ queries consistently in the simulation.

Next, $\mathcal{B}$ chooses a random exponent $\bar{y} \xleftarrow{R} \mathbb{Z}_p$ and a random value $k' \xleftarrow{R} \mathcal{K}$. It adds the mapping $[g^{\bar{x}} \mapsto k']$ to T_2, computes $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}}', \overline{\mathsf{exk}}) = \mathsf{PRG}(k')$ and $\overline{\mathsf{atk}} = \mathsf{Extract}(\overline{\mathsf{exk}}, H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}}))$.

Algorithm $\mathcal{B}$ then begins the simulation of the key-exchange security game for $\mathcal{A}$. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$'s actions as follows. Recall that in the selective security model, the adversary commits to the initiator of the protocol at the beginning of the security game. In the following description, we again write $\bar{C} \in \{\bar{P}, \bar{Q}\}$ to denote the client in the test session, and $\bar{S} \in \{\bar{P}, \bar{Q}\}$ to denote the server in the test session.

- **Broadcast queries.** If the adversary activates $\bar{S}$ to initiate the broadcast $(\bar{S}, \overline{\mathsf{bid}})$, the simulator uses $g^{\bar{s}}$ from the Strong-DH challenge as the semi-static DH share in the broadcast message. For other broadcast queries, algorithm $\mathcal{B}$ chooses a fresh DH exponent and constructs the broadcast message exactly as in the real protocol.
- **Client initialization queries.** When $\mathcal{A}$ activates a party P to initiate a session $(P, \mathsf{bid}, \mathsf{sid})$, if $(P, \mathsf{bid}, \mathsf{sid}) \neq (\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, $\mathcal{B}$ chooses a fresh DH exponent and prepares the message exactly as in the real scheme.

  Otherwise, if $(P, \mathsf{bid}, \mathsf{sid}) = (\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, $\mathcal{B}$ sets $g^{\bar{x}}$ from the Strong-DH challenge to be the

DH share in its message. Next, $\mathcal{B}$ sets $k = k'$ and $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k)$, where k' is the key $\mathcal{B}$ chose at the beginning of the simulation. The remaining steps of the query processing are handled exactly as in the real experiment.

- **Server response queries.** When $\mathcal{A}$ activates a server P to respond to a session $(P, \mathsf{bid}, \mathsf{sid})$, if $(P, \mathsf{bid}) \neq (\bar{S}, \overline{\mathsf{bid}})$, then $\mathcal{B}$ chooses a fresh ephemeral DH exponent and constructs the response message exactly as in the real scheme. Note that since it chose the semi-static DH share for (P, bid), it can construct the broadcast message exactly as in the real protocol.

  If $(P, \mathsf{bid}) = (\bar{S}, \overline{\mathsf{bid}})$, then let g^x be the ephemeral DH share in the activation message. Algorithm $\mathcal{B}$ checks if there is already a mapping $g^x \mapsto k$ in table T_2. If the mapping does not exist, then $\mathcal{B}$ chooses a new key $k \xleftarrow{R} \mathcal{K}$ and adds the mapping $g^x \mapsto k$ to T_2. Then, it derives $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k)$ and verifies the activation message (exactly as prescribed in the real protocol). If everything succeeds and $\mathsf{sid} = \overline{\mathsf{sid}}$, algorithm $\mathcal{B}$ uses $g^{\bar{y}}$ (chosen at the beginning of the experiment) as the ephemeral DH share when constructing its response. Otherwise, it chooses a fresh ephemeral share and constructs the response as in the real scheme.

- **Client finish queries.** When a client receives a response message for session $(P, \mathsf{bid}, \mathsf{sid})$, if $(P, \mathsf{bid}, \mathsf{sid}) \neq (\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, $\mathcal{B}$ constructs the outputs as prescribed by the real scheme (this is possible since $\mathcal{B}$ chose the client's ephemeral DH share in this case). If $(P, \mathsf{bid}, \mathsf{sid}) = (\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, $\mathcal{B}$ proceeds as described in the real scheme, except it sets $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k')$, where k' is the key $\mathcal{B}$ chose at the beginning of the simulation.

- **Exposure queries.** Algorithm $\mathcal{B}$ answers all admissible StateReveal, BroadcastReveal, KeyReveal and Corrupt queries as $\bar{\mathcal{S}}$. In the case of an admissible StateReveal, BroadcastReveal, or Corrupt query, the simulator can answer since it chose all of the associated secrets. For KeyReveal queries, we use the fact that $\mathcal{B}$ is already able to process the server response and client finish queries, which includes the computation of the secret session key.

- **Oracle queries to H_1.** Whenever $\mathcal{A}$ queries the random oracle H_1 at $(g^\alpha, g^\beta, g^\gamma)$, $\mathcal{B}$ responds as follows:

  - If there is already a mapping of the form $[(g^\alpha, g^\beta, g^\gamma) \mapsto k]$ in T_1, then $\mathcal{B}$ replies with k.
  - If either $g^\alpha \neq g^{\bar{s}}$, or $g^\alpha = g^{\bar{s}}$ and $\mathcal{O}(g^\beta, g^\gamma) = 0$, then $\mathcal{B}$ chooses $k \xleftarrow{R} \mathcal{K}$, adds the mapping $[(g^\alpha, g^\beta, g^\gamma) \mapsto k]$ to T_1 and replies with k.
  - If $g^\alpha = g^{\bar{s}}$ and $\mathcal{O}(g^\beta, g^\gamma) = 1$, then $\mathcal{B}$ checks whether there is a mapping of the form $[g^\beta \mapsto k]$ in T_2. If so, $\mathcal{B}$ adds the mapping $[(g^\alpha, g^\beta, g^\gamma) \mapsto k]$ to T_1 and replies with k. If no such mapping exists in T_2, then $\mathcal{B}$ samples $k \xleftarrow{R} \mathcal{K}$, and adds the mapping $[(g^\alpha, g^\beta, g^\gamma) \mapsto k]$ to T_1, and the mapping $[g^\beta \mapsto k]$ to T_2.

  Moreover, if $g^\alpha = g^{\bar{s}}$ and $g^\beta = g^{\bar{x}}$ and $\mathcal{O}(g^\beta, g^\gamma) = 1$, $\mathcal{B}$ aborts the simulation and outputs g^γ.

