A Provably Secure Trapdoor Hash Function Based on k-ECAA

Fuw-Yi Yang,*, Su-Hui Chiu

*Computer Science and Information Engineering, Chaoyang University of Technology, Taiwan,

1Office of Accounting, Chaoyang University of Technology, Taiwan,

*Corresponding author’s email: yangfy@cyut.edu.tw

ABSTRACT--- The integration of trapdoor hash function and scheme of digital signature not only enhances the security of the signature scheme, but also reduces the online computation during the construction of signatures. Many schemes of trapdoor hash function have been proposed. However, many of them are not provably secure. This paper proposes a trapdoor hash function based on an extension of k-CAA assumption, i.e. k-ECAA. On the assumption of random oracle model and adaptively chosen message attack, a forgery of collision implies solution of k-ECAA instance.

Keywords---- Digital signature, k-CAA assumption, k-ECAA assumption, Trapdoor hash function.

1. INTRODUCTION

Hash functions are commonly applied in digital signatures. Digital signing can be divided into three major phases: devising signature keys, signing documents and generating signatures, and certifying signatures. In a typical document-signing process, hash functions are first used to extract the abstract of the document to be signed, and the abstract is then digitally signed.

Collision-resistance is a crucial property of traditional hash functions but is selectively established for chameleon functions [Krawczyk& Rabin, 2000; Chen, 2014] or trapdoor hash functions[Shamir & Tauman, 2001; Yang, 2009 and 2013]. This indicates that trapdoor key attackers are unknown, and no algorithms of polynomial time complexity are available for calculating collision information (collided preimages). However, trapdoor key holders can efficiently identify other collision information and generate identical hash. For example, assume \( TH() \) represents the trapdoor hash functions, and hash value \( v = TH(h_1) \). After determining \( h_1 \), trapdoor key holders can calculate \( h_2 \), causing \( v = TH(h_2) = TH(h_1) \).

This paper presents the design and practical example of a trapdoor hash functions on the basis of mathematics problem: collusion attack algorithms with \( k \) traitors (\( k \)-CAAs). In practical example, the system parameter designs, trapdoor hash functions, and security properties are discussed.

2. INTRODUCTION TO COLLUSION ATTACK ALGORITHMS WITH \( k \) TRAITORS

The pairing function \( e \) is the primary element in bilinear pairing cryptosystems and pairs the elements in groups \( G_1 \) and \( G_2 \) to another group \( G_T \), specifically \( e: G_1 \times G_2 \to G_T, G_1 \) and \( G_2 \) are typically expressed as additive groups, and \( G_T \) is expressed as a multiplicative group. The three groups consist of identical orders, which are the large prime \( q \). In practice, the EC additive group \( G_1 = G_2 \) can typically be adopted. \( G_T \) is the multiplicative group of a finite field. The pairing function \( e \) adopts either Weil or Tate pairing and comprises bilinear and non degenerative properties:

**Bilinearity**: Bilinearity is found among the elements \( P, P_1 \), and \( P_2 \) of the additive group \( G_1 \) and \( Q, Q_1 \), and \( Q_2 \) of group \( G_2 \).

\[
\begin{align*}
e(P_1 + P_2, Q) &= e(P_1, Q) \cdot e(P_2, Q) \\
e(P, Q_1 + Q_2) &= e(P, Q_1) \cdot e(P, Q_2)
\end{align*}
\]
Non degenerative property: For the multiplicative group \(G_T\), 1 is used as the unit element, whereas 0 denotes the unit element used in the additive groups \(G_1\) and \(G_2\). For all elements \(P\) in \(G_1\), \(e(P, Q) = 1\); otherwise, \(Q = O\). For all elements \(Q\) in \(G_2\), \(e(P, Q) = 1\); otherwise, \(P = O\).

When \(G_1 = G_2\) and \(P \in G_1\), \(e(P, Q)\) generates the multiplicative group \(G_T\). For \(Q \in G_1\), \(e(P, Q) \in G_T\) can be calculated. However, provided \(e \in G_T\) and \(Q \in G_1\), calculating \(P \in G_1\), causes \(e(P, Q)\) to be difficult.

