APPENDIX A
PROOF OF THEOREM 2

Proof. As defined in section 3, the adversary $\mathcal{M}$ acts as a dishonest user. We start by observing that since the session key of the test session is computed as $sk = H_2(\sigma)$ for some 5-tuple σ.

Let $\mathcal{M}$ be any AKE adversary against CB-AKE protocol. We will show how to use the ability of $\mathcal{M}$ to construct an solver $\mathcal{S}$ to solve the ECGDH problem. $\mathcal{S}$ first chooses $P_0 \in G$ at random, sets P_0 as the CA's master public key P_{pub}, selects the system parameter params $= \{F_p, E/F_p, G, P, P_{pub}, H_2\}$, and sends params to $\mathcal{M}$. Since $\mathcal{H}_2$ and $\mathcal{H}$ is a random oracle, $\mathcal{M}$ can only distinguish a session key from a random string with probability significantly greater than $\frac{1}{2}$ in one of the following cases:

- Case 1 **Guessing attack:** $\mathcal{M}$ correctly guesses the session key.
- Case 2 **Key replication:** $\mathcal{M}$ forces two distinct non-matching sessions to have the same session key. In that case $\mathcal{M}$ can select one of the sessions as the test session and query the key of the other session.
- Case 3 **Forging attack:** At some point in its run, the adversary $\mathcal{M}$ queries $\mathcal{H}_2$ on the value $\left(ID_I, ID_J, K_1^A, K_2^A, K_3^A\right)$ in the test session owned by $\mathcal{A}$ communicating with $\mathcal{B}$.

The probability of guessing the output of $\mathcal{H}_2$ and $\mathcal{H}$ are $\mathcal{O}\left(1/2^\lambda\right)$, which is negligible; thus **Case 1** can be ruled out.

The input to the key derivation includes all information contained in the session identifier. Since two non-matching sessions cannot have the same communicating parties and ephemeral public keys (except with negligible probability), key replication is equivalent to finding a collision for $\mathcal{H}_2$ and $\mathcal{H}$. Therefore **Case 2** occurs with probability $\mathcal{O}\left(s(\lambda)^2/2^\lambda\right)$, which is negligible.

It remains to consider **Case 3-forging attacks**. The rest of this section is devoted to the analysis of this event. Following the standard approach we will show how to construct a ECGDH challenger, that uses an adversary $\mathcal{M}$ that succeeds with non-negligible probability in a forging attack. Let E be an elliptic curve defined over a finite field $GF(p)$, P be a point on E of order n, and X, Y and Z be points on E such that $X = x \cdot P$, $Y = y \cdot P$ and $Z = z \cdot P$ for some unknown $x, y, z \in [0, n-1]$. The challenger $\mathcal{S}$ is given a P, X, Y and an oracle access to $ECDDH(\cdot,\cdot,\cdot)$. The challenger is also given a $ECDDH$ oracle which on input (P, X, Y, Z), returns 1 if $Z = ECCDH\left(P, X, Y\right)$ and 0 otherwise. $\mathcal{S}$ simulates the game outlined above. During the game, $\mathcal{S}$ has to answer all queries of the adversary $\mathcal{M}$.

The following two sub-cases should be considered.

- $C_{3.1}$ The Test session has a matching session owned by another honest party.
- $C_{3.2}$ No honest party owns a session matching with the Test session.

A.1 The analysis of case $C_{3.1}$

Assume that $\mathcal{M}$ selects a test session for which the matching session exists. Then $\mathcal{S}$ modifies the experiment as follows.

$\mathcal{S}$ selects at random matching sessions executed by some honest parties $\mathcal{A}$ and $\mathcal{B}$ (in fact, $\mathcal{S}$ selects two sessions at random and continues only if they are matching - $\mathcal{S}$ successfully guesses them with probability $2/k^2$). Denote by $comm_A$ and $comm_B$ the communications sent by the respective parties in these matching sessions. When either of these sessions is activated, $\mathcal{S}$ does not follow the protocol.