To complete the proof, we show that $\mathcal{B}$ perfectly simulates H_0 for $\mathcal{A}$. First, the exponents $\bar{s}$ and $\bar{x}$ from the Strong-DH challenge are distributed uniformly over $\mathbb{Z}_p$, so they are properly distributed. By construction of the tables T_1 and T_2 and programming the outputs of the random oracle H_1 accordingly, $\mathcal{B}$ ensures that all sessions involving the broadcast $(\bar{S}, \overline{\mathsf{bid}})$ are properly simulated.

Note that in the simulation, k' plays the role of $H_1(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$, so both the sessions $(\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ and $(\bar{S}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ are properly simulated.

Since $\mathcal{B}$ perfectly simulates H_0 for $\mathcal{A}$, with non-negligible probability, $\mathcal{A}$ will query H_1 on $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$. Then $\mathcal{B}$ succeeds in the Strong-DH game with the same advantage. □

Claim E.4. *Hybrids* H_1 *and* H_2 *are computationally indistinguishable if* PRG *is a secure pseudo-random generator.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between H_1 and H_2. We use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ that can break PRG security. Algorithm $\mathcal{B}$ operates as follows. At the beginning of the game, the PRG challenger samples a seed $s \xleftarrow{R} \mathcal{K}$, and gives $\mathcal{B}$ a string $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}})$ where either $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}}) = \mathsf{PRG}(s)$ or $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}}) \xleftarrow{R} \mathcal{K}^3$.

Algorithm $\mathcal{B}$ starts simulating the protocol execution experiment for $\mathcal{A}$. It chooses all of the parameters as in the real scheme, and aborts if any of the abort conditions in H_1 and H_2 are triggered. During the simulation, if $\mathcal{B}$ ever needs to derive keys $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}) = \mathsf{PRG}(k)$ using $k = H_1(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$, it instead uses the keys $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}})$ from the PRG challenger.

If $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}})$ was the output of the PRG, then $\mathcal{B}$ correctly simulates H_1. In particular, algorithm $\mathcal{B}$ has exactly simulated an execution of the real scheme where the value of the random oracle H_1 at $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$ is the PRG seed s. Since s is sampled uniformly at random by the PRG challenger, this value is properly distributed. Moreover, even though the simulator does not know the seed s, it can still correctly answer all of the adversary's queries because it never needs to query H_1 at $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$. If it did, then the experiment aborts. Thus, the adversary's view of the protocol execution is simulated perfectly according to the specification of hybrid H_1.

Conversely, if the keys $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}'}, \overline{\mathsf{exk}})$ were chosen uniformly at random, then $\mathcal{B}$ has correctly simulated H_2. Thus, by security of the PRG, hybrids H_1 and H_2 are computationally indistinguishable. □

Claim E.5. *Hybrids* H_2 *and* H_3 *are computationally indistinguishable if the underlying encryption scheme is an authenticated encryption scheme and that* SIG *is a secure signature scheme (and that the certificates are authenticated).*

Proof. Suppose an adversary $\mathcal{A}$ is able to distinguish between hybrids H_2 and H_3 with non-negligible probability. First, suppose the adversary is able to produce a protocol execution where $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ does not match $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. We consider two possibilities:

- Suppose that $\bar{P}$ is the server in the session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. By definition, the session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ matches $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ if it is incomplete. Thus, it suffices to argue that if $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes, it completes with peer $\bar{P}$. By admissibility, $\bar{Q}$ must not have been corrupted before the completion of $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. The claim then follows by Lemma E.2.

- Suppose that $\bar{P}$ is the client in the session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. As in the previous case, the session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ is matching if it is incomplete.

  Instead, suppose that $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes with a peer $P \neq \bar{P}$. Since neither $\bar{P}$ nor $\bar{Q}$ are corrupt before the completion of their respective session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ and $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the following must have occurred:

– If $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes with peer $\bar{Q}$, $\bar{P}$ must have received an authenticated encryption of a message containing $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{\bar{P}})$ under $\overline{\mathsf{htk}}'$.

– If $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes with peer P, then $\bar{Q}$ must have encrypted a message containing $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{P})$ under some key (possibly $\overline{\mathsf{htk}}'$). Moreover, this is the only message an honest $\bar{Q}$ ever encrypts under *any* key that contains $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$.

We can use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks the ciphertext integrity of the underlying authenticated encryption scheme. At the beginning of the ciphertext integrity game, the challenger samples a random encryption key. This key will play the role of $\overline{\mathsf{htk}}'$ in the reduction. During the simulation, algorithm $\mathcal{B}$ is given access to an encryption oracle. Algorithm $\mathcal{B}$ responds to all queries as described in H_2, choosing all parameters other than $\overline{\mathsf{htk}}'$ for itself. During the simulation, whenever $\mathcal{B}$ needs to encrypt a message under $\overline{\mathsf{htk}}'$, it instead forwards it to the encryption oracle to obtain the ciphertext. When $\mathcal{A}$ delivers a response message containing a ciphertext $\overline{\mathrm{CT}}$ to the session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, $\mathcal{B}$ submits $\overline{\mathrm{CT}}$ to the ciphertext integrity challenger as its forgery. Otherwise, during the simulation, whenever $\mathcal{B}$ needs to decrypt a ciphertext using the key $\overline{\mathsf{htk}}'$, $\mathcal{B}$ checks whether this was one of the ciphertexts it previously submitted to the encryption oracle. If so, it looks up the corresponding plaintext and uses that value to simulate the response. If not, $\mathcal{B}$ substitutes the value $\perp$ for the decryption when simulating the response.

We argue first that $\mathcal{B}$ correctly simulates H_2 for $\mathcal{A}$. Since, $\mathcal{B}$ chooses all quantities other than $\overline{\mathsf{htk}}'$ as described in H_2 and $\overline{\mathsf{htk}}'$ is properly sampled by the ciphertext integrity challenger, it suffices to argue that $\mathcal{B}$ correctly simulates the client computation of the finish message where it needs to decrypt a ciphertext under $\overline{\mathsf{htk}}'$ and verify the integrity of the underlying message. Certainly, if the ciphertext that needs to be verified was supplied to $\mathcal{B}$ by the encryption oracle, $\mathcal{B}$ correctly simulates the decryption operation. If the ciphertext was not one that it received from the encryption oracle, then either it is invalid, in which case decryption in H_2 would also have produced $\perp$, or it is valid, in which case it is a forgery. Thus, assuming the underlying encryption scheme provides ciphertext integrity, $\mathcal{B}$ correctly simulates H_2 (this statement can be formalized by introducing another hybrid experiment).

