Toward Hierarchical Identity-Based Encryption

Jeremy Horwitz[1] and Ben Lynn[2]

[1] Stanford University, Stanford, CA 94305, USA
horwitz@cs.stanford.edu
[2] Stanford University, Stanford, CA 94305, USA
blynn@cs.stanford.edu

Abstract. We introduce the concept of hierarchical identity-based encryption (HIBE) schemes, give precise definitions of their security and mention some applications. A two-level HIBE (2-HIBE) scheme consists of a root private key generator (PKG), domain PKGs and users, all of which are associated with primitive IDs (PIDs) that are arbitrary strings. A user's public key consists of their PID and their domain's PID (in whole called an address). In a regular IBE (which corresponds to a 1-HIBE) scheme, there is only one PKG that distributes private keys to each user (whose public keys are their PID). In a 2-HIBE, users retrieve their private key from their domain PKG. Domain PKGs can compute the private key of any user in their domain, provided they have previously requested their domain secret key from the root PKG (who possesses a master secret). We can go beyond two levels by adding subdomains, subsubdomains, and so on. We present a two-level system with total collusion resistance at the upper (domain) level and partial collusion resistance at the lower (user) level, which has chosen-ciphertext security in the random-oracle model.

1 Introduction

Shamir asked for an identity-based encryption (IBE) cryptosystem in 1984 [9], but a fully-functional IBE scheme was not found until recent work by Boneh and Franklin [1] and Cocks [4]. Recall that an IBE scheme is a public-key cryptosystem where any arbitrary string is a valid public key. The corresponding private keys must be computed by a trusted third party called the private key generator (PKG) (who possesses a master secret). Users of the system request their private key from the PKG.

We note that the public key infrastructure associated with standard public-key cryptosystems also includes a trusted third party (in the form of a root certificate authority) and allows a hierarchy of certificate authorities [12]: the root certificate authority can issue certificates for other certificate authorities, who in turn can issue certificates for users in their respective domains.

The original system of Boneh and Franklin does not allow for such structure. However, a hierarchy of PKGs is desirable in an IBE system, as it greatly reduces the workload on master server(s) and allows key escrow at several levels. For instance, if the users of the system are employees of corporations, then it is natural to want each corporation to be able to generate the private keys for their employees, so that employees request their keys from their corporation, rather than the top-level PKG. Only corporations request their domain secret (and only once per corporation) from the top-level PKG. This is the idea behind a hierarchical IBE (HIBE) system. In particular, this is an example of a two-level HIBE (2-HIBE) scheme. (The advantage of an HIBE system over standard PKI is that senders can derive the recipient's public key from their address without an online lookup.)

More precisely, there are three types of entities in a 2-HIBE scheme. There is the root PKG, who possesses a master key. In the upper level, there are domain PKGs, who can request their domain key from the root PKG. Lastly, there are users, who can request private keys from their domain PKG. Each user and each domain has a primitive ID (PID), which is an arbitrary string. (If Alice works for Company.com and her email address is `alice@company.com`, her PID is `alice` and her company's PID is `company.com`.) The public key of a user consists of a tuple of PIDs: the PID of the user and the PID of the user's domain (this public key is also called the user's address) and, as with IBE systems, it is clear that a sender can derive the receiver's public key offline. We can generalize to HIBE schemes with more levels by allowing subdomains, subsubdomains, and so on.

Another application for HIBE systems is generating short-lived keys for portable computing devices. Suppose Alice is planning to embark on a week-long business trip and wants to be able read her encrypted mail while on the road. However, she is also worried that her laptop may be stolen or otherwise compromised, so she does not want to simply copy her private key to the laptop. This dilemma is readily solved with a 2-HIBE system: this time, the upper level consists of people, such as Alice, and the lower level consists of dates, and when an arbitrary user Bob wants to send a message to Alice he uses the tuple of Alice's PID and the PID for the current date as her address. Alice can generate (for example) seven days' worth of keys (from her private key that she has previously requested from a PKG) and transfer these to her laptop. Now if the laptop is compromised, the damage is limited. We note that collusion at the bottom level is not an issue, as Alice will only put a small number of keys on her laptop. This problem can also be solved with a standard (non-hierarchical) IBE scheme by having Alice run her own IBE system [1], but in this case Bob must get Alice's system parameters before he can communicate with her.

In this paper, we give formal security definitions that can model plausible real-life attack scenarios on HIBE systems. In addition to chosen-ciphertext attacks, we must also worry about attacks involving collusion by entities on arbitrary levels. In our example above, for instance, if the domain PKG of one corporation A colludes with employees of another corporation B, they should not be able to decrypt messages of other employees of corporation B (or of any other corporation C, for that matter). In general, an adversary should not be able to decrypt a message encrypted for a particular user in a particular domain (and subdomain, subsubdomain, etc.), even if they have access to the private key of every other user and of every other domain (and subdomain, subsubdomain, etc.), in addition to information obtained from a decryption oracle.

We present a 2-HIBE scheme with total collusion resistance at the upper level and partial collusion resistance at the lower level. (This limitation does not affect its applicability to the above laptop example.) In terms of the corporate setting, even if an arbitrary number of corporations collude, the master secret is safe, but, at the lower level, if more than certain number of employees of a corporation C collude, they can expose C's private key.

Our system requires a bilinear map with certain properties. A suitable map can constructed from the Weil pairing (which is described in [1]). Its performance is sufficiently fast for practical purposes, provided the number of colluding parties allowed in the lower level is not too large. (Its running time and key size involve a term linear in this number.) Additionally, we can employ the same techniques used with the Boneh-Franklin IBE scheme to split secrets across several servers and achieve robustness for free.

2 Definitions

An identity-based encryption scheme (IBE) is specified by four randomized algorithms: Setup, KeyGen (called $\mathsf{Extract}$ in [1]), $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$. In brief, Setup generates system parameters that are publicly released and a master key that is given to the PKG only; KeyGen is run by the PKG to generate private keys corresponding to a given primitive ID (PID); $\mathsf{Encrypt}$ encrypts a message using a given PID (PIDs are public keys); and $\mathsf{Decrypt}$ decrypts a ciphertext given a private key. We shall always take the message space to be $\mathcal{M} = \{0,1\}^m$.

