Multiple Hierarchy Wildcard Encryption

Christopher A. Seaman[1], Kent D. Boklan[2], and Alexander W. Dent[3]

[1] Graduate Center, City University of New York, USA
[2] Queens College, City University of New York, USA
[3] Royal Holloway, University of London, UK

1 Introduction and Motivation

1.1 Identity-Based Encryption

Identity-based encryption is most often considered in the context of one-to-one communication within a single Trusted Authority (TA); every encrypted message is sent between two individual entities. In this paper we consider a coalition of TA's desiring secure one-to-many communication, with each TA retaining the security of its secret keys. This secrecy requirement is natural in the setting of dynamic coalitions.

1.2 Known Extensions

One-to-many secure communication is a powerful cryptographic tool. Take a MANETs setting for example, where transmission is much more expensive than computation, one-to-many communication allows for a single transmitted message to be read by any number of valid recipients within range. Multiple recipients may decrypt the same message through the use of one or more 'wildcards'. A wildcard is a special character, commonly '*', that may be used in an address in lieu of specifying a particular aspect of an identity. Anyone with an identifier matching the non-wildcard portion is then able to read the message.

The method of employing wildcards into hierarchical identity-based encryption structures (Abdallah et al., 2006) allows an individual at any level within one TA to send messages to entire levels within that TA.

The wildcard method of multicast communication is limited by the structure of the hierarchy: it cannot distinguish between entities within a single hierarchical level. For example, a single message to "*@school.edu" could be read by "alice@school.edu", "bob@school.edu", and "eve@school.edu"; however, it would not be possible to send a single message to Alice and Bob without also allowing Eve to read it. Likewise, a message may be addressed to a single specified TA or to all TA's by wildcard; it is impossible to select multiple TA's to receive a message without making the message readable across all TA's. It is interesting to note that a message may be broadcast to all TA's in a coalition and remain secure against non-members of the coalition.

1.3 Our Contributions

In addition to one-to-many communication, this paper assumes that the coalition of TA's may change over time and allows those changes without compromising the security of communication or any TA's secret key. TA's may be removed or added to the coalition with minimal configuration. Upon a change in coalition make-up the participating TA's exchange public information, and based on non-public secret information they are able to communicate. The private keys of subordinate entities of each TA must be updated, but each TA can accomplish this through a single broadcast message exclusively readable to members of that TA. This flexible reconfiguration ability similarly allows a fixed set of TA's in coalition to schedule secure reconfigurations with minimal communication long before any secret key has a chance to become stale or compromised.

Should the members of the coalition of TA's change at some point, an interesting aspect of this system is that it is quickly reconfigurable. Reconfiguration of the system in play relies on the hierarchical organization of secret keys, the ability to broadcast messages to all members of a TA. In our example instantiation based on the Boneh-Boyen HIBE as extended by Abdalla et al. we rely only on the security assumptions initially made by Boneh and Boyen, the difficulty of the Bilinear Decision Diffie-Hellman (BDDH) problem.

Reconfiguring keys to adapt to changing coalitions is made more efficient through one-to-many communication that is non-confidential, but integrity protected. Coalition member TA's conduct a round of communication at the highest level to create TA-specific key adjustment parameters. Then each TA securely broadcasts this information to all of its subordinates in a single message in lieu of sending a separate message to each subordinate with their unique private key.

1.4 Usage Scenario

As an example of our techniques, we propose the following highly simplistic usage scenario. We suppose that individuals in a force can be identified using the hierarchical identifier (force name, mission name, individual name). For example, an individual could be identified as (US force, Mission 12, John Smith). Our scheme allows an architecture in which a central facility for each force generates a master key-pair. The master public key is publicly distributed and is intended for long-term use. We will assume that all master public keys are known by all individuals in the scenario. The master private key is used by the central facility to derive keys for each hierarchical name (force name, mission name, individual name).

These individual hierarchical encryption schemes can be used as independent WIBEs. In other words, any user in possession of a force's master public key and a hierarchical identifier (force name, mission name, individual name) may send a message that can only be read by that individual. Furthermore, the hierarchical identifier may include wildcard characters, denoted *. So, for example, a user could send a message to every individual on a mission using the hierarchical identifier (force name, mission name, *) or to an individual on any mission using the hierarchical identifier (force name, *, individual name).

However, our scheme allows coalitions to be formed between forces. If two (or more) forces which to form a coalition, then the central facilities exchange some (public) information which then allow the central facility to send a broadcast message to every member of the force. This broadcast message allows the individuals to derive coalition decryption keys from their individual decryption keys. The individuals must be assured that the broadcast message is from their force's central facility, but the message is short and need not be confidential. These coalition keys allow individuals to decrypt messages sent to multiple forces simultaneously.

For example, for a particular mission, the US and UK force may form a coalition. This allows messages to be sent to hierarchical identifiers of the form (coalition name, mission name, individual name) where the coalition name is consists of both the US and UK forces. Therefore, to send a message to all individuals involved in the mission, one would merely have to send the message using the hierarchical name (US and UK force, mission name, *). This message could only be decrypted by an individual on the mission who has received the (public) broadcast update from their central facility which allows them to form the coalition key for the coalition containing the US forces and the UK forces.

Furthermore, these coalitions can be dynamic in nature. So, in the above scenario, if the German force was later included in the mission, then broadcast messages could be sent to update coalition decryption keys to allow the US, UK, and German forces to decrypt messages. The German force members would not be able to read earlier messages, which were encrypted for use the US and UK forces coalition only. If the US force retired from the mission, then the coalition keys could be updated so that only the UK and German forces could decrypt messages.

2 Syntax

We define a Trusted Authority (TA_i) as a hierarchy of identities of the form $\boldsymbol{ID} = (ID_1, ID_2, ..., ID_k)$ with the same first identity ($ID_1 = TA_i$) and maximum depth of $k \leq \ell$. Given a population of TA's $\mathcal{U} = \{TA_1, TA_2, ..., TA_n\}$ we define a coalition $\mathcal{C} = \{TA_a, TA_b, ..., TA_k\} \subseteq \mathcal{U}$. A multi-hierarchy WIBE consists of the following PPT algorithms/protocols:

- $\texttt{CreateTA}(TA_i)$: This algorithm is run once by TA_i, it generates a public key and private key for that TA $(pk_i, sk_i) \xleftarrow{\$} \texttt{CreateTA}$ and publishes pk_i.

- `SetupCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j) \forall TA_j \in \mathcal{C})$: This algorithm calculates messages $w_{i,j}$ to be distributed from TA_i to each $TA_j \in \mathcal{C} \backslash \{TA_i\}$ who haven't received messages from previous instantiations of the algorithm. These values are stored by TA_j for later use in the `SetupCoalitionKeys` algorithm.
- `SetupCoalitionKeys`$(TA_j, sk_j, (TA_i, w_{i,j}) \forall TA_j \in \mathcal{C})$: The algorithm completes the setup of the coalition. After every member TA_i of the coalition has provided a message $w_{i,j}$ to TA_j. It outputs a message v_j to be broadcast to every member of TA_j's hierarchy in order who then run the `ExtractCoalitionKey` algorithm to obtain their $\mathcal{C}$ rcoalition-specific secret keys.

