Multiple Hierarchy Wildcard Encryption

1 Syntax

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

- `CreateTA`(TA_i): This algorithm is run once by TA_i, it generates a public key and private key for that TA $(pk_i, sk_i) \xleftarrow{\$}$ `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 $TA_j \in \mathcal{C}\setminus\{TA_i\}$ 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`$(TA_j, sk_j, (TA_i, w_{i,j})\forall TA_j \in \mathcal{C})$: 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 `InviteCoalition` algorithms, while new members $\mathcal{C} \setminus \mathcal{C}'$ execute the `JoinCoalition` algorithms. Excluded members $\mathcal{C}' \setminus \mathcal{C}$ are excluded and unable to communicate under the new coalition.

- `InviteCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j)\forall TA_j \in \mathcal{C}')$: For an existing coalition $\mathcal{C}$ containing TA_i to become a new coalition $\mathcal{C}' = (TA_{a'}, TA_{b'}, \ldots, TA_i, \ldots, TA_{m'})$ with new members in the set $\mathcal{C}' \setminus \mathcal{C}$. This algorithm outputs a list of messages $w'_{i,j}$ to be sent from persisting coalition member TA_i to all new coalition members $TA_j \in \mathcal{C}' \setminus \mathcal{C}$.
- `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 each $TA_j \in \mathcal{C}'$.
- `UpdateCoalitionKeys`$(TA_i, pk_i, sk_i, (TA_j, pk_j, w'_{j,i})\forall TA_j \in \mathcal{C}' \setminus \{TA_i\})$: TA_i runs `SetupCoalitionKeys` for the new coalition $\mathcal{C}'$ using the new messages $w'_{i,j}$. Note that this step allows the creation of a coalition $\mathcal{C}'$ that excludes previous members $TA_k \in \mathcal{C} \setminus \mathcal{C}'$
- `RefreshCoalitionBroadcast`$(TA_i, sk_i, (TA_j, pk_j)\forall TA_j \in \mathcal{C}')$: Allows TA_i to refresh it's key broadcast messages $w_{i,j}$. TA_i runs `SetupCoalitionBroadcast` for the coalition $\mathcal{C}'$ outputing new messages $w'_{i,j}$. For the refresh to take effect, each TA must then run `UpdateCoalitionKeys`.

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:

- `CreateTA`(TA): The oracle computes $(pk_i, sk_i) \xleftarrow{\$}$ `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.
- `InviteCoalition`, `JoinCoalition`, `UpdateCoalitionKeys`, and `RefreshCoalitionBroadcast` oracles are similar to the above...
- `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.
- `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 $\mathcal{C}$.
- `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:

- `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

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

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

– `InviteCoalition`, `JoinCoalition`, `RefreshCoalitionBroadcast` are similar to `SetupCoalitionBroadcast`.

– `UpdateCoalitionKeys` is similar to `SetupCoalitionKeys`.

– `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

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

– `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'.

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. $\pi(i) = i - |W(P_{\leq i}^*)|$. $\mathcal{B}$ 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:

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

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 $ID_i \rightarrow ID_{\pi(i)}$.
   – `SetupCoalitionBroadcast`: $\mathcal{B}$ forwards this request to its own oracle and returns the response.
   – `RefreshCoalitionBroadcast`, `InviteCoalitionBroadcast`, and `JoinCoalitionBroadcast`: $\mathcal{B}$ forwards an appropriate `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`: $\mathcal{B}$ forwards an appropriate `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

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

       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

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

       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

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

$\mathcal{B}$ 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

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

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_i): If $TA_i = TA^*$ then $\mathcal{B}$ returns pk_{TA}. If $TA \neq TA^*$ then $\mathcal{B}$ generates $\beta_i \xleftarrow{\$} \mathbb{Z}_p$ and returns $pk_i = g_1^{\beta_i}$.
   - `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

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

   $\mathcal{B}$ 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

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

   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. 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