These algorithms must satisfy the standard consistency constraint, namely, when d is the private key generated by algorithm KeyGen when it is given the PID A as the public key, then

$$\forall M \in \mathcal{M} : \mathsf{Decrypt}(\mathsf{params}, A, C, d) = M \ ,$$

where $C = \mathsf{Encrypt}(\mathsf{params}, A, M)$.

An ℓ-HIBE has a family of ℓ key-generation algorithms (KeyGen_i for $1 \leq i \leq \ell$) instead of just one, and public keys are now ℓ-tuples of PIDs instead of just a single PID.

Definition 1. *A* primitive ID (PID) *is an arbitrary string, i.e., an element of* $\{0,1\}^*$.

Definition 2. *An* address *is an ℓ-tuple of* PID*s.*

An address fully specifies a user's public key.

Definition 3. *A* prefix address (or prefix) *is an i-tuple of* PID*s for some* $0 \leq i \leq \ell$. *A prefix address* $\langle S_1, \ldots, S_i \rangle$ *is said to be a* prefix *of the prefix address* $\langle T_1, \ldots, T_j \rangle$ *if* $i \leq j$ *and* $S_a = T_a$ *for* $1 \leq a \leq i$.

Notice that addresses also happen to be prefix addresses.

Definition 4. *For a non-negative integer ℓ, an* ℓ-level hierarchical identity-based encryption scheme (ℓ-HIBE) *is specified by $\ell + 3$ randomized algorithms:* Setup, KeyGen_i *(for $1 \leq i \leq \ell$),* $\mathsf{Encrypt}$, *and* $\mathsf{Decrypt}$*:*

Setup**:** *Input: security parameter $k \in \mathbb{Z}$. Output: system parameters* params *and a master key* mk_ϵ *(which we also call the level-0 key).*

KeyGen_i**:** *(for $1 \leq i \leq \ell$)***:** *Input:* params, $\mathsf{mk}_{\langle S_1,\ldots,S_{i-1}\rangle}$ *(a level-$(i-1)$ key), and an i-tuple of* PID*s (a prefix address). Output:* $\mathsf{mk}_{\langle S_1,\ldots,S_i\rangle}$ *(a level-i key).*

$\mathsf{Encrypt}$**:** *Input:* params*, an address, and a message. Output: a ciphertext.*

$\mathsf{Decrypt}$**:** *Input:* params*, an address, a ciphertext, and a private key* $\mathsf{mk}_{\langle S_1,\ldots,S_\ell\rangle}$*. Output: the corresponding plaintext.*

These algorithms must satisfy the standard consistency constraint, namely, when d is the private key generated by algorithm KeyGen_ℓ *when it is given the address N as the public key, then*

$$\forall M \in \mathcal{M} : \mathsf{Decrypt}\big(\mathsf{params}, N, C, \mathsf{mk}_{\langle S_1,\ldots,S_\ell\rangle}\big) = M \ ,$$

where $C = \mathsf{Encrypt}(\mathsf{params}, N, M)$*.*

Remark 1. For certain values of ℓ, an HIBE is the same as other familiar structures:

- When $\ell = 0$, this definition captures the essence of public-key encryption schemes: the level-0 key corresponds to a private key and the params correspond to the public key (the address is empty when calling $\mathsf{Encrypt}$; each system is associated with only one private key/public key pair).
- When $\ell = 1$, we have a definition of a standard IBE.

2.1 Security

In order to cover realistic attacks, we assume that an attacker may be able obtain private keys at any level except for the master secret, and extend the standard model of chosen-ciphertext security accordingly. We note that if the master secret is compromised, the effects are at least as disastrous as when the root certificate authority is compromised in a public-key cryptosystem. Thus we assume that the precautions taken to guard the master secret are similar to those taken to guard a root certificate authority in real life (e.g., secret splitting, tamper-resistant hardware), rendering it unassailable. Consider the following game played by two parties, an adversary and a challenger:

1. The challenger runs the Setup algorithm (for a given security parameter k) and gives params to the adversary. It does not divulge mk_ϵ.

2. The adversary submits any number of decryption and/or key-generation queries adaptively (i.e., each query may depend on the replies to previous queries). In a decryption query, the adversary sends a ciphertext and an address and is given the corresponding plaintext under the unique key associated with that address (assuming the ciphertext and address are valid). For a key-generation query, the adversary submits any prefix address $\langle S_1, \ldots, S_i \rangle$ (for some $1 \leq i \leq \ell$), and is told the output K_i (where K_j is defined to be $\mathsf{KeyGen}_j(K_{j-1}, \langle S_1, \ldots, S_j \rangle)$ for $0 < j \leq i$ and K_0 is the key returned by Setup).
   In other words, not only can the adversary learn the decryption of any chosen ciphertext, it can also obtain the key corresponding to any prefix address.
3. The adversary then outputs any two plaintexts $M_0, M_1 \in \mathcal{M}$ and any address N on which it wishes to be challenged, subject to the restriction that no prefix of N has been queried in the previous step.
4. The challenger picks $b \in \{0, 1\}$ randomly and computes the ciphertext $C = \mathsf{Encrypt}(\mathsf{params}, N, M_b)$. It then sends the challenge C to the adversary.
5. The adversary again issues any number of decryption and/or key-generation queries adaptively, except that it now may not ask for the key corresponding to any prefix of N or for the plaintext corresponding to C under the private key corresponding to N.
6. The adversary outputs $b' \in \{0, 1\}$, and wins if $b = b'$.

We call such an adversary an ID-CCA attacker.

Definition 5. *We define an HIBE to be* secure against adaptive chosen-ciphertext attack (ID-CCA) *if no polynomially-bounded adversary has a non-negligible advantage in the above game, that is, for any polynomial f and for any probabilistic polynomial-time algorithm $\mathcal{A}$, $Adv(\mathcal{A}) := \left|\Pr[b = b'] - \frac{1}{2}\right|$ is less than $1/f(k)$. (The probability is over the random bits used by the two parties.)*

Finding even a 2-HIBE that satisfies this security requirement remains an open problem. We describe a 2-HIBE that is secure provided the adversary is limited to n KeyGen_2 queries within its domain (for a given n; unlimited KeyGen_1 queries are allowed). In other words, our system resists arbitrary collusion at the domain level, but resists only limited collusion at the user level.