Instead, there are three cases as follows:

- $C_{3.1.1}$ $\mathcal{S}$ generates t_A and t_B normally but sets $comm_1^A \leftarrow X_1$ (in place of T_A) and $comm_1^B \leftarrow Y_1$ (in place of T_B).
- $C_{3.1.2}$ $\mathcal{S}$ generates d_A, r_A and d_B, r_B normally, and it also has $e_A = H_n(Cert_A)$ and $e_B = H_n(Cert_B)$ normally. But it sets $comm_2^A$ and $comm_2^B$ in two ways. The first method is $\mathcal{S}$ sets $comm_2^A \leftarrow X_2$ (in place of Q_A) and $comm_2^B \leftarrow Y_2$ (in place of Q_B). The second method is $\mathcal{S}$ sets $R_A \leftarrow X_2$, $comm_2^A \leftarrow e_A X_2 + e_A R_{CA}^A + P_0$ (in place of $e_A r_A P + e_A R_{CA}^A + P_0$)and $R_B \leftarrow Y_2$, $comm_2^B \leftarrow e_B Y_2 + e_B R_{CA}^B + P_0$(in place of $e_B r_B P + e_B R_{CA}^B + P_0$).
- $C_{3.1.3}$ $\mathcal{S}$ generates t_A, d_A, r_A and t_B, d_B, r_B normally and it also has $e_A = H_n(Cert_A)$ and $e_B = H_n(Cert_B)$ normally. But it sets $comm_3^A$ and $comm_3^B$ in two ways. The first method is $\mathcal{S}$ sets $comm_3^A \leftarrow X_3$ (in place of $\left(t_A + d_A\right)P$ and $comm_3^B \leftarrow Y_3$ (in place of $\left(t_B + d_B\right)P$). The second method is $\mathcal{S}$ sets $T_A \leftarrow X_3^1$, $R_A \leftarrow X_3^2$, $comm_3^A \leftarrow X_3^1 + e_A X_3^2 + e_A R_{CA}^A + P_0$ (in place of $t_A P + e_A r_A P + e_A R_{CA}^A + P_0$) and $T_B \leftarrow Y_3^1$, $R_B \leftarrow Y_3^2$, $comm_3^B \leftarrow Y_3^1 + e_B Y_3^2 + e_B R_{CA}^B + P_0$ (in place of $t_B P + e_B r_B P + e_B R_{CA}^B + P_0$).

  With probability $1/k^2$ $\mathcal{M}$ picks one of the selected sessions as the test session and another as its matching session. We claim that if $\mathcal{M}$ wins in the forging attack, $\mathcal{S}$ can solve the ECCDH challenge. Indeed, the supposed session key for the selected session is $H_2(\sigma)$, where the 5-tuple σ includes the value $ECCDH(X_1, Y_1), ECCDH(X_2, Y_2), ECCDH(X_3, Y_3)$ or $ECCDH(X_1, Y_1)$, $ECCDH(e_A X_2 + e_A R_{CA}^A + P_0, e_B Y_2 + e_B R_{CA}^B + P_0)$, $ECCDH(X_3^1 + e_A X_3^2 + e_A R_{CA}^A + P_0, Y_3^1 + e_B Y_3^2 + e_B R_{CA}^B + P_0)$.

  To win, $\mathcal{M}$ must have queried σ to the random oracle H_2.

If the selected session is indeed the test session, $\mathcal{M}$ is allowed to reveal a subset of $\{r_A, r_B, d_A, d_B, t_A \; and \; t_B\}$, but it is not allowed to reveal both (r_A, d_A, t_A) or both (r_B, d_B, t_B). We observe that in this case, the only way that $\mathcal{M}$ can distinguish this simulated eCK experiment from a true eCK experiment is if $\mathcal{M}$ queries (r_A, d_A, t_A) or (r_B, d_B, t_B) (this way, $\mathcal{M}$ will find out that $comm_{\mathcal{A}}$ and $comm_{\mathcal{B}}$ were not computed correctly). Proposition probability that $\mathcal{M}$ makes such queries is at most

$$2n \cdot \mathbf{Adv}^{\mathrm{ECDLOG}}(\mathcal{T})$$

for some discrete logarithm solver $\mathcal{T}$.

Therefore (assuming that $\mathcal{M}$ always selects a test session which has a matching session)

$$\mathbf{Adv}^{\mathrm{ECGDH}}(\mathcal{S}) \geq \frac{2}{k^2} \cdot \mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{CBAP}}(\mathcal{M}) - 2n \cdot \mathbf{Adv}^{\mathrm{ECDLOG}}(\mathcal{T}) - O\left(\frac{k^2}{2^\lambda}\right).$$

Note that in this case $\mathcal{S}$ doesn't make any queries to the ECDDH oracle and runs in time $O(t)$.

A.2 The analysis of case $C_{3.2}$

Now assume that $\mathcal{M}$ selects a test session for which no matching session exists. In this case $\mathcal{S}$ modifies the experiment as follows.

