Attribute-based Key Exchange with General Policies

Vladimir Kolesnikov
Bell Labs
kolesnikov@research.bell-labs.com

Hugo Krawczyk
IBM Research
hugo@ee.technion.ac.il

Yehuda Lindell
Bar-Ilan University
yehuda.lindell@biu.ac.il

Alex J. Malozemoff
University of Maryland
amaloz@cs.umd.edu

Tal Rabin
IBM Research
talr@us.ibm.com

Abstract

Attribute-based methods provide authorization to parties based on whether their set of attributes (e.g., age, organization, etc.) fulfills a policy. In attribute-based encryption (ABE), authorized parties can decrypt, and in attribute-based credentials (ABC) systems, authorized parties can authenticate themselves. In this paper, we combine elements of ABE and ABC together with garbled circuits to construct attribute-based key exchange (ABKE). Our focus is on interactive solutions involving a client that holds a certificate (issued by an authority) vouching for that client's attributes and a server that holds a policy computable on such a set of attributes. The goal is for the server to establish a shared key with the client but only if the client's certified attributes satisfy the policy. Our solutions enjoy strong privacy guarantees for both the client and the server, including attribute privacy and unlinkability of client sessions.

Our main contribution is a construction of ABKE for *arbitrary circuits* with high (concrete) *efficiency*. Specifically, we support general policies expressible as boolean circuits computed on a set of attributes. Even for policies containing hundreds of thousands of gates the performance cost is dominated by two pairing computations per policy input. Put another way, for a similar cost to prior ABE/ABC solutions, which can only support small formulas efficiently, we can support *vastly* richer policies.

We implemented our solution and report on its performance. For policies with 100,000 gates and 200 inputs over a realistic network, the server and client spend 957 ms and 176 ms on computation, respectively. When using offline preprocessing and batch signature verification, this drops to only 243 ms and 97 ms.

1 Introduction

The increasing need and complexity of authentication in the digital world, alongside ever growing privacy concerns, has given rise to encryption and authentication mechanisms that combine privacy aspects (anonymity, unlinkability, etc.) with credentials that go well beyond asserting an identity of a client but rather vouch for a full set of attributes (age, rank, role, etc.). These mechanisms allow for authentication and encryption that build on authorization policies computed on the provided set of attributes. For example, in a hospital setting, access to a patient's records can be provided to the patient, her doctor, nurses while on duty, or to the director of the hospital ward, and this can be formalized as a policy.

The prime examples of these mechanisms are attribute-based credentials (ABC) [Cha81, Bra00, CL01, CL04, BCC+09, ide, upr] and attribute-based encryption (ABE) [SW05, GPSW06, BSW07,

Wat11]. The former is mainly directed towards identification and access control based on a set of attributes, and emphasizes privacy aspects such as anonymity, unlinkability, and attribute privacy. While ABC generally assumes an interactive setting between a client and a server, ABE focuses on (non-interactive) encryption where authorization is enforced through an encryption scheme that ties a ciphertext to a policy and a decryption key that ensures that only clients that have attributes that satisfy the policy can decrypt (note that we focus on the "ciphertext policy" setting of ABE [BSW07]). Since ABE schemes are non-interactive (and the party decrypting is completely passive), they have certain implicit privacy properties of ABC solutions such as unlinkability and attribute privacy. An essential requirement of the above primitives is that of collusion resistance. This means that different clients of the system cannot combine their attributes in order to pass policy verification that neither could have individually passed.

In many practical settings simply communicating a message to a party or just establishing rights (such as validating a function of attributes) is not enough. For example, authentication to an online service is usually followed with further communications that also need to be protected. In other words, the goal in these systems is the use of credentials to bootstrap a key exchange protocol that provides the parties with keys to protect a session.

In this paper we combine elements of ABE and ABC to build attribute-based key exchange (ABKE) where our focus is on interactive solutions involving a client that holds a certificate (issued by an authority CA) vouching for the client's attributes and a server that holds a policy computable on the set of attributes. The goal is for the server to establish a shared key with the client if and only if the client's certified attributes satisfy the policy.[1]

Of course, the above goal is easy to achieve if the client is willing to reveal its attributes to the server. The objective of our work is to enable the ABKE functionality while keeping the attributes of the client private alongside ensuring additional important properties. The main features of our ABKE solutions are summarized next.

General policies. We support any policy expressible as a polynomial-size boolean circuit computed on a set of attributes.

Attribute privacy. Client attributes are never disclosed. Of course, the server learns whether the key exchange succeeded and thus learns that the client's attributes fulfill the policy used in the exchange. However, nothing beyond this fact is revealed.

Unlinkability. Multiple communications with the same client (with one or more servers) cannot be linked together.

Collusion resistance. It is not possible for an adversary given keys associated with multiple clients with different attributes (certified by the CA) to succeed in an exchange in which no single client with its associated attributes fulfills the policy. In particular, attributes from different clients cannot be mixed-and-matched.

1.1 Overview of Our Solutions

Our main contributions are a simple and powerful definition (cf. §4) and realization (cf. §6) of attribute-based key exchange (ABKE) for public (circuit-based) policies.

[1] Note that we focus on the client-server setting where the client authenticates to the server. Server authentication usually happens with regular public key certificates that identify the server and can use standard tools such as TLS. Extensions of our system to the mutual authentication setting are possible but not treated here.

ABKE using garbled circuits. Our construction uses garbled circuits in order to achieve ABKE. The use of garbled circuits enables us to obtain a solution that both supports arbitrarily-complex policies (without requiring heavy machinery like multilinear maps or fully homomorphic encryption) and is concretely efficient. In our approach, the server generates a garbled circuit and sends it to the client. The client then obtains the garbled values on the input wires of the circuit, depending on its attributes. This is achieved by encrypting the garbled values on the input wires using a type of encryption that enables the client to decrypt only those values associated with its attributes. We call this notion attribute selective encryption (ASE) (cf. §5). The main technical difficulty comes with ensuring that the client obtains input labels corresponding to its credentials in a *private*, *unlinkable*, and *collusion-free* manner. At a high level, we construct such an encryption scheme using a rerandomizable set of public keys and a rerandomizable signature binding the public keys together. The client then presents a set of rerandomized keys (along with a signature on them), and the server encrypts the garbled labels knowing that the client can only decrypt the appropriate set. We introduce and utilize the notion of extractable linearly homomorphic (ELH) signatures (cf. §7) to construct two instantiations of ASE: one based on identity-based encryption (cf. §8) and the other built directly from ELH signatures (cf. §9). The *extractability* requirement ensures that a simulator can extract the *original* message that was signed, even though the adversary presents a *randomized* message. We prove this extractability property using the knowledge-of-exponent (KEA) assumption.

Our use of garbled circuits is a careful adaptation of the zero-knowledge-using-garbled-circuits approach of Jawurek et al. [JKO13]. As shown in their work, we can use a *single* garbled circuit while still achieving malicious security; this is discussed in more detail in §6.

Concrete performance. At a cost similar to that of prior ABE/ABC solutions, which only run efficiently on (small) formulas, we can support vastly richer policies represented by large circuits. Specifically, we instantiate our construction over bilinear groups requiring a number of pairings proportional to the number of *input* attributes to the policy circuit. Then a garbled circuit computation of the policy circuit is performed with cost that is not noticeable for policies of even relatively large circuit size[2].

To directly measure the performance of our scheme, we implemented it and ran various experiments; see §10. For example, in our implementation, the server and client computation time for a 1,000-gate policy and 10 attributes is **67 ms** and **11 ms**, respectively; for a 100,000-gate policy and 200 attributes the times are only **957 ms** and **176 ms**. We also note that much of the computation can be moved offline and we can use batch signature verification on the server side. Again with a 100,000-gate policy and 200 attributes, this optimized time is only around **243 ms** for the server, when assuming the server is batching ten messages in its signature verification, and **97 ms** for the client.

Additional features. Our construction can be easily extended to provide additional useful features, as detailed below.

- *Credential expiration*, by having attributes encode the expiration date.
- *Delegation of attributes*. This follows directly from the projectability property of our ASE definition (cf. §5).

[2]For example, we can garble (resp., evaluate) an AND gate in roughly 46 (resp., 28) cycles per gate using privacy-free garbled circuits [FNO15, ZRE15].

- *Multi-authority.* This can be achieved generically by having credentials from different CAs encode, as a sequence of attributes, a unique certified serial number which is verified to be the same during ABKE. A more efficient alternative is offered by our ELH-based ASE construction (cf. §9) by using a common value u in the clients' public keys in lieu of a unique serial number.
- *Unlinkability with respect to* CA. Our IBE-based construction for ASE (cf. §8) provides information theoretic unlinkability, which implies unlinkability even against a colluding server and CA. Such unlinkability is also achieved by our ELH-based solution provided that the public key components g and h are generated jointly between the client and CA.

Future work. In this work we consider public policies only. However, our techniques can be used to provide some notion of *private* policies and we leave this for future work. Likewise, our focus here has been on achieving *practical efficiency*, and we achieve this using the KEA assumption and the random-oracle model. The goal of achieving comparable efficiency under standard assumptions only and without a random oracle is important and we leave it for future work.

2 Related Work

Our ABKE notion relates to ciphertext-policy attribute-based encryption (CP-ABE) and attribute-based credentials (ABC). CP-ABE gives rise to a single-message key exchange (KE) solution in which a session key is encrypted under regular ABE encryption and hence is implicitly authenticated by clients that can decrypt. Since the same key is distributed to any client with a set of attributes satisfying the policy, multiple clients may share the same key. This is the solution proposed by Gorantla et al. [GBG10], who provide a game-based definition of attribute-based authenticated key exchange (under the abbreviation AB-AKE) and note that such a scheme is more in line with group key exchange than standard AKE.

ABE-based AKE requires several public-key operations per gate of the policy formula. Recent solutions to ABE for general circuits [GGH+13, GVW15], while sufficient to show feasibility, are mainly of theoretical interest due to the use of heavy underlying primitives. By using garbled circuits, our protocol costs are dramatically lower than either of the above ABE-based solutions.

Since most key exchange settings allow for interaction (the session that they protect is in itself typically interactive), our work leverages interaction to improve policy expressiveness as well as performance. In this sense we are closer to the ABC setting, where clients own the credentials they use in an interaction with a verifier. Our work inherits many of the challenges of ABC, particularly in the area of client privacy, with properties such as attribute-privacy and unlinkability being central to our work. We note, however, two important differences.

First and foremost, prior ABC protocols and systems focus on (but are not limited to) small *formula-based policies* [Cha81, Bra00, CL01, CL04, BCC+09, ide, upr, ABC] due to the high cost of needing several public-key operations per gate. Besides the cost, difficulty of policy design and analysis of non-trivial hand-generated small formulas is the reason that today's deployed systems mainly implement conjunction policies. In this work, we dramatically increase the computation power of the policy by enabling its implementation via garbled circuits. We believe that in addition to improving efficiency of existing ABC use cases, our work enables a much larger application scope for ABCs, due to the ability to run (large) policies auto-generated from easy-to-understand high-level code.

Secondly, the ABC literature focuses on verification of credentials and not on bootstrapping an authenticated session. In general, the ability to verify client credentials (i.e., a yes/no result) is insufficient for authenticating a session, even if the communication is carried over a server-authenticated channel (e.g., TLS). The relationship between credentials and key exchange is explicitly studied by Camenisch et al. [CCGS10], but their implementations do not cover rich policies, and do not outperform ABCs.

ABCs provide several practical features which we regard as future work, such as credential revocation and CA-verifier collusion. Other features, such as non-boolean and multi-authority credentials can be easily and cheaply built within our system (cf. §1.1).

In a very elegant concurrent and independent work, Chase et al. [CGM16] approach the problem of ABCs for non-boolean attributes by relying on garbled circuits to represent policies and, as a consequence, allow general circuit-based policies. The method of delivery of wire labels to the prover (in our notation, the client) is indeed the technical core of both of our approaches. Chase et al. allow the prover to enter arbitrary inputs to the garbled circuit, requiring a zero-knowledge proof that its garbled circuit inputs are consistent with arithmetic committed values, which, in turn, are consistent with the credential vector on which there exists a valid CA signature. This results in a number of exponentiations per boolean attribute, even in cases where a small subset of them are used in the policy. Chase et al. offer an alternative algorithm to reduce the number of public-key operations at the expense of message authentication code computation inside the garbled circuit, which introduces a significant communication overhead but may be a worthy trade-off for provers with many attributes. This approach, too, scales with the total number of attributes. In contrast, in our approach the client needs to only compute public-key operations per *policy* attribute (rather than over all the client's attributes as is required by Chase et al.), which may be significantly faster in many settings. However, our improved performance is a trade-off for using stronger assumptions. Additionally, we present the first implementation of general circuit ABKE (and hence ABC), and report on its concrete performance.

Finally, Sakai et al. [SAH16] very recently proposed attribute-based signatures for circuits based on bilinear maps. In their setting, only signers satisfying a certain policy on their attributes could successfully sign a message. Their scheme could be a basis for an ABC solution; however, they require several public-key operations and about 1 Kb of data sent per circuit gate; our garbled circuit-based solution is much more efficient (16 bytes and several symmetric key operations per circuit gate).

3 Preliminaries

Let $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$ and $\mathsf{S}_1, \ldots, \mathsf{S}_t$ be the set of clients and servers, respectively, and let $A = \{a_1, \ldots, a_m\}$ be the universe of all possible attributes. We associate an m-bit string $\chi_i = \chi_i[1] \cdots \chi_i[m] \in \{0,1\}^m$ with each P_i such that $\chi_i[j] = 1$ if and only if P_i has attribute a_j. A policy is a (polynomial sized) circuit C with m inputs and a single-bit output. We say that P_i satisfies policy C if and only if $\mathrm{C}(\chi_i) = 1$.

Garbled circuits. One of our main building blocks is garbled circuits. As the circuit description is public and only one party has input, we can utilize privacy-free garbled circuits [FNO15], which are more efficient than standard garbled circuits. We use the garbled circuit notation of Bellare et al. [BHR12], with one function (verification) introduced by Jawurek et al. [JKO13]. We only consider circuits with a single bit of output, as this is all that is needed in our setting.

We define a *verifiable garbling scheme* by a tuple of functions $\mathcal{G} = (\mathsf{Gb}, \mathsf{Ev}, \mathsf{Ve})$ with each function defined as follows:

- *Garbling* algorithm $\mathsf{Gb}(1^n, \mathrm{C})$: A randomized algorithm which takes as input the security parameter and a circuit $\mathrm{C} : \{0,1\}^m \to \{0,1\}$ and outputs a tuple of strings $(\mathrm{GC}, \{X_j^0, X_j^1\}_{j\in[m]}, \{Z^0, Z^1\})$, where GC is the garbled circuit, the values $\{X_j^0, X_j^1\}_{j\in[m]}$ denote the input-wire labels, and the values $\{Z^0, Z^1\}$ denote the output-wire labels.