We will also utilize a weaker notion of security in intermediate steps of our proofs. Consider another game played by two parties, an adversary and a challenger:

1. The challenger runs the Setup algorithm (for a given security parameter k) and gives params to the adversary.
2. The adversary submits some number of key-generation queries adaptively, that is, for each query, the adversary submits any prefix address $\langle S_1, \ldots, S_i \rangle$ (for some $1 \leq i \leq \ell$), and is told the output K_i (where K_j is defined to be $\mathsf{KeyGen}_j(K_{j-1}, \langle S_1, \ldots, S_j \rangle)$ for $0 < j \leq i$ and K_0 is the key returned by Setup).

3. The adversary then outputs any address N on which it wishes to be challenged, subject to the restriction that no prefix of N has been queried in the previous step.
4. The challenger picks a message $M \in \mathcal{M}$ randomly and computes $C = \mathsf{Encrypt}(\mathsf{params}, N, M)$. It sends the challenge C to the adversary.
5. The adversary again issues some number of key-generation queries adaptively, except that it now may not ask for the key corresponding to any prefix of N.
6. The adversary outputs some message $M' \in \mathcal{M}$, and wins if $M = M'$.

We call such an adversary an ID-OWE attacker.

Definition 6. *We define an HIBE to be a* one-way identity-based encryption scheme (ID-OWE) *if no polynomially-bounded adversary has a non-negligible advantage in the above game.*

Both these definitions are generalizations of definitions given by Boneh and Franklin [1].

3 An HIBE Resistant against Domain Collusion

We present a two-level system resistant to collusion at the domain level. The system is based on bilinear forms between two prime-order groups.

3.1 The BDH Assumption

We briefly review definitions given by Boneh and Franklin [1, 2].

Definition 7. *Let G_1, G_2 be groups with prime order q. Then we say a map $e\colon G_1 \times G_1 \to G_2$ is* bilinear *if, for all $g, h \in G_1$ and $a, b \in \mathbb{F}_q$, we have $e\big(g^a, h^b\big) = e(g, h)^{ab}$.*

Definition 8. *The* Bilinear-Diffie-Hellman problem *(BDH) for a bilinear function $e\colon G_1 \times G_1 \to G_2$ such that $|G_1| = |G_2| = q$ is prime is defined as follows: given $g, g^a, g^b, g^c \in G_1$, compute $e(g, g)^{abc}$, where g is a generator and a, b, c are randomly chosen from $\mathbb{F}_q$. An algorithm is said to solve the BDH problem with an advantage of ε if*

$$\Pr\big[\mathcal{A}\big(g, g^a, g^b, g^c\big) = e(g, g)^{abc}\big] \geq \varepsilon\ .$$

Definition 9. *A randomized algorithm $\mathcal{IG}$ that takes as input a security parameter $k \in \mathbb{Z}$ (in unary) is a* BDH parameter generator *if it runs in time polynomial in k and outputs the description of two groups G_1, G_2 and a bilinear function $e\colon G_1 \times G_1 \to G_2$. We further require that the groups have prime order (which we call q), and denote the output of the algorithm by $(G_1, G_2, e) = \mathcal{IG}\big(1^k\big)$.*

Definition 10. *We say that* $\mathcal{IG}$ satisfies the BDH assumption *if no probabilistic polynomial-time algorithm* $\mathcal{A}$ *can solve BDH (for* $\mathcal{IG}(1^k)$*) with non-negligible advantage.*

For the remainder of the paper we make use of some fixed BDH parameter generator $\mathcal{IG}$ that satisfies the BDH assumption, and use the symbols G_1, G_2, e, q to represent the constituents of its output. Boneh and Franklin [1] also give details on how to implement such a generator (their system also required one), based on the Weil pairing. (In their construction, G_1 is a group of points on a certain elliptic curve and G_2 is a certain subgroup of $\mathbb{F}_{p^2}^{\times}$, for some prime p.)

This assumption was implicitly used by Joux [7] to build a one-round three-party Diffie-Hellman protocol. Other constructions also require the BDH assumption ([8, 10, 11]). Additionally, a bilinear function is needed in a recently described short signature scheme [3].

3.2 A Game Transformation

The BDH assumption is closely tied to the CDH assumption. Recall that the CDH problem asks for g^{ab} given g, g^a, g^b, whereas the goal in the CDH problem is to compute $e(g,g)^{abc} = e(g^{ab}, g^c)$ given g^c in addition to g, g^a, g^b. This similarity between the BDH and CDH assumptions naturally leads to the following transformation on games:

Definition 11. *Using the notation of the previous section, suppose* $\mathcal{G}$ *is a game where the goal of the adversary is to compute a particular element* $g \in G_1$*. Then the* e-transformation of $\mathcal{G}$ *is the same game as* $\mathcal{G}$ *except now the adversary is also given a random* $h \in G_1$ *and the adversary's goal is to compute* $e(g, h)$*.*

We can transform assumptions by applying this transformation to the underlying game. For example, we obtain the BDH assumption (associated with a particular e) when we apply this transformation to the CDH assumption.

It is possible to formulate our assumptions differently: we could have started with assuming that e is a bilinear function such that if a game $\mathcal{G}$ is hard then its e-transformation is also hard. This would simplify our exposition (for example, we need only assume the CDH problem is hard, as that implies that the BDH problem is hard). However, such an assumption is really an abstract description of a class of assumptions, and we prefer the readability gained by relying on a small number of concrete assumptions instead.

Clearly, if an adversary can win a game $\mathcal{G}$, then it can easily win the e-transformation of $\mathcal{G}$. The converse is far from clear.

We shall see that transformed assumptions are required to show that schemes are ID-OWE; without transformation, the assumptions are more natural, but we can only show that an adversary cannot recover a user's private key.

3.3 Linear e-One-Way Functions

We now build up to the definition of a linear e-one-way function, from which one could build an HIBE. We then construct a function that is weaker than e-one-way that will allow for efficiently building a 2-HIBE which is secure against any collusion at the domain level and limited collusion at the user level.

Suppose that we have a function $h\colon G \times X \to G_1$, where G and G_1 are groups, G is of prime order p, X is a set, and $h(g^a, x) = h(g, x)^a$ for all $g \in G$, $x \in X$, $a \in \mathbb{F}_p$.

