Privacy, Discovery, and Authentication for the Internet of Things*

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) [24, 25], Apple Bonjour [3], Bluetooth Low Energy (BLE) [5], and Universal Plug-N-Play (UPnP) [6]. These mechanisms require only a broadcast communication channel between the devices (unlike older discovery protocols [4, 27, 57] that need a directory service). Moreover, these protocols adhere to the zero configuration networking charter (*Zeroconf*) [2] and can operate with minimal user intervention.

*The full version of this paper with complete proofs is available at `http://arxiv.org/abs/1604.06959`.

Privacy is an important feature often missing in zero-configuration service discovery protocols (e.g., Zeroconf) [37, 38, 40, 49]. 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 [40] 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 [12, 46, 36, 11, 35, 31].

Private mutual authentication. A closely related privacy problem arises during authentication between mutually suspicious entities. Most existing mutual authentication protocols (SIGMA [23, 41], JFK [10], and TLS [28]) 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.[3]

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

[3]While protocols like SIGMA-I [41, 23] and TLS 1.3 [43, 50] can ensure privacy against passive adversaries, they do *not* provide privacy against active attackers.

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.

1.1 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).

Protocol construction. Our protocols are designed for distributed public-key infrastructures, such as the Simple Distributed Security Infrastructure (SDSI) [51]. 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 [50, 43].

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 users 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 [44], which can be constructed using off-the-shelf identity-based encryption (IBE) schemes [20, 19].

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 the full version [54], we provide a rigorous proof of the security and privacy of both protocols in the Canetti-Krawczyk key-exchange model [22, 23, 41].

Implementation and evaluation. We implemented and deployed our protocols in the *Vanadium* open-source distributed application framework [1]. 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. 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 recent versions of OS X or iOS. 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. We describe the protocol in the full version of this paper.

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 (via TLS 1.2 with client authentication[4]) 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. 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.

2.2 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. At a high level, our primary privacy objectives, which should hold in the presence of both passive and active network attackers, are as follows:

- **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.

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

– **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.
– **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.

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 [51]. 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. 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. These prefix-based policies can be used to support group-based access control policies by viewing "subdomains" (e.g., `Alice/Family`) as groups.

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 $x \xleftarrow{\text{R}} S$ to denote that x is drawn uniformly at random from S.

Identity-based encryption and prefix encryption. Identity-based encryption (IBE) [53, 20, 26, 19] is a generalization of public-key encryption where public keys can be arbitrary strings, or *identities*. We give more details in the full version [54]. Prefix encryption [44] is a generalization of IBE where the secret key sk_{ID} for an identity $\textsc{id}$ can decrypt all ciphertexts encrypted to any identity $\textsc{id}'$ that is a prefix of $\textsc{id}$ (in IBE, decryption succeeds only if $\textsc{id} = \textsc{id}'$). 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 [44]. The key for an identity $\textsc{id} = s_1/s_2/\cdots/s_n$ 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 $\textsc{id}'$ is just IBE encryption to the identity $\textsc{id}'$. Given a secret key $\textsc{sk}_{\textsc{id}}$ for $\textsc{id}$, if $\textsc{id}'$ is a prefix of $\textsc{id}$, then $\textsc{sk}_{\textsc{id}}$ contains an IBE identity key for $\textsc{id}'$.

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 $\mathsf{PE.Enc}(\textsc{mpk}, \pi, m)$ to denote an encryption algorithm that takes as input the public key $\textsc{mpk}$, a message m, a prefix-constrained policy π, and outputs a ciphertext $\textsc{ct}$. When there is no ambiguity, we will treat $\textsc{mpk}$ as an implicit parameter to $\mathsf{PE.Enc}$. We write $\mathsf{PE.Dec}(\textsc{sk}_{\textsc{id}}, \textsc{ct})$ for the decryption algorithm that takes in a ciphertext $\textsc{ct}$ and a secret key $\textsc{sk}_{\textsc{id}}$ (for an identity $\textsc{id}$) and outputs a message if $\textsc{id}$ matches the ciphertext policy π, and a special symbol $\perp$ otherwise.

Other cryptographic primitives. We write $\{m\}_k$ to denote an authenticated encryption [13, 52, 15] of a message m under a key k, and $\mathsf{KDF}(\cdot)$ to denote a key-derivation function [29, 42]. We describe these additional primitives as well as the cryptographic assumptions (Hash Diffie-Hellman and Strong Diffie-Hellman [9]) we use in our security analysis in the full version.

Key-exchange model. We analyze the security of our private mutual authentication and privacy service discovery protocols in the Canetti-Krawczyk [22, 23, 41] 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 the full version.