- *Evaluation* algorithm $\mathsf{Ev}(\mathrm{GC}, \{X_j\}_{j\in[m]})$: A deterministic algorithm which evaluates garbled circuit GC on input-wire labels $\{X_j\}_{j\in[m]}$.

- *Verification* algorithm $\mathsf{Ve}(\mathrm{C}, \mathrm{GC}, \{X_j^0, X_j^1\}_{j\in[m]})$: A deterministic algorithm which takes as input a circuit C, garbled circuit GC, and input-wire labels $\{X_j^0, X_j^1\}_{j\in[m]}$, and outputs accept if GC is a valid garbling of C and reject otherwise.

A verifiable garbling scheme must satisfy three security properties: (1) *correctness*, (2) *authenticity*, and (3) *verifiability*. The definitions for correctness and authenticity are standard: correctness enforces that a correctly garbled circuit, when evaluated, outputs the correct output of the underlying circuit; authenticity enforces that the evaluator can only learn the output label that corresponds to the value of the function. *Verifiability* [JKO13] allows one to check that the garbled circuit indeed implements the specified plaintext circuit C.

t-KEA assumption. We recall the t-KEA assumption used in our implementation of extractable linearly homomorphic signature from §7. The assumption was formulated in [BCCT12, BCC+14]. See these papers and [GS14] for a good discussion and further references related to this assumption and its recent use. See also [AF07, Gro10] for a proof of security for t-KEA in the generic (bilinear) group model. The formulation below is simplified by not including an auxiliary input that, if present, is the same for both algorithms E and E'.

Definition 3.1. (t-KEA [BCCT12, BCC+14]) *Let G be a cyclic group of prime order q. Consider algorithms that on input t random elements $g_1, \ldots, g_t$ in G and t values $g_1^x, \ldots, g_t^x$ for $x \in_R Z_q$, output a pair (f, f') in G^2. Such an algorithm E is said to be a t-KEA algorithm if with non-negligible probability (over the choice of inputs to E and E's random coins) E outputs (f, f') such that $f' = f^x$. We say that* the t-KEA assumption holds *over G if for every efficient t-KEA algorithm E in G there exists another efficient algorithm E' for which the following property holds except for a negligible probability: Let $g_1, \ldots, g_t, g_1^x, \ldots, g_t^x$ be an input to E and ρ a vector of random coins for E on which E outputs $(f, f' = f^x)$ then on the same inputs (and random coins) E' outputs a vector $(f, f' = f^x, x_1, \ldots, x_n)$ such that $f = g_1^{x_1} \cdots g_n^{x_n}$.*

Auxiliary functionalities. Our construction makes use of two (standard) functionalities for commitments ($\mathcal{F}_{\mathsf{com}}$) and secure coin-tossing ($\mathcal{F}_{\mathsf{cointoss}}$). We recall them here for completeness; see Figure 3.1 and Figure 3.2.

Anonymous channels. Our protocol assumes that the parties interact over anonymous channels. In practice, the anonymity provided by the network used by the clients is the level of anonymity that they achieve. For the purpose of proving security, we assume a *perfect* anonymous channel. In the SUC framework, all messages to and from functionalities have public headers consisting of the type of operation, and the private content itself; the public header is revealed to the adversary but not the private content. However, the adversary is always given the identity of the party

$\mathcal{F}_{\mathsf{com}}$ runs between a sender and a receiver and works as follows.

1. On sender input $(\mathsf{commit}, \mathsf{sid}, i, m)$, if no message of the form $(\mathsf{sid}, i, \cdot)$ is already stored, store (sid, i, m) and send $(\mathsf{committed}, \mathsf{sid}, i, |m|)$ to the receiver.
2. On sender input $(\mathsf{reveal}, \mathsf{sid}, i)$, if a message of the form (sid, i, m) is stored, send $(\mathsf{reveal}, \mathsf{sid}, i, m)$ to the receiver and halt.

Figure 3.1: Functionality $\mathcal{F}_{\mathsf{com}}$ for commitment, taken mostly verbatim from Jawurek et al. [JKO13, Fig. 4].

$\mathcal{F}_{\mathsf{cointoss}}$ runs between two parties P_i and P_j and works as follows.

1. On input $(\mathsf{toss}, \mathsf{sid})$, if no message of the form (sid) is stored, store (sid), otherwise choose $r \leftarrow\!\!\$\ \{0,1\}^n$, send $(\mathsf{tossed}, \mathsf{sid}, r)$ to both parties, and halt.

Figure 3.2: Functionality $\mathcal{F}_{\mathsf{cointoss}}$ for secure coin-tossing.

sending the message to the functionality and the identity of the party receiving the message from the functionality. Thus, in order to model anonymous channels, all parties must send and receive together. (This actually makes sense since in principle, an adversary who can view the entire network can break anonymity unless every party interacts in each round. Nevertheless, here we use this simply as a way to model the requirements.) The $\mathcal{F}_{\mathsf{anon}}$ functionality appears in Figure 3.3. In the functionality all parties send a message to all other parties in each round. Note that if a party has no message at all to send, or it only needs to send to some parties, then it can simply use an empty message. We stress again that in practice not all parties need to interact in each round; this is merely for the purpose of modeling.

4 Security Definition

All of our definitions and proofs are in the simple-UC (SUC) model [CCL15]. As was shown in the aforementioned work [CCL15], any protocol that is secure in the SUC framework is also secure in the full UC framework.

Attribute-based key exchange. We present a functionality $\mathcal{F}_{\mathsf{abke}}$ for attribute-based key exchange supporting attribute privacy, unlinkability, and collusion resistance. The functionality is initialized with a set of attribute vectors $\{\chi_i\}$, where χ_i corresponds to the attribute vector of client P_i. The functionality begins by waiting for a message from a server S_j that contains a circuit C representing S_j's policy. The functionality stores this information and broadcasts a notification to all parties $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$ that a policy is available. Upon receiving a response by one of the parties, say, P_i, the functionality proceeds as follows. If $\mathrm{C}(\chi_i) = 1$, the policy is satisfied and so the functionality forwards a random key k to both P_i and S_j. If $\mathrm{C}(\chi_i) = 0$, then $\mathcal{F}_{\mathsf{abke}}$ sends $\perp$ to both P_i and S_j. The full description of $\mathcal{F}_{\mathsf{abke}}$ can be found in Figure 4.1.

Attribute privacy is captured by the fact that S_j never receives the attribute vector χ_i of client P_i. Collusion resistance is handled by the fact that each party's attribute vector is fixed upon functionality initialization and cannot be changed. Thus, parties cannot use any attribute vector that differs from their initial ones. This implies that collusions between parties to effectively use a different attribute vector are impossible. Finally, unlinkability follows since the functionality

$\mathcal{F}_{\mathsf{anon}}$ works with clients $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$ as follows:

1. Upon receiving a message $(\mathsf{send}, \mathsf{sid}, \mathsf{P}_i, (m_1^i, \ldots, m_\ell^i))$ from P_i, $\mathcal{F}_{\mathsf{anon}}$ stores the message.
2. After $\mathcal{F}_{\mathsf{anon}}$ receives a send message from *every* client $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$, $\mathcal{F}_{\mathsf{anon}}$ sends $(\mathsf{receive}, \mathsf{sid}, \mathsf{P}_j, (m_j^1, \ldots, m_j^\ell))$ to client P_j for $j = 1, \ldots, \ell$.

The public header of each message is $(\mathsf{send}, \mathsf{sid}, \mathsf{P}_i)$ and $(\mathsf{receive}, \mathsf{sid}, \mathsf{P}_j)$, respectively, for send and receive messages. The private contents is the vector of messages.

Figure 3.3: Anonymous communications functionality $\mathcal{F}_{\mathsf{anon}}$.

$\mathcal{F}_{\mathsf{abke}}$ runs with clients $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$ with attribute vectors $\chi_1, \ldots, \chi_\ell \in \{0,1\}^m$, and servers $\mathsf{S}_1, \ldots, \mathsf{S}_t$, and works as follows:

1. Upon receiving $(\mathsf{policy}, \mathsf{sid}, \mathrm{C})$ from some S_j, where C is either a circuit $\mathrm{C}' : \{0,1\}^m \to \{0,1\}$ or $\perp$, send $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$ to all $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$. If $\mathrm{C} = \perp$ then halt, and otherwise store $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$.
2. Upon receiving $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j)$ from S_j and $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j, \mathsf{P}_i)$ from P_i, if some message $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$ is stored, then:
   - If $\mathrm{C}(\chi_i) = 1$ then choose $k \in_R \{0,1\}^n$ and send $(\mathsf{completed}, \mathsf{sid}, k)$ to P_i and S_j.
   - If $\mathrm{C}(\chi_i) = 0$ then send $(\mathsf{completed}, \mathsf{sid}, \perp)$ to P_i and S_j.
3. Upon receiving $(\mathsf{abort}, \mathsf{sid})$ from Sim, clear any message $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$ that is stored, send $(\mathsf{abort}, \mathsf{sid})$ to P_i and S_j, and halt.

The public header of each message is: $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$, $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j)$, and $(\mathsf{completed}, \mathsf{sid})$; all other content is private.

Figure 4.1: Attribute-based key exchange functionality $\mathcal{F}_{\mathsf{abke}}$ with attribute privacy, unlinkability and collusion resistance.

does not pass on the identity of the client P_i to the server S_j at any time. We note that we do not provide server anonymity in our definition, since it does not seem to be required for the ABKE setting. Thus, the server's identity is revealed in the functionality definition.

We also introduce a functionality $\mathcal{F}_{\mathsf{setup}}$ for providing each party with the ASE keys used in our protocol construction; see Figure 4.2.

5 Attribute Selective Encryption

We introduce the notion of attribute selective encryption (ASE). ASE is related to ABE in the sense that clients' keys and decryption capabilities are related to the attributes they possess. In ASE a plaintext is comprised of a *set* of messages, and a client's credentials determine *which subset* can be decrypted. In more detail, each client has an m-bit vector $\chi \in \{0,1\}^m$ representing a set of attributes: $\chi[j]$ is set to 1 *if and only if* the client possesses the jth attribute. The cleint holds public and secret keys associated with χ. Anyone can encrypt a set of $2m$ messages $\begin{pmatrix} x_{1,0} & \cdots & x_{m,0} \\ x_{1,1} & \cdots & x_{m,1} \end{pmatrix}$ under the client's public key, and ASE enforces an OT-like property such that the client can decrypt

Upon initialization with length parameter m, $\mathcal{F}_{\mathsf{setup}}$ runs $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$ and stores $(\mathsf{mVK}, \mathsf{mSK})$.

1. Upon receiving $(\mathsf{generate}, \mathsf{sid}, \chi_i)$ from player P_i, $\mathcal{F}_{\mathsf{setup}}$ checks if there exists a record $(i, \cdot, \cdot, \cdot)$. If so, $\mathcal{F}_{\mathsf{setup}}$ sends $(\mathsf{result}, \mathsf{sid}, \perp)$ to P_i. Otherwise, $\mathcal{F}_{\mathsf{setup}}$ runs $(pk, sk) \leftarrow \mathsf{GenCert}(\mathsf{mSK}, \chi_i)$, records (i, χ_i) and sends $(\mathsf{result}, \mathsf{sid}, \mathsf{mVK}, pk, sk)$ to P_i.

The public header of each message is: $(\mathsf{generate}, \mathsf{sid})$ and $(\mathsf{result}, \mathsf{sid})$; all other content is private.

Figure 4.2: Setup functionality $\mathcal{F}_{\mathsf{setup}}$.

using its secret key only one of each $(x_{i,0}, x_{i,1})$, depending on $\chi[j]$. That is, the client decrypts the messages $x_{1,\chi[1]}, \ldots, x_{m,\chi[m]}$, and nothing else. We stress that ASE, unlike ABE, encrypts under a specific client's public key, and only that client can decrypt.

Besides the basic semantic security notion of ASE, we consider four additional properties: *attribute privacy*, *collusion resistance*, *unlinkability*, and *projectability*. Each property on its own is easy to achieve, yet the combination, especially that of collusion resistance and unlinkability, makes the construction challenging.

Attribute privacy. This property requires that the public key pk_χ hides attribute vector χ. That is, publishing pk_χ does not reveal which attributes the client holds. The following trivial solution achieves attribute privacy: generate a set of $2m$ public keys and define the secret key to consist of only one of the secret key in each pair.

Collusion resistance. A set $\mathcal{P}$ of clients with attribute vectors $\mathcal{X} = \{\chi_i\}_{P_i \in \mathcal{P}}$ and corresponding keypairs must not be able to construct a keypair representing $\chi \notin \mathcal{X}$ (or χ representing a subset of attributes not implied by $\mathcal{X}$ — cf. *projectability* below). Collusion resistance can be achieved by combining the trivial solution from above with a secure signature scheme; i.e., by providing a signature on the set of the client's public keys. This prevents clients from mixing and matching the individual keys in their public keys, giving collusion resistance.

Unlinkability. Unlinkability is the inability to link between different uses of the same public key. Specifically, we require that it be possible to *randomize* a public key using some algorithm Unlink so that the pair $(pk_\chi, \mathsf{Unlink}(pk_\chi))$ looks like two independent public keys. Without the requirement of collusion resistance, unlinkability is easy to achieve (e.g., by using ElGamal keys). However, as we are interested in collusion resistance we thus need to enable the creation of a signature on the randomized key. For this we need to use *homomorphic signatures*. However, existing signature schemes do not provide the capabilities that are need for our schemes. Thus, combining signatures with unlinkability is not straightforward.

Projectability. We require that given a keypair associated with a vector χ, one can generate a keypair that is associated with any orthogonal projection of χ onto some subset $S \subseteq [m]$. We stress that the subset S is *explicit* in the projection (otherwise, the encrypting party cannot know what the projection is, and this could be used to obtain unauthorized decryptions). As each public key needs to be certified, this implies that the certificate for the new key also needs to be derived from the certificate of pk_χ.

5.1 Formal Definition

Let n denote the security parameter and let m be the length of the attribute vector. We assume for simplicity that the client receives a public key on the entire attribute vector.

