Multiple Hierarchy Wildcard Encryption

All Who Want To Play

...

Abstract. expressionism is an art movement typically characterised by its non-realistic representation of non-tangible nouns (such as emotions or situations). It centred in the New York in the post-surrealistic decades after the second World War, and, in particular, around Peggy Guggenheim's gallery "Art of this Century".

Alex wrote:
What's to prevent an attacker setting up his own TA under the name of a real coalition member and then hijacking the update coalition protocol? Are we going to assume trusted distribution of master public keys?

Chris wrote:
In BBG, UpdateCoalitionBroadcast has a dependency on the correlation between the public messages and the secret key. In BB, the power α_i to which we raise g_2 is the same in all messages, but irretrievable due to the random element $r_{i,j}$. UpdateCoalitionBroadcast is inherently insecure if if doesn't require knowledge of a TA's secret key, e.g. if we allow a TA to change α_i and all $r_{i,j}$'s at the same time and in the clear. A non-corrupted TA should be safe from hijack if either UpdateCoalitionBroadcast messages are sent encrypted to members of the existing coalition or if broken into two algorithms: first an AdmitNewMembersBroadcast which sends $w_{i,j}$ messages for $TA_i \in \mathcal{C} \cap \mathcal{C}'$ and $TA_j \in \mathcal{C}' \setminus \mathcal{C}$ and secondly a (possibly broadcast-) encrypted CoalitionKeyRefresh to update keys in an existing coalition (perhaps just prior to expansion or after contraction).

How is JoinCoalitionBroadcast different from SetupCoalitionBroadcast in terms of inputs and outputs? In other words, could joining members of a coalition run SetupCoalBcast for the expanded coalition with equivalent results? (eliminating one algorithm)

Similarly, aren't UpdateCoalitionKeys and JoinCoalitionKeys essentially the same as SetupCoalitionKeys once the messages $w_{i,j}$ are out there? Do we keep them separate for generality, and thus need to also keep separate UpdateCoalitionKey and ExtractCoalitionKey algorithms?

Should the coalition key adjustment (u_i) and the coalition key ($c_{\boldsymbol{ID}}$) be indexed by coalition ($\mathcal{C}$) in addition to the identity ($\boldsymbol{ID}$)?

Why do we record which TAs have been corrupted?

1 Syntax

A multi-hierarchy WIBE consists of the following PPT algorithms/protocols:

– $\texttt{Setup}(1^k, TA_i)$: This algorithm is run once by TA_i and outputs a public key and private key for that TA $(pk_i, sk_i) \leftarrow \texttt{Setup}(1^k, TA_i)$.

- `SetupCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j) \forall TA_j \in \mathcal{C} \setminus \{TA_i\})$: This algorithm calculates messages $w_{i,j}$ to be distributed from TA_i to each $TA_j \in \mathcal{C} \setminus \{TA_i\}$ in order to initiate the coalition $\mathcal{C} = \{TA_a, TA_b, \ldots, TA_i, \ldots, TA_k\} \subset TA_1, TA_2, \ldots, TA_n$. After each $TA_i \in \mathcal{C}$ runs this algorithm, coalition keys may be generated using the `SetupCoalitionKeys` algorithm.
- `SetupCoalitionKeys`$(TA_j, sk_j, (TA_i, w_{i,j}) \forall TA_j \in \mathcal{C} \setminus \{TA_i\})$: 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 u_j to be broadcast to every member of TA_j's hierarchy in order to run extract or update

The system should be able to dynamically update the coalition. We may wish to change a coalition $\mathcal{C}$ into a coalition $\mathcal{C}'$. We assume that members $\mathcal{C} \cap \mathcal{C}'$ execute the `UpdateCoalition` algorithms, while new members $\mathcal{C} \setminus \mathcal{C}'$ execute the `JoinCoalition` algorithms. Excluded members $\mathcal{C}' \setminus \mathcal{C}$ are simply informed that they are no longer members of the coalition.