Bilinear pairing enables the DLP of \(G_1\) to be simplified to that of \(G_T\), suggesting that the DLP for \(G_1\) is not as difficult than that for \(G_T\). Moreover, \(G_1\) demonstrates that decisional Diffie–Hellman problems [Boneh, 1998] can be easily solved, whereas computational Diffie–Hellman problems remain challenging, which is referred to as gap groups [Okamoto & Pointcheval, 2001]. These results facilitate the security certification of cryptographic protocols. Researchers have recently studied bilinear pairing cryptosystems [Boneh, Lynn, & Shacham, 2001; Lin, 2010; Hoffstein, Pipher, & Silverman, 2014]. These have yielded crucial applications of these cryptosystems, including the k-CAA [Mitsunari, Sakai, & Kasahara, 2002; Yang, Ma, & Wang, 2006; Dutta, Barua, & Sarkar, 2004; Tso, Yi, & Hung, 2008; Tsai, Wuand, & Hsu, 2011].

**Definition 2.1 k-CAA problems**: \(k\) is an integer, \(P \in G_1\), and \(x \in \mathbb{Z}_q\). \((P, Q = xP, h_1, h_2, …, h_k \in \mathbb{Z}_q, P(h_1 + x), P(h_2 + x), …, P(h_k + x))\) is provided, and \(P(h + x)\) is calculated, in which \(h \notin \{h_1, h_2, …, h_k\}\).

The symbol \((\varepsilon, t)\)-break-k-CAA denotes the presence of an algorithm \(A\) that solves the \(k\)-CAA problem within the time limit \(t\) with a probability no less than \(\varepsilon\).

**Definition 2.2 (\(\varepsilon, t\))-k-CAA assumption**: No \((\varepsilon, t)\)-break-k-CAA algorithms exist in cyclic group \(G_1\).

The numerator of \(P(h + x)\) is multiplied by \((r/h)\), in which \(r \in \mathbb{Z}_q\). The originally defined k-CAA problem is rewritten. \(K\) is an integer, \(P \in G_1\), \(x \in \mathbb{Z}_q\), and \(\{P, P_{pub} = xP, R_{2k} = \{(h_i, r_i)\} | (h_i, r_i) \in \mathbb{Z}_q^2, \text{and } i = 1, 2, …, k\}\) as well as \(\{P(r_i/h_i, x + r), P(r_i/h_i, x + r)\} \in \mathbb{Z}_q^2\) and \(P(r_i/h_i, x + r)\) are provided. This outputs are \((h_i, r_i) \in \mathbb{Z}_q\) and \(P(r_i/h_i, x + r)\) in \(G_1\), in which \((h_i, r_i) \notin R_{2k}\). Additionally, the relationship between \(P(r_i/h_i, x + r_i)\) in \(G_1\) and \((h_i, r_i)\) can be specified using the equation \(e(P(r_i/h_i, x + r), r_i/h_i) = e(P, P)\), thus obtaining the following k-CAA problem:

**Definition 2.3. The extension of the collusion attack algorithms with \(k\)-traitors (k-ECAA) problem**: \(k\) is an integer, \(P \in G_1\), \(x \in \mathbb{Z}_q\). \((P, P_{pub} = xP, R_{2k} = \{(h_i, R_i)\} | (h_i, R_i) \in \mathbb{Z}_q \times G_1\), and \(i = 1, 2, …, k\), and \(S_{1k} = \{S_i \in G_1 \times \mathbb{Z}_q \times G_1 | P(h, R, S) = e(P, P)\), \((h_i, R_i) \in R_{2k}\) and \(i = 1, 2, …, k\) are provided, outputting \((h, R, S) \in \mathbb{Z}_q \times \{G_1\}^2\) in which \((h, R) \notin R_{2k}\).

**Inference 2.4 (\(\varepsilon, t\))-k-ECAA assumption**: k-ECAA problems are not less complicated than k-CAA problems.

**Proof**. The relationship between \(S_i = P((r_i/h_i, x + r_i, R_i) \in G_1\) and \((h_i, r_i, R_i)\) is specified using the equation \(e(P((r_i/h_i, x + r_i), r_i/h_i, P_{pub} + R_i) = e(P, P)\). A unique \(R_i\) can satisfy every \(h_i\) that is provided. Subsequently, the numerator of \(P(h_i + x)\) is multiplied by \((r_i/h_i)\), in which \(r_i \in \mathbb{Z}_q\). Because \(r_i\) is a random number, multiplying the numerator by a random number cannot reduce the complexity of the problem. Therefore, k-ECAA problems are not less complicated than k-CAA problems.