Definition 5.1. An attribute selective encryption (ASE) scheme with *attribute privacy*, *collusion resistance*, and *unlinkability* is a tuple of probabilistic-polynomial time algorithms ($\mathsf{Setup}, \mathsf{GenCert}, \mathsf{Vrfy}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Unlink}, \mathsf{Project}$) as follows:

- $\mathsf{Setup}(1^n, m)$ takes as input an attribute set size m, and outputs a master verification key and a master secret key $(\mathsf{mVK}, \mathsf{mSK})$ along with public parameters PP. All the following algorithms implicitly take PP as input.
- $\mathsf{GenCert}(\mathsf{mSK}, \chi)$ takes as input the master secret key and attribute vector $\chi \in \{0,1\}^m$, and outputs a certified keypair (pk_χ, sk_χ) associated with χ.
- $\mathsf{Vrfy}(\mathsf{mVK}, pk_\chi)$ takes as input the master verification key and a public key pk_χ, and outputs 1 if and only if pk_χ is a valid public key.
- $\mathsf{Enc}(pk_\chi, \vec{x})$ takes as input a public key pk_χ, and a vector $\vec{x}$, where $\vec{x} = \begin{pmatrix} x_{1,0} & \cdots & x_{m,0} \\ x_{1,1} & \cdots & x_{m,1} \end{pmatrix}$ is a series of $2m$ messages. The function outputs an encryption c. For simplicity, we assume that each $x_{i,b}$ is of length n (this suffices for our use).
- $\mathsf{Dec}(sk_\chi, c)$ takes as input a secret key sk_χ and a ciphertext c, and outputs a set of m plaintexts based on χ.
- $\mathsf{Unlink}(pk_\chi, sk_\chi)$ takes as input a public key pk_χ and its associated private key sk_χ, and outputs a new keypair (pk'_χ, sk'_χ) for the same χ.
- $\mathsf{Project}(pk_\chi, sk_\chi, S)$ takes as input a public key pk_χ, its associated secret key sk_χ, and a set $S \subseteq \{0,1\}^m$ which defines χ' by specifying which attributes of χ are to be preserved. $\mathsf{Project}$ outputs a keypair $(pk'_{\chi'}, sk'_{\chi'})$ on the projected attribute vector χ'.

We require the following properties on the algorithms:

- (Correctness) For every $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$, $\chi \in \{0,1\}^m$, $(pk_\chi, sk_\chi) \leftarrow \mathsf{GenCert}(\mathsf{mSK}, \chi)$, and $\vec{x} = \begin{pmatrix} x_{1,0} & \cdots & x_{m,0} \\ x_{1,1} & \cdots & x_{m,1} \end{pmatrix}$, it holds that $\mathsf{Dec}\left(sk_\chi, \mathsf{Enc}(pk_\chi, \vec{x})\right) = (x_{1,\chi_1}, \ldots, x_{m,\chi_m})$.
- The output distribution of Unlink is the same as the output distribution of $\mathsf{GenCert}$, and correctness holds for every $(pk'_\chi, sk'_\chi) \leftarrow \mathsf{Unlink}(pk_\chi, sk_\chi)$.
- For every $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$, $\chi \in \{0,1\}^m$, and $(pk_\chi, sk_\chi) \leftarrow \mathsf{GenCert}(\mathsf{mSK}, \chi)$, it holds that $\mathsf{Vrfy}(\mathsf{mVK}, pk_\chi) = 1$.
- The algorithm Enc is a committing encryption scheme.

Finally, we require the existence of the following two algorithms, which are used in our security definitions:

- $\mathsf{GenCert}^*(\mathsf{mSK})$ takes as input the master secret key and outputs a certified keypair (pk, sk) associated with both the 0 and 1 value of each attribute.
- $\mathsf{Dec}^*(sk, c)$ takes as input a secret key sk generated by $\mathsf{GenCert}^*$ and a ciphertext c, and outputs the full set of $2m$ plaintexts.

We call an ASE scheme projectable if:

- For every $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$, $\chi \in \{0,1\}^m$, $(pk_\chi, sk_\chi) \leftarrow \mathsf{GenCert}(\mathsf{mSK}, \chi)$, $S \subseteq \{0,1\}^m$, the output of $\mathsf{Project}(pk_\chi, sk_\chi, S)$ is distributed according to $\mathsf{GenCert}(\mathsf{mSK}, \chi')$ for χ' derived according to S.
- Correctness holds for every projected attribute vector.

Having defined the syntax, we now define security. We define this via experiments between a challenger $\mathcal{C}$ and an adversary Adv for an ASE scheme π.

Collusion resistance. Our collusion resistance experiment guarantees that players can only obtain decryptions authorized by their attribute vectors. The adversary Adv is given oracle access to $\mathsf{GenCert}$ in order to model Adv corrupting multiple parties and learning their attribute vectors. Eventually, Adv sends a public key to $\mathcal{C}$, who responds with a random plaintext $\vec{x}$ encrypted under this public key. The adversary Adv responds with a set of potential plaintext messages. If some subset of this set corresponds to an attribute vector (or any of its projections) that were *not* queried by Adv to $\mathsf{GenCert}$, then Adv wins.

The reason we need to define collusion resistance in this way is that when proving security of our ABKE scheme, we extract the plaintext through the adversary's calls to the random oracle. Namely, the plaintext messages $x_{i,b}$ are input into the random oracle by the adversary. However, the adversary is not limited to just inputting the proper messages to the random oracle, and thus we need to consider the set of *all* queries to the random oracle, a subset of these which may contain the extracted plaintext.

Note that it is easy for the challenger to check whether such a subset exists as follows. It checks whether each message in $\mathcal{M}$ is a valid plaintext message $x_{i,b}$. Given this set of valid plaintext messages, the challenger can extract an attribute vector (based on the (i,b) values) and check whether such an attribute vector is unauthorized as per the definition.

Experiment $\mathsf{Expt}^{\text{collude}}_{\pi,\mathsf{Adv}}(1^n, m)$**:**

1. $\mathcal{C}$ computes $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$ and sends PP and mVK to Adv.
2. Adv, with oracle access to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$, outputs a public key pk. Let $\mathcal{X}$ be the set of attribute vectors Adv used as input to its oracle.
3. $\mathcal{C}$ chooses a random plaintext $\vec{x}$, as specified by the ASE syntax, and sends $\mathsf{Enc}(pk, \vec{x})$ to Adv.
4. Adv outputs a set $\mathcal{M}$ of potential plaintext messages.
5. The output of the experiment is 1 (and Adv wins) *if and only if* the following conditions all hold:
   (a) $\mathsf{Vrfy}(\mathsf{mVK}, pk) = 1$;
   (b) There exists some subset $\mathcal{M}' \subseteq \mathcal{M}$ such that either (1) the strings in $\mathcal{M}'$ correspond to $\{x_{i,\chi[i]}\}_{i\in[m]}$ for some attribute vector χ, *or* (2) there exist two strings $s, s' \in \mathcal{M}'$ such that $s = x_{i,0}$ and $s' = x_{i,1}$ for some $i \in [m]$.
   (c) $\chi \notin \mathcal{X}$, and χ is not a projection of any vector from $\mathcal{X}$.

Attribute privacy. We now consider an adversary who aims to infer χ from pk_χ. This follows a standard indistinguishability-based formulation. At a high level, the adversary is trying to distinguish a public key generated for some attribute vector χ with an "all-powerful" public key generated by $\mathsf{GenCert}^*$. Note that the inability to distinguish these two settings implies the inability to distinguish between any two attribute vectors by a simple hybrid argument.

Experiment $\mathsf{Expt}^{\text{att-priv}}_{\pi,\mathsf{Adv}}(1^n, m)$**:**

1. $\mathcal{C}$ computes $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$ and sends PP and mVK to Adv.
2. Adv, with oracle access to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$, sends attribute vector $\chi \in \{0,1\}^m$ to $\mathcal{C}$.
3. $\mathcal{C}$ chooses $b \in_R \{0,1\}$, and proceeds as follows:
   - If $b = 0$, compute $(pk, sk) \leftarrow \mathsf{GenCert}(\mathsf{mSK}, \chi)$ and send pk to Adv.
   - If $b = 1$, compute $(pk, sk) \leftarrow \mathsf{GenCert}^*(\mathsf{mSK})$ and send pk to Adv.
4. Adv outputs a bit b'.
5. The output of the experiment is 1 (and Adv wins) *if and only if* $b' = b$.

Unlinkability. Finally, we define an experiment to formalize the property of unlinkability. The definition is relatively weak in that we only need to prevent an adversary from determining whether a keypair has been run through Unlink or not. However, this is sufficient for our purposes. In particular, unlinkability of keys used in our ABKE protocols will hold due to the conjunction of the guarantees of both attribute privacy and unlinkability.

Experiment $\mathsf{Expt}^{\mathrm{link}}_{\pi,\mathsf{Adv}}(1^n, m)$**:**

1. $\mathcal{C}$ computes $(\mathsf{PP}, \mathsf{mVK}, \mathsf{mSK}) \leftarrow \mathsf{Setup}(1^n, m)$ and sends PP and mVK to Adv.
2. Adv with oracle access to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$ eventually sends χ to $\mathcal{C}$.
3. $\mathcal{C}$ computes $(pk_0, sk_0) \leftarrow \mathsf{GenCert}(\mathsf{mSK}, \chi)$ and $(pk_1, sk_1) \leftarrow \mathsf{Unlink}(pk_0, sk_0)$. $\mathcal{C}$ chooses $b \in_R \{0,1\}$ and sends (pk_b, sk_b) to Adv.
4. Adv outputs b'.
5. The output of the experiment is 1 (and Adv wins) *if and only if* $b' = b$.

Note that we cannot simply set Unlink to the identity function as we need the output distribution of Unlink to be the same as that of $\mathsf{GenCert}$, as required in Definition 5.1.

Security definition. We are now ready to define security.

Definition 5.2. A (projectable) attribute selective encryption scheme π with *attribute privacy*, *collusion resistance*, and *unlinkability* is secure if for every probabilistic-polynomial time adversary Adv there exists a negligible function μ such that for every n and every $\mathrm{X} \in \{\text{att-priv}, \text{link}\}$ it holds that

$$\Pr\left[\mathsf{Expt}^{\mathrm{X}}_{\pi,\mathsf{Adv}}(1^n, m) = 1\right] \leq \frac{1}{2} + \mu(n)$$

and

$$\Pr\left[\mathsf{Expt}^{\mathrm{collude}}_{\pi,\mathsf{Adv}}(1^n, m) = 1\right] \leq \mu(n).$$

ASE Instantiations. We present two schemes realizing Definition 5.2 in §8 and §9.

6 ABKE from ASE

We now construct ABKE for public policies by integrating ASE with garbled circuit-based zero-knowledge proofs [JKO13] and key agreement. Jawurek et al. [JKO13] observed that for zero-knowledge proofs, the verifier-constructed circuit may be opened to the prover post-evaluation since it has no private data. Carefully arranging the prover's and verifier's commitments and openings, they ensure that neither can cheat, and only a single garbled circuit needs to be garbled, sent, and evaluated. Specifically, their protocol proceeds by the server using a *sender-committing* oblivious transfer (OT) to transfer the input-wire labels to the client. Given the garbled circuit

The protocol Π_{abke} is between server S and client P with attribute vector χ. We assume a setup where each client P_i sends $(\mathsf{generate}, \mathsf{sid}, S, \chi_i)$ for some set $S \subseteq [m]$ and attribute vector χ_i to $\mathcal{F}_{\mathsf{setup}}$, receiving $(\mathsf{result}, \mathsf{sid}, \mathsf{mVK}, pk, sk)$ in response. We assume all messages are sent/received through the $\mathcal{F}_{\mathsf{anon}}$ functionality; for simplicity we omit the use of this functionality in the description below. Also for simplicity, we assume the evaluated policy C uses all m attributes (otherwise, a corresponding Projectoperation can be applied to the party's key).

1. S broadcasts policy circuit C to all parties. If C is not a valid policy then P outputs $\perp$.
2. S runs $(\mathrm{GC}, \{X_j^0, X_j^1\}_{j\in[m]}, \{Z^0, Z^1\}) \leftarrow \mathsf{Gb}_{\mathsf{ABKE}}(1^n, \mathrm{C})$.
3. P computes $(pk', sk') \leftarrow \mathsf{Unlink}(pk, sk)$ and sends pk' to S.
4. S runs $\mathsf{Vrfy}(\mathsf{mVK}, pk')$. If the output is zero then S aborts; otherwise, S sets $\vec{x} = \{X_j^0, X_j^1\}_{j\in[m]}$, computes $c \leftarrow \mathsf{Enc}(pk', \vec{x})$ and sends c and GC to P.
5. P computes $\vec{m} \leftarrow \mathsf{Dec}(sk', c)$, where $\vec{m} = \widehat{X}_0^{\chi_j^0}, \ldots, \widehat{X}_m^{\chi_j^m}$ and computes $Z \leftarrow \mathsf{Ev}_{\mathsf{ABKE}}(\mathrm{GC}, \{\widehat{X}_i^{\chi_j^i}\}_{i\in[m]})$; P sets $Z := \perp$ if $\mathsf{Ev}_{\mathsf{ABKE}}$ fails.
6. P sends $(\mathsf{commit}, \mathsf{sid}, 1, Z)$ to $\mathcal{F}_{\mathsf{com}}$, which sends $(\mathsf{committed}, \mathsf{sid}, 1, |Z|)$ to S.
7. S sends the wire labels $\{X_i^b\}$ and the randomness r used in the encryption to P, who verifies that the encryptions match the wire labels and then computes $\mathsf{Ve}_{\mathsf{ABKE}}(\mathrm{C}, \mathrm{GC}, \{X_i^0, X_i^1\}_{i\in[m]})$. If either the wire labels did not match the encrypted values or the output of $\mathsf{Ve}_{\mathsf{ABKE}}$ is reject then P outputs $\perp$. Likewise, if $\mathrm{C}(\chi_j) = 0$ then P outputs $\perp$. Otherwise, P sends $(\mathsf{reveal}, \mathsf{sid}, 1)$ to $\mathcal{F}_{\mathsf{com}}$, which sends $(\mathsf{reveal}, \mathsf{sid}, 1, Z)$ to S.
8. S checks that $Z = Z^1$, if not, it sends $\perp$ to P and halts. Otherwise, the parties both send $(\mathsf{toss}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{cointoss}}$, receive $(\mathsf{tossed}, \mathsf{sid}, k)$, and output k.

Figure 6.1: Protocol Π_{abke} realizing $\mathcal{F}_{\mathsf{abke}}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{cointoss}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.

and input-wire labels, the client can evaluate the garbled circuit and commit the output-wire label to the server. Now, the server can decommit to its inputs of the OT, allowing the client to verify that the garbled circuit was constructed correctly. If so, the client can open the commitment to its output and the server can verify that the client indeed computed the correct output-wire label.

We adapt this protocol to realize $\mathcal{F}_{\mathsf{abke}}$ by replacing sender-committing OT with ASE. That is, instead of the parties running the OT step in Jawurek et al.'s protocol, the client sends its (randomized) ASE public key to the server, who encrypts each input-wire label of the garbled circuit with ASE, guaranteeing that the client is only able to decrypt labels corresponding to its attribute vector. Next, the client evaluates the garbled circuit and commits to the output-wire label it computed. The server can then open all the encrypted values, allowing the client to verify the circuit was correctly garbled (before the client reveals anything). If the circuit is correct, the client decommits the output-wire label it computed, allowing the server to verify that indeed the client satisfied the policy. The parties then run a secure coin-tossing protocol to derive the shared key. See Figure 6.1 for the full protocol description.

Garbling scheme for ABKE. For us to successfully reduce to the collusion experiment in our ASE definition, we need to extract the plaintext from a malicious client to feed to the challenger in the experiment. This plaintext corresponds to the input-wire labels of the garbled circuit. Thus, we need some way to do this extraction. We do this by using a random oracle: we can monitor

the inputs to the random oracle and use these as "potential" plaintexts which we can feed to the challenger in the collusion experiment. Thus, we construct a simple modified garbling scheme which allows us to do this extraction.