Since $\mathcal{B}$ correctly simulates the view of H_2 for $\mathcal{A}$ with non-negligible probability, $\mathcal{A}$ is able to activate $\bar{P}$ with a server response message containing a valid encryption $\overline{\mathrm{CT}}$ of a message containing $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{\bar{P}})$ under $\overline{\mathsf{htk}}'$. As argued above, the only ciphertext an uncorrupted $\bar{Q}$ would construct containing $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ is for a tuple with the prefix $(\mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}}) \neq (\mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{\bar{P}}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. Thus, during the simulation $\mathcal{B}$ never needs to query the encryption oracle on the tuple $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{\bar{P}})$, and so it can submit $\overline{\mathrm{CT}}$ as its forgery. We conclude that as long as the underlying encryption scheme is an authenticated encryption scheme, then $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ cannot complete with a peer $P \neq \bar{P}$.

Next, suppose that the adversary is able to produce a protocol execution where the session key output by the test session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ is not $\overline{\mathsf{atk}}$. We again consider two possibilities:

- Suppose that $\bar{P}$ is the server in the session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. Since $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes with peer $\bar{Q}$ and $\bar{P}$ is the server, it must be the case that $\bar{P}$ received a message containing a signature on a tuple $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \mathrm{ID}_{\bar{P}}, \mathrm{ID}_{\bar{Q}}, g^{\bar{s}}, g^{x'})$ under the key identified by $\mathrm{ID}_{\bar{Q}}$ for some $x' \in \mathbb{Z}_p$.

From the above analysis, with overwhelming probability, the session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ matches $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. By the admissibility requirement, this means that $\bar{Q}$ has not been corrupted before the completion of $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. Thus, $\bar{Q}$ would only sign a message containing $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ if it was activated to initiate a session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. By construction, the simulator uses $\bar{x}$ as the ephemeral exponent when initiating this session. Moreover, the only message an honest $\bar{Q}$ signs that contains $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ also contains $g^{\bar{x}}$ as the client's DH share. Thus, it must be the case that $x' = \bar{x}$, since otherwise, the adversary can be used to forge signatures under the public key bound to $\mathrm{ID}_{\bar{Q}}$. Finally, if the remaining validation checks pass and $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes with $\bar{Q}$, then $\bar{P}$ uses $\bar{y}$ as its ephemeral exponent and derives the session key atk using the DH shares $g^{\bar{s}}, g^{\bar{x}}, g^{\bar{y}}$. In this case, $\mathsf{atk} = \overline{\mathsf{atk}}$, as desired.

- Suppose that $\bar{P}$ is the client in the session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. This means that $\bar{P}$ received an ephemeral share $g^{y'}$ and an authenticated encryption of the tuple $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \mathrm{ID}_{\bar{Q}}, \mathrm{ID}_{\bar{P}}, g^{\bar{s}}, g^{\bar{x}}, g^{y'})$ for some $y' \in \mathbb{Z}_p$ under the key $\overline{\mathsf{htk}}'$. If so, then $\bar{P}$ derives the session key atk from $g^{\bar{s}}$, $g^{\bar{x}}$, and $g^{y'}$. Since $\bar{Q}$ has not been corrupted before the completion of $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, it sends at most one encryption of a message containing $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. From the specification of the simulator, we also have that when $\bar{Q}$ is activated as a responder to the session $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, it uses $\bar{y}$ as its ephemeral exponent. Thus, the only ciphertext an uncompromised $\bar{Q}$ constructs that encrypts $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ also contains $g^{\bar{y}}$. By the same argument as above, we appeal to the security of the authenticated encryption scheme (in particular, ciphertext integrity) to argue that $y' = \bar{y}$ with overwhelming probability. Otherwise, the adversary must have been able to create a new ciphertext (under $\overline{\mathsf{htk}}'$) encrypting $(\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ and $g^{y'}$ for $y' \neq \bar{y}$, thereby breaking ciphertext integrity. We conclude that if $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ completes, $\bar{P}$ derives the shared key from $g^{\bar{s}}$, $g^{\bar{x}}$, and $g^{\bar{y}}$, in which case $\mathsf{atk} = \overline{\mathsf{atk}}$. Note that this also shows that if two matching session complete, then they both derive the same session key. □

Claim E.6. *Hybrids* H_3 *and* H_4 *are computationally indistinguishable if the* $\mathsf{Extract}$ *function is a secure pseudorandom function.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between H_3 and H_4. It is straightforward to use $\mathcal{A}$ to build a PRF adversary for $\mathsf{Extract}$. In hybrids H_3 and H_4, the extraction key $\overline{\mathsf{exk}}$ used as the key to $\mathsf{Extract}$ is sampled uniformly at random (and independently of other scheme parameters). Moreover, the responses to the adversary's queries (other than $\mathsf{Corrupt}$ queries to $\bar{P}$ and $\bar{Q}$, which are not admissible), are independent of $\overline{\mathsf{exk}}$ in hybrids H_3 and H_4.

We now use $\mathcal{A}$ to build a PRF adversary $\mathcal{B}$. Algorithm $\mathcal{B}$ makes a single query $H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}})$ to the PRF oracle and sets $\overline{\mathsf{atk}}$ to be the output from the PRF oracle. In the pseudorandom world (where the response is the output of the PRF instantiated with a random key), then $\mathcal{B}$ has correctly simulated H_3 (where $\overline{\mathsf{exk}}$ plays the role of the PRF key). If the output is a uniformly random string, then $\mathcal{B}$ has correctly simulated H_4. Thus, by PRF security of $\mathsf{Extract}$, H_3 and H_4 are computationally indistinguishable. □

To conclude the proof, we argue that in H_4, the view of the adversary is independent of the challenge bit. First, the challenger's response to the Test query is uniformly random over $\mathcal{K}$ regardless of the challenge bit. Finally, it suffices to argue that $\overline{\mathsf{atk}}$ is never used anywhere else in H_4, and thus, is still uniform given the adversary's view of the protocol execution. By construction of the simulator, $\overline{\mathsf{atk}}$ is only used in sessions when the shares $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{y}}, g^{\bar{s}\bar{x}}, g^{\bar{x}\bar{y}})$ are used to derive the session key. By admissibility, the adversary is not allowed to expose session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ or

its matching session, $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. In all other honest sessions on which the adversary could issue a $\mathsf{KeyReveal}$ query, the simulator must have chosen at least one of the DH exponents s, x, or y. Since the adversary can only initiate a polynomial number of sessions and the exponents s, x, y are sampled uniformly from $\mathbb{Z}_p$ where p is super-polynomial in the security parameter λ, we conclude that with overwhelming probability, at least one of $s \neq \bar{s}$, $x \neq \bar{x}$, or $y \neq \bar{y}$. Thus, the simulator's response to all of the adversary's queries (other than the Test query) is independent of $\overline{\mathsf{atk}}$, which proves the claim. Moreover, we note that in each of the reductions, the simulator always picked $\bar{y} \xleftarrow{R} \mathbb{Z}_p$ for itself. Thus, in each case, the simulator is able to respond to a $\mathsf{StateReveal}$ query against the server in the test session. This completes the analysis of Case 1.