- `UpdateCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j) \forall TA_j \in \mathcal{C}' \setminus \{TA_i\})$: This algorithm updates an existing coalition $\mathcal{C}$ containing TA_i to become a new coalition $\mathcal{C}' = (TA_{a'}, TA_{b'}, \ldots, TA_{n1}, \ldots, TA_{m'})$. This algorithm outputs a list of messages $w'_{i,j}$ to be sent from TA_i to TA_j. It should be noted that some $w'_{i,j}$ may be empty, particularly if $TA_j \in \mathcal{C} \cap \mathcal{C}'$.
- `JoinCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j) \forall TA_j \in \mathcal{C}')$: A new authority TA_i which is joining an existing coalition to form a new coalition $\mathcal{C}'$ uses this algorithm to produce a series of messages $w_{i,j}$ to be sent from TA_i to TA_j.
- `UpdateCoalitionKeys`$(TA_i, pk_i, sk_i, (TA_j, pk_j, w_{j,i}) \forall TA_j \in \mathcal{C}')$: The algorithm completes the updating of the coalition for existing members. After every member TA_j of the coalition has provided a (non-empty) message $w_{j,i}$ for TA_i. It outputs a message u_i to be broadcast to every member of its hierarchy.
- `JoinCoalitionKeys`$(TA_i, pk_i, sk_i, (TA_j, pk_j, w_{j,i}) \forall TA_j \in \mathcal{C}')$: This algorithm completes the joining of an existing coalition for new members. After every member TA_j of the coalition has provided a (non-empty) message $w_{j,i}$ for TA_i, this algorithm outputs a message u_i to be broadcast to every member of its hierarchy.

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`$(\boldsymbol{ID} \in TA_i, u_i, d_{\boldsymbol{ID}})$: 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`: It does what you'd expect...

2 Security Model

The security model is parameterized by a bit b involves a PPT attacker $\mathcal{A}$ which is initially given the input 1^k and access to the following oracles:

– $\texttt{CreateTA}(TA)$: The oracle computes $(pk_i, sk_i) \xleftarrow{\$} \texttt{Setup}(1^k, TA)$ for the TA identity TA_i and returns pk_i. This oracle can only be queried once for each identity TA_i.

– $\texttt{SetupCoalitionBroadcast}(TA_i, \mathcal{C})$: This oracle runs the SetupCoalitionBroadcast algorithm for coalition $\mathcal{C}$ containing TA_i and returns messages $w_{i,j} \forall TA_j \in \mathcal{C} \setminus TA_i$.

– $\texttt{SetupCoalitionKeys}(TA_i, w_{j,i} \forall TA_j \in \mathcal{C} \setminus TA_i)$: 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.

– UpdateCoalition oracles are similar to the above...

– $\texttt{Corrupt}(\boldsymbol{ID})$: 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 and records TA_i is corrupt.

– $\texttt{UserDecrypt}(\boldsymbol{ID}, C^*)$: This oracle decrypts the ciphertext with the decryption key $d_{\boldsymbol{ID}}$.

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

– $\texttt{Test}(\mathcal{C}^*, P, m_0, m_1)$: 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 attacker terminates by outputting a bit b'. The attacker's advantage is defined to be:

$$Adv_{\mathcal{A}}^{\texttt{IND}}(k) = |Pr[b' = 1|b = 1] - Pr[b' = 1|b = 0]|$$

The disallowed oracle queries:

1. A Corrupt query for any $\boldsymbol{ID}$ in the test coalition matching pattern P
2. A Corrupt query for any $\boldsymbol{ID}$ in the test coalition 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

3 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:

– $\texttt{CreateTA}(TA_i)$: 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 $g_2^{\alpha_i}$.

– $\texttt{Extract}(\boldsymbol{ID}, ID', d_{\boldsymbol{ID}})$: Under $TA_i = ID_0$, the first level identity is $\boldsymbol{ID} = (ID_0, ID_1)$. The TA generates $r_0, r_1 \xleftarrow{\$} \mathbb{Z}_p$ and computes the key $d_{\boldsymbol{ID}} = (h, a_0, a_1)$ where $h \leftarrow g_2^{\alpha_i}(u_{0,0} \cdot u_{0,1}^{ID_0})^{r_0}(u_{1,0} \cdot u_{1,1}^{ID_1})^{r_1}$, $a_0 \leftarrow g_1^{r_0}$ and $a_1 \leftarrow g_1^{r_1}$. An identity $(ID_1, \ldots, ID_\ell)$ with private key $(h, a_0, a_1, \ldots, a_\ell)$ 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 $a_{\ell+1} \leftarrow g_1^{r_{\ell+1}}$.

– $\texttt{SetupCoalitionBroadcast}(TA_i, \mathcal{C})$: 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 $u_i \leftarrow ((w_{j,i})) \forall TA_j \in \mathcal{C} \setminus TA_i$

– `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

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

$$a_0' \leftarrow a_0 \prod_{j=1}^{k} w_{j,1} = g_1^{r + \sum r_j}$$

– `UpdateCoalitionBroadcast` and `JoinCoalitionBroadcast`: For persistent and joining members respectively of an updated coalition. Members may use `SetupCoalitionBroadcast` with respect to the new coalition $\mathcal{C}'$.

– `UpdateCoalitionKeys` and `JoinCoaltitionKeys`: For persistent and joining members respectively of an updated coalition. Members may use `SetupCoalitionKeys` with respect to the updated coalition $\mathcal{C}'$ and the corresponding broadcasts for the new coalition.