ABKE garbling scheme $\mathcal{G}_{\mathsf{ABKE}} = (\mathsf{Gb}_{\mathsf{ABKE}}, \mathsf{Ev}_{\mathsf{ABKE}}, \mathsf{Ve}_{\mathsf{ABKE}})$ for a circuit C with m inputs.

$\mathsf{Gb}_{\mathsf{ABKE}}$ is defined as follows.

1. Generate $2m$ random labels $X_i^b \in_R \{0,1\}^n$, where X_i^0 and X_i^1 correspond to input wire i. The set of all X_i^b form the input-wire labels to be encrypted. Let $h_i^b := \mathrm{RO}(i \| b \| X_i^b)$, where RO is a random oracle.
2. Using any secure garbling scheme Gb, generate the garbled circuit, including the $2m$ input-wire labels W_i^b.
3. Append to the generated garbled circuit the following input-wire translation tables: for wire i, append $\begin{pmatrix} \mathsf{Enc}_{h_i^0}(W_i^0) \\ \mathsf{Enc}_{h_i^1}(W_i^1) \end{pmatrix}$. Set the input-wire labels to be the set $\{X_i^b\}$ and the output-wire labels to be those set by Gb.

The $\mathsf{Ev}_{\mathsf{ABKE}}$ and $\mathsf{Ve}_{\mathsf{ABKE}}$ functions are defined naturally from Ev, Ve, and $\mathsf{Gb}_{\mathsf{ABKE}}$.

Clearly, the scheme allows evaluation and verification in the same manner as the underlying garbling scheme once a label per each wire is obtained. At the same time, any party evaluating a garbled circuit must make a call to the random oracle per input-wire label in order to learn the "real" underlying label for the garbled circuit. Thus, the underlying garbling scheme Gb cannot be decrypted without a random oracle evaluation on a input-wire label of $\mathsf{Gb}_{\mathsf{ABKE}}$, which is exactly the property we need for the reduction to the collusion experiment.

Theorem 6.1. *Assume that the encryption scheme used in Π_{abke} is a secure attribute selective encryption scheme. Then Π_{abke} securely computes $\mathcal{F}_{\mathsf{abke}}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{cointoss}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model, in the random-oracle model.*

Proof. We consider an adversary Adv that corrupts any number of parties. While we allow for arbitrary corruptions among parties, there are only two interesting cases: a (potentially colluding) corrupted server talking to an honest client, and an honest server talking to a (potentially colluding) corrupted client. We construct simulators for each of these cases.

Honest client: We construct a simulator Sim for an adversary corrupting servers $\{\mathsf{S}_j\}_{j \in \mathcal{J}}$ and clients $\{\mathsf{P}_i\}_{i \in \mathcal{I}}$ as follows:

1. Upon activation by $\mathcal{Z}$, Sim sends $(\mathsf{corrupt}, \mathsf{S}_j)$ to S_j for $j \in \mathcal{J}$ and $(\mathsf{corrupt}, \mathsf{P}_i)$ for $i \in \mathcal{I}$.

2. Upon receiving C from Adv, Sim proceeds as follows. If C is not a valid policy circuit, then Sim sends $(\mathsf{policy}, \mathsf{sid}, \perp)$ to $\mathcal{F}_{\mathsf{abke}}$ and halts as an honest client would, outputting whatever Adv outputs. Otherwise, Sim sends $(\mathsf{policy}, \mathsf{sid}, \mathrm{C})$ to $\mathcal{F}_{\mathsf{abke}}$.

3. Upon receiving $(\mathsf{policy}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{abke}}$, Sim proceeds as follows. It constructs a fake keypair $(pk, sk) \leftarrow \mathsf{GenCert}^*(\mathsf{mSK})$, computes $(pk', sk') \leftarrow \mathsf{Unlink}(pk, sk)$, and sends pk' to Adv.

4. Upon receiving c and GC from Adv, Sim computes $\begin{pmatrix} X_1^0 & \cdots & X_m^0 \\ X_1^1 & \cdots & X_m^1 \end{pmatrix} \leftarrow \mathsf{Dec}^*(sk', c)$, and checks that the garbled circuit sent by Adv indeed garbles C. If so, Sim sets $\mathsf{cheat} := 0$ and otherwise sets $\mathsf{cheat} := 1$. Sim sends $(\mathsf{committed}, \mathsf{sid}, 1, n)$ to Adv.

5. Upon receiving wire labels $\{X_i^b\}$ and randomness r from Adv, Sim proceeds as follows. If $\mathsf{cheat} = 1$ then Sim sends $(\mathsf{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{abke}}$ and outputs whatever Adv outputs. Otherwise, Sim sends $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j)$ to $\mathcal{F}_{\mathsf{abke}}$, receiving either $(\mathsf{completed}, \mathsf{sid}, k)$ or $(\mathsf{completed}, \mathsf{sid}, \perp)$. If Sim receives k then it sends $(\mathsf{reveal}, \mathsf{sid}, 1, Z^1)$ to Adv and otherwise it halts, outputting whatever Adv outputs.

6. Upon receiving $(\mathsf{toss}, \mathsf{sid})$ from Adv, Sim sends $(\mathsf{tossed}, \mathsf{sid}, k)$ to Adv and halts, outputting whatever Adv outputs.

We now show that the ideal and hybrid worlds are computationally indistinguishable. We do so by a series of hybrids.

1. Hybrid_1. Sim acts as an honest client would.

2. Hybrid_2. Upon receiving C from Adv, Sim checks whether C is a valid policy circuit. If so, Sim sends $(\mathsf{policy}, \mathsf{sid}, \mathrm{C})$ to $\mathcal{F}_{\mathsf{abke}}$, and otherwise Sim sends $(\mathsf{policy}, \mathsf{sid}, \perp)$ and halts.

   These two hybrids are perfectly indistinguishable, as the behavior of Sim communicating with $\mathcal{F}_{\mathsf{abke}}$ versus an honest party is exactly the same from the point of view of $\mathcal{Z}$.

3. Hybrid_3. Sim constructs a "fake" keypair $(pk, sk) \leftarrow \mathsf{GenCert}^*(\mathsf{mSK})$ and sends pk to Adv.

   These two hybrids are computationally indistinguishable by the attribute privacy property of the ASE scheme. Namely, if there exists an environment $\mathcal{Z}$ distinguishing these two hybrids then we can construct an adversary Adv' that wins the $\mathsf{Expt}^{\text{att-priv}}_{\pi,\mathsf{Adv}'}(1^n, m)$ experiment as follows.

   Upon receiving PP and mVK, Adv' proceeds as follows. Adv' runs $\mathcal{Z}$, learning all the attribute vectors of each client. For each party that $\mathcal{Z}$ corrupts, Adv' makes the appropriate call to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$. Next, Adv' sends χ, where χ is the attribute vector of the honest client, to $\mathcal{C}$. Otherwise, Adv' runs $\mathcal{Z}$ as normal and outputs whatever $\mathcal{Z}$ outputs.

   Note that if $\mathcal{C}$ chooses $b = 0$ then this is exactly Hybrid_2, whereas if $b = 1$ then this is exactly Hybrid_3. Thus, Adv' succeeds with the same probability that $\mathcal{Z}$ distinguishes the two hybrids, and thus the two hybrids are indistinguishable.

4. Hybrid_4. Sim computes $(pk', sk') \leftarrow \mathsf{Unlink}(pk, sk)$ and sends pk' to Adv.

   These two hybrids are computationally indistinguishable by the unlinkability property of the ASE scheme. Namely, if there exists an environment $\mathcal{Z}$ distinguishing these two hybrids then we can construct an adversary Adv' that wins the $\mathsf{Expt}^{\text{link}}_{\pi,\mathsf{Adv}'}(1^n, m)$ experiment as follows.

   Upon receiving PP and mVK, Adv' proceeds as follows. Adv' runs $\mathcal{Z}$, learning all the attribute vectors of each client. For each party that $\mathcal{Z}$ corrupts, Adv' makes the appropriate call to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$. Next, Adv' sends χ, where χ is the attribute vector of the honest client, to $\mathcal{C}$. Otherwise, Adv' runs $\mathcal{Z}$ as normal and outputs whatever $\mathcal{Z}$ outputs.

   Note that if $\mathcal{C}$ chooses $b = 0$ then this is exactly Hybrid_3, whereas if $b = 1$ then this is exactly Hybrid_4. Thus, Adv' succeeds with the same probability that $\mathcal{Z}$ distinguishes the two hybrids, and thus the two hybrids are indistinguishable.

5. Hybrid_5. Sim uses Dec^* to fully decrypt the ciphertext and uses the decrypted values to check whether the garbled circuit is a correct garbling of C. If so, Sim sets $\mathsf{cheat} := 0$ and otherwise

sets $\mathsf{cheat} := 1$. Now, upon receiving labels and randomness from Adv, if $\mathsf{cheat} = 1$ then Sim sends $(\mathsf{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{abke}}$ and otherwise Sim sends $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j)$ to $\mathcal{F}_{\mathsf{abke}}$.

These two hybrids are computationally indistinguishable by the verifiability property of the garbling scheme.

6. Hybrid_6. If Sim receives $(\mathsf{completed}, \mathsf{sid}, k)$ from $\mathcal{F}_{\mathsf{abke}}$ then it sends $(\mathsf{reveal}, \mathsf{sid}, 1, Z^1)$, where Z^1 is the one-bit output-wire label for the garbled circuit sent by Adv.

   These two hybrids are perfectly indistinguishable due to the use of $\mathcal{F}_{\mathsf{com}}$.

7. Hybrid_7. Sim sends $(\mathsf{tossed}, \mathsf{sid}, k)$ to Adv.

   These two hybrids are perfectly indistinguishable, due to the use of $\mathcal{F}_{\mathsf{cointoss}}$ and the fact that k is chosen uniformly at random by $\mathcal{F}_{\mathsf{abke}}$.

As Hybrid_7 is exactly the same as the simulator in the ideal world, we conclude that the real and ideal worlds are computationally indistinguishable.

Honest server: We construct a simulator Sim for an adversary corrupting servers $\{\mathsf{S}_j\}_{j \in \mathcal{J}}$ and clients $\{\mathsf{P}_i\}_{i \in \mathcal{I}}$ as follows:

1. Upon activation by $\mathcal{Z}$, Sim sends $(\mathsf{corrupt}, \mathsf{P}_i)$ to P_i for $i \in \mathcal{I}$, receiving attribute vectors $\{\chi_i\}_{i \in \mathcal{I}}$.

2. Sim receives $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$ from $\mathcal{F}_{\mathsf{abke}}$, and sends C to Adv.

3. Upon receiving pk' from Adv, Sim acts as an honest server would, sending c and GC to Adv.

4. Upon receiving $(\mathsf{commit}, \mathsf{sid}, 1, Z)$ from Adv, Sim proceeds as follows. If Z equals the one-bit output-wire label of GC, Sim finds some $i \in \mathcal{I}$ such that $\mathrm{C}(\chi_i) = 1$; if no such i exists, Sim outputs fail. Otherwise, it submits $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j, \mathsf{P}_i)$ to $\mathcal{F}_{\mathsf{abke}}$, receiving back $(\mathsf{completed}, \mathsf{sid}, k)$.

   If Z equals the zero-bit output-wire label of GC, Sim finds some $i \in \mathcal{I}$ such that $\mathrm{C}(\chi_i) = 0$; if no such i exists, Sim outputs fail. Otherwise, it submits $(\mathsf{exchange}, \mathsf{sid}, \mathsf{S}_j, \mathsf{P}_i)$ to $\mathcal{F}_{\mathsf{abke}}$, receiving back $(\mathsf{completed}, \mathsf{sid}, \perp)$, and halts, outputting whatever Adv outputs.

   If Z is not equal to *either* the one-bit or zero-bit output-wire label, Sim sends $(\mathsf{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{abke}}$, receiving back $(\mathsf{abort}, \mathsf{sid})$, and halts, outputs whatever Adv outputs.

5. Upon receiving $(\mathsf{toss}, \mathsf{sid})$ from Adv, Sim sends $(\mathsf{tossed}, \mathsf{sid}, k)$ to Adv and halts, outputting whatever Adv outputs.

We now show that the ideal and real worlds are computationally indistinguishable. We do so by a series of hybrids.

1. Hybrid_1. Sim acts as an honest server would.

2. Hybrid_2. Sim receives $(\mathsf{policy}, \mathsf{sid}, \mathsf{S}_j, \mathrm{C})$ from $\mathcal{F}_{\mathsf{abke}}$, and sends C to Adv.

3. Hybrid_3. Sim receives $(\mathsf{commit}, \mathsf{sid}, 1, Z)$ from Adv and proceeds as follows. If Z equals the one-bit output-wire label of the garbled circuit, then Sim finds some $i \in \mathcal{I}$ such that $C(\chi_i) = 1$, outputting fail if no such i exists.

These two hybrids are computationally indistinguishable. They differ only in that in Hybrid_2 the simulator Sim outputs fail if it cannot find an attribute vector χ_i for $i \in \mathcal{I}$ such that $C(\chi_i) = 1$. The only way this could happen is if Adv was able to obtain an output-wire label mapping to 1 while not having credentials for an attribute vector χ such that $C(\chi) = 1$. Note that by the authenticity property of the garbling scheme, Adv must have a set of valid input-wire labels that evaluate to 1. We can use this fact to construct an adversary Adv' that wins the $\mathsf{Expt}^{\text{collude}}_{\pi,\mathsf{Adv}'}(1^n, m)$ experiment.

Upon receiving PP and mVK, Adv' runs $\mathcal{Z}$ and simulates $\mathcal{F}_{\mathsf{setup}}$ to use PP and mVK. For each party corrupted by $\mathcal{Z}$, Adv' makes the appropriate oracle query to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$. Otherwise, Adv' acts as an honest server would. Let pk_χ be the public key sent by Adv. If $\mathsf{Vrfy}(\mathsf{mVK}, pk_\chi)$ fails then Adv' aborts. Otherwise, pk_χ is a valid public key for attribute χ which Adv' knows (as it controls the queries to $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$). Adv' submits pk_χ to $\mathcal{C}$, receiving ciphertext c. Adv' then uses the matching secret key (which it learned as output from the $\mathsf{GenCert}(\mathsf{mSK}, \cdot)$ queries) to decrypt c, learning $\{X_j^{\chi[j]}\}_{j\in[m]}$. It then creates a garbled circuit GC with random labels except those extracted from c. Adv' then sends c and GC to Adv, and then monitors the calls to the random oracle made by Adv until it receives $(\mathsf{commit}, \mathsf{sid}, 1, \cdot)$ from Adv. It takes all random oracle calls of the form $(i\|b\|X_i^b)$, extracts X_i^b, and sends the set $\mathcal{M}$ of all X's to $\mathcal{C}$.

Note that if Adv' succeeds, then it must be the case that $\mathcal{M}$ contains some subset which corresponds to valid plaintext messages for some attribute vector not known to $\mathcal{Z}$, and thus the probability of Adv' succeeding is equal to that of $\mathcal{Z}$ distinguishing, completing the reduction.