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 key-pair 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 $\textsc{msk}$), which may coincide with certificate authorities. 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 $\textsc{id}_P$ to denote a certificate chain binding P's public key to one of its identities. For a message m, we write $\textsc{sig}_P(m)$ 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 [41, 23] 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 [9] 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 $\textsc{id}_S$ using the prefix encryption scheme under its policy π_S to obtain an encrypted identity $\textsc{ct}_S \leftarrow \mathsf{PE.Enc}(\pi_S, \textsc{id}_S)$.
   (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 $\sigma = \textsc{sig}_S(\mathsf{sid}, \textsc{ct}_S, g^x, g^y)$ on its encrypted identity and the ephemeral session state, and encrypt $(\textsc{ct}_S, \sigma)$ using htk to obtain a ciphertext 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 $(\textsc{ct}_S, \sigma_S)$ and checks whether its identity $\textsc{id}_C$ satisfies the server's policy π_S (revealed by $\textsc{ct}_S$). If the client satisfies the server's policy, it decrypts $\textsc{ct}_S$ using its identity key $\textsc{sk}_C$ to obtain the server's identity $\textsc{id}_S$. If $\textsc{id}_S$ satisfies the client's policy π_C and σ_S is a valid signature on $(\mathsf{sid}, \textsc{ct}_S, g^x, g^y)$ under the public key identified by $\textsc{id}_S$, the client replies to the server with the session id sid and an encryption c' of $(\textsc{id}_C, \textsc{sig}_C(\mathsf{sid}, \textsc{id}_C, g^x, g^y))$ 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 $(\textsc{id}_C, \sigma_C)$ and verifies that $\textsc{id}_C$ satisfies its policy and that σ_C is a valid signature on $(\mathsf{sid}, \textsc{id}_C, g^x, g^y)$ under the public key identified by $\textsc{id}_C$. If so, the handshake completes with atk as the shared session key and where the client believes it is talking to $\textsc{id}_S$ and the server believes it is talking to $\textsc{id}_C$. Otherwise, the server aborts.

4.1 Protocol Analysis

In this section, we highlight some properties of our private mutual authentication protocol. In the full version [54], we also discuss policy privacy, unlinkability, and caching the encrypted certificate chains.

$$\begin{tikzcd}[column sep=huge]
C(\textsc{id}_C, \pi_C) \arrow[r, "\mathsf{sid},\ g^x"] & S(\textsc{id}_S, \pi_S) \\
{} & {} \arrow[l, "{\mathsf{sid},\ g^y,\ \{\overbrace{\mathsf{PE.Enc}(\pi_S,\ \textsc{id}_S)}^{\textsc{ct}_S},\ \textsc{sig}_S(\mathsf{sid}, \textsc{ct}_S,\ g^x,\ g^y)\}_{\mathsf{htk}}}"'] \\
{} \arrow[r, "{\mathsf{sid},\ \{\textsc{id}_C,\ \textsc{sig}_C(\mathsf{sid}, \textsc{id}_C,\ g^x,\ g^y)\}_{\mathsf{htk}}}"] & {}
\end{tikzcd}$$

Fig. 1. Message flow between the client C (with certificate $\textsc{id}_C$ and policy π_C) and the server S (with certificate $\textsc{id}_S$ 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 $\textsc{id}_C$ and $\textsc{id}_S$, respectively. For a message m, $\textsc{sig}_C(m)$ and $\textsc{sig}_S(m)$ 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 $\textsc{sk}_C$ for the prefix-based encryption scheme for the identity associated with its certificate $\textsc{id}_C$.

Comparison with SIGMA-I. Our authentication protocol is very similar to the SIGMA-I key-exchange protocol [41, §5.2], but with the following key difference: the server's certificate, $\textsc{id}_S$, 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 [23, §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 and show that the protocol in Figure 1 achieves this notion in the full version. In contrast, the SIGMA-I protocol does not provide such a guarantee as the identity of the server is revealed to active adversaries.

Security theorem. We state the security theorem for our private mutual authentication protocol here, but defer the formal proof to the full version [54].

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 [50, 43], 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 [9] 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 2.

Service broadcast message. To setup a new broadcast with broadcast id bid, the server S chooses a fresh DH exponent $s \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$, 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. Next, the client chooses an ephemeral DH exponent $x \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$. 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 $y \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$ and derives the session key $\mathsf{atk} = \mathsf{KDF}(g^s, g^x, g^{sx}, g^y, g^{xy})$.[5] The server encrypts the tuple $(\mathsf{bid}, \mathsf{sid}, \text{ID}_1, \text{ID}_2)$ un-

[5]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

der 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 $(\mathsf{bid}, \mathsf{sid}, \textsc{id}_S, \textsc{id}_C)$. 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.

Server's Broadcast:

$$\mathsf{bid}, \mathsf{PE.Enc}(\pi_S, (\textsc{id}_S, g^s, \textsc{sig}_S(\mathsf{bid}, \textsc{id}_S, g^s)))$$

0-RTT Mutual Authentication:

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

Fig. 2. Basic message flow between the client C (with certificate $\textsc{id}_C$ and policy π_C) and the server S (with certificate $\textsc{id}_S$ and policy π_S) for the private discovery protocol. The client can also include early application data in the first flow of the 0-RTT mutual authentication protocol.