The system should be able to dynamically update the coalition. We may wish to change from a coalition $\mathcal{C}$ to a coalition $\mathcal{C}'$. Persistent members of $\mathcal{C} \cap \mathcal{C}'$ execute the `SetupCoalitionBroadcast` algorithm to broadcast messages to new members of $\mathcal{C}' \setminus \mathcal{C}$. New members also run `JoinCoalition`, but broadcast messages to all members of coalition $\mathcal{C}'$. Excluded members $\mathcal{C}' \setminus \mathcal{C}$ are excluded and left unable to communicate under the new coalition.

We now describe the algorithms required by the individual users.

- `Extract`$(\boldsymbol{ID}, ID', d_{\boldsymbol{ID}})$: This algorithm outputs a decryption key $d_{\boldsymbol{ID}\|ID'}$ for the identity $\boldsymbol{ID}\|ID'$. The basic level has $\boldsymbol{ID} = TA_i$ and $d_{TA_i} = sk_i$.
- `ExtractCoalitionKey`$(\boldsymbol{ID} \in TA_i, v_i, d_{\boldsymbol{ID}})$: This algorithm outputs a user key $c_{\boldsymbol{ID}}$ for the coalition $\mathcal{C}$ by combining the broadcast message u_i corresponding to coalition $\mathcal{C}$ and their decryption key $d_{\boldsymbol{ID}}$.
- `UpdateCoalitionKey`$(\boldsymbol{ID} \in TA_i, u_i, c_{\boldsymbol{ID}}, d_{\boldsymbol{ID}})$: This algorithm outputs an updated user key $c'_{\boldsymbol{ID}}$ for the coalition $\mathcal{C}'$ by combining the broadcast key u_i corresponding to coalition $\mathcal{C}'$ with the user's decryption key $d_{\boldsymbol{ID}}$ and existing coalition key $c_{\boldsymbol{ID}}$ corresponding to the previous coalition $\mathcal{C}$.
- `Encrypt`$(P, m, (TA_i, pk_i) \forall TA_i \in \mathcal{C})$: This algorithm is used to encrypt a message m to entities satisfying the pattern P under the coalition $\mathcal{C} = \{TA_a, \ldots, TA_k\}$. It outputs a ciphertext C or the invalid symbol $\perp$.
- `Decrypt`$(\boldsymbol{ID}, c_{\boldsymbol{ID}}, d_{\boldsymbol{ID}}, C, (TA_i, pk_i) \forall TA_i \in \mathcal{C})$: Decrypts ciphertext C to output message m.

3 Security Model

The security of a multi-TA WIBE is parametrized by a bit $b \in \{0,1\}$ and a positive integer k through the following game. An attacker, $\mathcal{A}$, is tasked with guessing the value of b chosen by the challenger. Initially an attacker is given the public parameters for the system. These parameters include $(g_1, g_2, u_{1,0}, u_{1,1}, u_{2,0}, u_{2,1}, \ldots, u_{L,0}, u_{L,1}) \in \mathbb{G}$ defining a scheme of maximum depth L, an identity space $\mathcal{ID} = \{0,1\}^k$, and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}^*$. In the first phase the attacker is allowed to query the challenger. For chosen-plaintext security (IND-ID-CPA), the attacker is given polynomially many queries to the following oracles:

- `CreateTA`(TA): The oracle computes $(pk_i, sk_i) \overset{\$}{\leftarrow}$ `Setup`$(1^k, TA)$ for the TA identity TA_i and returns pk_i. This oracle can only be queried once for each identity TA_i.
- `SetupCoalitionBroadcast`$(TA_i, \mathcal{C})$: This oracle runs the `SetupCoalitionBroadcast` algorithm for coalition $\mathcal{C}$ containing TA_i and returns messages $w_{i,j} \forall TA_j \in \mathcal{C} \setminus TA_i$.
- `SetupCoalitionKeys`$(TA_i, w_{j,i} \forall TA_j \in \mathcal{C} \setminus TA_i)$: This oracle can only be queried if each TA_i, $TA_j \in \mathcal{C}$ has been queried to the `SetupCoalitionBroadcast` oracle with k TA's in the coalition. The oracles runs the `SetupCoalitionKeys` algorithm assuming that message $w_{j,i}$ was sent by TA_j. Note that this does not imply that all the TAs believe that they're in the same coalition.
- `Corrupt`$(\boldsymbol{ID})$: The oracle returns $d_{\boldsymbol{ID}}$ for the identity $\boldsymbol{ID}$. Note that if $\boldsymbol{ID} = TA_i$ then this method returns TA_i's secret key sk_i.

For chosen-ciphertext security, the attacker is additionally given access to decryption oracles:

- `UserDecrypt`($\boldsymbol{ID}, C^*$): This oracle decrypts the ciphertext with the decryption key $d_{\boldsymbol{ID}}$.
- `CoalitionDecrypt`($\mathcal{C}, \boldsymbol{ID}, C^*$): This oracle decrypts the ciphertext with the coalition decryption key $c_{\boldsymbol{ID}}$ corresponding to coalition $\mathcal{C}$.

To end the first phase the attacker outputs two equal-length messages m_1, m_2 and a challenge pattern $\boldsymbol{P}$=$(P_1, P_2, \ldots, P_\ell)*$ for $P_i \in \mathcal{ID} \cup \{*\}$. This is modeled as a single query to the oracle:

- `Test`($\mathcal{C}^*, P, m_0, m_1$): This oracle takes as input two messages (m_0, m_1) of equal length. It encrypts the message m_b for the coalition using $pk_i \forall TA_i \in \mathcal{C}^*$ under the pattern P. This oracle may only be access once and outputs a ciphertext C^*. We will let $\mathcal{C}^*$ denote the challenge coalition $(TA_a, \ldots, TA_k)$.

The relationship between the attacker's queries and choice of $\boldsymbol{P}^*$ are restricted. Specifically, the attacker is not allowed access to the decryption keys of nodes matching the challenge pattern nor nodes capable of generating those keys. The disallowed oracle queries are:

1. A `Corrupt` query for any $\boldsymbol{ID}$ matching pattern P
2. A `Corrupt` query for any $\boldsymbol{ID}$ that is an ancestor of pattern P, i.e. there exists $\boldsymbol{ID}^*$ such that $\boldsymbol{ID} \| \boldsymbol{ID}^*$ matches the pattern P
3. A `UserDecrypt` decrypt query for C^* and any user ID in the test coalition matching the pattern P or an ancestor thereof.