4. Hybrid_4. If Z equals the zero-bit output-wire label of the garbled circuit, then Sim finds some $i \in \mathcal{I}$ such that $C(\chi_i) = 0$, outputting fail if no such i exists.

   These two hybrids are computationally indistinguishable using a similar reduction to the collusion experiment as detailed in the previous hybrid.

5. Hybrid_5. If Z is not equal to *either* the one-bit or zero-bit output-wire label, Sim sends $(\mathsf{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{abke}}$.

   These two hybrids are perfectly indistinguishable, as the view from the point of view of $\mathcal{Z}$ is equivalent.

6. Hybrid_6. Sim sends $(\mathsf{tossed}, \mathsf{sid}, k)$ to Adv.

   These two hybrids are perfectly indistinguishable due to the use of $\mathcal{F}_{\mathsf{cointoss}}$ and the fact that k is chosen uniformly at random.

As Hybrid_6 is exactly the same as the simulator in the ideal world, we conclude that the real and ideal worlds are computationally indistinguishable, completing the proof. □

7 ELH Signatures

We introduce the notion of *extractable linearly homomorphic (ELH) signatures* and show an implementation using the Boneh-Lynn-Shacham (BLS) [BLS04] signature scheme. ELH signatures play a central role in our ASE constructions detailed in §8 and §9.

Definition 7.1. (Linearly homomorphic signatures) *Let* $\mathsf{Sig} = (\mathsf{Sign}, \mathsf{Vrfy})$ *be a signature scheme over a space of messages consisting of elements of a group* G *of prime order* q*, with signatures also lying in this group. We say that* Sig *is linearly homomorphic over* G *if for any two elements* $g_1, g_2 \in G$*, it holds that* $\mathsf{Sign}(g_1 g_2) = \mathsf{Sign}(g_1)\mathsf{Sign}(g_2)$*. The scheme is called unforgeable if no probabilistic polynomial-time algorithm given* n *pairs* $(g_i, \mathsf{Sign}(g_i))$ *for random elements* $g_1, \ldots, g_n \in G$ *and an additional random independent element* $g \in G$*, has non-negligible probability to output* $\mathsf{Sign}(g)$*.*

Note that being linearly homomorphic implies that given n signed elements $g_1, \ldots, g_n \in G$, one can compute (without the signing key) the signature of any linear combination (in the exponent) of $g_1, \ldots, g_n$; namely, for any $x_1, \ldots, x_n \in \mathbb{Z}_q$ we have that $\mathsf{Sign}(g_1^{x_1} \cdots g_n^{x_n}) = \mathsf{Sign}(g_1)^{x_1} \cdots \mathsf{Sign}(g_n)^{x_n}$. We note that the requirement of the signatures lying in the same group as the message space is not essential but it simplifies notation by using the same group operation for group elements and signatures, and is a property of our implementation using BLS signatures. This notion can be seen as a one-dimensional case of homomorphic signatures for linear spaces [BFKW09, GKKR10, BF11]. Also note that the unforgeability property holds only with respect to random messages (i.e., random elements in the group).

We now define the property of extractability. It captures the intuition behind linearly homomorphic signatures as allowing limited malleability. That is, anyone can generate signatures on a value g without possessing the signing key as long as it *knows* a representation of g as a linear combination (in the exponent) of previously signed elements. Extractability formalizes this knowledge similarly to existing knowledge extractability notions. In spite of being intuitively appealing we are not aware of this form of homomorphic signatures being defined in prior work.

Definition 7.2. (Extractable linearly homomorphic signatures) *Let* G *be a cyclic group of prime order* q *and* $\mathsf{Sig} = (\mathsf{Sign}, \mathsf{Vrfy})$ *a linearly homomorphic signature scheme over* G*. Consider algorithms that on input* t *random elements* $g_1, \ldots, g_t$ *in* G *and corresponding signatures* $\mathsf{Sign}(g_1), \ldots, \mathsf{Sign}(g_t)$*, output a pair* $(f, \mathsf{Sign}(f))$ *for* $f \in G$ *with non-negligible probability (over the choice of* g_i*s and the algorithm's random coins). We say that* Sig *is an* extractable linearly homomorphic *(ELH) signature scheme if for every polynomial-time algorithm* F *as above there exists another polynomial-time algorithm* F' *for which the following property holds, except for with negligible probability: Let* $\{g_i, \mathsf{Sign}(g_i)\}_{i \in [t]}$ *be an input to* F *on which* F *outputs* $(f, \mathsf{Sign}(f))$*, then on the same inputs (and internal random coins)* F' *outputs a vector* $(f, \mathsf{Sign}(f), x_1, \ldots, x_n)$ *with* $x_i \in \mathbb{Z}_q$ *such that* $f = g_1^{x_1} \cdots g_n^{x_n}$*.*

Interestingly, extractability in linearly homomorphic signatures implies unforgeability as shown next.

Lemma 7.3. *Let* $\mathsf{Sig} = (\mathsf{Sign}, \mathsf{Vrfy})$ *be an ELH signature scheme over a group* G *where the discrete logarithm problem is hard. Then* Sig *is unforgeable.*

Proof. Let F be a polynomial-time algorithm against Sig that is given signatures on elements $g_1, \ldots g_n \in_R G$ and is also given an additional random independent element $g \in G$. Assume F outputs $\mathsf{Sign}(g)$, then by extractability we get values $x_1, \ldots, x_n \in \mathbb{Z}_q$ such that $g = g_1^{x_1} \cdots g_n^{x_n}$. By Lemma 7.4, finding such a representation is infeasible under the hardness of discrete logarithm. □

Lemma 7.4. *Under the hardness of the discrete log over group* G*, given* $g_1, \ldots g_n, g \in_R G \setminus \{1\}$*, it is infeasible to find* $x_1, \ldots, x_n \in \mathbb{Z}_q$ *such that* $g = g_1^{x_1} \cdots g_n^{x_n}$*. Similarly, finding two representations* $\Pi_{i=1}^n g_i^{x_i} = \Pi_{i=1}^n g_i^{y_i}$ *such that there exists an* i *for which* $x_i \neq y_i$ *is also infeasible.*

Proof. Given two random generators $g, h \in G$, we find the discrete log of h with respect to g by setting $g_i = h$ for random $i, 1 \leq i \leq n$, and choosing the other g_i's as known random powers of g. Given values $x_1, \ldots, x_n \in \mathbb{Z}_q$ such that $g = g_1^{x_1} \cdots g_n^{x_n}$ and $x_i \neq 0$, one derives x such that $h = g^x$. □

7.1 Implementation of ELH Signatures

We now demonstrate an implementation of an ELH signature scheme using the Boneh-Lynn-Shacham (BLS) [BLS04] signature scheme, which we first recall.

Boneh-Lynn-Shacham (BLS) signature scheme. The scheme assumes groups (G_1, G_2, G_T) of prime order q with a bilinear pairing $e : G_1 \times G_2 \to G_T$ where the co-CDH assumption holds (i.e., given $g \in G_1, h, h^x \in G_2$, finding g^x is infeasible). The public/private keypair is (h^x, x), where $x \in_R \mathbb{Z}_q$, and $h \in_R G_2$. A signature on message m is computed as $H(m)^x$ where H is a hash function mapping messages to random elements in G_1. Verification of a signature σ on message m under public key $y = h^x$ is performed by checking the following equality: $e(\sigma, h) = e(H(m), y)$.

BLS* signature scheme. We define the BLS* scheme to be the same as BLS but the message space is the group G_1 itself and *no hash function is applied to the messages* (this is sufficient for our application that only requires unforgeability on random group elements).

The following lemma shows that BLS* leads to an implementation of ELH signatures under the t-KEA assumption [BCCT12, BCC+14].

Lemma 7.5. *Under the t-KEA assumption over group G_1, BLS* is an unforgeable extractable linearly homomorphic signature scheme.*

of Lemma 7.5. The scheme is obviously linearly homomorphic over G_1. We show it is extractable under the t-KEA assumption over group G_1. Denote the BLS* signing operation by Sign and let F be an algorithm that on input t random elements $g_1, \ldots, g_t$ in G_1 and corresponding signatures $\mathsf{Sig}(g_i), i = 1, \ldots, t$, outputs a pair $(g, \mathsf{Sign}(g))$ for $g \in G_1$ with non-negligible probability. Since $\mathsf{Sign}(g) = g^x$, we have that F, on input $g_1, \ldots, g_t, g_1^x, \ldots, g_t^x$, outputs a pair (g, g^x) with non-negligible probability. Hence, by t-KEA there exists an extractor F' that on the same input (and coin tosses) of F outputs a vector $(g, g^x, x_1, \ldots, x_n)$ such that $g = g_1^{x_1} \cdots g_n^{x_n}$, except for negligible probability. This is exactly the defining condition for ELH extractability. Unforgeability (over G_1) follows from the extractability property (cf. Lemma 7.3). □

8 ASE Using IBE

We now construct an attribute selective encryption scheme from identity based encryption (IBE) and extractable linearly homomorphic (ELH) signatures. The security of the protocol is based on the security of the underlying IBE and ELH signature schemes. In addition, we require that the master public key of the IBE scheme be from a group so that it can be rerandomized and that the ELH signature scheme works over the same group.

We first give a high level overview of how we use the IBE scheme in our construction. Recall that in an IBE scheme a central authority chooses a master secret key and publishes the correlated master public key. The master public key is used as part of the encryption key for all clients. Each client has an identity which is known to all and in addition each client receives a private secret key

that is computed using the master secret key and its identity. A message is encrypted using the master public key and the identity of the client for whom the message is intended. The client uses its secret key to decrypt.

The first switch that we make in our scheme is that the "identities" are associated with the attributes. Thus, if a client has an attribute then it receives the secret key relating to that attribute. However, that clearly is not sufficient as collusions can take place. A client receiving the secret key for attribute j_1 can collude with another client who has the secret key for j_2, enabling them to decrypt an unauthorized set of messages. Thus, we introduce our second switch which is that the center creates a "personalized" public master key for each client (by choosing a different master secret key) and modifies the secret keys of the client to relate to the personalized public key. Now this additional change prevents the clients from colluding as their secret keys relate to different master public keys. While this is the basic intuition, there are additional details that need to be added to satisfy all the requirements, which we describe below.

To present our scheme, we first recall the general construction of IBE and define its terminology. An IBE scheme is comprised of four parts: setup, key generation, encryption, and decryption.

- $\mathsf{Setup}_{\mathsf{IBE}}(1^n)$: Takes as input the security parameter. We split the setup into two parts: Part 1 outputs the public parameters $\mathsf{PP}_{\mathsf{IBE}}$, and Part 2 outputs the master public key $\mathsf{mpk}_{\mathsf{IBE}}$ and the master secret key $\mathsf{msk}_{\mathsf{IBE}}$. The public parameters are common to all key pairs, and are implictly given to all algorithms below.
- $\mathsf{KeyGen}_{\mathsf{IBE}}(\mathsf{msk}_{\mathsf{IBE}}, \mathsf{ID})$: Takes as input the master secret key and the identity ID of the client, and outputs the client's secret key $\mathsf{sk}_{\mathsf{IBE}}$.
- $\mathsf{Enc}_{\mathsf{IBE}}(\mathsf{mpk}_{\mathsf{IBE}}, m, \mathsf{ID})$: Takes as input the master public key, the plaintext message m, and the identity of a client, and outputs the ciphertext c.
- $\mathsf{Dec}_{\mathsf{IBE}}(c, \mathsf{sk}_{\mathsf{IBE}})$: Takes as input the ciphertext and the client's secret key and outputs the message.

In Appendix A, we provide a full definition of the security of IBE and a description of the Boneh-Franklin IBE scheme [BF01]. Here we informally state that encrypting with IBE is a secure encryption.

We are now ready to present our construction. Our IBE-ASE scheme is defined as follows.

- $\mathsf{Setup}(1^n, m)$:
  1. Run Part 1 of $\mathsf{Setup}_{\mathsf{IBE}}$, receiving $\mathsf{PP}_{\mathsf{IBE}}$.
  2. Choose $2m$ random strings, $g_{j,0}, g_{j,1}$, for $j \in [m]$, where $g_{j,0}$ corresponds to not having attribute j, and $g_{j,1}$ corresponds to having the attribute. These will be the "identities" of the system.
  3. Run a key generation protocol for an ELH signature scheme, receiving and setting mVK to the public verification key and mSK to the secret signing key.
  4. Output $\mathsf{PP} := (\mathsf{PP}_{\mathsf{IBE}}, \{g_{j,0}, g_{j,1}\}_{j\in[m]})$, mVK, and mSK.
- $\mathsf{GenCert}(\mathsf{mSK}, \chi)$:
  1. Run Part 2 of $\mathsf{Setup}_{\mathsf{IBE}}$, creating $\mathsf{msk}_{\mathsf{IBE}}$ and $\mathsf{mpk}_{\mathsf{IBE}}$. Set $pk_\chi := (\mathsf{mpk}_{\mathsf{IBE}}, \sigma = \mathsf{Sign}(\mathsf{mpk}_{\mathsf{IBE}}))$. In what follows we sometimes abuse notation and refer to pk_χ only as the public key and sometimes as both the public key and its signature.
  2. For each attribute $j \in [m]$, call $\mathsf{KeyGen}_{\mathsf{IBE}}$ on input $\mathsf{msk}_{\mathsf{IBE}}$ and identity $g_{j,\chi[j]}$. This returns a secret key sk_j.

3. Set the client's secret key to $\mathsf{sk}_\chi := (\mathsf{sk}_1, \ldots, \mathsf{sk}_m)$.

Note that now we can discard the master secret key $\mathsf{msk}_{\mathsf{IBE}}$. We use the signature to compensate for the fact that $\mathsf{mpk}_{\mathsf{IBE}}$ is not one of the public parameters of the system.

- $\mathsf{Vrfy}(\mathsf{mVK}, pk_\chi)$: Output a bit attesting to the validity of the public key pk_χ by checking the signature σ.
- $\mathsf{Enc}(pk_\chi, \{x_{j,0}, x_{j,1}\}_{j\in[m]})$:
  1. Verify that $\mathsf{Vrfy}(\mathsf{mVK}, pk_\chi) = 1$. If not then abort.
  2. For $j \in [m]$, compute $c_{j,0} \leftarrow \mathsf{Enc}_{\mathsf{IBE}}(pk_\chi, x_{j,0}, g_{j0})$ and $c_{j,1} \leftarrow \mathsf{Enc}_{\mathsf{IBE}}(pk_\chi, x_{j,1}, g_{j1})$.
  3. Output $c := c_{1,0}, c_{1,1}, \ldots, c_{m,0}, c_{m,1}$.
- $\mathsf{Dec}(sk_\chi, c)$: Output $x_j := \mathsf{Dec}_{\mathsf{IBE}}(c_{j,\chi[j]}, \mathsf{sk}_{\chi[j]})$ for all $j \in [m]$.
- $\mathsf{Unlink}(pk_\chi, sk_\chi)$: In our implementation using the Boneh-Franklin IBE scheme, the public key pk_χ has the form g^z for some value z. We implement the unlink operation by raising pk_χ to a random exponent s. All other values are also raised to s, including the ELH signature and every component of the secret key.
- $\mathsf{Project}(pk_\chi, sk_\chi, S)$: The project function in the IBE case is trivial; all that needs to be done is to remove from the secret key the elements whose index is not in the set S. The public key remains the same.
- $\mathsf{GenCert}^*(\mathsf{mSK})$: Exactly as in $\mathsf{GenCert}$, except now the secret key sk contains secret keys $\mathsf{sk}_{j,b}$ for all $j \in [m]$ and $b \in \{0, 1\}$.
  This can be trivially achieved in two manners: first, by giving additional secret keys exactly in the format as given in $\mathsf{GenCert}$ or even simpler by just giving x to the client.
- $\mathsf{Dec}^*(sk, c)$: Same as Dec, except here the full set of $2m$ plaintexts are returned. This is completely straightforward, as the secret key now includes decryption keys for all messages.