Definition 12. *The elements $x, x_1, x_2, \ldots, x_n \in X$, $a \in \mathbb{F}_p$, and a generator $g \in G$ are chosen at random. Given x, g, and $\langle x_i, h(g^a, x_i)\rangle$ for $i = 1, 2, \ldots, n$, the problem of computing $h(g^a, x)$ is called the* linear one-way problem (of size n)*.*

Definition 13. *We say that h is a* linear one-way function *if no probabilistic polynomial-time (in n and $\log p$) algorithm can solve the linear one-way problem of any size.*

Remark 2. For example, if DDH is hard in $\mathbb{F}_{p^2}^{\times}$, the Weil pairing is an example of a linear one-way function. More generally, bilinear functions that satisfy BDH give rise to families of linear one-way functions. For example, suppose we have $(G_1, G_2, e) = \mathcal{IG}\big(1^k\big)$. Then fix a generator $g \in G_1$ and consider the function $f_g\colon G_1 \to G_2$ defined by $f_g(g_1) := e(g, g_1)$. Now, f_g is one-way, assuming DDH is hard in G_2. To see this, assume that f_g is easy to invert; DDH in G_2 can be solved as follows: given $x, x^a, x^b, x^c \in G_2$ we find their inverses y, y_a, y_b, y_c respectively, and check if $e(y, y_c) = e(y_a, y_b)$. We note that if $\mathcal{IG}$ is constructed as described by Boneh and Franklin, then G_2 is a subgroup of $\mathbb{F}_{p^2}^{\times}$, a group in which DDH is thought to be hard. (It is also possible to construct elliptic curves where the q-torsion points are contained in $\mathbb{F}_p$ for some large prime q. Inverting the Weil pairing on these curves is equivalent to breaking DDH in $\mathbb{F}_p$.) More generally, this is why the relationship between a game and its e-transform appears to be highly nontrivial: if an algorithm $\mathcal{A}$ could win a game $\mathcal{G}$, given an algorithm $\mathcal{B}$ that wins the e-transform of $\mathcal{G}$, then $\mathcal{A}$ is an algorithm that can invert f_g.

Now suppose that $(G_1, G_2, e) = \mathcal{IG}\big(1^k\big)$. Then the e-transformation of the linear one-way problem is called the linear e-one-way problem. (In this problem, we are also given g^r for some random $r \in \mathbb{F}_p$ (in addition to x, g, and $\langle x_i, h(g^a, x_i)\rangle$ for $i = 1, 2, \ldots, n$) and now the goal is to compute $e(h(g^a, x), g^r)$.)

Definition 14. *If no probabilistic polynomial-time algorithm can solve the linear e-one-way problem of any size, then we say that h is a* linear e-one-way function*.*

If we knew how to construct linear e-one-way functions, we could construct an HIBE scheme as follows:

Setup: Input: $k \in \mathbb{Z}$. Run $\mathcal{IG}(1^k)$ and set (G_1, G_2, e) to be the output. Construct a linear e-one-way function $h\colon G_1 \times \mathbb{F}_q \to G_1$. Choose a random $a \in \mathbb{F}_q$ and a random generator $g \in G_1$. Pick cryptographically-strong hash functions $H_1\colon \{0,1\}^* \to G_1$, $H_2\colon \{0,1\}^* \to \mathbb{F}_q$, and $H_3\colon G_2 \to \{0,1\}^m$ (for some m).
Output: $\mathsf{mk}_\epsilon := a$, and $\mathsf{params} := \langle G_1, G_2, e, g, g^a, H_1, H_2, H_3\rangle$.

KeyGen_1: Input: a prefix address $\langle S\rangle$ (the domain name).
Output: $\mathsf{mk}_{\langle S\rangle} := H_1(S)^a \in G_1$.

KeyGen_2: Input: an address $\langle S, T\rangle$ (S is the domain PID and T is the user PID).
Let $\mathsf{mk}_{\langle S\rangle} \in G_1$ be the domain key.
Output: $k = \mathsf{mk}_{\langle S,T\rangle} := h\big(\mathsf{mk}_{\langle S\rangle}, H_2(S \,\|\, T)\big) \in G_2$.

$\mathsf{Encrypt}$: Input: params, $N = \langle S, T\rangle$ (S is the recipient domain's PID and T is the recipient user's PID), and M.
Pick a random $r \in \mathbb{F}_q$.
Output: $C = \langle g^r, M \oplus H_3(s)\rangle$, where $s := e(h(H_1(S), H_2(S \,\|\, T)), g)^r$.

$\mathsf{Decrypt}$: Input: params, $N = \langle S, T\rangle$, a ciphertext $C = \langle g^r, V\rangle$, and a user's private key $k := \mathsf{mk}_{\langle S,T\rangle} \in G_1$.
Output: $M = V \oplus H_3(e(k, g^r))$.

It can be shown that this scheme is ID-OWE. By applying the Fujisaki-Okamoto [6] transformation, we obtain a scheme which is ID-CCA. Though finding a linear e-one-way function h remains an open problem, we are able to construct an h such that the linear e-one-way problem for a fixed n is hard, giving rise to a 2-HIBE system that is resistant to (unlimited) domain-level collusion and can tolerate up to n-party user-level collusion. We describe this in the following section. Briefly, we will define $h\colon G_1^{n+1} \times \mathbb{F}_q \to G_1$ (for some n; q is the prime order of G_1) as $h((g_0, g_1, \ldots, g_n), d) := g_0^{d^0} g_1^{d^1} \cdots g_n^{d^n}$. We then have a linear function h such that, given g and n pairs $\langle x_i, h(g, x_i)\rangle$, it appears hard to determine $\langle x', h(g, x')\rangle$ for any other x'.

3.4 Our Domain-Collusion Resistant Scheme

Let n denote the amount of collusion that we are willing to tolerate at the user level.

Setup: Input: $k \in \mathbb{Z}$. Run $\mathcal{IG}(1^k)$ and set (G_1, G_2, e) to be the output. Choose a random $a \in \mathbb{F}_q$ and a random $g \in G_1$. Pick cryptographically-strong hash functions $H_1\colon \{0,1\}^* \to G_1^{n+1}$, $H_2\colon \{0,1\}^* \to \mathbb{F}_q$, and $H_3\colon G_2 \to \{0,1\}^m$ (where $\mathcal{M} = \{0,1\}^m$ is the message space). For the security proof, we view the hash functions as random oracles.
Output: $\mathsf{mk}_\epsilon := a$ and $\mathsf{params} := \langle G_1, G_2, e, g, g^a, H_1, H_2, H_3\rangle$.