Case 2: s is compromised after the handshake. In this case, we rely on the security of the server's ephemeral DH share to ensure confidentiality of the session key. Our analysis is very similar to that of Case 1. We begin by describing our hybrid experiments:

- **Hybrid H_0:** This is the real protocol execution experiment (same as in Case 1).
- **Hybrid H_1:** Same as in Case 1, except the abort condition is only checked before the completion of the test session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$.
- **Hybrid H_2:** Same as in Case 1.
- **Hybrid H_3:** Same as in Case 1.
- **Hybrid H_4:** Same as H_4, except the value of $H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}})$ is replaced by a uniformly random value over $\mathcal{K}$.
- **Hybrid H_5:** Same as H_5, except $\overline{\mathsf{atk}}$ is replaced by a uniformly random value over $\mathcal{K}$.

Hybrids H_0 through H_3 are computationally indistinguishable by applying the same arguments as in Case 1 (Claims E.3 through E.5). Note that the proof of Claim E.3 continues to hold because we assume that $\bar{s}$ is uncompromised before the completion of the test session. This is the only part of the protocol where the claim is checked, so the same argument applies. If suffices then to show that H_3, H_4, and H_5 are computationally indistinguishable.

Claim E.7. *Hybrids H_3 and H_4 are computationally indistinguishable if the Hash-DH assumption holds in $\mathbb{G}$.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that can distinguish between hybrids H_3 and H_4. We use $\mathcal{A}$ to build a distinguisher $\mathcal{B}$ for the Hash-DH assumption. Algorithm $\mathcal{B}$ is given as input a Hash-DH challenge (g^x, g^y, T) and must decide whether $T = H_2(g^x, g^y, g^{xy})$ or if T is uniform over $\mathbb{G}$.

Algorithm $\mathcal{B}$ simulates the protocol execution with the exponents x and y from the Hash-DH challenge playing the roles of $\bar{x}$ and $\bar{y}$ and T playing the role of $H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}})$. Algorithm $\mathcal{B}$ chooses all of the other parameters that appear in the protocol execution as described in the real scheme. Since $\bar{x}$ and $\bar{y}$ are only used in the test session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ and its matching session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the simulator is able to answer all of the adversary's (admissible) queries exactly as in the real scheme. Thus, if $T = H_2(g^x, g^y, g^{xy})$, then $\mathcal{B}$ has simulated H_3 and if T is uniform, then $\mathcal{B}$ has simulated H_4 for $\mathcal{A}$. □

Claim E.8. *Hybrids* H_4 *and* H_5 *are computationally indistinguishable if* Extract *is a strong randomness extractor.*

Proof. In H_4, $\overline{\mathsf{atk}}$ is the output of Extract on a uniformly random string while in H_5, it is a uniformly random string. With overwhelming probability, the ephemeral DH exponents of all (non-corrupt) sessions other than the test session will not be $\bar{x}$ and $\bar{y}$ (since the simulator chooses the ephemeral exponents uniformly at random). Thus the value of $H_2(g^{\bar{x}}, g^{\bar{y}}, g^{\bar{x}\bar{y}})$ is only used once in the simulation and is independent of the adversary's view (by admissibility). The claim then follows from the fact that Extract is a strong randomness extractor. □

We conclude that hybrids H_0 and H_5 are computationally indistinguishable. As in Case 1, the adversary's view of the protocol execution in H_5 no longer depends on the test bit. Thus, the private discovery protocol in Figure 4 is a secure service-discovery protocol (Definition B.2).

E.2 Security of 0-RTT Protocol

One of the main advantages of the service discovery protocol from Figure 4 is its support for 0-RTT mutual authentication. To enable support for application data on the first flow, when the client prepare the initialization message, it derives an additional key using the PRG. More precisely, when generating the initialization query, it computes $k = H_1(g^s, g^x, g^{sx})$ and $(\mathsf{htk}, \mathsf{htk}', \mathsf{exk}, \mathsf{eadk}) = \mathsf{PRG}(k)$. In the first flow of the handshake, it includes any early application data encrypted under eadk. In this section, we show that this 0-RTT protocol is secure in the one-pass security model of [KW15, §4.3]. In [KW15, §5.3], Krawczyk and Wee perform a similar analysis for the 0-RTT mode of the OPTLS protocol in TLS 1.3.

In the one-pass setting, it is not possible to achieve perfect forward secrecy for the lifetime of the server's broadcast (since in a 0-RTT protocol, the server is unable to contribute an ephemeral secret to the session setup). Moreover, the eadk in the 0-RTT protocol is vulnerable to replays of the client's message (again, for the lifetime of the server's broadcast). One way to address the replay problem is to have each client choose their session id at random and have each server maintain a list of session ids that have been used for the lifetime of each broadcast.

One-pass security model. We now specify the one-pass security model more formally in the discovery setting. In this model, when the adversary activates a client to initiate a session $(P, \mathsf{bid}, \mathsf{sid})$ with a broadcast message B, if the client does not abort the protocol then it outputs a public tuple $(P, \mathsf{bid}, \mathsf{sid}, Q)$ and a secret key eadk. We refer to this tuple as the "one-pass session output" to distinguish it from the normal session output. The one-pass session output for the server is defined to be its usual session output. We note that if a client's one-pass session output is $(P, \mathsf{bid}, \mathsf{sid}, Q)$, then if the session $(P, \mathsf{bid}, \mathsf{sid})$ completes, it must complete with the same output $(P, \mathsf{bid}, \mathsf{sid}, Q)$. In the one-pass setting, we only reason about the first message in the protocol, and so we work *only* with the one-pass session outputs.

As usual, the adversary has full control over the network, and can activate parties to initiate and respond to messages. As before, it can perform BroadcastReveal, StateReveal, KeyReveal, and Corrupt queries, subject to the usual admissibility constraints. The difference is that now the KeyReveal query returns eadk instead of the session key. We also modify the adversary's goal to be to distinguish the early application data key of a target session rather than the session key. In the one-pass setting, we impose an additional restriction on the adversary:

- The adversary cannot issue a $\mathsf{BroadcastReveal}$ query on (Q, bid) nor corrupt Q before the expiration of (Q, bid).