### 3. TRAPDOOR HASH FUNCTION BASED ON K-ECAA

This section elucidates the k-ECAA trapdoor hash functions, which comprise algorithms for setting system parameters and keys as well as generating and verifying collision information:

**Parameter Setup**: Operating cyclic groups \(G_1\) and \(G_2\), bilinear pairing function \(e\), and hash functions \(H(\cdot)\): \([0, 1]^* \rightarrow \mathbb{Z}_q\) are selected, in which \(H(\cdot)\) must satisfy the properties of secure hash functions. Next, the system parameters \(params = (G_1, G_2, e, q, P)\) and \(H(\cdot)\) are made public.

**Key setup**: The user \(ID_a\) selects and secretly collects the trapdoor key \(u \in \mathbb{R} \mathbb{Z}_q\) and calculates and makes public the validation key \(U = uP \in G_1\).

**Generating collision information**: \(ID_a\) owns the trapdoor key and therefore can efficiently calculate the collision information \((R, S)\) to restore the validation key. Provided that \(ID_a\) has formulated message \((m)\), the collision information can be calculated as follows:

\[
\begin{align*}
    r &\in \mathbb{Z}_q, R = r \cdot P \in G_1 \\
    h &= H(m, R)
\end{align*}
\]
\[ S = P(h(u + r)) \in G_1 \]

The validation key \( U \) and collision information \((R, S)\) are equivalent to the public signature information \( P_{pub} \) and signature [Zhang, Safavi-Naini, & Susilo, 2004], but a probability approach is adopted here to generate the collision information.

**Verifying collision information:** When the authenticator receives the collision information \((R, S)\) and message \( m \), the checking procedure is performed as follows:

\[
\begin{align*}
    h &= H(m, R) \\
    e(S, hU_1 + R) &= e(P, P)
\end{align*}
\]

where, \( ^\dagger \) denotes testing the equivalence of the terms on the two sides of the symbol. When the two sides equate, authentication succeeds, and \((R, S, m)\) is confirmed as the collision information and message of the validation key \( U \); otherwise, authentication fails. The term \( e(P, P) \) on the right-hand side of the certification equation may demonstrate various concepts other than those of the restored validation key \( U \), but this difference results from the simplification process. When \( S = up(h(u + r)) \) is adopted to calculate the collision information, the right-hand side of the certification equation becomes \( e(U_1, P) = e(P, P) \dagger \), which is the restored validation key.

4. SECURITY OF THE PROPOSED TRAPDOOR HASH FUNCTION

The model assumptions of ROM [Bellare and Rogaway, 1993] and adaptively chosen message attack [Goldwasser, Micali, and Rivest, 1988] are adopted to prove the security of the proposed scheme. Theorem 4.1 proves that when \((e, t, q_s, q_t)\)-forge algorithm exists in the \( k \)-CAA trapdoor hash functions, the \((e', t')\)-break-\( k \)-ECAA algorithm also exist in the cyclic group \( G_t \), in which \( t' = t \), and \( e' = e(q_sq_t) \dagger \).

**Theorem 4.1.** If the \( k \)-ECAA trapdoor hash function is \((e, t, q_s, q_t)\)-forge, then the \((e', t')\)-break-\( k \)-ECAA algorithm exists in the cyclic group \( G_t \), in which \( e' = e(q_sq_t) \dagger \), \( t' = t \), and \( k > q_t \).

**Proof.** Symbol \( A \) is used to represent the **challenger**, and \( F \) is used to represent the **forger** \((e, t, q_s, q_t)\)-forge. \( A \) is provided with the system parameters \( \text{params} = (G_t, G_\ast, e, P, H(\cdot)) \), an actual \( k \)-ECAA example \( \{P, P_{pub} = x \cdot P, R_{st} = \{(h_i, m_i, R_i)\} \in \mathbb{Z}_p \vdash H(m_i, R_i) \in G_\ast, m_i \in \{0, 1\}^\ast, R_i \in G_\ast, i = 1, 2, \ldots, q_t \} \), and \( S_{st} = \{S_i \in G_\ast, i = 1, 2, \ldots, q_t \} \). \( A \) simulates hash functions and generates hashes and signature information to respond to the demand of **Forger** \( F \). The aim is to use the ability of \( F \) to forge to calculate \((h, m, R) \) and \( S \), causing \( h = H(m, R) \) and \( e(S, hP_{pub} + R) = e(P, P) \), in which \((h, m, R) \in R_{st} \).