KeyGen_1: Input: a prefix address $\langle S\rangle$ (the domain name). Let $\langle g_0, g_1, \ldots, g_n\rangle = H_1(S)$ (so each g_i lies in G_1).
Output: $\mathsf{mk}_{\langle S\rangle} := \langle g_0^a, g_1^a, \ldots, g_n^a\rangle \in G_1^{n+1}$.

KeyGen_2: Input: an address $\langle S, T\rangle$ (S is the domain PID and T is the user PID).
Set $d := H_2(S \| T)$ (which is an element of $\mathbb{F}_q$).
Let $\mathsf{mk}_{\langle S\rangle} = \langle g_0^a, g_1^a, \ldots, g_n^a\rangle \in G_1^{n+1}$ be the domain key.
Output: $k = \mathsf{mk}_{\langle S,T\rangle} := \prod_{i=0}^{n} g_i^{ad^i} \in G_1$.

$\mathsf{Encrypt}$: Input: params, $N = \langle S, T\rangle$ (S is the recipient domain's PID and T is the recipient user's PID), and a message M.
Set $\langle g_0, g_1, \ldots, g_n\rangle := H_1(S)$. Set $d := H_2(S \| T)$.
Pick a random $r \in \mathbb{F}_q$. Then compute $w := e\left(\prod_{i=0}^{n} g_i^{d^i}, g^a\right)^r \in G_2$.
Output: the ciphertext $\langle g^r, M \oplus H_3(w)\rangle$.

$\mathsf{Decrypt}$: Input: params, $N = \langle S, T\rangle$, a ciphertext $C = \langle U, V\rangle$, and a private key $k = \mathsf{mk}_{\langle S,T\rangle} \in G_1$.
Output: $M = V \oplus H_3(e(k, U))$.

The scheme is consistent because, by the bilinearity of e, we have $e(k, U) = e\left(\prod_{i=0}^{n} g_i^{d^i}, g^a\right)^r$, when $U = g^r$.

3.5 Proof of Security

Recall that we are restricting the adversary to at most n KeyGen_2 queries from the same domain.

Theorem 1. *Suppose $\mathcal{A}$ is an ID-OWE attacker of our cryptosystem with an advantage of ε. Then, if we model H_1, H_2, and H_3 as random oracles, there exists an algorithm $\mathcal{B}$ that can solve the BDH problem with an advantage of $\varepsilon / \left(2(Q_{K_1} + 2Q_{K_2})Q_{H_1}\binom{Q_{H_2}}{n}\mathbf{e}\right)$, where Q_{K_i} is the total number of* KeyGen_i *queries, Q_{H_i} is the number of H_i queries issued by $\mathcal{A}$, and $\mathbf{e}$ is the base of the natural logarithm.*

Proof. The proof of the theorem is broken into several lemmata. In Lemma 1, we show that an attacker $\mathcal{B}$, whose KeyGen queries are restricted to only KeyGen_2 queries from the same domain as the challenge address, is essentially as strong as an arbitrary attacker $\mathcal{A}$. We do so in a manner similar to that used in the analysis of the Boneh-Franklin scheme [1], which is itself partly based on a technique of Coron [5]. In Lemma 2, we define the Bilinear Polynomial Diffie-Hellman (BPDH) game, and give a reduction from the attack by the $\mathcal{B}$ described above to an attack by (an attacker) $\mathcal{C}$ on the BPDH game. Lastly we produce a reduction from an attack by $\mathcal{C}$ on the BPDH game to an attack by $\mathcal{D}$ on the BDH problem. The combination of the three lemmata leads immediately to the theorem. $\square$

Lemma 1. *Suppose there exists an ID-OWE attacker $\mathcal{A}$ with an advantage of ε. Let Q_i be a bound on the number of H_i queries made by $\mathcal{A}$ (for $i = 1, 2$). If we model H_1 as a random oracle, then there exists an ID-OWE attacker $\mathcal{B}$ with an advantage of $\varepsilon/\mathbf{e}Q$, where $Q = Q_1 + 2Q_2$, whose key-generation queries are all* KeyGen_2 *queries from the same domain as the challenge domain.*

Proof. After receiving the system parameters, $\mathcal{B}$ passes them on to $\mathcal{A}$. Without loss of generality, we may assume that every key-generation query for a prefix address (domain name) $\langle S \rangle$ or address $\langle S, T \rangle$ has been preceded by an H_1 (domain-level) hash query on the domain PID S. We may also assume that $\mathcal{A}$ issues an H_1 hash query on the challenge domain PID before revealing it.

We will need some auxiliary functions and global variables:

Initially, L is an empty list that will hold information on $\mathcal{B}$'s responses to H_1 queries, and $s_{\text{CHALLENGE}}$ is a string that is set to a special value NULL. Additionally, we will use a unique value REAL (not in G, $\mathbb{F}_q$, etc.) in the proof.

When $\mathcal{A}$ issues an H_2 query for an address $\langle S, T \rangle$ (i.e., a hash query on $S \| T$), $\mathcal{B}$ returns $H_2(S \| T)$.

When $\mathcal{A}$ issues an H_1 query on a domain PID S, $\mathcal{B}$ runs the following algorithm:

1. If L contains a tuple whose first element is S, then
   (a) If L contains $\langle S, r_0, r_1, \ldots, r_n \rangle$, then return $\langle g^{r_0}, g^{r_1}, \ldots, g^{r_n} \rangle$.
   (b) If L contains $\langle S, \text{REAL} \rangle$, then return $H_1(S)$.
2. Otherwise, flip a coin that takes the value 1 with probability p and 0 otherwise (p will be determined later).
   (a) If COIN $= 1$, then pick random $r_0, r_1, \ldots, r_n \in \mathbb{F}_q$. Insert the tuple $\langle S, r_0, r_1, \ldots, r_n \rangle$ into L, and return $\langle g^{r_0}, g^{r_1}, \ldots, g^{r_n} \rangle$.
   (b) Otherwise, COIN $= 0$. In this case, insert $\langle S, \text{REAL} \rangle$ into L and return $H_1(S)$.

Since we are modelling H_1 as a random oracle, $\mathcal{A}$ cannot distinguish between this simulation and the real H_1.