For the second phase the attacker is once again given access to the queries from the first phase (still restricted as above). The attacker ends the game by outputting a bit b' as its guess for the value of bit b. We define the attacker's advantage as:

$$Adv_{\mathcal{A}}^{\texttt{IND}}(k) = |Pr[b' = b] - \tfrac{1}{2}|$$

4 BB-Based Construction

We may instantiate this notion using the Boneh-Boyen WIBE. We assume that there exists a set of bilinear map groups G, G_T of large prime order p and a bilinear map $e : G \times G \to G_T$. We assume the existence of two randomly chosen generators $g_1, g_2 \xleftarrow{\$} G^*$. We also assume that there exist $2L + 2$ randomly chosen group elements $u_{i,j} \xleftarrow{\$} G_2$ where $i \in \{0, 1, 2, \ldots, L\}$ and $j \in \{0, 1\}$ and L is a limit on the maximum depth in a hierarchy. The MTA-WIBE is described as follows:

- `CreateTA`(TA_i): The TA generates $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$ and computes master public key $pk_i \leftarrow g_1^{\alpha_i}$. The master private key is defined to be $g_2^{\alpha_i}$.
- `Extract`($\boldsymbol{ID}, ID', d_{\boldsymbol{ID}}$): Under $TA_i = ID_1$, the first level identity is $\boldsymbol{ID} = (ID_1, ID_2)$. The TA generates $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and computes the key $d_{\boldsymbol{ID}} = (h, a_1, a_2)$ where $h \leftarrow g_2^{\alpha_i}(u_{0,0} \cdot u_{0,1}^{ID_1})^{r_1}(u_{1,0} \cdot u_{1,1}^{ID_2})^{r_2}$, $a_1 \leftarrow g_1^{r_1}$ and $a_2 \leftarrow g_1^{r_2}$. An identity $(ID_1, \ldots, ID_\ell)$ with private key $(h, a_1, a_2, \ldots, a_\ell)$ and $\ell < L$ can compute the decryption key for its child $(ID_1, \ldots, ID_\ell, ID_{\ell+1})$ by choosing a random value $r_{\ell+1}$ and computing the private key as the tuple $(h', a_0, a_1, \ldots, a_\ell, a_{\ell+1})$ where $h' \leftarrow h(u_{\ell+1,0} \cdot u_{\ell+1,1}^{ID_{\ell+1}})^{r_{\ell+1}}$ and $a_{\ell+1} \leftarrow g_1^{r_{\ell+1}}$.
- `SetupCoalitionBroadcast`($TA_i, \mathcal{C}$): For $TA_j \in \mathcal{C}$, TA_i randomly generates $r_j \xleftarrow{\$} \mathbb{Z}_p$ and computes $w_{i,j,0} \leftarrow g_2^{\alpha_i}(u_{0,0} \cdot u_{0,1}^{TA_j})^{r_j}$ and $w_{i,j,1} \leftarrow g_1^{r_j}$ where $TA_j \in \mathbb{Z}_p$ is the identity of TA_j. The algorithm sets $w_{i,j} = (w_{i,j,0}, w_{i,j,1})$ and outputs a list of TA/message pairs $(TA_j, w_{i,j}) \forall TA_j \in \mathcal{C} \setminus TA_i$.
- `SetupCoalitionKeys`($TA_i, sk_i, (TA_j, pk_j, w_{j,i}) \forall TA_j \in \mathcal{C}$): The algorithm outputs the message $u_i \leftarrow (\prod w_{j,i,0}, \prod w_{j,i,1}) \forall TA_j \in \mathcal{C} \setminus TA_i$

– `ExtractCoalitionKey`$(\mathcal{C}, u_i, d_{\boldsymbol{ID}})$: Parse u_i as $(w_{j,i})$ where $w_{j,i,0} = g_2^{\alpha_j}(u_{0,0} \cdot u_{0,1}^{TA})^{r_j}$ and $w_{j,i,1} = g_1^{r_j}$ (where $g_2^{\alpha_j}$ is the private key of TA_j). A user with private key $(h, a_0, a_1, a_2, \ldots, a_\ell)$ can form a coalition key $(h', a_0', a_1, a_2, \ldots, a_\ell)$ where

$$h' \leftarrow h \prod_{j=1}^{k} w_{j,0} = g_2^{\sum \alpha_j}(u_{0,0} \cdot u_{0,1}^{TA})^{\sum r_j}$$
$$a_0' \leftarrow a_0 \prod_{j=1}^{k} w_{j,1} = g_1^{r + \sum r_j}$$

– `Encrypt`$(P, m, (TA_i, pk_i) \forall TA_i \in \mathcal{C})$: Let ℓ be the depth of the pattern and $W(P)$ be the set of levels which have wildcard characters. The sender chooses $t \xleftarrow{\$} \mathbb{Z}_p$ and computes the ciphertext $C = (C_1, C_{2,0}, C_{2,1}, ..., C_{2,\ell}, C_3)$ where

$$C_1 \leftarrow g_1^t$$
$$C_{2,i} \leftarrow \begin{cases} (u_{i,0} \cdot u_{i,1}^{ID_i})^t & \text{if } i \notin W(P) \\ (u_{i,0}^t, u_{i,1}^t) & \text{if } i \in W(P) \end{cases}$$
$$C_3 \leftarrow m \cdot e(\prod_{j=1}^{n} pk_j, g_2)^t$$

– `Decrypt`$(\boldsymbol{ID}, c_{\boldsymbol{ID}}, C)$: Parse $\boldsymbol{ID}$ as $(ID_1, \ldots, ID_\ell)$, c_{ID} as $(h, a_0, \ldots, a_\ell)$, and C as $(C_1, C_{2,0}, \ldots, C_{2,\ell}, C_3)$. For each $i \in W(P)$, parse $C_{2,i}$ as $(v_{i,0}, v_{i,1})$. We recover a complete HIBE ciphertext by setting $C_{2,i}' \leftarrow C_{2,i}$ if $i \notin W(P)$, and $C_{2,i} \leftarrow v_{i,0} \cdot v_{i,1}^{ID_i}$ if $i \in W(P)$. Recover

$$m' \leftarrow C_3 \frac{\prod_{i=0}^{\ell} e(a_i, C_{2,i}')}{e(C_1, h)}$$

and return m'.

A HIBE scheme may be designed analogously except that we remove the possibility of the pattern containing wildcards.

Theorem 1. *Suppose that there exists an attacker $\mathcal{A}$ against the selective-identity multiple TA Boneh-Boyen WIBE that runs in time t and with advantage ϵ, then there exists an attacker $\mathcal{B}$ against the selective-identity multiple TA Boneh-Boyen HIBE that runs in time t' and with advantage ϵ' where $t \approx t'$ and $\epsilon' = \epsilon$.*

Proof We directly describe the algorithm $\mathcal{B}$ which breaks the HIBE using the algorithm $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{B}$ runs as follows:

1. $\mathcal{B}$ runs $\mathcal{A}$ on the security parameter. $\mathcal{A}$ responds by outputting a description of the challenge coalition $TA^* = (TA_1^*, \ldots, TA_n^*)$ and the challenge pattern $P^* = (P_1^*, \ldots, P_{\ell^*}^*)$. Let π be a map which identifies the number of non-wildcard entries in the first i layers of P^*, i.e. $\pi(i) = i - |W(P_{\leq i}^*)|$. $\mathcal{B}$ outputs the challenge coalition TA^* and the challenge identity $\hat{ID}^* = (\hat{ID}_1^*, \ldots, \hat{ID}_{\pi(\ell^*)}^*)$ where $\hat{ID}_i^* = P_{\pi(i)}^*$.
2. The challenger responds with HIBE parameters $param = (\hat{g}_1, \hat{g}_2, \hat{u}_{0,0}, \ldots, \hat{u}_{L,1})$. $\mathcal{B}$ generates WIBE parameters as follows:

$$\begin{array}{rll} (g_1, g_2) & \leftarrow (\hat{g}_1, \hat{g}_2) & \\ u_{i,j} & \leftarrow \hat{u}_{\pi(i),j} & \text{for } i \notin W(P^*), j \in \{0,1\} \\ u_{i,j} & \leftarrow g_1^{\beta_{i,j}} & \text{for } i \in W(P^*), j \in \{0,1\} \text{ where } \beta_{i,j} \xleftarrow{\$} \mathbb{Z}_p \\ u_{i,j} & \leftarrow \hat{u}_{i-|W(P^*)|,j} & \text{for } i \in \{\ell+1, \ldots, L\}, j \in \{0,1\} \end{array}$$