Theorem 8.1. *The above scheme is a secure ASE scheme when instantiated with the Boneh-Franklin IBE and an ELH signature scheme. The ELH signature scheme needs to work over the same group G, as defined in Boneh-Franklin.*

Proof. We prove each property in turn.

Collusion Resistance. Given an adversary Adv that can win in $\mathsf{Expt}^{\text{collude}}_{\pi,\mathsf{Adv}}$, we can construct an adversary $\mathcal{B}$ that breaks the underlying IBE scheme.

First, $\mathcal{B}$ chooses an index $i \in [q]$ where q is the number of queries χ that Adv can ask, and an index $\ell \in [m]$. $\mathcal{B}$ will succeed if Adv creates a collusion with client i on attribute ℓ.

$\mathcal{B}$ starts constructing the view for Adv as follows: it receives $(\mathsf{PP}_{\mathsf{IBE}} = (G, G_1, g, H), \mathsf{mpk}_{\mathsf{IBE}} = g^x)$ from $\mathcal{C}$, the challenger of the IBE system. It generates key pair $(\mathsf{mVK}, \mathsf{mSK})$ for the ELH signature scheme, and chooses "identities" of the attributes $g_{1,0}, g_{1,1}, \ldots, g_{m,0}, g_{m,1}$ and sends $(\mathsf{PP}_{\mathsf{IBE}}, \mathsf{mVK}, \{g_{j,b}\}_{j\in[m],b\in\{0,1\}})$ to Adv. It sets pk_i, the public key response for query i that Adv will ask, to be mskIBE, i.e., $pk_i = \mathsf{mpk}_{\mathsf{IBE}} = g^x$. We also call this element g^{x_i}.

For all queries $\chi_{i'}$ for $i' \neq i$, $\mathcal{B}$ runs $\mathsf{GenCert}(\mathsf{mSK}, \chi_{i'})$ and provides Adv with the $pk_{i'} = g^{x_{i'}}$ and $sk_{i'}$.

For the i^{th} query χ, it already set the public above and $\mathcal{B}$ will query the IBE challenger, $\mathcal{C}$, as follows. For each $\chi[j], j \in [1..m]$ it sends as "id" the value $g_{j,\chi[j]}$ and receives the secret IBE key for that identity, $\mathsf{sk}_{\mathsf{IBE}}[j]$. It creates the response to the query χ setting the secret key $sk_i = \mathsf{sk}_{\mathsf{IBE}}[1], \ldots, \mathsf{sk}_{\mathsf{IBE}}[m]$.

Now Adv send to $\mathcal{B}$ its challenge, a public key pk and the signature on this key. From pk, its certificate and due to the ELH signatures $\mathcal{B}$ derives $r_1, \ldots, r_q$ such that $pk = pk_1^{r_1} \cdot \cdots \cdot pk_q^{r_q}$. If $r_i = 0$ $\mathcal{B}$ announces failure as the i^{th} key was not included in pk and thus it will not be able to drive a break for the IBE. This happens with probability $1/q$.

Otherwise, $\mathcal{B}$ prepares its response to Adv in the following way. It chooses messages $x_{j,0}, x_{j1}$ for $j \in [m]$ and calls $\mathsf{Enc}(pk, \{x_{j,0}, x_{j,1}\}_{j\in[m]})$ and receives $c_{1,0}, c_{1,1}, \ldots, c_{m,0}, c_{m,1}$. Recall that $\mathcal{B}$ chose attribute ℓ as the location to use for breaking the IBE scheme and let $b = \chi[\ell]$. Thus, to complete the creation of the challenge $\mathcal{B}$ turns to $\mathcal{C}$ and provides the identity $g_{\ell,1-b}$ and two messages m_0 and m_1. It receives a cipher text c of either m_0 or m_1 denote it by M. Given that this is a Boneh-Franklin IBE c is of the format: $(g^r, M \cdot e(g^{x_i}, H(g_{\ell,1-b}))^r)$.

It creates the cipher for position $c_{\ell,1-b}$ by modifying the first element in the encryption c to be $(g^r)^{1/r_i} = g^{r/r_i}$ the second element remains the same. It substitutes this pair instead of what it computed above.

Adv completes the attack and returns a set of plaintexts as the decryptions of the ciphertexts provided by $\mathcal{B}$. The probability of failure in this part if $1/2m$ that Adv did not decrypt the message in location $\ell, 1-b$.

If all plantexts are known to $\mathcal{B}$ from the set $x_{j,0}, x_{j1}$ for $j \in [m]$ it declares failure.

Otherwise, for the value, D, that is not from the set it computes $D \cdot e(g^r, H(g_{\ell,1-b}))^{R/r_i}$ to receive M. It checks whether $M = m_0$ or m_1. If there are multiple unrecognized values for some reason, then this computation can be repeated for all of them.

The reason that the above computation yields the correct value is that: $y = \Sigma_{i=1}^{q} x_i r_i$. This value is only known implicitly as $\mathcal{B}$ does not know x_i. However, it does know the value $R = \Sigma_{j\neq 1} x_j r_j$. Adv computed the decryption of the message based on the public key $pk = g^y$ whose correlated secret key is $(H(g_{\ell,1-b}))^y$, thus we have

$$\begin{aligned} D &= M \cdot \frac{e(g^{x_i}, H(g_{\ell,1-b}))^r}{e((H(g_{\ell,1-b}))^y, g^{r/r_i})} \\ &= M \cdot \frac{e(g, H(g_{\ell,1-b}))^{rx_i}}{e(g, H(g_{\ell,1-b}))^{(x_i r_i + R)r/r_i}} \\ &= M \cdot \frac{e(g, H(g_{\ell,1-b}))^{rx_i}}{e(g, H(g_{\ell,1-b}))^{(x_i r_i)r/r_i} e(g, H(g_{\ell,1-b}))^{Rr/r_i}} \\ &= \frac{M}{e(g, H(g_{\ell,1-b}))^{Rr/r_i}} = \frac{M}{e(g^r, H(g_{\ell,1-b}))^{R/r_i}}. \end{aligned}$$

We know g^r, R, r_1 and thus can compute the dominator in the above expression as $e(g_{1d}c_1)^{R/r_1} = e(g_{1d}, g)^{r\cdot R/r_1}$.

The probability of success of this process is $1/2mq$.

Attribute Privacy. As the public key pk_χ generated by GenCert is the original master public key of the IBE scheme it is completely independent of χ and thus attribute privacy easily follows.

Unlinkability. For two key pairs $g^x, \mathsf{Sign}(g^x)$ and $g^y, \mathsf{Sign}(g^y)$ and a challenge $g^r, \mathsf{Sign}(g^r)$ it is information theoretically impossible to know whether it is g^x raised to r_1 or g^y raised to r_2 as there exist r_1, r_2 such that $r = x \cdot r_1 = y \cdot r_2$. And this applies to the signatures as well.

Projectability. It is easy to see that projection of a public key for attribute vector χ into a subset of attributes χ' is distributed identically to an output of GenCert for attribute vector χ'.

Committing Encryption. This follows immediately from the fact that the Boneh-Franklin is a committing encryption.

This completes the proof. □

9 ASE using ELH Signatures

We present an instantiation of an attribute-selective encryption (ASE) scheme (cf. §5) based on extractable linearly homomorphic (ELH) signatures (cf. §7). The key generation and certification mechanism uses the homomorphic property of the signatures while their extractability properties are used to prove security. ASE encryption is based on simple ElGamal encryption with the per-attribute encryption keys certified via homomorphic signatures which, in turn, allow for key and signature randomization to support unlinkability. In order to enable projections of public keys that only include a subset of attributes, signatures are applied to the individual per-attribute ElGamal encryption keys. This raises the difficulty of how to bind the different per-attribute signatures together against collusion (mix-and-match) attacks. This is solved by mixing a per-public-key (randomizable) identifier u into all the per-attribute signatures. The resultant scheme is "minimal" in the sense that the removal of any element in the scheme leads to an explicit attack.

The ELH-ASE scheme is defined as follows.

- $\mathsf{Setup}(1^n, m)$: Let $\mathsf{Sig} = (\mathsf{Sign}, \mathsf{Vrfy})$ be an ELH signature scheme over a group G of order q. Setup uses Sig to sign several elements in each client's public key (defined below). Each element type (g, h, u) has a dedicated signature key and there is also a per-attribute signature key to sign elements of the form ue_j. For readability, we denote the above four types of signatures by $\mathsf{Sign}_g, \mathsf{Sign}_h, \mathsf{Sign}_u$ and Sign_j for $j \in [m]$.
  The set of public verification keys for the above signatures form the master verification key mVK and the corresponding secret signing keys form the master secret key mSK.
  We refer to the party running the Setup function as the CA.
- $\mathsf{GenCert}(\mathsf{mSK}, \chi)$: A public key pk_χ associated with an attribute vector $\chi = (\chi_1, \ldots \chi_m)$ is generated as follows:

  1. Choose random elements $g, h, u \in G \setminus \{1\}$ and compute signatures $\mathsf{Sign}_g(g)$, $\mathsf{Sign}_h(h)$, and $\mathsf{Sign}_u(u)$.
  2. Choose $r_1, \ldots, r_m \in_R \mathbb{Z}_q^*$ and set $e_j = g^{r_j}$ if $\chi[j] = 0$ and $e_j = h^{r_j}$ if $\chi[j] = 1$; compute signatures $\mathsf{Sign}_j(ue_j)$ for $j \in [m]$.

  Set the public key pk_χ to $(g, h, u, \{e_j\}_{j\in[m]}, \mathsf{Sign}_g(g), \mathsf{Sign}_h(h), \mathsf{Sign}_u(u), \{\mathsf{Sign}_j(ue_j)\}_{j\in[m]})$ and the secret key sk_χ to $\{r_j\}_{j\in[m]}$.
- $\mathsf{Vrfy}(\mathsf{mVK}, pk_\chi)$: Check that $g, h, u \in G \setminus \{1\}$ and use mVK to check all signatures.
- $\mathsf{Enc}(pk_\chi, \{x_{j,0}, x_{j,1}\}_{j\in[m]})$: For $j \in [m]$, choose $s_j, t_j \in_R \mathbb{Z}_q$ and set $c_{j,0} := \left(g^{s_j}, e_j^{s_j} \cdot x_{j,0}\right)$ and $c_{j,1} := \left(h^{t_j}, e_j^{t_j} \cdot x_{j,1}\right)$. The ciphertext is the sequence $\{(c_{j,0}, c_{j,1})\}_{j\in[m]}$.
  (Note: We assume for simplicity that the random values $x_{j,b}$, that correspond to input wire labels X_j^b in protocol Π_{abke} of Fig. 6.1 are random elements in G (these values are later hashed into strings h_j^b as part of the garbling scheme $\mathcal{G}_{\mathsf{ABKE}}$).)
- $\mathsf{Dec}(sk_\chi, \{(c_{j,0}, c_{j,1})\}_{j\in[m]})$: For $j \in [m]$, set (C_1, C_2) to the pair c_{j,χ_j} and compute $x_{j,\chi_j} := C_2 / {C_1}^{r_j}$.
- $\mathsf{Unlink}(pk_\chi, sk_\chi)$: Choose a random value $r \in \mathbb{Z}_q$ and raise every element of pk to the power

of r; output:

$$pk'_\chi := (g^r, h^r, u^r, \{e_j^r\}_{j\in[m]}, (\mathsf{Sign}_g(g))^r, (\mathsf{Sign}_h(h))^r, \\ (\mathsf{Sign}_u(u))^r, \{(\mathsf{Sign}_j(ue_j))^r\}_{j\in[m]})$$

and $sk'_\chi := \{r \cdot r_j\}_{j\in[m]}$.

- $\mathsf{Project}(pk_\chi, sk_\chi, S)$: Output a new public key by omitting any component e_j for $j \notin S$, and set the corresponding secret key to $\{r_j\}_{j\in S}$.
- $\mathsf{GenCert}^*(\mathsf{mSK})$: Generation of the pair (pk_χ, sk_χ) is the same as for $\mathsf{GenCert}$ except that h is set to g^τ for known $\tau \in_R \mathbb{Z}_q$ and e_j is set to g^{r_j} for all $j \in [m]$ (as in the case $\chi = 0^m$). The secret key sk is comprised of the set $\{r_j\}_{j\in[m]}$ and the value τ. This enables Dec^* as follows.
- $\mathsf{Dec}^*(sk, \{c_{j,0}, c_{j,1}\}_{j\in[m]})$: For $j \in [m]$, set (C_1, C_2) to the pair $c_{j,0}$ and compute $x_{j,0} := C_2/{C_1}^{r_j}$. Then, for $j \in [m]$, set (C_1, C_2) to the pair $c_{j,1}$ and compute $x_{j,1} := C_2/{C_1}^{\tau \cdot r_j}$.

Theorem 9.1. *If* Sig *is an extractable linearly homomorphic signature scheme over a DDH group G then under the DDH and KEA assumptions over G, the ELH-ASE scheme has the properties of collusion resistance, attribute privacy, unlinkability, projectability and committing encryption.*

Proof. We prove each property in turn.

Collusion Resistance. See Lemma 9.4.

Attribute Privacy. Experiment $\mathsf{Expt}^{\text{att-priv}}_{\pi,\mathsf{Adv}}$ requires that the application of $\mathsf{GenCert}$ on any attribute vector χ of length m be indistinguishable from the output of $\mathsf{GenCert}^*$. It is readily seen that public keys produced by $\mathsf{GenCert}$ are identically distributed regardless of the input attribute χ (the latter only influences the e_j basis but these elements are distributed uniformly over G regardless of the basis, hence for any χ). The output of $\mathsf{GenCert}^*$ has this exact same distribution too (e.g., it is identically distributed as $\mathsf{GenCert}(\mathsf{mSK}, \chi = 0^m)$).

