Iterated Random Oracle: A Universal Approach for Finding Loss in Security Reduction

Fuchun Guo[1(✉)], Willy Susilo[1], Yi Mu[1], Rongmao Chen[1,2], Jianchang Lai[1], and Guomin Yang[1]

[1] Centre for Computer and Information Security Research, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia
{fuchun,wsusilo,ymu,rc517,jl967,gyang}@uow.edu.au
[2] College of Computer, National University of Defense Technology, Changsha, China

Abstract. The indistinguishability security of a public-key cryptosystem can be reduced to a computational hard assumption in the random oracle model, where the solution to a computational hard problem is hidden in one of the adversary's queries to the random oracle. Usually, there is a *finding loss* in finding the correct solution from the query set, especially when the decisional variant of the computational problem is also hard. The problem of finding loss must be addressed towards tight(er) reductions under this type. In EUROCRYPT 2008, Cash, Kiltz and Shoup proposed a novel approach using a trapdoor test that can solve the finding loss problem. The simulator can find the correct solution with overwhelming probability 1, if there exists a trapdoor test for the adopted hard problem. The proposed approach is efficient and can be used for many Diffie-Hellman computational assumptions. The only limitation is the requirement of a trapdoor test that must be found for the adopted computational assumptions.

In this paper, we introduce a universal approach for finding loss, namely *Iterated Random Oracle*, which can be applied to all computational assumptions. The *finding loss* in our proposed approach is very small. For 2^{60} queries to the random oracle, the success probability of finding the correct solution from the query set will be as large as $1/64$ compared to $1/2^{60}$ by a random pick. We show how to apply the iterated random oracle for security transformation from key encapsulation mechanism with one-way security to normal encryption with indistinguishability security. The security reduction is very tight due to a small finding loss. The transformation does not expand the ciphertext size. We also give the application of the iterated random oracle in the key exchange.

Keywords: Random oracle · Indistinguishability security under computational assumptions · Finding loss

This work was partially supported by ARC Discovery Grant DP130101383.

J.H. Cheon and T. Takagi (Eds.): ASIACRYPT 2016, Part II, LNCS 10032, pp. 745–776, 2016.
DOI: 10.1007/978-3-662-53890-6_25

1 Introduction

Security reduction is a kind of reduction techniques in cryptography where we construct a simulator that uses an adversary's attack to solve a mathematically hard problem. According to the type of attack and the type of hard problem, cryptosystems have the following two popular types of security reduction.

- Unforgeability security based on a computational hard problem (UF-CHP). This type of security reduction has been used to prove the security of digital signature schemes. We construct a simulator that uses a forged signature from the adversary to solve a computational hard problem.
- Indistinguishability security based on a decisional hard problem (IND-DHP). This type of security reduction has been used to prove the security of encryption schemes. We construct a simulator that uses the guess of random message in the challenge ciphertext from the adversary to decide whether a solution in a given instance is correct or incorrect.

Roughly speaking, a computational problem is to find a correct solution to a given instance, while a decisional problem is to decide whether or not a solution in a given instance is correct. A computational hard problem is always harder than its decisional variant. However, without any additional assumption, it seems impossible to carry out a security reduction for a cryptosystem with indistinguishability security based on a computational hard problem. We call this type of reduction IND-CHP for short. This is because the guess from the adversary only has two answers: 0 or 1, which cannot provide sufficient information to find a correct solution. Fortunately, IND-CHP reduction becomes possible with the help of random oracles. Random oracles were first introduced by Bellare and Rogaway in [5] for designing efficient protocols. In the random oracle model, at least one hash function namely H is treated as a random oracle where responses on queries are assumed to be uniformly distributed. Anyone especially the adversary has no advantage in guessing the hash value of an input before querying the input to the random oracle. With the help of this "magical" property, many cryptosystems such as asymmetric encryption and key exchange can achieve IND-CHP security reduction.

The IND-CHP security reduction is programmed as follows. Suppose the simulator aims to compute $\mathcal{C}[I, P]$ as the solution to a given instance I under a computational hard problem P. The simulator who controls the random oracle programs the simulation using the instance I. In the simulation, the adversary must make a set of queries including a challenge query denoted by $\overline{\mathcal{Q}}^*$ to the random oracle to break the security, and the solution $\mathcal{C}[I, P]$ can be extracted from this challenge query. Different from UF-CHP and IND-DHP security reductions, the simulator solves the hard problem using the adversary's query set to random oracles instead of the adversary's forgery or guess. This distinctive security reduction arises a very important and interesting question:

How to find the correct solution from the adversary's query set?