3. $\mathcal{B}$ executes $\mathcal{A}$ on the public parameters $(g_1, g_2, u_{0,0}, \ldots, u_{L,1})$. $\mathcal{A}$ may make the following oracle queries:
   - `CreateTA`: $\mathcal{B}$ forwards this request to its own oracle and returns the response.
   - `Corrupt`: $\mathcal{B}$ forwards this request to its own oracle and returns the response. Note that the disallowed queries for this oracle are preserved by the map $ID_i \to ID_{\pi(i)}$.
   - `SetupCoalitionBroadcast`: $\mathcal{B}$ forwards this request to its own oracle and returns the response.
   - `SetupCoalitionKeys`: $\mathcal{B}$ forwards this request to its own oracle and returns the response.
   - `Extract`: To an extract a decryption key for an identity $ID = (ID_1, \ldots, ID_\ell)$ which does not match the challenge pattern, $\mathcal{B}$ computes the projection of the identity onto the HIBE identity space to give a projected identity $\hat{ID} = (\hat{ID}_1 \ldots, \hat{ID}_{\hat{\ell}})$.
     - If $\ell \leq \ell^*$ then $\hat{\ell} \leftarrow \pi(\ell)$ and $\hat{ID}_{\pi(i)} \leftarrow ID_i$ for $i \notin W(P^*_{\leq \ell})$. Since ID does not match the challenge pattern for the WIBE, we have that $\hat{ID}$ does not match the challenge identity for the HIBE. $\mathcal{B}$ queries its `Extract` on $\hat{ID}$ and receives $(\hat{h}, \hat{a}_0, \ldots, \hat{a}_{\hat{\ell}})$ in response. $\mathcal{B}$ now "retro-fits" to find a complete key, by setting

$$\begin{array}{ll} a_0 \leftarrow \hat{a}_0 & \\ a_i \leftarrow \hat{a}_{\pi(i)} & \text{for } 1 \leq i \leq \ell \text{ and } i \notin W(P^*_{\leq \ell}) \\ a_i \leftarrow g_1^{r_i} & \text{for } 1 \leq i \leq \ell \text{ and } i \in W(P^*_{\leq \ell}) \text{ where } r_i \xleftarrow{\$} \mathbb{Z}_p \\ h \leftarrow \hat{h} \prod_{i=1, i \in W(P^*_{\leq \ell})}^{\ell} (u_{i,0} \cdot u_{i,1}^{ID_i})^{r_i} & \end{array}$$

       and returning the key $(h, a_0, \ldots, a_\ell)$.
     - If $\ell > \ell^*$, then $\hat{\ell} = \ell - |W(P^*)|$, $\hat{ID}_{\pi(i)} \leftarrow ID_i$ for $1 \leq i \leq \ell^*$ and $i \notin W(P^*)$, and $\hat{ID}_{i-|W(P^*)|} \leftarrow ID_i$ for $\ell^* < i \leq \ell$. Since ID does not match the challenge pattern for the WIBE, we have that $\hat{ID}$ does not match the challenge identity for the HIBE. $\mathcal{B}$ queries its `Extract` on $\hat{ID}$ and receives $(\hat{h}, \hat{a}_0, \ldots, \hat{a}_{\hat{\ell}})$ in response. $\mathcal{B}$ now "retro-fits" to find a complete key, by setting

$$\begin{array}{ll} a_0 \leftarrow \hat{a}_0 & \\ a_i \leftarrow \hat{a}_{\pi(i)} & \text{for } 1 \leq i \leq \ell^* \text{ and } i \notin W(P^*) \\ a_i \leftarrow g_1^{r_i} & \text{for } 1 \leq i \leq \ell^* \text{ and } i \in W(P^*) \text{ where } r_i \xleftarrow{\$} \mathbb{Z}_p \\ a_i \leftarrow \hat{a}_{i-|W(P^*)|} & \text{for } \ell^* < i \leq \ell \\ h \leftarrow \hat{h} \prod_{i=1, i \in W(P^*)}^{\ell^*} (u_{i,0} \cdot u_{i,1}^{ID_i})^{r_i} & \end{array}$$

       and returning the key $(h, a_0, \ldots, a_\ell)$.
     - `Test`: If $\mathcal{A}$ queries the test oracle on two equal-length messages (m_0, m_1), then $\mathcal{B}$ forwards this query on to its own `Test` oracle. The oracle returns $(C_1^*, C_{2,0}^*, \hat{C}_{2,1}^*, \ldots, \hat{C}_{2,\pi(\ell^*)}^*, C_3^*)$. $\mathcal{B}$ retro-fits this to form a challenge ciphertext for $\mathcal{A}$, by setting

$$\begin{array}{ll} C_{2,i}^* \leftarrow \hat{C}_{2,\pi(i)}^* & \text{for } 1 \leq i \leq \ell^*, i \notin W(P^*) \\ C_{2,i}^* \leftarrow (\psi(C_1^*)^{\beta_{i,0}}, \psi(C_1^*)^{\beta_{i,1}}) & \text{for } 1 \leq i \leq \ell^*, i \in W(P^*) \end{array}$$

       $\mathcal{B}$ returns $(C_1^*, C_{2,0}^*, \ldots, C_{2,\ell^*}^*, C_3)$ to $\mathcal{A}$ as the challenge ciphertext.

   $\mathcal{A}$ terminates by outputting a big b' as its guess for the challenge bit b.
4. $\mathcal{B}$ outputs the bit b'.

The algorithm $\mathcal{B}$ correctly simulates the oracles to which $\mathcal{A}$ has access; furthermore, $\mathcal{B}$ wins the HIBE game if and only if $\mathcal{A}$ wins the game. Hence, we have the theorem. □

Theorem 2. *If there exists an attacker $\mathcal{A}$ against the selective-identity multiple TA IND-WID-CPA secure of the Boneh-Boyen HIBE that runs in time t and has advantage ϵ, then there exists an algorithm $\mathcal{B}$ that solves the DBDH problem that runs in time $t' = O(t)$ and has advantage $\epsilon' \geq \epsilon - q_K/p$.*

Proof We directly describe the algorithm $\mathcal{B}$ against the DBDH problem:

1. $\mathcal{B}$ receives the input (g, g^a, g^b, g^c, Z).