Unlinkability. To show that the public key randomization procedure (Unlink) of ELH-ASE is secure (i.e., unlinkable) with respect to experiment $\mathsf{Expt}^{\text{link}}_{\pi,\mathsf{Adv}}$, note first that public keys generated by $\mathsf{GenCert}$ consist of $m + 3$ random independent group generators plus the corresponding ELH signatures on them. The application of the Unlink operation to such a public key, raises all these values to the same random power r. By the DDH assumption, the result is indistinguishable from raising these elements to independent random powers. Hence the resultant randomized elements are indistinguishable from random uniform generators in G, each with a correct signature (that is adjusted through randomization thanks to the homomorphic property) and therefore indistinguishable from an output of $\mathsf{GenCert}$.

Projectability. It is readily verified that the projection of a public key for attribute vector χ into a subset of attributes χ' is distributed identically to an output of $\mathsf{GenCert}$ for attribute vector χ'.

Committing Encryption. The ASE Enc procedure is committing as the ciphertexts commit to the encryption randomness s_j and t_j via the injective mapping g^{s_j}, h^{t_j}.

This completes the proof. □

Proof of collusion resistance. The reminder of the section contains the proof of collusion resistance for the ELH-ASE scheme. It follows from the following three lemmas.

Lemma 9.2. *Under the DDH assumption on group G, given $\mathsf{Enc}(pk_\chi, \{x_{j,0}, x_{j,1}\}_{j\in[m]})$ it is infeasible to decrypt two values $x_{j,0}, x_{j,1}$ for the same j.*

Proof. We prove the claim even for the party that possesses the secret key sk_χ corresponding to the public key $pk_\chi := (g, h, u, \{e_j\}_{j\in[m]}, \mathsf{Sign}_g(g), \mathsf{Sign}_h(h), \mathsf{Sign}_u(u), \{\mathsf{Sign}_j(ue_j)\}_{j\in[m]})$ under which the encryption is applied. Assume $\chi[j] = 0$ (the case $\chi[j] = 1$ is analogous) and let r_j denote the j-th component of sk_χ. In this case we have $e_j = g^{r_j}$. Thus, the encryption of the value $x_{j,1}$ is of the form $(h^k, x_{j,1} \cdot (g^{r_j})^k)$. Now, the tuple $(h, g^{r_j}, h^k, (g^{r_j})^k)$ is a DDH tuple for which the client knows the random generators h, g^{r_j} as well as the value h^k for random (and unknown to the client) k. Hence by DDH, $(g^{r_j})^k$ is indistinguishable from a random group element, thus learning $x_{j,1}$ is infeasible. □

Lemma 9.3. *Given a successful run of the attacker in experiment $\mathsf{Expt}^{\mathrm{collude}}_{\pi,\mathsf{Adv}}$, one extracts the secret key sk corresponding to the attacker's chosen public key pk.*

Proof. Thanks to Lemma 9.2 we know that the attacker cannot succeed in option (b)(2) of experiment $\mathsf{Expt}^{\mathrm{collude}}_{\pi,\mathsf{Adv}}$, hence we assume that Adv wins in option (b)(1).

We first consider the following case. Let pk be a public key provided by the CA (running Setup) for attribute vector χ, namely, $pk := (g, h, u, \{e_j\}_{j\in[m]}, \mathsf{Sign}_g(g), \mathsf{Sign}_h(h), \mathsf{Sign}_u(u), \{\mathsf{Sign}_j(ue_j)\}_{j\in[m]})$ where $e_j = g^{r_j}$ if $\chi[j] = 0$ and $e_j = h^{r_j}$ if $\chi[j] = 1$. Assume $\chi[j] = 0$ and consider an encryption $(C_1, C_2) = (g^s, e_j^s \cdot x)$ of a *random* group element x where $s \in_R \mathbb{Z}_q$ and $x \in_R G$ are chosen by the encryptor (the randomness of the plaintext x is specified by $\mathsf{Expt}^{\mathrm{collude}}_{\pi,\mathsf{Adv}}$.[3] Let D be an algorithm that on inputs pk and (C_1, C_2) outputs x. We claim that under the KEA assumption there exists an algorithm D' that outputs r_j in addition to x.

Indeed, consider the relevant information D has for decryption. It has values g, g^s and $e_j = g^{r_j}$, and it also has the element $e_j^s \cdot x$ which by the randomness of x perfectly hides e_j^s. Yet, D is able to compute e_j^s (since it finds x). The KEA assumption states that for any algorithm D that on input (g, g^s) outputs a pair (f, f^s) there is an algorithm D' that outputs r such that $f = g^r$. In our case we have $f = e_j$ hence the output from D' is $r = r_j$.

We note that the above use of KEA requires g, g^s to be chosen uniformly in G and s to be unknown to D. This is indeed the case here since g is chosen by the CA hence random and s is chosen by the encryptor. Furthermore, note that even if pk includes a value g' not directly chosen by the CA, the fact that the ELH signature $\mathsf{Sign}_g(g')$ verifies, implies that g' was generated as a *known* linear combination of CA-chosen g values. This suffices for the above argument (based on the randomness of g) to work with any g' that is part of a valid pk. We conclude that if an attacker presents a valid pk and can decrypt m values x_{j,b_j} from an ASE ciphertext $\mathsf{Enc}(pk_\chi, \{x_{j,0}, x_{j,1}\}_{j\in[m]})$ (where all $x_{j,b} \in_R G$) then we extract all the secret key values $\{r_j\}_{j\in[m]}$.

We now can apply these considerations to the actual collude experiment $\mathsf{Expt}^{\mathrm{collude}}_{\pi,\mathsf{Adv}}$. In this experiment we get from the attacker a set $\mathcal{M}'$ that includes m plaintexts x_{j,b_j} but we are not told to which ciphertexts they correspond (and the set $\mathcal{M}'$ may contain other values as well). Thus, in order to obtain the values e_j^s (or e_j^t) from which to extract r_j, we need to couple each of the $2m$ ciphertexts in $\mathsf{Enc}(pk_\chi, \vec{x})$ with each of the m' possible plaintexts in $\mathcal{M}'$, and output the corresponding candidate value of e_j^s or e_j^t. In each case that we pair the correct values, we get the

[3] For non-random encrypted values one can hash the ElGamal pad (e.g., e_j^s) under a random oracle and obtain the same extractability result as in this theorem.

correct e_j^s (or e_j^t) and also extract the corresponding r_j (and we can test that this is the correct coupling).

In all, we have that given a successful run of the collude attacker (that generates valid pk and a set $\mathcal{M}'$ as required), we obtain all the secret key values $\{r_j\}_{j\in[m]}$. □

Lemma 9.4. *Let G be a DDH group that satisfies the KEA assumption. Then, any polynomial-time attacker against the ELH-ASE scheme over G has negligible probability to win the collusion experiment* $\mathsf{Expt}^{\text{collude}}_{\pi,\mathsf{Adv}}$.

Proof. Let Adv be an attacker running the collusion experiment $\mathsf{Expt}^{\text{collude}}_{\pi,\mathsf{Adv}}$ against the ELH-ASE scheme and assume it wins the experiment with non-negligible probability. This means that Adv presents a valid public key pk_χ for a vector χ such that no public key was issued by the CA for χ or for any χ' of which χ is a projection, and (by Lemma 9.2) Adv wins the option (b)(1) of the experiment. Hence by Lemma 9.3 one can invoke an extractor to obtain the secret key $sk = \{r_j\}_{j\in[m]}$ corresponding to pk_χ. We will reach a contradiction by showing that the presented public key was either output by the CA (or a projection of such pk) or it is invalid.

Denote by $pk_1, \ldots, pk_n$ the public keys issued by the CA using the Setup function where pk_i is issued for an attribute vector χ_i and its contents are denoted by $pk_i := (g_i, h_i, u_i, \{e_{ij}\}_{j\in[m]}, \mathsf{Sign}_g(g_i), \mathsf{Sign}_h(h_i), \mathsf{Sign}_u(u_i), \{\mathsf{Sign}_j(u_i e_{ij})\}_{j\in[m]})$.

Hereafter, we denote the public key presented by the attacker in $\mathsf{Expt}^{\text{collude}}_{\pi,\mathsf{Adv}}$ by pk_0 and the corresponding attribute vector by χ_0, namely, $pk_0 := (g_0, h_0, u_0, \{e_{0j}\}_{j\in[m]}, \mathsf{Sign}_g(g_0), \mathsf{Sign}_h(h_0), \mathsf{Sign}_u(u_0), \{\mathsf{Sign}_j(u_0 e_{0j})\}_{j\in[m]})$ and assume pk_0 to pass verification. Thus, since all the signatures under Sign_g, $\mathsf{Sign}_h, \mathsf{Sign}_u$, and Sign_j for $j \in [m]$, are valid, then by the extractability property of the ELH signature scheme, one can extract from pk_0 values x_i, y_i, z_i, s_{ij} such that:

$$g_0 = \prod_{i=1}^{n} g_i^{x_i} \quad h_0 = \prod_{i=1}^{n} h_i^{y_i} \quad u_0 = \prod_{i=1}^{n} u_i^{z_i}$$

and

$$u_0 e_{0j} = \prod_{i=1}^{n} (u_i e_{ij})^{s_{ij}} \tag{1}$$

where e_{0j} has the form ${g_0}^{r_{0j}}$ or ${h_0}^{r_{0j}}$ depending on whether $\chi_0[j]$ is 0 or 1, respectively. We assume for concreteness that the base is g_0; the other case is analogous. In either case, by Lemma 9.3 we can extract the values $\{r_{0j}\}_{j\in[m]}$.

Thus we can write

$$u_0 e_{0j} = u_0 g_0^{r_{0j}} = \prod_{i=1}^{n} u_i^{z_i} \prod_{i=1}^{n} g_i^{x_i r_{0j}}. \tag{2}$$

Combining equations (1) and (2) we get

$$\prod_{i=1}^{n} u_i^{s_{ij}} \prod_{i=1}^{n} e_{ij}^{s_{ij}} = \prod_{i=1}^{n} u_i^{z_i} \prod_{i=1}^{n} g_i^{x_i r_{0j}}$$

and thanks to Lemma 7.4 we get that it must be that the exponents on the u_i values are the same in the two sides of the equation, namely, $s_{ij} = z_i$ for $i \in [n]$ and for all js. Removing the u_i factors

from the equation and replacing s_{ij} with z_i we get

$$\prod_{i=1}^{n} e_{ij}^{z_i} = \prod_{i=1}^{n} g_i^{x_i r_{0j}}.$$

Let $S_j = \{\chi_i[j] \neq \chi_0[j]\}_{i\in[n]}$ and let $\bar{S}_j$ denote its complement. We can then write the last equation as

$$\prod_{i\in\bar{S}_j} g_i^{r_{ij}z_i} \prod_{i\in S_j} h_i^{r_{ij}z_i} = \prod_{i=1}^{n} g_i^{x_i r_{0j}}.$$

Due to Lemma 7.4 we get that for $i \in S_j$ the exponents z_i must be 0.

We now claim that for every i there must be a j for which $i \in S_j$ and therefore $z_i = 0$ for all $i \in [n]$. This implies that $u_0 = 1$ in contradiction to the assumption that pk_0 was a valid public key. To see that for every i there must be a j for which $i \in S_j$, or equivalently that $\chi_i[j] \neq \chi_0[j]$, note that if there was an i such that for all j, $\chi_i[j] = \chi_0[j]$, then we would have that χ_0 is same as a previously certified attribute vector, and hence pk_0 would not be a successful forgery. □

10 Performance

As the bottleneck in terms of computation is the pairings and exponentiations (versus garbling and evaluating the policy), we now give a concrete count of the number of pairings and exponentiations required for each of the two ASE schemes presented in §8 and §9.

- *IBE-based scheme (§8).* For concreteness we calculate the cost using the Boneh-Franklin IBE scheme [BF01].
  - The client computes two exponentiations to randomize both its "master public key" and its associated signature.
  - The server computes two pairings to verify the signature of the client's "master public key". To encrypt $2m$ messages, the server computes $2m$ pairings and $2m$ exponentiations.
  - The client computes m pairings and exponentiations to decrypt m messages.
- *ELH signature-based scheme (§9).*
  - The client computes a total of $6 + 2m$ exponentiations to randomize both the basis and its associated signature (6) and the public keys and their associated signatures ($2m$).
  - The server computes $3 + m$ signature verifications, which requires $6 + 2m$ pairings. To encrypt the $2m$ messages it computes $4m$ exponentiations.
  - The client computes m exponentiations to decrypt m messages.

We note several important points regarding the performance of the ELH signature-based scheme. First, the scheme requires the client to only compute exponentiations as opposed to pairings. This could be meaningful in a setting where the client is a small computing device. Second, the server can batch multiple signature verifications from different clients. The CA's signature keys for g, h, u and the attributes are the same for all clients. Using techniques of Ferrara et al. [FGHP09] for batching pairing-based signatures can help us achieve better amortized run-times.

Implementation and results. We implemented the scheme described in Figure 6.1 using the ELH signature-based ASE scheme (cf. §9) utilizing all the optimizations mentioned above. We

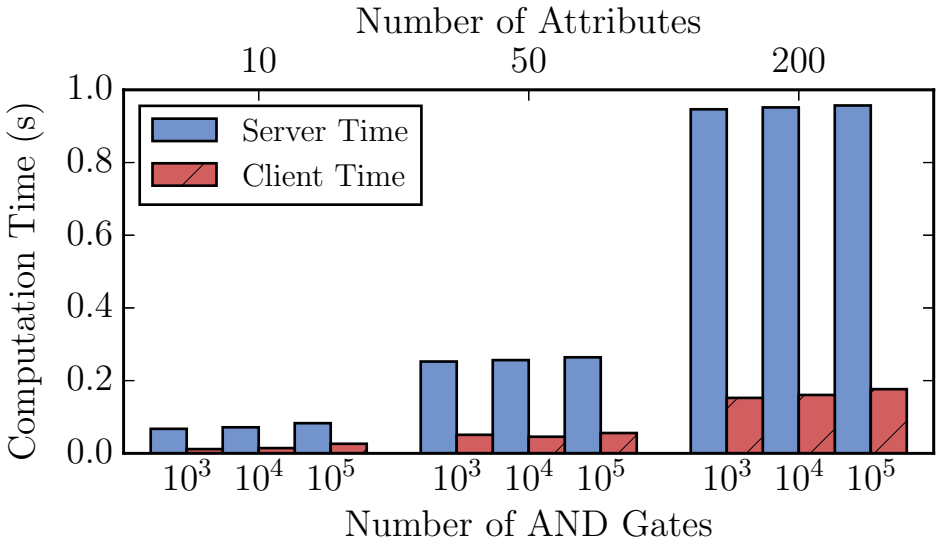

Figure 10.1: Computation time of the server and client for various numbers of attributes and sizes of the policy. The bottom x-axis gives the number of gates in the policy, the top x-axis gives the number of attributes, and the y-axis gives the computation time in seconds.

instantiate the coin-tossing and commitment functionalities using SHA-1, and use the privacy-free garbling technique of Zahur et al. [ZRE15]. The code as well as all the scripts for generating our experimental results is available at `https://github.com/amaloz/abke`.