When $\mathcal{A}$ issues a KeyGen_1 query on a prefix address (domain name) $\langle S \rangle$, $\mathcal{B}$ runs the following algorithm:

By assumption, $\mathcal{A}$ has already issued an H_1 query for S.
1. If $\langle S, r_0, r_1, \ldots, r_n \rangle$ is on the list L, return $\langle g^{ar_0}, g^{ar_1}, \ldots, g^{ar_n} \rangle$.
2. Otherwise, $\langle S, \text{REAL} \rangle$ appears on L: output FAILURE and halt.

When $\mathcal{A}$ issues a KeyGen_2 query on an address $\langle S, T \rangle$, $\mathcal{B}$ runs the following algorithm:

Again by assumption, a hash query on S has already been issued. Let $d = H_2(S \| T)$.
1. If L contains $\langle S, r_0, r_1, \ldots, r_n \rangle$, return $\left\langle g^{ar_0 d^0}, g^{ar_1 d^1}, \ldots, g^{ar_n d^n} \right\rangle$.
2. Otherwise, $\langle S, \text{REAL} \rangle \in L$:
   (a) If $s_{\text{CHALLENGE}} = S$, then $\mathcal{B}$ issues the KeyGen_2 query (recall that $\mathcal{B}$ is allowed to do this for the challenge domain).
   (b) If $s_{\text{CHALLENGE}} \neq$ NULL then $\mathcal{B}$ outputs FAILURE and halts.
   (c) Otherwise, $\mathcal{B}$ sets $s_{\text{CHALLENGE}} := S$ and issues the KeyGen_2 query.

Eventually, $\mathcal{A}$ outputs a challenge address $\langle S, T\rangle$. If $\langle S, \text{REAL}\rangle \notin L$, then output FAILURE. If $\langle S, \text{REAL}\rangle \in L$ and $s_{\text{CHALLENGE}} \neq \text{NULL}$ and $s_{\text{CHALLENGE}} \neq S$, then output FAILURE. Otherwise (when $\langle S, \text{REAL}\rangle \in L$ and ($s_{\text{CHALLENGE}} = \text{NULL}$ or $s_{\text{CHALLENGE}} = S$)), set $s_{\text{CHALLENGE}} := S$.

The next round of queries is handled in the same manner as in the first round.

Finally, $\mathcal{A}$ will output a guess M and halt; then $\mathcal{B}$ outputs M and halts. Clearly, if $\mathcal{A}$ is successful, then so is $\mathcal{B}$.

Recall that Q_1 is the number of KeyGen_1 queries. Then the probability that FAILURE is not output during such a query is at least p^{Q_1} (it is sufficient to have $\text{COIN} = 1$ for each query).

Recall that Q_2 is the number of KeyGen_2 queries. In the worst case, for every KeyGen_2 query on an address $\langle S, T\rangle$, L contains $\langle S, \text{REAL}\rangle$, and, once $s_{\text{CHALLENGE}}$ has been set, any other value for S will cause failure. So the probability that failure is not output during KeyGen_2 queries is bounded from below by p^{Q_2-1}.

After $\mathcal{A}$ outputs the challenge address, the probability that $\langle S, \text{REAL}\rangle$ is on the list L is $1-p$, and the probability $s_{\text{CHALLENGE}} = S$ or $s_{\text{CHALLENGE}} = \text{NULL}$ is at least p^{Q_2}. (In the worst case, every KeyGen_2 query is in a different domain, and, trivially, the probability that $s_{\text{CHALLENGE}} = S$ or $s_{\text{CHALLENGE}} = \text{NULL}$ is no less than the probability that $s_{\text{CHALLENGE}}$ remains NULL.)

Let $k = Q_1 + 2Q_2 - 1$. Then $p^k(1-p)$ is a lower bound on the probability that a failure state is not reached. It is minimized when $p = k/(k+1)$, which makes the probability of not reaching a failure state bounded from below by $1/\mathbf{e}(k+1)$.

With $Q = Q_1 + 2Q_2$, we see that $\mathcal{B}$ has an advantage of at least $\varepsilon/\mathbf{e}Q$. $\square$

Definition 15. *The* Computational Polynomial Diffie-Hellman (CPDH) game (of degree n) for a function $H\colon X \to G_1$*, where* X *is a set, is the following game:*

A polynomial $f(x) = c_0 + c_1 x + \cdots + c_n x^n$ with coefficients in $\mathbb{F}_q$ is chosen at random. An element a is chosen at random from $\mathbb{F}_q$.

The attacker is given $g, g^a, g^{c_0}, g^{c_1}, \ldots, g^{c_n}$ and $d \in \mathbb{F}_q$.

Then the attacker picks any $s \in X$, and learns $g^{af(H(s))}$. This step is repeated up to n times. (The attack may be adaptive.)

Lastly, the attacker wins if it can output the value of $g^{af(d)}$.

Remark 3. For $n = 0$, this reduces to the CDH problem. (The adversary is not allowed to make any queries.)

Definition 16. *The* Bilinear Polynomial Diffie-Hellman (BPDH) game (of degree n) for a function $H\colon X \to G_1$ *is the e-transformation of the corresponding CPDH game, i.e., it is the same as the previous game except that the attacker is also given* g^r *for some random* $r \in \mathbb{F}_q$ *and now the attacker's goal is to compute* $e(g,g)^{arf(d)}$.

Remark 4. For $n = 0$ this reduces to the BDH problem.

Lemma 2. *Suppose there exists an ID-OWE attacker* $\mathcal{B}$ *with an advantage of* ε *whose key-generation queries are always* KeyGen_2 *queries of addresses from* *the same domain as the challenge domain, and furthermore, $\mathcal{B}$ makes at most n such queries. ($\mathcal{B}$ makes* no KeyGen_1 *queries.) Then, there exists an attacker $\mathcal{C}$ that can win the BPDH game for H_2 with an advantage of $\varepsilon/(2Q)$, where Q is a bound on the number of H_1 queries that $\mathcal{B}$ makes.*

Proof. Again we may assume that any key-generation query for an address is preceded by a hash query for that address, and that before the challenge address is output, a hash query for the challenge address will have been issued. We may also assume that each query (for any hash function) is distinct (since previous results can simply be cached). Also, without loss of generality, we may assume that $\mathcal{B}$ makes *exactly* n KeyGen_2 queries.

The algorithm $\mathcal{C}$ is given as input $g, g^a, g^{c_0}, g^{c_1}, \ldots, g^{c_n}, g^r, d$ (using the notation employed in the description of the BPDH game; its goal is to compute $e(g,g)^{arf(d)}$). $\mathcal{C}$ begins by giving $\mathcal{B}$ the system parameters g, g^a.