**Public key:** \( P \) denotes the public key, \( P_{pub} \in G_1 \), and \( \text{params} \) represents the system parameters.

**A prepares the hash table \( T_{st} \):** \( A \) generates a hashed set \( H_t = \{h_1, h_2, \ldots, h_{q_t}\} \), in which \( H_t = \{a_i \in \mathbb{Z}_p \mid i = 1, 2, \ldots, q_t \} \) and \( \mathbb{Z}_p = H_t \cap H_p \). The hash table \( T_{st} \) has a \( q_t \times 4 \) format. For \( j = 1, 2, \ldots, q_t \), \( A \) arbitrarily selects elements from the union of sets \( H_t, H_s \), and \( H_p \) and places the elements into \( T_{st}[j, 1] \) without repetition. When \( T_{st}[j, 1] = h_i \), selections are made in set \( H_t \); corresponding signature information \( S_i \) is then filled into \( T_{st}[j, 2] \), and message \((m_i, R_i)\) is filled into \( T_{st}[j, 3] \) and \( T_{st}[j, 4] \) as null data.

**F requests the hashes for \((m, R)\):** \( F \) searches the hash table \( T_{st}[j, 3] \) according to sequence, beginning from \( j = 1 \). For any corresponding data on the \( j \)-th row that satisfies \( T_{st}[j, 3] = m_i \) and \( T_{st}[j, 4] = R_i \), \( T_{st}[j, 1] \) is returned to \( F \); otherwise, \( A \) fills \((m_i, R_i)\) into \( T_{st}[j, 3] \) and \( T_{st}[j, 4] \) and returns \( T_{st}[j, 1] \) to \( F \), in which \( j \leq q_t \) is the minimal integer rendering \( T_{st}[j, 3] \) and \( T_{st}[j, 4] \) as null data.

**F requests the signature information for message \( m \):** Assuming the hashes for \((m_i, R_i)\) have been designated, \( A \) searches the hash table \( T_{st}[j, 3] \) beginning from \( j = 1 \) to identify the corresponding data \( T_{st}[j, 3] = m_i \), on the \( j \)-th row. If \( T_{st}[j, 2] \) is null data, \( A \) fails; otherwise, signature information \( T_{st}[j, 3] \) and \( T_{st}[j, 4] \) are returned to \( F \).

**Forger \( F \):** \( F \) can request the hash and signature information for arbitrary information \((m, R)\) at any time and may ultimately generate forged signature information \((m^\ast, R^\ast, S^\ast)\) for \( A \), in which \( m^\ast = m_i \) and \( 1 \leq i^\ast \leq q_t \). Through \( q_t \), number of requests for signature information, the condition \( H(m^\ast, R^\ast) \neq H_{st} \) can be confirmed. Therefore, \( A \) obtains \((h_t, m, R)\) and \( S \), causing \( h = H(m, R) \) and \( e(S, hP_{pub} + R) = e(P, P) \), in which \( h \neq H_t \) \( \in \{h_1, h_2, \ldots, h_{q_t}\} \). Because \( A \) generates signature information at a success rate of \( q_t/ q_t \), the probability of successful requests for signature information by \( F \) after \( q_t \) trials is \( (q_t/ q_t)^{q_t} \).

Specifically, when \( F \) can generate the \((e, t, q_s, q_t)\)-forge signature information, the algorithm \((e', t')\)-break-\( k \)-ECAA exists in cyclic group \( G_t \), in which \( e' = e(q_sq_t) \dagger \), and \( t' = t \).
5. CONCLUSIONS

Trapdoor hash functions can aid any signature scheme to securely generate signatures online efficiently. This paper extends the assumption of k-CAA to the k-ECAA. The later assumption is at least as hard as the first assumption. Then, a trapdoor hash function based on the intractability of k-ECAA is proposed. The discussion of security uses the ROM and adaptively chosen message attack to simulate the adversary’s attack. A forgery of hash collision means an instance of k-ECAA is solved. Thus, certifies the security of the proposed scheme.

6. REFERENCES