– `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 $(ID_0, \ldots, ID_\ell)$, 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'.

We define a HIBE in exactly the same way except that we remove the possibility that the pattern can contain 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. $\pi(i) = i - |W(P_{\le i}^*)|$. $\mathcal{B}$ 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.
   – `SetupCoalitionBroadcast`: $\mathcal{B}$ forwards this request to its own oracle and returns the response.
   – `JoinCoalitionBroadcast` and `UpdateCoalitionBroadcast`: $\mathcal{B}$ forwards a `SetupCoalitionBroadcast` request to its own oracle and returns the response.
   – `SetupCoalitionKeys`: $\mathcal{B}$ forwards this request to its own oracle and returns the response.
   – `UpdateCoalitionKeys` and `JoinCoaltitionKeys`: $\mathcal{B}$ forwards a `SetupCoalitionKeys` 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

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

       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

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

       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

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

       $\mathcal{B}$ 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. $\mathcal{A}$ computes the challenge parameters

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

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 queries, then $\mathcal{B}$ answers queries as follows:
   - `CreateTA`(TA): If $TA \in TA^*$ then $\mathcal{B}$ returns mpk_{TA}. If $TA \notin TA^*$ then $\mathcal{B}$ generates $\beta_{TA} \xleftarrow{\$} \mathbb{Z}_p$ and returns $g_1^{\beta_{TA}}$.
   - `CorruptTA`(TA): For this query to be valid, we have $TA \notin TA^*$; hence, $\mathcal{B}$ returns $g_2^{\beta_{TA}}$.
   - `SetupCoalitionBroadcast`$(TA, (TA_1, \ldots, TA_n))$: For this to be a valid request, we must have $TA \notin \{TA_1, \ldots, TA_n\}$. For $TA \neq TA_1^*$, $\mathcal{B}$ can compute the private key directly; hence, $\mathcal{B}$ can return the correct value using the appropriate algorithm. For $TA = TA_1^*$, $\mathcal{B}$ 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 $((TA_i, w_i))_{i=1}^n$.
   - `CorruptUser`$(TA, \boldsymbol{ID})$: For this to be a valid query, 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.

– $\texttt{Test}(m_0, m_1)$: 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 `CorruptUser` oracle 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 `CorruptUser` oracle. To show that if the simulator doesn't abort, the `CorruptUser` returns a correct key, not that it suffices to show that

$$msk_{TA_1^*}\left(u_{j,0} \cdot u_{j,1}^{ID_j}\right)^r = g_2^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} \cdot g_2^{-\sum_{i=2}^{n^*}\beta_i} \qquad \text{for} \qquad r = -\frac{b}{\alpha_{0,1}(ID_j - ID_j^*)}\,.$$

We note that $msk_{TA_1^*} = g_2^{a - \sum_{i=2}^{n^*}\beta_i}$. The correct decryption key is

$$\begin{aligned}
msk_{TA_1^*}\left(u_{j,0} \cdot u_{j,1}^{ID_j}\right)^r &= g_2^{a-\sum_{i=2}^{n^*}\beta_i}\left(g^{\alpha_{j,0}} \cdot (g^a)^{-\alpha_{j,1}ID_j^*} \cdot (g^a)^{\alpha_{j,1}ID_j}\right)^{-\frac{b}{\alpha_{j,1}(ID_j - ID_j^*)}} \\
&= g^{ab} g_2^{-\sum_{i=2}^{n^*}\beta_i}\left(g^{\alpha_{j,0}} \cdot (g^a)^{\alpha_{j,1}(ID_j - ID_j^*)}\right)^{-\frac{b}{\alpha_{j,1}(ID_j - ID_j^*)}} \\
&= g^{ab} g_2^{-\sum_{i=2}^{n^*}\beta_i}(g^b)^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} g^{-ab} \\
&= g_2^{\frac{-\alpha_{j,0}}{\alpha_{j,1}(ID_j - ID_j^*)}} g_2^{-\sum_{i=2}^{n^*}\beta_i}
\end{aligned}$$

Hence, $\mathcal{B}$'s simulation returns a correct decryption key. A similar calculation shows that 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

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

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 $\epsilon - q_K/p$. □

4 Still to be done

– This whole treatment ignores the update coalition protocols.

- The use of ROM to turn selective-identity into non-selective identity should be proven.
- CHK transform.
- Can we use the same techniques to prove the security of a multiple TA BBG-WIBE and a multiple TA Waters-WIBE? Can we do it for some general treatment? The Waters version would be particular good as it provides CPA security in the non-selective-identity model.
- What about the Gentry-Halevi HIBE?
- What about the other things on the list, especially node-to-node coalitions?