We call this problem as a **finding problem** and the reduction has a **finding loss**, if the simulator can only succeed in finding the correct solution from the query set with a probability less than 1. When the decisional variant of the computational hard problem P is easy, there is no finding loss by verifying all solutions extracted from each query. However, when the decisional variant is also hard, it seems finding loss cannot be avoided. In this work, we focus on the non-trivial case that the decisional variant of P is also hard.

1.1 Finding Loss in Previous Approaches

In the IND-CHP security reduction, when the adversary can break a scheme simulated using an instance I, the challenge query will appear in the adversary query set and contain the solution $\mathcal{C}[I, P]$ to the instance I. The reduction after disclosing the simulation is equivalent to that the adversary who is given an instance I will make a set of queries including a challenge query $\overline{\mathcal{Q}}^* = \mathcal{C}[I, P]$. Using this disclosed reduction, we can use the following theories to describe how the finding problem is addressed.

The traditional approach in the literature is described in Theory 1. It has been applied to many cryptosystems such as [8] for IND-CHP security reductions.

Theory 1 (Traditional Approach). *Suppose an adversary, who is given an instance I generated by the simulator, must make a set of queries $\mathbb{Q}$ ($|\mathbb{Q}| = q$) including a challenge query $\overline{\mathcal{Q}}^* = \mathcal{C}[I, P]$ to the random oracle. We can construct a simulator who controls the random oracle to solve the hard problem P using the query set $\mathbb{Q}$ in $O(1)$ time with success probability $1/q$.*

It is easy to construct such a simulator. Given an instance I, the simulator forwards the instance to the adversary. Then, the challenge query is equal to the solution for the simulator. A random pick from the query set with q number of queries therefore has the success probability $1/q$.

In the security reduction, the adversary can make a polynomial number of queries to the random oracle. The query number q can be as large as $q = 2^{60}$, and hence the success probability of finding the correct solution is $1/2^{60}$. It means that all cryptosystems using this traditional approach in reduction will have at least 60-bit security loss. In the concrete security of group-based cryptosystems, we must expand the corresponding group size with 60-bit more security to compensate the security loss. This compensation at least requires 120-bit length more of security parameter in group choice, and it is therefore accompanied with inefficient group operation and large group representation.

In EUROCRYPT 2008, Cash, Kiltz and Shoup [10] introduced the first novel approach for finding loss. They proposed a new computational problem called the twin Diffie-Hellman problem. This new problem is as hard as the Computational Diffie-Hellman (CDH) problem even given access to a corresponding decision oracle. The heart of their approach is a trapdoor test, which allows the simulator to simulate an effective decision oracle without knowing any of the corresponding discrete logarithms. Their approach can be summarized using a theory described as follows.

Theory 2 (Cash-Kiltz-Shoup). *Suppose an adversary, who is given instances* (I_1, I_2) *generated by the simulator, must make a set of queries* $\mathbb{Q}$ $(|\mathbb{Q}| = q)$ *including a challenge query* $\overline{\mathcal{Q}}^* = \mathcal{C}[I_1, P] \,||\, \mathcal{C}[I_2, P]$ *to the random oracle. We can construct a simulator who controls the random oracle to solve the hard problem* P *using the query set* $\mathbb{Q}$ *in* $O(q)$ *time with nearly success probability* 1, *if there exists a trapdoor test on solutions to a given instance and a created instance under the hard problem* P.

The simulator can be constructed as follows. Given an instance I, the simulator sets $I_1 = I$, Then, it randomly chooses a trapdoor and creates the second instance I_2 from I_1 and the trapdoor. The trapdoor test holds with the property that a query $\overline{\mathcal{Q}} = Q_1 \,||\, Q_2$ can pass the trapdoor test run by the simulator if and only if $Q_1 = \mathcal{C}[I_1, P]$ and $Q_2 = \mathcal{C}[I_2, P]$ except with a negligible probability. Therefore, only the challenge query can pass the test and the simulator can successfully find the correct solution $\mathcal{C}[I, P]$ without any finding loss after all queries are tested.