This restriction reflects the fact that perfect forward secrecy is not achievable for the lifetime of the server's broadcast. We now prove the following theorem, which proceeds very similarly to the analysis of Case 1 in the proof of Theorem E.1 in Appendix E.1.1.

Theorem E.9. *The protocol in Figure 4 (adapted to the 0-RTT setting) achieves one-pass security for early application data in the random oracle model, assuming the Strong-DH assumption in $\mathbb{G}$, and the security of the underlying cryptographic primitives (the signature scheme, the PRG, the authenticated encryption scheme, and the extraction algorithm).*

Proof. We proceed as in the proof of Theorem E.1 from Appendix E.1.1. In particular, we define an analogous simulator $\bar{S}$ for the one-pass security experiment. We define the following sequence of hybrid experiments:

- **Hybrid H_0:** This is the real experiment where $k = H_1(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$ and $(\overline{\mathsf{htk}}, \overline{\mathsf{htk}}', \overline{\mathsf{exk}}, \overline{\mathsf{eadk}}) = \mathsf{PRG}(k)$.
- **Hybrid H_1:** Same as H_0, except the simulator aborts if $\mathcal{A}$ queries H_1 on the input $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$.
- **Hybrid H_2:** Same as H_1, except $\overline{\mathsf{htk}}$, $\overline{\mathsf{htk}}'$, $\overline{\mathsf{exk}}$, and $\overline{\mathsf{eadk}}$ are all replaced by uniformly random values over $\mathcal{K}$.

Hybrids H_0 and H_1 are computationally indistinguishable by the same argument as in the proof of Claim E.3. Next, hybrids H_1 and H_2 are computationally indistinguishable by PRG security using the same argument as in the proof of Claim E.4.

To complete the proof, we argue that $\overline{\mathsf{eadk}}$ is independent of the adversary's view of the protocol execution (before it makes the Test query). By construction $\overline{\mathsf{eadk}}$ is only used in sessions where the client's ephemeral DH share is $\bar{x}$ and the server's semi-static DH share is $\bar{s}$. Let $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \bar{Q})$ be the test session. As usual, let $\bar{C}, \bar{S} \in \{\bar{P}, \bar{Q}\}$ denote the client and server, respectively in the test session. We consider several cases:

- Consider a session $(P, \mathsf{bid}, \mathsf{sid})$ where P is the client and $(P, \mathsf{bid}, \mathsf{sid}) \neq (\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. Since the simulator chooses the ephemeral DH exponent x uniformly at random in this session, with overwhelming probability $x \neq \bar{x}$. Thus, $\overline{\mathsf{eadk}}$ is independent of all parameters and messages associated with this session.
- Consider a session $(P, \mathsf{bid}, \mathsf{sid})$ where P is the client and $(P, \mathsf{bid}, \mathsf{sid}) = (\bar{C}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. If $P = \bar{P}$, then this is the test session and cannot be exposed. Consider the case where $P = \bar{Q}$. Since $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ matches $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}}, \bar{Q})$, the adversary can only expose $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ if this session completes with a peer $R \neq \bar{P}$. In this case, $\bar{P}$ is the server, so applying Lemma E.2, session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ can only complete with peer $\bar{P}$. Thus $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ will always match the test session and cannot be exposed.
- Consider a session where $(P, \mathsf{bid}, \mathsf{sid})$ where P is the server and $(P, \mathsf{bid}) \neq (\bar{S}, \overline{\mathsf{bid}})$. Since the simulator chooses the semi-static DH exponent s uniformly at random for the broadcast (P, bid), with overwhelming probability $s \neq \bar{s}$. Again, $\overline{\mathsf{eadk}}$ is independent of the session parameters.

- Consider a session $(P, \mathsf{bid}, \mathsf{sid})$ where $(P, \mathsf{bid}) = (\bar{S}, \overline{\mathsf{bid}})$ but $\mathsf{sid} \neq \overline{\mathsf{sid}}$. Suppose the session completes with a peer Q and let g^x be the client's ephemeral DH share in $(P, \mathsf{bid}, \mathsf{sid})$. We argue that with overwhelming probability, $x \neq \bar{x}$. Suppose for sake of contradiction that a session $(P, \mathsf{bid}, \mathsf{sid})$ completes with peer Q and where Q's ephemeral DH share is $g^{\bar{x}}$. This means that P must have received an encryption CT_Q of the message $(\text{ID}_P, \text{ID}_Q, \sigma_Q)$ under the handshake traffic key $\overline{\mathsf{htk}}$ derived from $(g^{\bar{s}}, g^{\bar{x}}, g^{\bar{s}\bar{x}})$. Here, σ_Q is Q's signature on a string containing the tuple $(\mathsf{bid}, \mathsf{sid}) \neq (\overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. Whenever the adversary activates a party to initialize a session other than $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$, the simulator chooses an ephemeral DH share x uniformly at random. With overwhelming probability $x \neq \bar{x}$. Thus, with overwhelming probability, the simulator will only construct a single message encrypted under $\overline{\mathsf{htk}}$, namely the message for session $(\bar{P}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. In particular, CT_Q is not a ciphertext constructed by the simulator. Since $\overline{\mathsf{htk}}$ is uniformly random (and unknown to the adversary), and the encryption scheme provides ciphertext integrity, this happens with negligible probability. Thus, $x \neq \bar{x}$ with overwhelming probability, and $\overline{\mathsf{eadk}}$ is independent of the session parameters.

- Consider a session $(P, \mathsf{bid}, \mathsf{sid})$ where P is the server and $(P, \mathsf{bid}, \mathsf{sid}) = (\bar{S}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. The case $P = \bar{P}$ corresponds to the test session and cannot be exposed. Suppose $P = \bar{Q}$. Then, the session $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$ matches the test session unless it completes with a peer $R \neq \bar{P}$. Suppose this happens, and let g^x be the client's ephemeral DH share in $(\bar{Q}, \overline{\mathsf{bid}}, \overline{\mathsf{sid}})$. By the same argument as in the previous case, we can argue that $x \neq \bar{x}$ with overwhelming probability assuming that the underlying encryption scheme provides ciphertext integrity.