There is a list L that is used to store H_3 queries and is initially empty.

$\mathcal{C}$ picks a random i between 1 and Q. On the ith H_1 query for S (that $\mathcal{B}$ makes), $\mathcal{C}$ sets $s_{\text{CHALLENGE}} := S$ and returns $\langle g^{c_0}, g^{c_1}, \ldots, g^{c_n}\rangle$. For all other H_1 queries, $\mathcal{C}$ returns $H_1(S)$. Since we are modelling H_1 as a random oracle, the algorithm $\mathcal{B}$ cannot distinguish between this simulation of H_1 and the real H_1.

When $\mathcal{B}$ issues an H_2 query for $\langle S, T\rangle$, $\mathcal{C}$ returns $H_2(S \| T)$.

When $\mathcal{B}$ issues an H_3 query for $s \in G_2$, $\mathcal{C}$ returns $H_3(s)$, and inserts $\langle s, H_3(s)\rangle$ into L.

When $\mathcal{B}$ issues a KeyGen_2 query on $\langle S, T\rangle$, if $S \neq s_{\text{CHALLENGE}}$, then $\mathcal{C}$ outputs FAILURE and halts. Otherwise, $\mathcal{C}$ issues a query for $g^{af(H_2(S\|T))}$ and returns the result to $\mathcal{B}$.

Eventually, $\mathcal{B}$ gives the challenge address $N = \langle S, T\rangle$ to $\mathcal{C}$. If $S \neq s_{\text{CHALLENGE}}$, then $\mathcal{C}$ outputs FAILURE and halts. Otherwise, $\mathcal{C}$ chooses a random $R \in \{0,1\}^k$, and $\mathcal{C}$ gives the ciphertext $C := \langle g^r, R\rangle$ to $\mathcal{B}$.

The next round of queries is handled in the same manner as in the first round.

Eventually, $\mathcal{B}$ outputs its guess M and halts. Then, the algorithm $\mathcal{C}$ looks for a tuple of the form $\langle s, M \oplus R\rangle$ in L; if it cannot find it, $\mathcal{C}$ outputs FAILURE. If $\mathcal{B}$ is successful (i.e., if $\mathsf{Encrypt}(\mathsf{params}, N, M) = C$), then $M = R \oplus H_3(s)$, where $s = e(g,g)^{arf(H_2(S\|T))}$. If $\mathcal{C}$ finds the tuple in L, $\mathcal{C}$ then knows $n+1$ values of the function $x \mapsto e(g,g)^{arf(x)}$, so $\mathcal{C}$ can compute $e(g,g)^{arf(d)}$ using Lagrange interpolation.

Notice that, since H_3 is a random oracle, if $\langle s, H_3(s)\rangle$ is not found in L, then the decryption of the ciphertext C is independent of the knowledge $\mathcal{B}$ accumulated from its various queries, which means that $\mathcal{B}$ succeeds in this case with probability $1/2^k$.

The probability of success is $(1/Q)(1 - 1/2^k)$ (i must be guessed correctly), which is at least $1/(2Q)$. □

Lemma 3. *Suppose there exists an algorithm $\mathcal{C}$ that can win the BPDH game for a function $H\colon X \to G_1$ with an advantage of ε and suppose H may be modelled as a random oracle. Then there exists an attacker $\mathcal{D}$ that can solve the BDH problem with an advantage of $\varepsilon/\binom{Q}{n}$, where Q is a bound on the number of H queries that $\mathcal{C}$ makes.*

Proof. We may assume that $Q \geq n$, that all H queries are distinct (previous results can be cached), and that a query for $g^{af(H(s))}$ implies that $\mathcal{C}$ has already issued a query for $H(s)$.

The algorithm $\mathcal{D}$ is given g, g^x, g^y, g^z for randomly chosen $x, y, z \in \mathbb{F}_q$ (its goal is to compute $e(g,g)^{xyz}$). Set $y_0 := y$. There is a list L that holds responses to H queries that is initially empty. $\mathcal{D}$ picks random $a_0, a_1, \ldots, a_n, y_1, y_2, \ldots, y_n \in \mathbb{F}_q$. $\mathcal{D}$ then solves the system of equations

$$\begin{aligned}
g_0^{a_0^0} \cdot g_1^{a_0^1} \cdots g_n^{a_0^n} &= g^{y_0} \\
g_0^{a_1^0} \cdot g_1^{a_1^1} \cdots g_n^{a_1^n} &= g^{y_1} \\
&\vdots \\
g_0^{a_n^0} \cdot g_1^{a_n^1} \cdots g_n^{a_n^n} &= g^{y_n}
\end{aligned}$$

for the g_i. If we define the matrix A by $A_{ij} := a_i^j$, then, with high probability, the a_i are distinct (so A is a Vandermonde matrix), thus guaranteeing a unique solution for the g_i.

Then, $\mathcal{D}$ hands $\mathcal{C}$ the input $\langle g, g^x, g_0, g_1, \ldots, g_n, g^z, a_0 \rangle$.

Let Q be a bound on the number of H queries made by $\mathcal{C}$. $\mathcal{D}$ chooses a random subsequence I of length n from the sequence $(1, 2, \ldots, Q)$.

Let $s_j \in X$ be the jth element on which $\mathcal{C}$ makes an H query. $\mathcal{D}$ answers that query as follows:

If j is the ith element of I, then $\mathcal{D}$ responds with a_i and inserts $\langle s_j, i \rangle$ into L. Otherwise, $j \notin I$ and $\mathcal{D}$ responds with a random number. Since the a_i were chosen at random, the algorithm $\mathcal{C}$ cannot distinguish between our simulation of a random oracle and a real random oracle.

If $\mathcal{C}$ asks for $g^{af(H(s))}$, then, if $\langle s, i \rangle \in L$ for some i, $\mathcal{D}$ replies with g^{y_i}. Otherwise, it outputs FAILURE and halts.

Eventually, $\mathcal{C}$ will output its guess g' for $e(g,g)^{xzf(a_0)}$ and halt; then $\mathcal{D}$ outputs g' and halts.