5.1 Protocol Analysis

We now describe some of the properties of our private service discovery protocol in Figure 2. We give a more detailed discussion in the full version of this paper.

0-RTT security. The security analysis of the 0-RTT mutual authentication protocol in Figure 2 is similar to that of the OPTLS protocol in TLS 1.3 [43] and relies on the Strong-DH and Hash-DH assumptions [9] in the random oracle model [14]. Note that in contrast to the OPTLS protocol which only provides client 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 [47]) 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.

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.

for all subsequent application-layer messages (encrypted under atk). We discuss the perfect forward secrecy properties of this protocol in Section 5.1.

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. 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. 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 the full version, we show that the security of the session is preserved even if the server's semi-static secret is compromised but the ephemeral secret is uncompromised. This method of combining a semi-static key with an ephemeral key also features in the OPTLS [43] and QUIC [47] protocols.

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 the full version.

Security theorem. We conclude by stating the security theorem for our private service discovery protocol. We give the formal proof in the full version [54].

Theorem 5.1 (Private Service Discovery). *The protocol in Figure 2 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 [1]. Due to space constraints, we defer some of the deployment details to the full version of this paper. 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. 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. Interactions between parties are encrypted and mutually authenticated based on the blessings bound to each end.

We implement our protocols to enhance the privacy of the Vanadium discovery framework. Our entire implementation is in Go (with wrappers for interfacing with third-party C libraries).

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 [19, §5] over the 256-bit Barreto-Naehrig (`bn256`) [48] pairings curve. We chose the BB_2 IBE scheme for its efficiency: it only requires a single pairing evaluation during decryption. We apply the Fujisaki-Okamoto transformation [32] to obtain CCA-security. For the underlying symmetric encryption scheme in the Fujisaki-Okamoto transformation, we use the authenticated encryption scheme from NaCl [16, 17]. All of our cryptographic primitives are chosen to provide at least 128 bits of security. In the full version, we give some microbenchmarks of the different IBE operations on several devices and describe how we integrate IBE into the Vanadium infrastructure.

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. 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 1. 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.

	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 1. Private mutual authentication benchmarks.

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. 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 the structure shown in Figure 2. Our implementation of prefix encryption ($\mathsf{PE.Enc}$) has a ciphertext overhead 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 [24, 25], which suffices for service advertisements.

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 [5], which is far too small to fit a full service advertisement. In the full version [54], we show how an auxiliary service can be used to host the encrypted advertisements, and thus, enable private service discovery over BLE and other similarly space-constrained advertisement protocols.

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. We provide more details in the full version [54].

7 Extensions

In the full version, we describe several ways to extend our protocols. These include ways to hide the server's authorization policy and allowing non-IBE roots to manage and issue prefix encryption keys for their subdomains.

8 Related Work

Private mutual authentication. The term "private authentication" was first introduced by Abadi and Fournet [7, 8]. However, the protocols in [8] 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. Many other cryptographic primitives have also been developed for problems related to private mutual authentication, including secret handshakes [12, 36, 11], oblivious signature-based envelopes [46], oblivious attribute certificates [45], hidden credentials [35, 31, 21], 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 [46], oblivious attribute certificates [45] and hidden credentials [35, 31, 21] 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 [18, 33] and predicate encryption [39, 34], 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 [57] for a survey. Broadly, these protocols can be categorized into two groups: "directory-based" protocols and "directory-free" protocols.

In directory-based discovery protocols [27, 55, 4], 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 [37, 38, 56, 58], 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 [30] 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 [49], 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.

Acknowledgments

We thank Martín Abadi, Mike Burrows, and Adam Langley for many helpful comments and suggestions. 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

[1] Vanadium. `http://vanadium.github.io/`.
[2] IETF zero configuration networking (zeroconf). `https://datatracker.ietf.org/doc/charter-ietf-zeroconf`, 2004.
[3] Bonjour printing specification version 1.2, 2013.
[4] Jini(TM) network technology specifications – Apache river version 2.2.0, 2013.
[5] Bluetooth specification version 4.2, 2014.
[6] UPnP(TM) device architecture 2.0, 2015.
[7] Martín Abadi. Private authentication. In *PETS*, pages 27–40, 2003.
[8] Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322:427–476, 2004.
[9] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In *CT-RSA*, pages 143–158, 2001.
[10] 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.
[11] Giuseppe Ateniese, Jonathan Kirsch, and Marina Blanton. Secret handshakes with dynamic and fuzzy matching. In *NDSS*, 2007.

[12] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[27] 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.

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

[29] 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.

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

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

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

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

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

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

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

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

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

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

[40] 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.

[41] 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.

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

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

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

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

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

[47] 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.

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

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

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

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

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

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

[54] David J. Wu, Ankur Taly, Asim Shankar, and Dan Boneh. Privacy, discovery, and authentication for the Internet of Things. *CoRR*, abs/1604.06959, 2016. Available at `http://arxiv.org/abs/1604.06959`.

[55] 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.

[56] 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.

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

[58] 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.