2. $\mathcal{B}$ runs $\mathcal{A}$ to obtain the challenge coalition $TA^* = \{TA_1^*, \ldots, TA_{n^*}^*\}$ and the challenge identity $\boldsymbol{ID^*} = (ID_1^*, \ldots, ID_{\ell^*}^*)$ under the challenge trust authority TA_1^*. (We assume, without loss of generality, that the challenge identity is under the trusted authority TA_1^*.)
3. If $\ell^* < L$ then randomly generates $ID_{\ell^*+1}^*, \ldots, ID_L^* \xleftarrow{\$} \mathbb{Z}_p$.
4. $\mathcal{B}$ computes the challenge parameters

$$\begin{gathered} g_1 \leftarrow g \qquad g_2 \leftarrow g^b \qquad k_{i,j}, \alpha_j \xleftarrow{\$} \mathbb{Z}_p^* \text{ for } 0 \leq i \leq L, j \in \{0,1\} \\ pk_1 \leftarrow g^a / g^{\sum_{j=2}^{n^*} \alpha_j} \qquad pk_j \leftarrow g^{\alpha_j} \text{ for } 2 \leq j \leq n^* \\ u_{0,0} \leftarrow g_1^{k_{0,0}} \cdot (g^a)^{-TA_1^* 5 k_{0,1}} \qquad u_{0,1} \leftarrow (g^a)^{\alpha_{0,1}} \\ u_{i,0} \leftarrow g_1^{k_{i,0}} \cdot (g^a)^{-ID_i^* \alpha_{i,1}} \qquad u_{i,1} \leftarrow (g^a)^{\alpha_{i,1}} \qquad \text{for } 1 \leq i \leq L \end{gathered}$$

5. $\mathcal{B}$ runs $\mathcal{A}$ on the public parameters $(g_1, g_2, u_{0,0}, u_{0,1}, \ldots, u_{L,0}, u_{L,1})$. If $\mathcal{A}$ makes an oracle query and $\mathcal{B}$ answers queries as follows:
   – `CreateTA`(TA_i): If $TA_i = TA_1^*$ then $\mathcal{B}$ returns $pk_1 = g^a / g^{\sum_{j=2}^{n^*} \alpha_j}$. If $TA \neq TA^*$ then $\mathcal{B}$ generates $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$ and returns $pk_i = g_1^{\alpha_i}$ as described above.
   – `Corrupt`$(\boldsymbol{ID})$: For $\boldsymbol{ID} = TA_i$, we have $TA_i \neq TA^*$ for this to be a valid query; hence, $\mathcal{B}$ returns $g_2^{\beta_i}$. For $\boldsymbol{ID}$ as a subordinate user, we require that $TA \neq TA_1^*$ or $\boldsymbol{ID}$ is not ancestor of $\boldsymbol{ID^*}$ where $\boldsymbol{ID} = (ID_1, \ldots, ID_\ell)$. If $TA \neq TA_1^*$ then we may extract a decryption key using the extract algorithm and the master secret key of TA. If $TA = TA_1^*$ and there must exist an index $1 \leq j \leq L$ such that $ID_j = ID_j^*$, then $\mathcal{B}$ generates $r_0, \ldots, r_\ell \xleftarrow{\$} \mathbb{Z}_p$ and computes the decryption key $(h, a_0, \ldots, a_j)$ for $(ID_1, \ldots, ID_j)$ as

$$\begin{aligned} h &\leftarrow g_2^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} \cdot g_2^{\sum_{j=2}^{n^*} \beta_j} \cdot \left(u_{0,0} \cdot u_{0,1}^{TA}\right)^{r_0} \cdot \prod_{i=1}^{j} \left(u_{i,0} \cdot u_{i,1}^{ID_i}\right)^{r_i} \\ a_i &\leftarrow g_1^{r_i} \text{ for } 0 \leq i \leq j-1 \\ a_j &\leftarrow g_2^{-\frac{1}{\alpha_{j,1}(ID_j - ID_j^*)}} \cdot g_1^{r_j} \end{aligned}$$

   $\mathcal{B}$ computes the decryption key for $\boldsymbol{ID}$ using the key derivation algorithm and returns the result. If no such j exists then $\mathcal{B}$ aborts.
   – `SetupCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j) \forall TA_j \in \mathcal{C})$: For this to be a valid request we must have $TA_i \in \mathcal{C}$. For $TA_i \neq TA_1^*$, $\mathcal{B}$ can compute the private key directly; hence, $\mathcal{B}$ can return the correct value using the appropriate algorithm. For $TA_i = TA_1^*$, $\mathcal{B}$ generates $r_0 \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$\begin{aligned} w_{i,0} &\leftarrow g_2^{\frac{-\alpha_{0,0}}{\alpha_{0,1}(TA_i - TA_1^*)}} \cdot g_2^{\sum_{j=2}^{n^*} \beta_j} \cdot \left(u_{0,0} \cdot u_{0,1}^{TA_i}\right)^{r_0} \\ w_{i,1} &\leftarrow g_2^{-\frac{1}{\alpha_{0,1}(TA_i - TA_1^*)}} g_1^{r_0} \end{aligned}$$

   for $1 \leq i \leq n$ and sets $w_i \leftarrow (w_{i,0}, w_{i,1})$. $\mathcal{B}$ outputs a list of TA/message pairs $((TA_i, w_i))_{i=1}^n$.
   – `Test`(m_0, m_1): For this query to be valid, we require that $|m_0| = |m_1|$. $\mathcal{B}$ chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and computes the ciphertext

$$C^* \leftarrow (g^c, (g^c)^{\alpha_{1,0}}, \ldots, (g^c)^{\alpha_{\ell^*,0}}, m_b \cdot Z)\,.$$

   $\mathcal{B}$ returns the ciphertext C^*.

   $\mathcal{A}$ terminates with the output of a bit b'.
6. If $b = b'$ then $\mathcal{B}$ outputs 1. Otherwise, outputs 0.

The `Corrupt` oracle for subordinates works perfectly providing that $\mathcal{B}$ does not abort. The simulator only occurs if $\boldsymbol{ID^*} \neq \boldsymbol{ID}$, $\boldsymbol{ID^*}$ is an ancestor of $\boldsymbol{ID}$, and $\boldsymbol{ID}$ is an ancestor of $(ID_1^*, \ldots, ID_L^*)$. In particular, this means that $ID_{\ell^*+1} = ID^*_{\ell^*+1}$, which occurs with probability $1/p$ as this value is information theoretically hidden from $\mathcal{A}$. Hence, the probability that this does not occur in the entire execution of $\mathcal{A}$ is q_K/p where q_K is the number of queries to the `Corrupt` oracle. To show that if the simulator doesn't abort, the `Corrupt` returns a correct key, not that it suffices to show that

$$sk_1 \left(u_{j,0} \cdot u_{j,1}^{ID_j}\right)^r = g_2^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} \cdot g_2^{-\sum_{i=2}^{n^*} \beta_i} \qquad \text{for} \qquad r = -\frac{b}{\alpha_{0,1}(ID_j - ID_j^*)}\,.$$

We note that $sk_1 = g_2^{a - \sum_{i=2}^{n^*} \beta_i}$. The correct decryption key is

$$\begin{aligned} sk_1 \left(u_{j,0} \cdot u_{j,1}^{ID_j}\right)^r &= g_2^{a-\sum_{i=2}^{n^*} \beta_i} \left(g^{\alpha_{j,0}} \cdot (g^a)^{-\alpha_{j,1} ID_j^*} \cdot (g^a)^{\alpha_{j,1} ID_j}\right)^{-\frac{b}{\alpha_{j,1}(ID_j - ID_j^*)}} \\ &= g^{ab} g_2^{-\sum_{i=2}^{n^*} \beta_i} \left(g^{\alpha_{j,0}} \cdot (g^a)^{\alpha_{j,1}(ID_j - ID_j^*)}\right)^{-\frac{b}{\alpha_{j,1}(ID_j - ID_j^*)}} \\ &= g^{ab} g_2^{-\sum_{i=2}^{n^*} \beta_i} (g^b)^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} g^{-ab} \\ &= g_2^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} g_2^{-\sum_{i=2}^{n^*} \beta_i} \end{aligned}$$

Hence, $\mathcal{B}$'s simulation returns a correct decryption key. Similarly, the `SetupCoalitionBroadcast` algorithm gives correct broadcast messages for TA_1^*. All other oracles that $\mathcal{B}$ provides (except, perhaps, the `Test` oracle) correctly simulate the security model for $\mathcal{A}$.