Let $r_1, r_2, \ldots, r_n \in \mathbb{F}_q$ be such that $g_0 = g^{r_0}, g^{r_1}, \ldots, g_n = g^{r_n}$ (we will never need to explicitly compute the r_i). If $\mathcal{D}$ has not yet failed, and, if $\mathcal{C}$ wins its game, then $\mathcal{C}$ will output $g' = e(g,g)^{xzf(a_0)} = e(g,g)^{xyz}$ (where f is the polynomial $f(u) = r_0 + r_1 u + \cdots + r_n u^n$), which means that $\mathcal{D}$, by outputting $e(g,g)^{xyz}$, will win the BDH game.

Thus, the probability of $\mathcal{D}$ succeeding, given that $\mathcal{C}$ is successful, is at least $1/\binom{Q}{n}$ (it is sufficient for the set I to correspond exactly to the n elements of X for which $\mathcal{C}$ issues $g^{af(H(s))}$ queries). $\square$

Applying the Fujisaki-Okamoto transformation to the cryptosystem yields an ID-CCA 2-HIBE system that tolerates user collusion up to size n; in this system, the Setup algorithm also selects two hash functions $H'\colon \{0,1\}^k \times \{0,1\}^k \to \mathbb{F}_q$ and $H''\colon \{0,1\}^k \to \{0,1\}^k$, and there is an extra level of hashing during encryption and decryption, as follows (we use the same notation as in the description of the cryptosystem): $\mathsf{Encrypt}$ now also picks a random $\sigma \in \{0,1\}^k$, computes $r := H'(\sigma, M)$ (instead of picking a random $r \in \mathbb{F}_q$), and outputs

$C := \langle g^r, \sigma \oplus H_3(s), M \oplus H''(\sigma)\rangle$ (as opposed to $\langle g^r, M \oplus H_3(s)\rangle$). Decrypt is modified similarly.

The proof of Lemma 3 involves a reduction that is exponential in n, rendering the system untrustworthy for large n. One could simply assume that the Bilinear Polynomial Diffie-Hellman game is hard to win in order to rectify this, but it would be better to find a polynomial-time reduction from winning the Bilinear Polynomial Diffie-Hellman game to a more natural assumption.

4 Conclusions and Open Problems

We introduced hierarchical identity-based encryption schemes and their related security definitions. We presented a concrete two-level HIBE scheme that is totally collusion-resistant on the upper level and partially collusion-resistant on the lower level, and showed it to be secure under the BDH assumption (in the random-oracle model). An open problem is to construct a two-level HIBE scheme that is totally collusion-resistant on the lower level and at least partially (if not totally) collusion-resistant on the upper level.

One could try to extend our approach to finding a 2-HIBE system to attempt to construct an ℓ-HIBE system for any given ℓ by using a family of $\ell - 1$ linear e-one-way functions such that the output of one is used as the first input of the next. Ideally, there would be a single linear e-one-way function h: $G_1 \times X \to G_1$, and one could apply h recursively to obtain a family of any given size.

Unfortunately, this may not be enough to guarantee that an ID-OWE attacker has negligible advantage. A major hurdle in a proof of security is that it is not clear how to build a simulator that can answer an algorithm's queries for addresses within the same domain but within a different subdomain than the challenge address. Additionally, if we model hash functions as random oracles, then the most obvious approaches will involve security reductions exponential in ℓ (informally, it seems a simulator must guess which hash query to rig at each level).

In any event, there are no known families of linear e-one-way functions of any size (that can accompany a bilinear function satisfying the BDH assumption). However, if we are willing to tolerate n-party collusion at every level, then we may construct a family of functions by extending our scheme. In our system, KeyGen_2 produces one group element from n group elements. This suggests a scheme where a private key at the top level consists of $n^{\ell-1}$ group elements: to generate keys for the next level, sets of n elements are used to produce one group element each; private keys at a given level contain n times the number of group elements in private keys of the level below. At the bottom level a private key consists of a single group element which can be fed into the bilinear function.

Clearly, in this scheme, the key size at the top level increases exponentially with ℓ, so it does not scale well. Even so, for some applications, setting ℓ slightly greater than 2 may be beneficial; e.g., if we take domains, subdomains and users to be corporations, departments and employees, then it may be less likely for n department key generators to collude than for n^2 employees to collude. (It is still possible for n^2 employees to collude and expose the corporation's key, but this requires n employees from each of n different departments.)

References

1. D. Boneh and M. Franklin, "Identity Based Encryption from the Weil Pairing", *Advances in Cryptology: CRYPTO 2001 (LNCS 2139)*, pp. 213–229, 2001.
2. D. Boneh and M. Franklin, "Identity Based Encryption from the Weil Pairing", *Cryptology ePrint Archive*, Report 2001/090, 2001. `http://eprint.iacr.org/2001/090/`
3. D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing", *Advances in Cryptology: ASIACRYPT 2001 (LNCS 2248)*, pp. 514–532, 2001.
4. C. Cocks, "An Identity Based Encryption Based on Quadratic Residues", *Cryptography and Coding (LNCS 2260)*, pp. 360–363, 2002.
5. J. Coron, "On the Exact Security of Full Domain Hash", *Advances in Cryptology: CRYPTO 2000 (LNCS 1880)*, pp. 229–235, 2000.
6. E. Fujisaki and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes", *Advances in Cryptology: CRYPTO '99 (LNCS 1666)*, pp. 537–554, 1999.
7. A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman", *Algorithmic Number Theory : 4th International Symposium, ANTS-IV (LNCS 1838)*, pp. 385–394, 2000.
8. M. Kasahar, K. Ohgishi, and R. Sakai, "Cryptosystems Based on Pairing", *The 2001 Symposium on Cryptography and Information Security*, Oiso, Japan, 2001.
9. A. Shamir, "Identity-Based Cryptosystems and Signature Schemes", *Advances in Cryptology: CRYPTO '84 (LNCS 196)*, pp. 47–53, 1985.
10. E. Verheul, "Evidence That XTR Is More Secure than Supersingular elliptic curve cryptosystems", *Advances in Cryptology: EUROCRYPT 2001 (LNCS 2045)*, pp. 195–210, 2001.
11. E. Verheul, "Self-Blindable Credential Certificates from the Weil Pairing", *Advances in Cryptology: ASIACRYPT 2001 (LNCS 2248)*, pp. 533–551, 2001.
12. ISO/IEC 9594-8, "Information Technology — Open Systems Interconnection — The Directory: Authentication Framework", International Organization for Standardization, Geneva, Switzerland, 1995 (equivalent to ITU-T Recommendation X.509, 1993).