Based on this theory, Cash, Kiltz and Shoup [10] proposed many twin schemes based on original schemes using two key pairs, whose IND-CHP security reductions are tight(er) without any finding loss. The price to pay for an encryption scheme is two times less efficient in terms of key size and computations compared to the original one, but the size of ciphertext is not changed. However, this theory has a limitation. It can only be applied to those cryptosystems whose underling computational assumptions have a corresponding trapdoor test. The trapdoor test proposed in [10] is a very special construction and it can be adopted by some computational Diffie-Hellman hard problems only.

1.2 Our Contribution

We propose a completely new approach for finding loss, namely *iterated random oracle*, which can be applied to all computational hard problems. Instead of using a trapdoor test to find the correct solution, the simulator in our approach can remove most of useless queries such that a random pick from remaining queries will merely have a small finding loss only. The corresponding theory is described as follows.

Theory 3 (Iterated Random Oracle). *Let* H *be a random oracle. Suppose an adversary, who is given instances* $(I_1, I_2, \cdots, I_n)$ *generated by the simulator, must make a set of queries* $\mathbb{Q}$ $(|\mathbb{Q}| = q)$ *including a challenge query* $\overline{\mathcal{Q}}^* = \overline{\mathcal{Q}}_*^{(n)}$ *to the random oracle, where* $\overline{\mathcal{Q}}_*^{(n)}$ *is defined as*

$$\overline{\mathcal{Q}}_*^{(i)} = H(\overline{\mathcal{Q}}_*^{(i-1)}) \,||\, \mathcal{C}[I_i, P] \,||\, i : \quad i \in [1, n], \quad H(\overline{\mathcal{Q}}_*^{(0)}) = 0_\epsilon \text{ is an empty string.}$$

We can construct a simulator who controls the random oracle to solve the hard problem P *using the query set* $\mathbb{Q}$ *in* $O(n)$ *time with success probability at least* $1/(nq^{\frac{1}{n}})$.

The simulator construction and probability analysis are given in Sect. 3. We give an example in the next subsection to overview the simulator construction and the probability analysis. When this theory holds, the success probability is $1/640$ for $q = 2^{60}$ and $n = 10$. We can further increase the success probability to $1/64$ by repeating hash operations for ten times. In comparison with the traditional approach with success probability $1/2^{60}$ only, our approach significantly improves the success probability even with a small integer n. We compare the different approaches for finding loss in Table 1.

We show how to apply the iterated random oracle in encryption and key exchange for tight(er) reduction. In the application to encryption, we show how to use a key encapsulation mechanism with one-way security to construct an encryption scheme with indistinguishability security against a chosen-plaintext attack and a chosen-ciphertext attack. The security transformation from one-way security to indistinguishability security will only have a small finding loss. Notice that the security reduction for encapsulation mechanism with one-way security does not have the finding loss because the adversary must return the encapsulation key, which can be programmed as the solution to the computational hard problem in the reduction. Therefore, our security transformation is equivalent to a provably secure encryption under IND-CHP security reduction with a small finding loss. The transformation is n times ($n = 10$) less efficient in terms of key size and computations. However, the transformation does not expand the ciphertext size when the generation of key encapsulation is independent of public key. Many encryption schemes such as the ElGamal encryption [21] and BF-IBE [8] can be modified into key encapsulation mechanisms capturing this property. We also study the application of the iterated random oracle in an identity-based non-interactive key exchange protocol and other key exchange protocols.

Table 1. Comparison of different approaches for finding loss. The finding efficiency refers to the time cost of picking a query from the query set. The query efficiency refers to the time cost of generating the challenge query. Here, q is the size of query set including the challenge query and n is the maximum iteration time.