We have shown that the simulator's response to all admissible queries are independent of $\overline{\mathsf{eadk}}$ in hybrid H_3. We conclude that the service discovery protocol in Figure 4 (adapted to the 0-RTT setting) achieves one-pass security. □

E.3 Privacy of Service Discovery Protocol

In this section, we show that the service discovery protocol in Figure 4 is private if the server encrypts its broadcast using a prefix encryption scheme (as shown in Figure 3). As was the case with our analysis of the private mutual authentication scheme, we assume the prefix encryption scheme is constructed from an IBE scheme as described in Section 3.1. In our analysis, we use the same general privacy model from Appendix C, but adapted to the service discovery setting (Appendix B.1). Because our protocol supports 0-RTT mutual authentication and client identification is provided on the *first* round in the protocol, we can only guarantee a meaningful notion of privacy if we restrict ourselves to the one-pass security setting introduced in Appendix E.2. Thus, in the following description, we match sessions based on their one-pass session outputs rather than their session outputs. We say a session completes (in the one-pass sense) if it produces a one-pass session output.

In particular, in the 0-RTT service discovery model, privacy is achievable as long as the adversary does not expose any session (in the one-pass sense) that involves the test party. Otherwise, the adversary is able to trivially learn the identity of the test party. We now state our admissibility requirements more precisely:

- The adversary does not corrupt ID_T or make a $\mathsf{StateReveal}$ query on any session $(P_T, \mathsf{bid}, \mathsf{sid})$ that completes (in the one-pass sense).

- Whenever a session $(P_T, \mathsf{bid}, \mathsf{sid})$ completes (in the one-pass sense) with public output Q, then

the adversary has not made a $\mathsf{StateReveal}$ query on the session $(Q, \mathsf{bid}, \mathsf{sid})$, and moreover, Q is not corrupt before the completion of $(P_T, \mathsf{bid}, \mathsf{sid})$.

- Whenever a session $(P, \mathsf{bid}, \mathsf{sid})$ completes (in the one-pass sense) with public output P_T, then the adversary has not made a $\mathsf{StateReveal}$ query on $(P, \mathsf{bid}, \mathsf{sid})$.
- Let Π_T denote the set of policies the adversary has associated with the test party P_T. Let $I \subseteq [n]$ be the indices of the parties the adversary has either corrupted or on which it has issued a $\mathsf{BroadcastReveal}$ query in the course of the protocol execution. Then, for all policies $\pi \in \Pi_T$ and indices $i \in I$, it should be the case that ID_i does not satisfy π.
- Whenever the adversary associated a policy π with a broadcast (P, bid) or a session $(P, \mathsf{bid}, \mathsf{sid})$, it must be the case that either $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ both satisfy π or neither satisfy π.

We show the following theorem:

Theorem E.10. *The protocol in Figure 4 (where the broadcasts are encrypted under the server's policy using a prefix encryption scheme) is private in the random oracle model assuming the IBE scheme used to construct the prefix encryption scheme is* $\mathsf{IND\text{-}ID\text{-}CCA}$*-secure, the Strong-DH and Hash-DH assumptions hold in* $\mathbb{G}$*, and the underlying cryptographic primitives (the signature scheme, the PRG, the authenticated encryption scheme, and the extraction algorithm) are secure.*

Proof. The proof is similar to that of Theorem D.2. As in the proof of Theorem D.2, we first define a simulator that simulates the role of the challenger for the adversary $\mathcal{A}$ in the protocol execution environment. We then define a series of hybrid experiments.

Specifically, the simulator $\mathcal{S}$ takes as input the number of parties n, the security parameter λ, and the adversary $\mathcal{A}$, and plays the role of the challenger in the protocol execution experiment with $\mathcal{A}$. At the beginning of the simulation, $\mathcal{A}$ submits a tuple of distinct identities $(\text{ID}_1, \ldots, \text{ID}_n)$ and two test identities $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$. During the protocol execution, the simulator chooses the parameters for each party and responds to the adversary's queries according to the specification of the hybrid experiment. We now define our sequence of hybrid experiments:

- **Hybrid** H_0**:** This is the real experiment Expt_0.
- **Hybrid** H_1**:** Same as H_0, except $\text{ID}_T^{(1)}$ is used in place of $\text{ID}_T^{(0)}$ when simulating handshake messages for sessions where P_T is the client.
- **Hybrid** H_2**:** Same as H_1, except $\text{ID}_T^{(1)}$ is used in place of $\text{ID}_T^{(0)}$ when simulating handshake messages for sessions where P_T is the server.
- **Hybrid** H_2**:** This is the real experiment Expt_1.

We now show that each consecutive pair of hybrid experiments is computationally indistinguishable.

Claim E.11. *Hybrids* H_0 *and* H_1 *are computationally indistinguishable in the random oracle model assuming the Strong-DH and Hash-DH assumptions hold in* $\mathbb{G}$ *and the security of the underlying cryptographic primitives.*

Proof. In the service discovery protocol, the client in the protocol execution always encrypts both its and the server's identity under the handshake traffic key htk. Similarly, in the server's response message, the server replies with an encryption of both the client's and server's identity under the handshake traffic key htk'. Similar to the proof of Claim D.3, it suffices to argue that the adversary's view of the handshake secret keys htk and htk' in all sessions where P_T is the client is computationally indistinguishable from uniform. The claim then follows by semantic security of the underlying encryption scheme.

Formally we use the following hybrid argument. Let q be a bound on the number of times the adversary $\mathcal{A}$ activates the test party P_T to initiate a session $(P_T, \mathsf{bid}, \mathsf{sid})$. We define a sequence of $q+1$ hybrid experiments $\mathsf{H}_{0,0}, \ldots, \mathsf{H}_{0,q}$ where in hybrid experiment $\mathsf{H}_{0,i}$, the simulator uses $\textsc{id}_T^{(0)}$ as the identity in P_T's message for the first $q-i$ sessions and $\textsc{id}_T^{(1)}$ as the identity in the last i sessions. By construction, $\mathsf{H}_0 \equiv \mathsf{H}_{0,0}$ and $\mathsf{H}_1 \equiv \mathsf{H}_{0,q}$. It suffices to argue that for each $i \in [q]$, hybrids $\mathsf{H}_{0,i-1}$ and $\mathsf{H}_{0,i}$ are computationally indistinguishable.