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

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 $\epsilon - q_K/p$. □

4 Features

– Completed

    - Multi-TA HIBE
    - Wildcard one-to-many addressing in multi-TA WIBE
    - Instantiation in Boneh-Boyen-Goh
    - Security proof of instantiation with coalition updates allowed
    - Coalition updates
    - Within TA broadcast messages
    - Efficient subordinate key updates for coalition updates
  - In Process
    - CHK transform from CPA to CCA
    - Subordinate-initiated coalitions (Dual-HIBE $\rightarrow$ Dual-WIBE)
    - Instatiation and proof based on Waters HIBE for non-selective ID-CPA
    - Waters + CHK for true IND-WID-CCA
  - Desired
    - Many levels without loss of security (Gentry's fabulous matrix method?)
    - TA-anonymity of message (sender anonymity? Sender and recipients?)
    - Reduced SetupCoalition requirements (currently $O(n^2)$)
    - Security proof on abstracted model (e.g. all commutative blinding schemes)

5 Comments

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?

6 Orphaned Discussion from Prior Writing

6.1 Introduction & Motivation

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 security of it's secret keys. This secrecy requirement is natural in the setting of dynamic coalition forming and dissolution.

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. In this paper, multiple recipients may decrypt the same message through the use of one or more ”wildcards”. A wildcard is a special character that may be used in an address in lieu of specifying a particular aspect of an identity, allowing anyone matching the non-wildcard portion to read the message.

In the single-TA case, 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 hierarchy of a TA could be as simple as an email address, every entity under the TA (say ”school.edu”) has a name. Considered as an IBE setting, one might desire to send a single message to an address of the form ”name@school.edu”, but a message pertaining to everyone at the school might be better sent to *@school.edu in a wildcard IBE setting.

In the multi-TA case, we could consider schools as separate Trusted Authorities. These TA's could agree upon a protocol such as the one described in this paper to allow secure one-to-many hierarchical IBE. As such, a message addressed to the leadership of universities could be sent to ”provost@*.edu” or a message for computer system administrators might be addresses sysadmin@*.*. 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”, 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.

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.

An interesting consequence and possible drawback of using a hierarchical IBE is that messages sent to a subordinate entity may be decrypted by direct ancestors of that entity. Explicitly, a message to ”bob@school.edu” could be read by the entity ”school.edu”, but not by ”eve@school.edu” or by an entity at ”university.edu”. The ability to decrypt is possible because an entity's ancestors are able to generate new secret keys for subordinates, so ”school.edu” can make keys for ”alice@school.edu”, ”bob@school.edu”, ad infinitum. With the use of a centralized key distributor it may be possible to avoid this vulnerability by choosing keys in such a way as to isolate the levels of hierarchy from each other.

6.2 Reconfiguration

Should the members of the coalition of TA's change, 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. The security of reconfiguration relies on only the security assumptions made previously: difficulty of the BDDH under e and the discrete logarithm problem in $\mathbb{G}^+$.

Once a new coalition is determined, the members choose β_i and $s_{i,j}$ to replace α_i and $r_{i,j}$ and publish values of $(\beta_i \cdot g_1)$ for all i and $(\beta_i \cdot g_2 + s_{i,j} \cdot (u_{1,0} + ID_{j,1} \cdot u_{1,1}))$for all $i \neq j$. As before, each TA$_j$ may then calculate their private key based on the withheld $i = j$ value, and replace the previous secret key $(a_0, a_1) = (((\Sigma\alpha_i) \cdot g_2 + \Sigma r_{i,j} \cdot (u_{1,0} + ID_{j,1} \cdot u_{1,1})), \Sigma r_{i,j} \cdot g_1)$ with the new secret key $d_{TA_j} = (b_0, b_1) = (\Sigma\beta_i \cdot g_2 + \Sigma s_{i,j} \cdot (u_{1,0} + ID_{j,1} \cdot u_{1,1}), \Sigma s_{i,j} \cdot g_1)$ (summation runs on the i index). The public key $\Sigma\beta_i \cdot g_1$ is also calculable from the public information during this setup and should be assumed available to all TA's and subordinates. To disseminate the new private information, the TA's could calculate an adjustment term:

$$(b_0 - a_0, b_1 - a_1) = ((\Sigma\beta_i - \Sigma\alpha_i) \cdot g_2 + (\Sigma s_{i,j} - \Sigma r_{i,j}) \cdot (u_{1,0} + ID_{j,1} \cdot u_{1,1}), (\Sigma s_{i,j} - \Sigma r_{i,j}) \cdot g_1)$$