A.2.1 $C_{3.2.1}$: assume that $\mathcal{B}$ is the owner

$\mathcal{S}$ selects a random party $\mathcal{B}$ and sets $Q_B \leftarrow X_2$ as its long-term public key, it also sets $R_B \leftarrow X_2^*$ as its long-term public point. Note that $\mathcal{S}$ doesn't know long-term secret key corresponding to this long-term public key, and $\mathcal{S}$ also doesn't know long-term secret number via this long-term public point. Thus it cannot properly simulate eCK sessions executed by $\mathcal{B}$. $\mathcal{S}$ handles eCK sessions executed by $\mathcal{B}$ as follows.$\mathcal{S}$ randomly selects t_B, picks h_2 and h_3 at random from Z_q^* and computes $e_B = H_n(Cert_B)$.

$\mathcal{S}$ sets $comm_1^{\mathcal{B}} = t_B \cdot P$, $comm_2^{\mathcal{B}} = h_2 \cdot P$ instead of $ECDLOG(X_2)P$ or $e_B ECDLOG(X_2^*)P + e_B R_{CA}^B + P_0$, $comm_3^{\mathcal{B}} = h_3 \cdot P$ instead of $(t_B + ECDLOG(X_2))\, P$ or $t_B P + e_B ECDLOG(X_2^*)P + e_B R_{CA}^B + P_0$. $\mathcal{S}$ sets a session key sk (which is supposed to be $H_2(ECCDH(t_B P, comm_1^{\mathcal{C}}), ECCDH(X_2, comm_2^{\mathcal{C}}), ECCDH(t_B P + X_2, comm_3^{\mathcal{C}}), \mathcal{B}, \mathcal{C}))$ or $H_2(ECCDH(t_B P, comm_1^{\mathcal{C}}),\ ECCDH(e_B X_2^* + e_B R_{CA}^B + P_0, comm_2^{\mathcal{C}}),\ ECCDH(t_B P + e_B X_2^* + e_B R_{CA}^B + P_0, comm_3^{\mathcal{C}}, \mathcal{B}, \mathcal{C}))$ to be a random value.

Note that $\mathcal{S}$ can handle session key and ephemeral secret key reveals by revealing sk, t_B, but cannot handle long-term secret key reveals or long-term secret value reveals.

If $\mathcal{C}$ is an adversary-controlled party, $\mathcal{M}$ can compute the session key on its own, reveal the session key sk and detect that it is fake. To address this issue, $\mathcal{S}$ watches $\mathcal{M}$'s random oracle queries and if $\mathcal{M}$ ever queries $(Z_1, Z_2, Z_3, \mathcal{B}, \mathcal{C})$ to H_2 (for some $Z_1, Z_2, Z_3 \in G$), $\mathcal{S}$ checks if

$$ECDDH\left(t_B P, comm_1^{\mathcal{C}}, Z_1\right) = 1,$$
$$ECDDH\left(X_2, comm_2^{\mathcal{C}}, Z_2\right) = 1,$$
$$ECDDH\left(t_B P + X_2, comm_3^{\mathcal{C}}, Z_3\right) = 1$$

or

$$ECDDH\left(t_B P, comm_1^{\mathcal{C}}, Z_1\right) = 1,$$
$$ECDDH\left(e_B X_2^* + e_B R_{CA}^B + P_0, comm_2^{\mathcal{C}}, Z_2\right) = 1,$$
$$ECDDH\left(t_B P + e_B X_2^* + e_B R_{CA}^B + P_0, comm_3^{\mathcal{C}}), Z_3\right) = 1$$

and if yes, replies with the session key sk. Similarly, on the computation of sk, $\mathcal{S}$ checks if sk should be equal to any previous response from the random oracle. Because of these checks $\mathcal{S}$ runs in quadratic time of the number of random oracle's queries.

$\mathcal{M}$ cannot detect that it is in the simulated eCK experiment unless it either queries $\mathcal{H}_2$ or reveals a long-term secret key and long-term secret number of $\mathcal{B}$. The first event reveals $ECDLOG\,(X_2)$,$ECDLOG\,(X_2^*)$and allows $\mathcal{S}$ to solve the $ECCDH$ problem, it happens with probability at most

$$n \cdot \mathbf{Adv}^{\mathrm{ECDLOG}}(\mathcal{T})$$

for some discrete logarithm solver $\mathcal{T}$. The second event is impossible as otherwise the test session will no longer be clean.