We now argue that $\mathsf{H}_{0,i-1}$ and $\mathsf{H}_{0,i}$ are computationally indistinguishable. It suffices to argue that the key htk the test party P_T uses to encrypt its identity in the $(q-i+1)^{\text{th}}$ session is uncompromised. Let $(P_T, \mathsf{sid}, \mathsf{bid})$ be the $(q-i+1)^{\text{th}}$ session initiated at P_T. Let htk and htk' be the handshake keys P_T derives in this session to encrypt its identity. We first argue that htk is uncompromised. An identical argument applies to htk'. By construction, if P_T sent an initialization message in session $(P_T, \mathsf{sid}, \mathsf{bid})$, it must have completed its session (in the one-pass sense) with a peer Q. The overall argument now follows identically to that used to argue that the early application data key eadk in the client's message is uncompromised in the proof of Theorem E.9 (since htk and htk' play an analogous role as eadk in that proof). To apply the argument from the proof of Theorem E.9, we let $(P_T, \mathsf{sid}, \mathsf{bid})$ be the test session, and verify that all the legal queries in the privacy setting are also admissible in the one-pass security setting.

In the private discovery setting, the adversary cannot corrupt or issue a $\mathsf{StateReveal}$ query on a session $(P_T, \mathsf{bid}, \mathsf{sid})$. This is because in the one-pass setting, if the adversary activates P_T to initiate a session $(P_T, \mathsf{bid}, \mathsf{sid})$, and P_T does not abort when processing the initialization query, then the session must have completed in the one-pass sense.

It suffices to check that the adversary does not corrupt Q or issue a $\mathsf{BroadcastReveal}$ query on the broadcast (Q, bid) where Q is the peer of P_T in the target session. Since $(P_T, \mathsf{bid}, \mathsf{sid})$ completes, by the admissibility condition, Q must not have been corrupt at the time the session completed. Thus, we can invoke Lemma E.2 to conclude that P_T was activated to initiate a session with the broadcast message output by (Q, bid). Note that the proof of Lemma E.2 only depends on the broadcast message and the first message in the mutual authentication handshake, so the statement applies even in our one-pass setting.

By admissibility in the private discovery setting, if the adversary issued a $\mathsf{BroadcastReveal}$ query on (Q, bid), then the policy π_Q must not satisfy P_T. But in this case, an honest P_T would have aborted the session $(P_T, \mathsf{bid}, \mathsf{sid})$. We conclude that an admissible adversary could not have made a $\mathsf{BroadcastReveal}$ on the peer's broadcast (Q, bid). In this case, the target session $(P_T, \mathsf{bid}, \mathsf{sid})$ is admissible in the one-pass security setting, and so by the same argument as in the one-pass security proof (Theorem E.9), we conclude that the handshake security keys htk and htk' are uniform and completely hidden to the adversary. The claim then follows by semantic security of the underlying authenticated encryption scheme. □

Claim E.12. *Hybrids* H_1 *and* H_2 *are computationally indistinguishable in the random oracle model*

assuming Strong-DH and Hash-DH assumptions hold in $\mathbb{G}$ and the security of the underlying cryptographic primitives.

Proof. The proof of this statement follows analogously to that of Claim E.11. We show that in all sessions where P_T is the server, the handshake encryption keys htk and htk' for that session have not been compromised.

In the one-pass model, a client's initialization message for a session $(P, \mathsf{bid}, \mathsf{sid})$ will contain the identity of the test party ID_T if and only if $(P, \mathsf{bid}, \mathsf{sid})$ completes with one-pass session output $(P, \mathsf{bid}, \mathsf{sid}, P_T)$. It suffices to show that $(P, \mathsf{bid}, \mathsf{sid})$ is not exposed, in which case the handshake encryption keys htk and htk' are uniform and hidden from the adversary (by a similar argument as that used in the proof of Theorem E.9). The claim then follows by a hybrid argument similar to the one used in the proof of Claim E.11.

Consider a session $(P, \mathsf{bid}, \mathsf{sid})$ that completes with peer P_T. Since the session completes, P could not have been corrupt. In the discovery model, P_T also cannot be corrupted. Next, by admissibility, the adversary cannot issue a $\mathsf{StateReveal}$ query on $(P, \mathsf{bid}, \mathsf{sid})$. By one-pass security, the handshake encryption keys for this session are uncompromised. The claim follows by semantic security of the underlying authenticated encryption scheme. □

Claim E.13. *Hybrids H_2 and H_3 are computationally indistinguishable if the underlying IBE scheme is* IND-ID-CCA*-secure.*

Proof. This proof proceeds very similarly to that of Claim D.4. In particular, we let q be an upper bound on the number of sessions where $\mathcal{A}$ activates the test party P_T to initiate a broadcast. We define a sequence of $q+1$ hybrid experiments $\mathsf{H}_{2,0}, \ldots, \mathsf{H}_{2,q}$ where hybrid experiment $\mathsf{H}_{2,i}$ is defined as follows:

- Same as H_2 except the first i times P_T is activated to initialize a broadcast, P_T substitutes the identity $\text{ID}_T^{(1)}$ for $\text{ID}_T^{(0)}$ in its broadcast. In all subsequent times P_T is activated to initialize a broadcast, it uses the identity $\text{ID}_T^{(0)}$.

We now show that for all $i \in [q]$, hybrids $\mathsf{H}_{2,i-1}$ and $\mathsf{H}_{2,i}$ are computationally indistinguishable assuming that the IBE scheme is IND-ID-CCA-secure. Suppose $\mathcal{A}$ is able to distinguish $\mathsf{H}_{2,i-1}$ from $\mathsf{H}_{2,i}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the IND-ID-CCA-security game. First, $\mathcal{B}$ is given the public parameters MPK for the IBE scheme. Then, $\mathcal{B}$ begins running $\mathcal{A}$ and obtains a tuple of identities $(\text{ID}_1, \ldots, \text{ID}_n)$ and test identities $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$. Algorithm $\mathcal{B}$ simulates the setup procedure in H_2 by choosing signing and verification keys for each party P_i and P_T. It also issues certificates binding ID_i to the verification key for each P_i. It prepares two certificates binding P_T to identities $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$. Finally, $\mathcal{B}$ gives MPK to $\mathcal{A}$ and begins simulating the protocol execution experiment for $\mathcal{A}$:

- **Server broadcast queries.** When adversary $\mathcal{A}$ activates a party P to initiate a broadcast (P, bid), if $P \neq P_T$, algorithm $\mathcal{B}$ simulates the response as in the real scheme. If $P = P_T$, then let ℓ be the number of times $\mathcal{A}$ has activated P_T to initiate a broadcast. Let π be the policy specified by $\mathcal{A}$. Algorithm $\mathcal{B}$ then responds as follows:
  - If $\ell < i - 1$, $\mathcal{B}$ constructs the broadcast as in H_3, that is using $\text{ID}_T^{(1)}$.
  - If $\ell \geq i$, $\mathcal{B}$ constructs the broadcast as in H_2, that is, using $\text{ID}_T^{(0)}$.