If $Z = e(g,g)^{abc}$ then the `Test` oracle provides a correct encryption of m_b. This is because an encryption using the random value c would have

$$\begin{aligned} C_1 &= g_1^c = g^c \\ C_{2,0} &= (u_{0,0} \cdot u_{0,1}^{TA_1^*})^c = (g^{\alpha_{0,0}})^c = (g^c)^{\alpha_{0,0}} \\ C_{2,i} &= (u_{i,0} \cdot u_{i,1}^{ID_i^*})^c = (g^c)^{\alpha_{i,0}} \qquad \text{for } 1 \le i \le \ell^* \\ C_3 &= m_b \cdot e(\prod_{i=1}^{n^*} pk_i, g_2)^c = m_b \cdot e(g^a, g^b)^c = m_b \cdot e(g,g)^{abc} \end{aligned}$$

The probability that $\mathcal{B}$ outputs 1 in this situation is the probability that $b = b'$ in the mTA-IND-WID-CPA game for the attacker $\mathcal{A}$. If Z is random then the `Test` oracle information theoretically hides b and so the probability that $\mathcal{B}$ outputs 1 in this situation is $1/2$. Hence, the probability that $\mathcal{B}$ wins the DBDH is $\epsilon - q_K/p$. $\square$

5 Non-Selective Identity Security in the Random Oracle Model

Like the Boneh-Boyen HIBE it is based on [**?**], the multi-*TA* scheme is only proven secure in the (selective-identity) multi-TA IND-sWID-CPA model. For non-wildcard single *TA* schemes, selective-identity IND-s(H)ID security in the standard model can be transformed into non-selective-identity secure IND-(H)ID schemes in the random oracle model [**?**]. This technique was extended to securely enable wildcards in IND-sHID-CPA WIBE schemes [**?**] and we further extend it to prove non-selective-identity security of our proposed multi-*TA* scheme in the random oracle model.

The security game in the random oracle model is the same as in the standard model with the addition of one new oracle [**?**]. This oracle responds to each query with a uniformly random element chosen from its output domain. Additionally, for each specific query the response is always the same whenever that query is made.

To prove security in a non-selective-identity setting we alter the routines of the multi-TA WIBE scheme to use the hash of an identity ($H_i(ID_i)$) rather than the identity (ID_i) for each level i. The key derivation and encryption routines pass the identity to a hash function (modeled as a random oracle) and use the result in place of the identity in the relevant algorithm. This hash function maps from the finite space of identifiers to an equal or smaller sized target space. This transform changes the pattern $\boldsymbol{P} = (P_1, P_2, ..., P_\ell)$ into pattern $H(P) = \boldsymbol{P'} = (P'_1, P'_2, ..., P'_\ell)$ as follows:

$$P'_i = H_i(P_i) \text{ for } i \notin W(P), \text{ otherwise } H(P_i) = * = P'_i$$

We note that a finite family of independent random oracles of predetermined maximum size may be simulated using a single random oracle. Given a single random oracle $H(\text{query})$, a WIBE of maximum depth L. and a fixed-length format to enumerate $i \in \{1 \dots L\}$, we may create i independent random oracles. Each oracle $H_i(\text{query})$ corresponding to level i of the WIBE is simulated by prepending the query with the number of the oracle it is addressed to, $H_i(\text{query}) = H(i||\text{query})$.

Theorem 3. *Suppose that there exists a polynomial-time attacker $\mathcal{A}$ against the non-selective-identity multiple TA IND-WID-CPA Boneh-Boyen WIBE with advantage ϵ with access to q_H queries from each of the L random oracles associated with hierarchy levels, then there exists a polynomial-time attacker $\mathcal{B}$ against the selective-identity multiple TA IND-sWID-CPA Boneh-Boyen WIBE with advantage ϵ' when allowed q_K key derivation queries with ϵ' bounded:*

$$\epsilon' \geq \left(\frac{\epsilon}{2^n L(q_H+1)^L}\right)\left(1 - \frac{(q_H+1)}{|ID|}\right)^L$$

Proof Suppose there exists adversary $\mathcal{A}$ against the multi-TA IND-WID-CPA scheme (with hashed identities), we will construct an adversary $\mathcal{B}$ which gains an advantage against the multi-TA IND-sWID-CPA scheme using the algorithm $\mathcal{A}$ as a black box.

For a multi-TA WIBE of maximum depth L consisting of a maximum of k $TA's$, the algorithm $\mathcal{B}$ runs as follows:

1. $\mathcal{B}$ randomly chooses a length $\ell^* \in \{1 \dots L\}$ for its challenge identity vector. Also, $\mathcal{B}$ randomly chooses a coalition $\boldsymbol{TA}^* = (TA^*_1, \dots, TA^*_n)$ from all possible non-empty combinations of the k TA's. The numbering assignment of TA's in the coalition is assigned randomly.
2. $\mathcal{B}$ chooses a challenge pattern $\boldsymbol{P}^*$ as follows. First, $\mathcal{B}$ randomly assigns a family $t_i \xleftarrow{\$} \{0, 1, ..., q_H\}$ for all $i \in \{1, ..., \ell\}$. If $t_i = 0$ then $\mathcal{B}$ assigns $P_i = *$. For $t_1 \neq 0$ $\mathcal{B}$ randomly chooses $P_1 \xleftarrow{\$} \{TA_1, *\}$. For $i > 1$ with $t_i \neq 0$ $\mathcal{B}$ randomly chooses $P_i \xleftarrow{\$} \mathcal{ID}$
3. The challenger responds with WIBE parameters $param = (\hat{g}_1, \hat{g}_2, \hat{u}_{0,0}, \dots, \hat{u}_{L,1})$. $\mathcal{B}$ initiates $\mathcal{A}$ with the same parameters and a family of hashes H_i for each level i allowed in the HIBE.
4. Since both $\mathcal{B}$ and $\mathcal{A}$ are in WIBE environments playing the chosen plaintext game, they have access to the same set of oracles with $\mathcal{A}$ having access to an additional random oracle. To prepare for random oracle queries from $\mathcal{A}$, $\mathcal{B}$ initializes associative lists J_i for each level i allowed in the HIBE to answer queries for each oracle H_i. Initially empty, $\mathcal{B}$ must either track the number of entries made on each list with a counter or equivalently through measure of its size, we let $|J_i| \leftarrow 1$ denote the size of empty lists. When $\mathcal{A}$ queries random oracle H_i on identity ID, $\mathcal{B}$ answers as follows:
   - If $J_i(ID)$ has been previously defined, then $\mathcal{B}$ returns $J_i(ID)$.
   - Otherwise:
     - For $t_i = J_i$, if $i = 1$ then $\mathcal{B}$ assigns the value $J_1(ID) \leftarrow TA_1$. If $i \neq 1$ then $\mathcal{B}$ assigns the value $J_i \leftarrow P^*_i$.
     - For $t_i \neq J_i$, $\mathcal{B}$ assigns $J_i(ID) \leftarrow H_i(ID)$ and increments the counter $|J_i|$.
     - $\mathcal{B}$ then returns the value $J_i(ID)$
5. For other queries, $\mathcal{B}$ hashes the relevant identity pattern $\boldsymbol{P}$ as a random oracle query with result $\boldsymbol{P'}$. Note that for TA-level queries with $\boldsymbol{ID} = (TA_i)$ the pattern is still transformed as $\boldsymbol{ID'} = (J_1(TA_i))$. $\mathcal{A}$ may query:

- `CreateTA`(TA_i) $\mathcal{B}$ computes $TA'_i = J_1(TA_i)$ and queries its own oracle as `CreateTA`(TA'_i), returning the result.
- `Corrupt`$(\boldsymbol{P})$ If $J(\boldsymbol{P}) \in_* \boldsymbol{P^*}$ then $\mathcal{B}$ aborts because it would have to make an illegal query. Otherwise, $\mathcal{B}$ computes $\boldsymbol{P'} = J(\boldsymbol{P})$ and forwards the query using its own oracle as `Corrupt`$(\boldsymbol{P'})$, returning the result.
- `SetupCoalitionBroadcast`$(TA_i, \mathcal{C})$ For each TA_j in the coalition $\mathcal{C}$, $\mathcal{B}$ computes the hashed TA identity $TA'_j = J_1(TA_j)$ and adds it to coalition $\mathcal{C}'$. $\mathcal{B}$ also computes the hashed identity $TA'_i = J_1(TA_i)$ then forwards the query as `SetupCoalitionBroadcast`$(TA'_i, \mathcal{C}')$, returning the result.
- `SetupCoalitionKeys`$(TA_j, sk_j, \mathcal{C}, \hat{W}_j)$ For each TA_j in the coalition $\mathcal{C}$, $\mathcal{B}$ computes the hashed TA identity $TA'_j = J_1(TA_j)$ and adds it to coalition $\mathcal{C}'$. $\mathcal{B}$ then computes $TA'_j = J_1(TA_j)$ and forwards the query as `SetupCoalitionKeys`$(TA'_j, sk_j, \mathcal{C})$, returning the result.
- `ExtractCoalitionKey`$(\mathcal{C}, v_j, d_{\boldsymbol{ID}})$ For each TA_j in the coalition $\mathcal{C}$, $\mathcal{B}$ computes the hashed TA identity $TA'_j = J_1(TA_j)$ and adds it to coalition $\mathcal{C}'$. $\mathcal{B}$ then computes $\boldsymbol{ID'} = J(\boldsymbol{ID})$ and queries `ExtractCoalitionKey`$(\mathcal{C}', v_j, d_{\boldsymbol{ID'}})$, returning the result.

6. $\mathcal{A}$ ends this stage of the game by outputting a challenge pattern $\widehat{\boldsymbol{P}} = (\widehat{P}_1, ..., \widehat{P}_\ell)$ and challenge coalition $\widehat{\boldsymbol{TA}} = \{\widehat{TA}_1, ..., \widehat{TA}_k\}$ and two equal-length messages (m_0, m_1). If any $J_i(\widehat{P}_i)$ has not yet been defined then $\mathcal{B}$ sets $J_i(\widehat{P}_i) \leftarrow H_i(\widehat{P}_i)$.
7. $\mathcal{B}$ must abort if:
   - if $\ell \neq \ell^*$
   - if there exists $i \in W(\widehat{\boldsymbol{P}})$ such that $i \notin W(\boldsymbol{P}^*)$
   - if $J_i(\widehat{P}_i) \neq P^*_i$
8. If $\mathcal{B}$ has not aborted it outputs the messages (m_0, m_1) to the challenger.
9. The challenger computes the challenge ciphertext C^* by encrypting message m_β for $\beta \xleftarrow{\$} \{0, 1\}$ and returns it to $\mathcal{B}$.
10. $\mathcal{B}$ in turn returns the challenge ciphertext to $\mathcal{A}$, which outputs a bit $\widehat{\beta}$ as a guess for the value of β.
11. In turn, $\mathcal{B}$ outputs $\widehat{\beta}$ as its guess for the value of β.

The algorithm $\mathcal{B}$ wins the IND-sWID-CPA game if $\mathcal{A}$ wins the IND-WID-CPA game and $\mathcal{B}$ does not abort. $\mathcal{B}$ may abort if it has guess the challenge coalition and pattern incorrectly, or if it was forced to make an illegal query. $\mathcal{B}$ uses the t_i's to guess which of $\mathcal{A}$'s queries will be used to define the i^{th} level of its challenge pattern with $t_i = 0$ corresponding to a wildcard.

To avoid $\mathcal{B}$'s making of an illegal query we require that there are no collisions in the hash. For distinct identifiers $ID \neq ID'$ we require that the hash output $H_i(ID) \neq H_i(ID')$. With a random oracle such a collision could only occur by chance. Given $(q_H + 1)$ random oracle queries for a level the probability of a collision has an upper bound of $(q_H + 1)/|ID|$. Then the probability of successfully avoiding a collision has a lower bound of $(1 - (q_H + 1)/|ID|)$. With L independent random oracles, the probability of avoiding a collision on all levels is then $(1 - (q_H + 1)/|ID|)^L$. If a collision occurs then $\mathcal{B}$ will gain no benefit from $\mathcal{A}$'s advantage as it cannot properly simulate the WIBE environment. Of $\mathcal{A}$'s original advantage ϵ, $\mathcal{B}$ may only leverage an advantage of $\epsilon(1 - (q_H + 1)/|ID|)^L$.

We require that $\mathcal{B}$ correctly choose the challenge pattern and coalition in order to win the game. The probability that $\ell = \ell^*$ is $1/L$, the probability that t_i correctly identifies P_i is $1/(q_H + 1)$ for each level $i \leq L$, and the probability of the challenge coalition $\mathcal{C}$ being correct as $1/2^n$. Each of these factors reduce $\mathcal{B}$'s advantage multiplicatively in comparison to $\mathcal{A}$'s. If $\mathcal{B}$ correctly guesses these values and $\mathcal{A}$ never forces $\mathcal{B}$ to make an illegal query then $\mathcal{B}$ has advantage:

$$\epsilon' \geq \left(\frac{\epsilon}{2^n L (q_H + 1)^L}\right) \left(1 - \frac{(q_H + 1)}{|ID|}\right)^L$$

□

6 Chosen-Ciphertext Security

Transforming an adaptive chosen-plaintext secure (CPA) secure IBE scheme into a CCA-secure public-key scheme was originally proposed by Canetti, Halevi, and Katz [?]. Their method was later improved for efficiency by Boneh and Katz [?], and further research by Boyen, Mei, and Waters [?] allowed for transformations with fewer dependencies outside the initial IBE scheme. Each of these methods allows for a parallel transformation of selective-identity CPA-secure IBE schemes in the standard model into nonselective-identity CCA-secure public-key scheme in the random oracle model using a proof similar to the one above.