A.2.2 $C_{3.2.2}$: $\mathcal{S}$ also randomly selects an eCK session in which $\mathcal{B}$ is the peer

Denote the owner of this session by $\mathcal{A}$. When the selected session is activated, $\mathcal{S}$ follows the protocol only partially:

- C_a $\mathcal{S}$ generates t_A and t_B normally but sets $comm_1^A \leftarrow X_1$ (in place of T_A) and $comm_1^B \leftarrow Y_1$ (in place of T_B).
- C_b $\mathcal{S}$ generates d_A, r_A and d_B, r_B normally, and it also has $e_A = H_n(Cert_A)$ and $e_B = H_n(Cert_B)$ normally. But it sets $comm_2^A$ and $comm_2^B$ in two ways. The first method is $\mathcal{S}$ sets $comm_2^A \leftarrow X_2$ (in place of Q_A) and $comm_2^B \leftarrow Y_2$ (in place of Q_B). The second method is $\mathcal{S}$ sets $R_A \leftarrow X_2$, $comm_2^A \leftarrow e_A X_2 + e_A R_{CA}^A + P_0$ (in place of $e_A r_A P + e_A R_{CA}^A + P_0$)and $R_B \leftarrow Y_2$, $comm_2^B \leftarrow e_B Y_2 + e_B R_{CA}^B + P_0$(in place of $e_B r_B P + e_B R_{CA}^B + P_0$).
- C_c $\mathcal{S}$ generates t_A, d_A, r_A and t_B, d_B, r_B normally and it also has $e_A = H_n(Cert_A)$ and $e_B = H_n(Cert_B)$ normally. But it sets $comm_3^A$ and $comm_3^B$ in two ways. The first method is $\mathcal{S}$ sets $comm_3^A \leftarrow X_3$ (in place of $(t_A + d_A)\, P$ and $comm_3^B \leftarrow Y_3$ (in place of $(t_B + d_B)\, P$). The second method is $\mathcal{S}$ sets $T_A \leftarrow X_3^1$, $R_A \leftarrow X_3^2$, $comm_3^A \leftarrow X_3^1 + e_A X_3^2 + e_A R_{CA}^A + P_0$ (in place of $t_A P + e_A r_A P + e_A R_{CA}^A + P_0$) and $T_B \leftarrow Y_3^1$, $R_B \leftarrow Y_3^2$, $comm_3^B \leftarrow Y_3^1 + e_B Y_3^2 + e_B R_{CA}^B + P_0$ (in place of $t_B P + e_B r_B P + e_B R_{CA}^B + P_0$).
  With probability at least $1/nk(1/n$ to pick the correct party $\mathcal{B}$ and $1/k$ to pick the correct session), $\mathcal{M}$ picks the selected session as the test session, and if it wins, it solves the ECCDH problem. The supposed session key for the selected session is $H_2(\sigma)$, where the 5-tuple σ includes the value $ECCDH(X_1, Y_1)$, $ECCDH(X_2, Y_2)$, $ECCDH(X_3, Y_3)$ or $ECCDH(X_1, Y_1)$, $ECCDH(e_A X_2 + e_A R_{CA}^A + P_0, e_B Y_2 + e_B R_{CA}^B + P_0)$, $ECCDH(X_3^1 + e_A X_3^2 + e_A R_{CA}^A + P_0, Y_3^1 + e_B Y_3^2 + e_B R_{CA}^B + P_0)$. To win, $\mathcal{M}$ must have queried σ to the random oracle H_2.

If the selected session is indeed the test session, $\mathcal{M}$ is not allowed to reveal both r_A, d_A and t_A and is not allowed to corrupt $\mathcal{B}$. In this case, the only way that $\mathcal{M}$ can distinguish this simulated eCK experiment from a true eCK experiment is if M queries (r_A, d_A, t_A). However, by Case $C_{3.1}$ it happens with probability at most for some discrete logarithm solver $\mathcal{T}$. Overall, if $\mathcal{M}$ always selects a test session which doesn't have a matching session then the success probability of $\mathcal{S}$ is at most

$$n \cdot \mathbf{Adv}^{\mathrm{ECDLOG}}(\mathcal{T})$$

for some discrete logarithm solver $\mathcal{T}$. Overall, if $\mathcal{M}$ always selects a test session which doesn't have a matching session then the success probability of $\mathcal{S}$ is at most

$$\mathbf{Adv}^{\mathrm{ECGDH}}(\mathcal{S}) \geq \frac{1}{nk} \cdot \mathbf{Adv}_{\mathcal{M}}^{\mathrm{CB-AKE}} - O\left(\frac{k^2}{2^{\lambda}}\right) - 2n \cdot \mathbf{Adv}^{\mathrm{ECDLOG}}(\mathcal{T}),$$

where $\mathcal{T}$ is some discrete logarithm solver. $\mathcal{S}$ runs in time $O(kt)$.

Finally, under the ECGDH assumption, $\mathbf{Adv}^{\mathrm{ECGDH}}(\mathcal{S})$ is negligible. Therefore, $\mathbf{Adv}_{\mathcal{M}}^{\mathrm{CB-AKE}}$ is negligible and CB-AKE protocol has eCK security. $\square$