– If $\ell = i - 1$, $\mathcal{B}$ chooses a random DH share $s \xleftarrow{R} \mathbb{Z}_p$, and computes signatures $\sigma_0 = \text{SIG}_{P_T}(\mathsf{bid}, \text{ID}_T^{(0)}, g^s)$ and $\sigma_1 = \text{SIG}_{P_T}(\mathsf{bid}, \text{ID}_T^{(1)}, g^s)$. It submits the tuples $(\text{ID}_T^{(0)}, g^s, \sigma_0)$ and $(\text{ID}_T^{(1)}, g^s, \sigma_1)$ to the IBE challenger with π as its challenge identity. It receives a ciphertext $\overline{\text{CT}}_S$ from the challenger. Algorithm $\mathcal{B}$ outputs the broadcast $(\mathsf{bid}, \overline{\text{CT}}_S)$.

- **Client initialization queries.** When an adversary activates a client P to initiate a session $(P, \mathsf{bid}, \mathsf{sid})$ with a broadcast $(\mathsf{bid}, \text{CT}_S)$ and server policy π_S, $\mathcal{B}$ does the following:

  1. If there is already a session $(P, \mathsf{bid}, \mathsf{sid})$, $\mathcal{B}$ aborts the session. If $P \neq P_T$ and ID_P does not satisfy π_S, $\mathcal{B}$ aborts the session. If $P = P_T$ and $\text{ID}_T^{(0)}$ does not satisfy π_S, $\mathcal{B}$ also aborts the session. Recall that our admissibility requirement states that either both $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ satisfy π_S or neither satisfy π_S.
  2. If $\mathcal{B}$ is still in the pre-challenge phase, or if $\mathcal{B}$ is in the post-challenge phase and either $\text{CT}_S \neq \overline{\text{CT}}_S$ or $\pi_S \neq \bar{\pi}$ (where $\bar{\pi}$ is the identity $\mathcal{B}$ submitted to the IBE challenger in the challenge phase), then $\mathcal{B}$ queries the IBE decryption oracle on CT_S and identity π_S to obtain a decrypted broadcast message (or $\perp$). Algorithm $\mathcal{B}$ performs the checks on the decrypted broadcast message and simulates the response as described in H_2.

     If $\mathcal{B}$ is in the post-challenge phase and $\text{CT}_S = \overline{\text{CT}}_S$ and $\pi_S = \bar{\pi}$, then $\mathcal{B}$ aborts the session if $\text{ID}_T^{(0)}$ does not satisfy the the client's policy (specified by the adversary). Again, by admissibility, either both $\text{ID}_T^{(0)}$ and $\text{ID}_T^{(1)}$ satisfy the client's policy or neither do. If $\mathcal{B}$ does not abort the session, then $\mathcal{B}$ simulates the client's response message as described in H_2 (always using the identity $\text{ID}_T^{(1)}$ for ID_T).

- **Server response queries.** These are handled exactly as in H_2 and H_3. They are independent of the IBE parameters.

- **Client finish queries.** These are handled exactly as in H_2 and H_3.

- StateReveal **and** KeyReveal **queries.** These are handled exactly as in H_2 and H_3.

- Corrupt **queries.** If $\mathcal{A}$ asks to corrupt a party $P \neq P_T$ (since $\mathcal{A}$ is admissible), $\mathcal{B}$ queries the IBE extraction oracle for the secret keys for ID_P and each prefix of ID_P. It gives these secret keys to $\mathcal{A}$, the long-term signing key associated with ID_P, and any ephemeral secrets for incomplete sessions currently in the local storage of P.

At the end of the game, adversary $\mathcal{A}$ outputs a guess for whether it is in H_2 or H_3. Algorithm $\mathcal{B}$ echoes this guess.

To complete the proof, we show that $\mathcal{B}$ is an admissible IBE adversary in the IND-ID-CCA-security game. By construction, $\mathcal{B}$ never asks the adversary to decrypt the challenge ciphertext. Similar to the proof of Claim D.4 we appeal to the admissibility of $\mathcal{A}$ to argue that algorithm $\mathcal{B}$ never needs to query the extraction oracle for the identity $\bar{\pi}$ in the challenge query during the simulation.

By construction, if $\mathcal{B}$ receives an encryption of $\text{ID}_T^{(0)}$ from the IBE challenger, then it has correctly simulated the broadcast queries according to the specification of hybrid $\mathsf{H}_{2,i-1}$ for $\mathcal{A}$. If it receives an encryption of $\text{ID}_T^{(1)}$ from the IBE challenger, then it has correctly simulated the broadcast queries according to the specification of hybrid $\mathsf{H}_{2,i}$ for $\mathcal{A}$.

To conclude the proof, we check that the client initialization queries are correctly simulated. The only non-trivial case is when the adversary submits $\mathrm{CT}_S = \overline{\mathrm{CT}}_S$ and $\pi_S = \bar{\pi}_S$. All other cases are processed exactly as in H_2 and H_3. In the case where $\mathrm{CT}_S = \overline{\mathrm{CT}}_S$ and $\pi_S = \bar{\pi}_S$, the ciphertext is either a valid encryption of a broadcast from P_T with identity $\mathrm{ID}_T^{(0)}$ or with identity $\mathrm{ID}_T^{(1)}$. By admissibility, the client will either accept both $\mathrm{ID}_T^{(0)}$ and $\mathrm{ID}_T^{(1)}$ or neither. Thus, in the real scheme, the client's decision to abort the session is *independent* of whether the server's identity is $\mathrm{ID}_T^{(0)}$ or $\mathrm{ID}_T^{(1)}$. The response generation step in H_2 and H_3 depends only on $\mathrm{ID}_T^{(1)}$, and is in particular, independent of the world bit for the IBE security game. Thus, correctness of the simulation follows. Thus, if the IBE scheme is $\mathsf{IND\text{-}ID\text{-}CCA}$ secure, then H_2 and H_3 are computationally indistinguishable. □

Combining Claims E.11 through E.13, we conclude that the service discovery protocol in Figure 4 (where the broadcasts are encrypted under the server's policy using the prefix encryption scheme) is a private service discovery protocol. □