We present a transformation descended from the Park and Lee's proof [?] which directly transforms a selective-identity CPA-secure $L+1$ level multi-TA WIBE into a selective-identity CCA-secure L level multi-TA WIBE in the standard model. While requiring a one-time signature scheme as in the original Canetti, Halevi, and Katz transformation, the Park and Lee transformation (and our extension thereof) does not require the padding of identities by one bit.

To achieve sWID-IND-CCA chosen-ciphertext security we require the addition of a strongly unforgeable signature scheme *Sig = (SigKeyGen, Sign, Verify)* and a collision resistant hash function mapping verification keys to $\mathbb{Z}_p$ where p is the order of $\mathbb{G}$. For simplicity, we will assume a natural map of verification keys to $\mathbb{Z}_p$ and treat the keys as members of the finite field directly. Recall that $\mathbb{G}$ is the group from which the public parameters of the HIBE are chosen, and that there is a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ mapping onto the message space $\mathbb{G}_T$. In mapping onto a WIBE of one fewer levels, the public parameters for the $L+1$ level are appropriated for the verification code. The new public parameters for the Boneh-Boyen based multi-TA WIBE become:

$$\{g_1, g_2, u_{1,0}, u_{1,1}, u_{2,0}, u_{2,1}, \dots, u_{L,0}, u_{L,1}, u_{v,0}, u_{v,1}\} \in \mathbb{G}$$

$$\text{And the set of } TA\text{s } \{TA_1, TA_2, \dots, TA_n\}$$

The algorithms for key delegation, encryption, and decryption must be altered to incorporate verification codes. The algorithms change as follows:

– $\texttt{KeyGen}(d_{\boldsymbol{ID}}, \boldsymbol{ID'})$ Node $\boldsymbol{ID} = (ID_1, \dots, ID_j)$ generates a secret key $d_{\boldsymbol{ID'}}$ for subordinate node $\boldsymbol{ID'} = (ID_1, \dots, ID_j, ID_{j+1}, \dots, ID_\ell)$ using its private key $d_{\boldsymbol{ID}} = (h, a_1, \dots, a_j)$ as follows:

$$d_{\boldsymbol{ID'}} = (h \cdot \prod_{k=1}^{\ell} (u_{k,0} \cdot u_{k,1}^{ID_k})^{r_k}, a_1 \cdot g^{r_1}, a_2 \cdot g^{r_2}, \dots, a_j \cdot g^{r_j}, g^{r_{j+1}}, g^{r_{j+2}}, \dots, g^{r_\ell})$$

– $\texttt{Encrypt}(m, \boldsymbol{P}, \mathcal{C})$ To encrypt message m to pattern $\boldsymbol{P} = (P_1, P_2, \dots, P_\ell)$ in coalition $\mathcal{C}$ with coalition-specific public key $pk_{\mathcal{C}}$ the sender randomly chooses $t \xleftarrow{\$} \mathbb{Z}_p$. The sender runs the *SigKeyGen* signing key generator algorithm to obtain a signing key S and corresponding verification key K. The sender then computes:

$$\begin{aligned} C_1 &\leftarrow g_1^t \\ C_{2,v} &\leftarrow (u_{v,0} \cdot u_{v,1}^K)^t \\ C_{2,i} &\leftarrow \begin{cases} (u_{i,0} \cdot u_{i,1}^{ID_i})^t & \text{if } i \notin W(P) \\ (u_{i,0}^t, u_{i,1}^t) & \text{if } i \in W(P) \end{cases} \\ C_3 &\leftarrow m \cdot e(pk_{\mathcal{C}}, g_2)^t \end{aligned}$$

$$C \leftarrow (C_1, C_{2,k}, C_3, Sign_K(C)) \text{ for } k \in \{1 \dots \ell, v\}$$

The ciphertext is then sent as:

$$\boldsymbol{CT} = (C, Sign_S(C), V) \text{ where } S \text{ and } V \text{ are related signing and verification keys respectively}$$

– `Decrypt`($\boldsymbol{CT}$, $\mathcal{C}$, $d_{\boldsymbol{ID}}$) A ciphertext $\boldsymbol{CT} = (C, \sigma, V)$ encrypted to pattern P under coalition $\mathcal{C}$ may be decrypted by an entity $\boldsymbol{ID} = \{ID_1, ID_2, \ldots, ID_\ell\} \in^* P$ holding key $d_{\boldsymbol{ID}} = (h, a_1, a2, \ldots, a_\ell)$ specific to coalition $\mathcal{C}$. The decrypting node first verifies that signature σ of ciphertext C is valid under verification key V, if invalid outputting $\perp$. If the verification succeeds the ciphertext is then decrypted by choosing randomly $s \xleftarrow{\$} \mathbb{Z}_p$ and computing:

$$C'_{2,i} \leftarrow \begin{cases} C_{2,i} & \text{if } i \notin W(P) \\ (C_{2,i,0} \cdot C_{2,i,1}^{ID_i}) & \text{if } i \in W(P) \end{cases}$$

$$m' \leftarrow C_3 \frac{\prod_{i=1}^{\ell} e(a_i, C'_{2,i}) \cdot e(C_{2,v}, g_1^s)}{e(C_1, h \cdot (g_1^V \cdot h)^s)}$$

Theorem 4. *Suppose that there exists a polynomial-time attacker $\mathcal{A}$ against the multiple TA IND-sWID-CCA signed-ciphertext Boneh-Boyen WIBE with advantage ϵ, then there exists a polynomial-time attacker $\mathcal{B}$ against the Bilinear Decisional Diffie-Hellman (BDDH) problem with advantage ϵ'.*

Proof Suppose there exists adversary $\mathcal{A}$ against the multi-TA IND-sWID-CCA scheme (with verification codes), we will construct an adversary $\mathcal{B}$ which gains an advantage against the multi-TA IND-sWID-CPA scheme using the algorithm $\mathcal{A}$ as a black box.

For a multi-TA WIBE of maximum depth L consisting of a maximum of k $TA's$, the algorithm $\mathcal{B}$ runs as follows:

1. $\mathcal{B}$ receives BDDH input (g, g^a, g^b, g^c, Z) for some unknown $a, b, c \in \mathbb{Z}_p^*$ and must decide whether $Z = g^{abc}$ and output 1 is true and 0 otherwise.
2. $\mathcal{B}$ initializes $\mathcal{A}$ for the decided number of WIBE levels and *TA*s. $\mathcal{A}$ returns challenges coalition C^* and challenge pattern $\boldsymbol{P}^* = (P_1^*, P_2^*, \ldots, P_\ell^*)$ with $P_i \in \mathbb{G} \cup \{*\}$ that $\mathcal{A}$ intends to attack. If $\boldsymbol{P^*}$ does not extend to the maximum level L then $\mathcal{B}$ randomly chooses elements $(P*_{l+1}, \ldots, P_L^*) \in \mathbb{Z}_p$. Next $\mathcal{B}$ runs *SigKeyGen* to obtain signing key S and verification key K and
3. $\mathcal{B}$ announces public parameters the