All experiments were conducted on an Intel Core i5-4210H CPU running at 2.90 GHz. We use the RELIC library [AG] for pairings, using the BN-P256 curve, and `libgarble` [Mal] as our garbled circuit library. On our benchmarking machine, pairings take roughly 1.88 ms and exponentiations in G_1 take roughly 160 μs (76 μs when using preprocessing). All experiments were run over localhost; however, to emulate a WAN environment we used the `tc` command in Linux to set the latency to 33 ms (the average latency in the United States [lat]) and the bandwidth to 200 Mbps. For each measurement, we ran 10 iterations of 10 runs, taking the mean of the medians from each run.

Figures 10.1 and 10.2 show the results of our experiments. We varied the number of attributes m between 10, 50, and 200, and varied the size of the policy (comprised of only AND gates) between 1,000, 10,000, and 100,000. Figure 10.1 depicts the *computation* time of the server and client, whereas Figure 10.2 depicts the *communication* time. We also list the number of bits sent by the server and client in Figure 10.2. As we can see, the computation time is fairly consistent for a fixed m, but grows as m increases. This validates our claim that the pairings and exponentiations account for most of the overhead as opposed to the garbling and evaluating of the policy. The computation time varies from 67 ms for the server and 11 ms for the client for a 1,000 gate policy with 10 attributes, to 957 ms for the server and 176 ms for the client for a 100,000 gate policy with 200 attributes.

Looking at the case of a 100,000 gate policy with 200 attributes (see also Table 10.4), we see that most of the overhead on the server side comes from verifying the public key sent by the client (857 ms), due to the $2m$ pairings needed. The next largest operation is encryption, which accounts for 82 ms. Meanwhile, garbling the policy takes only 5 ms. Regarding the client, the costliest

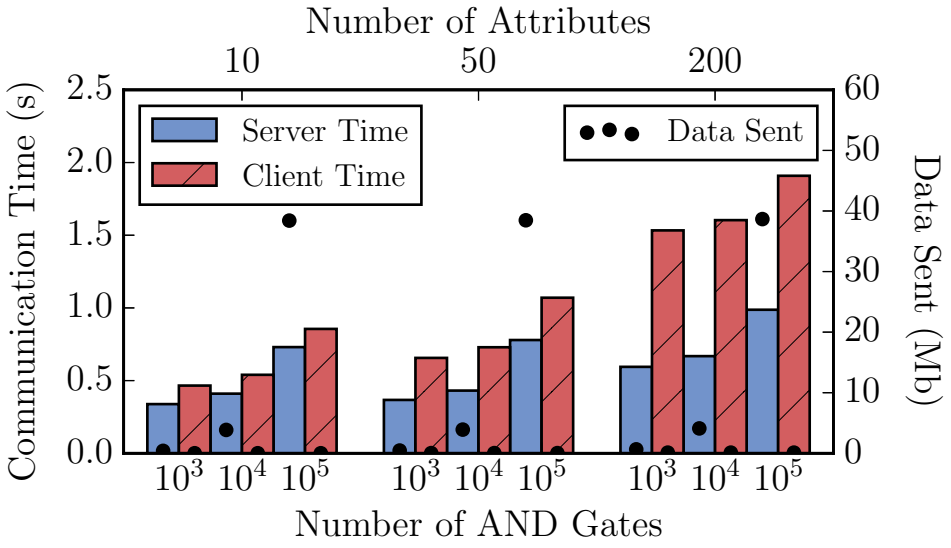

Figure 10.2: Communication time of the server and client for various numbers of attributes and sizes of the policy circuit. The bottom x-axis gives the number of gates in the policy, the top x-axis gives the number of attributes, the left y-axis gives the computation time (in seconds), and the right y-axis gives the number of bits sent (in Mb).

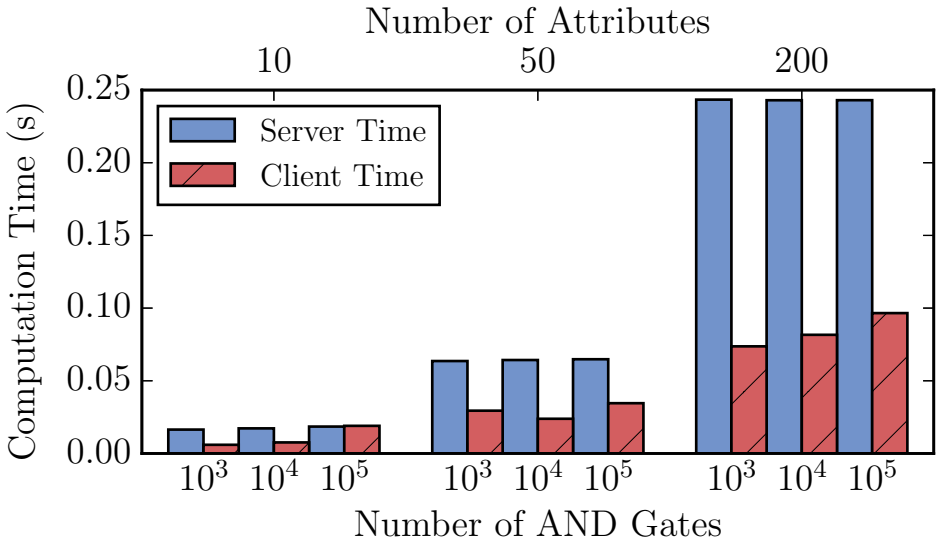

Figure 10.3: *Optimized* computation time (i.e., pushing the cost of randomizing the public key and garbling the policy to an offline stage, along with batching of the key verification) of the server and client for various numbers of attributes and sizes of the policy. The bottom x-axis gives the number of gates in the policy, the top x-axis gives the number of attributes, and the y-axis gives the computation time in seconds.

operation is checking that the encryption sent by the server is correct, which requires re-encrypting the m unopened wire-labels (42 ms), followed by randomizing its public key (78 ms). Decryption

Step	S	S [opt]	P	P [opt]
2 (Gb)	5 ms	—	—	—
3 (Unlink)	—	—	78 ms	—
4 (Vrfy)	857 ms	159 ms	—	—
4 (Enc)	82 ms	82 ms	—	—
5 (Dec)	—	—	28 ms	28 ms
5 (Ev)	—	—	3 ms	3 ms
6 (commit)	—	—	<1 ms	<1 ms
7 (Enc)	—	—	42 ms	42 ms
7 (Ve)	—	—	8 ms	8 ms
8 (cointoss)	<1 ms	<1 ms	<1 ms	<1 ms
Total	944 ms	241 ms	159 ms	81 ms

Table 10.4: Breakdown of server (S) and client (P) computation times for the various steps of Π_{abke} for a 100,000 gate policy with 200 attributes. [opt] denotes the optimized computation time (i.e., pushing computation to an offline stage and batching verification). See Figure 6.1 for a description of each step. The total cost is slightly less than that reported elsewhere due to rounding errors and not accounting for initialize/cleanup steps.

Operation	Cycles
BLS* sign	522,767
BLS* verify	12,316,919
BLS* batch verify	22,635,625

Table 10.5: Benchmarking BLS* signing and verification, along with the batch verification approach of Ferrara et al. [FGHP09] for *ten* messages.

is relatively cheap, requiring 28 ms. Meanwhile, evaluating the garbled circuit takes 3 ms, again demonstrating that the garbled circuit is not the bottleneck (at least with regards to computation).

Looking at the communication time (cf. Figure 10.2), we see that as both the number of attributes and number of gates grows so does the running time. Most of the server's communication time is spent sending the garbled circuit, whereas most of the client's time is spent receiving the garbled circuit and the ciphertext, this latter case due to the client blocking while the server verifies the (randomized) public key. We note that our network bandwidth of 200 Mbps is pessimistic, and running our protocol on Amazon EC2 or other networks with 1 Gbps bandwidth will all but eliminate the communication overhead of sending/receiving the garbled circuit (e.g., when running over localhost, the communication time is essentially the time spent blocking waiting for the other party to complete some computation).

Note that with regards to computation, most of the expensive operations (such as randomizing and verifying the public key) can be either done offline or batched. Thus, we also calculated an *optimized* computation time; see Figure 10.3. In these experiments, we ignore the cost of the client randomizing its public key and the server garbling its policy, as both of these can be done in an offline stage. To account for the batching optimization, we implemented and benchmarked the batching techniques of Ferrara et al. [FGHP09], see Table 10.5. We see a roughly 5.4× improvement when batch verifying ten messages. Thus, in our experiments we model a server operating over ten clients at a time by dividing the public key verification time by 5.4. We see upwards of a 4.4× and

2× improvement in running time for the server and client, respectively. This makes sense in light of the fact that randomizing and verifying the public key are the two most expensive operations.

A policy cost example. In light of the above results, we provide a rough calculation of the cost of a realistic policy, where the client succeeds if its geolocation (x_U, y_U) is within distance d of the server's location (x_S, y_S). Such a policy requires computing $(x_U - x_S)^2 + (y_U - y_S)^2 <_? d^2$. Using a 64-bit input and the CBMC circuit compiler [HFKV12], we can compile this function as a circuit containing 20,000 gates. Thus, as demonstrated by our performance results, the cost of the corresponding garbled circuit would be unnoticeable relative to the public key operations required by the server and client. In contrast, an ABE-based solution would require converting the policy circuit into a (very large) formula, and performing pairings proportional to its size, which is not practical in most settings.

Acknowledgments

This work was supported by the Office of Naval Research (ONR) contract number N00014-14-C-0113.

This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

Work of Alex J. Malozemoff conducted in part with Government support through the National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFG 168a, awarded by DoD, Air Force Office of Scientific Research.

References

[ABC] ABC4Trust EU project. https://www.abc4trust.eu.

[AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *TCC*, 2007.

[AG] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Crypto*, 2009.

[BCC+14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. Cryptology ePrint Archive, Report 2014/580, 2014.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, 2012.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *Crypto*, 2001.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Eurocrypt*, 2011.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC 2009*, 2009.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, 2012.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[Bra00] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000.

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

[CCGS10] Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. Credential authenticated identification and key exchange. In *Crypto*, 2010.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In *Crypto*, 2015.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In *Crypto*, 2016. To appear.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Eurocrypt*, 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Crypto*, 2004.

[FGHP09] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In *CT-RSA*, 2009.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Eurocrypt*, 2015.

[GBG10] M. Choudary Gorantla, Colin Boyd, and Juan Manuel González Nieto. Attribute-based authenticated key exchange. In *ACISP*, 2010.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *Crypto*, 2013.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *PKC*, 2010.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, 2006.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Asiacrypt*, 2010.

[GS14] Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. In *Indocrypt*, 2014.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In *Crypto*, 2015.

[HFKV12] Andreas Holzer, Martin Franz, Stefan Katzenbeisser, and Helmut Veith. Secure two-party computations in ANSI C. In *CCS*, 2012.

[ide] Identity mixer. http://idemix.wordpress.com.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *CCS*, 2013.

[lat] Global IP network latency. http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html. Retrieved February 8, 2016.

[Mal] Alex J. Malozemoff. `libgarble`: A garbled circuit library based on JustGarble. https://github.com/amaloz/libgarble.

[SAH16] Yusuke Sakai, Nuttapong Attrapadung, and Goichiro Hanaoka. Attribute-based signatures for circuits from bilinear map. In *PKC*, 2016.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In *Eurocrypt*, 2005.

[upr] Microsoft U-Prove. http://www.microsoft.com/uprove.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, 2011.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Eurocrypt*, 2015.

A IBE Security

We only require that the IBE is secure under chosen-plaintext attacks. Security for IBE under CPA is formalized as a regular encryption experiment, with the addition that the adversary is allowed to ask for key pairs associated with any identity it wishes. Furthermore, the adversary outputs a pair of (equal-length) messages *as well as an identity*, and one of the messages is encrypted and returned to the adversary. The adversary wins if it guesses which message was encrypted and it did not ask for the key pair of the given identity. Formally, we define the following experiment:

Experiment $\mathsf{Expt}^{\text{CPA-IBE}}_{\pi,\mathsf{Adv}}(1^n)$**:**

1. $(\mathsf{mpk}_{\mathsf{IBE}}, \mathsf{msk}_{\mathsf{IBE}}) \leftarrow \mathsf{Setup}(1^n)$
2. $(m_0, m_1, \mathsf{ID}) \leftarrow \mathsf{Adv}^{\mathsf{KeyGen}_{\mathsf{IBE}}(\mathsf{msk}_{\mathsf{IBE}}, \cdot)}(\mathsf{mpk}_{\mathsf{IBE}})$
3. Choose random $b \leftarrow \{0,1\}$ and compute $c \leftarrow \mathsf{Enc}_{\mathsf{IBE}}(\mathsf{mpk}_{\mathsf{IBE}}, m_b, \mathsf{ID})$
4. $b' \leftarrow \mathsf{Adv}^{\mathsf{KeyGen}_{\mathsf{IBE}}(\mathsf{msk}_{\mathsf{IBE}}, \cdot)}(c)$
5. The output of the experiment is 1 (and Adv wins) *if and only if* $b' = b$.

Definition A.1. *An identity-based encryption scheme* π *is* CPA-secure *if for every probabilistic-polynomial time adversary* Adv *there exists a negligible function* μ *such that for every* n*:*

$$\Pr\left[\mathsf{Expt}^{\text{CPA-IBE}}_{\pi,\mathsf{Adv}}(1^n) = 1\right] \leq \frac{1}{2} + \mu(n).$$

We say that π *is* group-based *if the public parameters* $\mathsf{PP}_{\mathsf{IBE}}$ *include the definition of a group* $\mathbb{G}$ *of order* q *with generator* g*, the master secret key is a random value* $\mathsf{msk}_{\mathsf{IBE}} \in \mathbb{Z}_q$ *and the master public key is the group element* $\mathsf{mpk}_{\mathsf{IBE}} = g^{\mathsf{msk}_{\mathsf{IBE}}} \in \mathbb{G}$*. We say that a group-based IBE is* CPA-secure for random identities *if*

$$\Pr\left[\mathsf{Expt}^{\text{RAND-IBE}}_{\pi,\mathsf{Adv}}(1^n) = 1\right] \leq \frac{1}{2} + \mu(n).$$

Boneh-Franklin IBE scheme [BF01]. The IBE scheme includes the selection of:

1. The public groups G with generator g and G_1 both of order q.
2. A random private master-key $\mathsf{msk}_{\mathsf{IBE}} = x \in \mathbb{Z}_q^*$.
3. A public key $\mathsf{mpk}_{\mathsf{IBE}} = g^x$.
4. A hash function H.

The secret key creation for the user with identity ID is as follows: $\mathsf{sk}_{\mathsf{IBE}} = (H(\mathsf{ID}))^x$. For encryption, given message m, the ciphertext c is obtained as follows:

1. Choose random $r \in Z_q^*$.
2. $c = (g^r, m \cdot e(g^x, (H(\mathsf{ID}))^r))$.

For decryption, given ciphertext $c = (u, v)$ we have $m = v/e(\mathsf{sk}_{\mathsf{IBE}}, u)$.