The savings in reconfiguration costs come from TAs needing to do a round of communication at the highest level and then allowing a broadcast of information to subordinates rather than being required to send a separate message to each subordinate using their unique private keys.

6.3 Properties of the Scheme

Our setup enables a group of TA's to use identity based encryption in a mixed trust situation. The "super secret" $\Sigma\alpha_i \cdot g_2$ is inseparable from the secret values $r_{,i,j}$ and α_i that the TA's do not share. In this way, no TA has enough information to decrypt messages destined for another. It should be noted that we inherit from a hierarchical cryptosystem the property that every subordinate's key is derived from its superior. From this it follows that every superior may read messages sent to any subordinates in the hierarchy,not just immediate subordinates. If there were an entity above the TA's, corresponding to the "super secret",it would be able to read all messages sent in the system; for this reason we force this value to be difficult to recover by any TA or group of TA's.

Collaborative secret creation requires revealing many pieces of information but results in a highly collusion-resistant secret. The generation process requires each TA to reveal n pieces of information which are each related to the secret α_i that the TA must not reveal. From the security of each TAs α_i, any two TAs colluding have no advantage over a single TA in recovering the unknown α_i. Because no one but the target TA has this information, bringing more TAs into an attacking coalition would bring no advantage either. This can be seen by imagining the worst case scenario, $(n-1)$ TAs attacking the secret of the lone target TA. In this case, the attacker has knowledge of all α_is and $r_{i,j}$s except for α_k and $r_{k,k}$ for target TA$_k$. The attackers have knowledge of $((\Sigma\alpha_i) \cdot g_2 + r_{i,k} \cdot (u_{1,0} + ID_{k,1} \cdot u_{1,1}))$ for all $i \neq k$. From their secret and public pieces of information the attackers would need to calculate $(\alpha_k \cdot g_2 + r_{k,k} \cdot (u_{1,0} + ID_{k,1} \cdot u_{1,1}))$, but they cannot because $r_{k,k}$ is secret and $alpha_k$ is occluded by the discrete logarithm problem. Similarly, the use of independent secrets in the setup ensures that no TA can read anothers messages without knowledge of either $\Sigma\alpha_i) \cdot g_2$ or the target TAs individual α_i. Solving either of these problems is equivalent to solving the Bilinear Diffie-Hellman problem. In the first case the attacker must be able to separate the two terms of the published $(\alpha_i \cdot g_2 + r_{i,j} \cdot (u_{1,0} + (ID_{j,1}) \cdot u_{1,1}))$, which relies on knowledge of $r_{i,j}$s. In the second case the attacker must recover α_i, from solving a discrete logarithm on public information such as $\alpha_i \cdot g_1$.

The parameters used in encryption and decryption in this scheme does not depend on the TA membership of the sender or recipient(s) of a message (except in the sense of recipient identity). There are no TA-specific pieces of public information necessary as is sometimes the case in identity-based encryption with multiple TAs. As such, ciphertext messages may be unencumbered by information about the sender. The system is truly identity-based because a ciphertext depends only on the identity of the recipient(s). One interesting question in this scenario is the security of sending a message across all TAs. Is it possible for an attacker to masquerade as an additional TA by using the public information available? It is interesting to note that this scheme allows for broadcast of information to many (or all) identities within TAs. The incorporation of wildcards into the scheme comes at the (low) cost of having more public parameters for the crypto-system.