	Theory 1	Theory 2	Theory 3 (Ours)
For all problems	✓	×	✓
Success probability	$\frac{1}{q}$	1	$\frac{1}{n \cdot q^{\frac{1}{n}}}$
Finding efficiency	$O(1)$	$O(q)$	$O(n)$
Query efficiency	1	2	$O(n)$

1.3 Overview of the Approach

For simplicity, we use the concrete CDH problem as an example to describe the overview of the approach in the iterated random oracle. Suppose an adversary,

who is given instances $I_i = (g,\ g^{a_i},\ g^b)$ for all $i \in [1,n]$ generated by the simulator, must make a query set $\mathbb{Q}$ ($|\mathbb{Q}| = q$) including a challenge query $\overline{\mathcal{Q}}^* = A_n$ to the random oracle, where A_n is defined as

$$A_i = H(A_{i-1}) \mid\mid g^{a_i b} \mid\mid i: \quad i \in [1,n], \quad H(A_0) = 0_\epsilon \text{ is an empty string.}$$

We can construct a simulator to solve the CDH problem using the query set $\mathbb{Q}$ with success probability at least $1/(nq^{\frac{1}{n}})$. Given as input an instance (g, g^a, g^b) under a cyclic group $\mathbb{G}$ of prime order p, the aim of the simulator is to find g^{ab} from the query set generated by the adversary. This reduction is mainly composed of two tasks: (1) how to generate the instances $I_i = (g,\ g^{a_i},\ g^b), i \in [1,n]$ for the adversary, namely *instance generation* and (2) how to pick the query from the adversary's query set, namely *query selection*.

Instance Generation. The simulator randomly chooses $d \in [1,n], a_1, a_2, \cdots, a_{d-1},\ a_{d+1}, \cdots, a_n \in \mathbb{Z}_p$ and sets $a_d = a$. Then, it gives $I_i = (g,\ g^{a_i},\ g^b)$ for all $i \in [1,n]$ to the adversary who is required to make a query set including A_n. It requires that the adversary does not know d. Since all instances are chosen randomly, this requirement holds trivially. In the instances given to the adversary, the simulator can compute $g^{a_i b} = (g^b)^{a_i}$ for all $i \in [d+1,n]$ by itself since all related a_i are known. This is very important in the query selection for a small finding loss.

Query Selection. In this phase, a query is defined as either a candidate query or a useless query. The simulator will randomly pick a query from candidate queries, after all useless queries are removed. Before introducing what are useless queries and how to remove them, we first introduce what all iterated queries look like.

The query $\overline{\mathcal{Q}} = H(\overline{\mathcal{Q}}') \mid\mid Q \mid\mid i$ in the iterated random oracle is an iterated query, composed of an oracle response, a weight (the solution will appear here) and an iteration time. All iterated queries to the random oracle can be depicted in an arbitrary tree, where a node denotes a response on a query and an edge denotes a query. The root is an empty string. The edge $\overline{\mathcal{Q}} = H(\overline{\mathcal{Q}}') \mid\mid Q \mid\mid i$ starts from the node $H(\overline{\mathcal{Q}}')$ and ends at the node $H(\overline{\mathcal{Q}})$, which is depicted at the level i. When the maximum iteration time is n, the height of this arbitrary tree is n. For example, the two queries $\overline{\mathcal{Q}}^{(1)}_{1,2} = 0_\epsilon \mid\mid Q^{(1)}_{1,2} \mid\mid 1$ and $\overline{\mathcal{Q}}^{(2)}_{2,1} = H(\overline{\mathcal{Q}}^{(1)}_{1,2}) \mid\mid Q^{(2)}_{1,2} \mid\mid 2$ can be depicted in a path from the root to a leaf shown in Fig. 1.

According to the property of random oracle, if $\overline{\mathcal{Q}}^* = A_n$ appears in the query set, all queries $A_1, A_2, \cdots, A_n$ must appear in the query set. Now, we can roughly describe what are useless queries. First, all queries with iteration time which is not equal to d are useless queries. Second, a query $\overline{\mathcal{Q}}$ with iteration time equal to d is a useless query if there is no valid path from the node $H(\overline{\mathcal{Q}})$ to a leaf node at the level n. Here, a valid path is the path where all edges for $i \in [d+1,n]$ in this path are valid queries whose weights are equal to $g^{a_i b}$. The simulator can verify whether a path is valid or not, because $g^{a_i b} = (g^b)^{a_i}$ for all $i \in [d+1,n]$ are computable using a_i. All queries with iteration time equal to n are candidate queries.

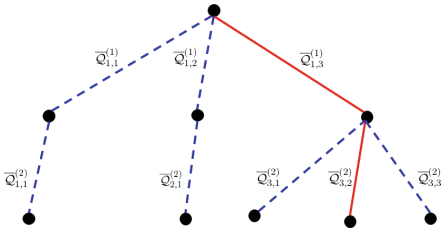

Fig. 1. Example 1

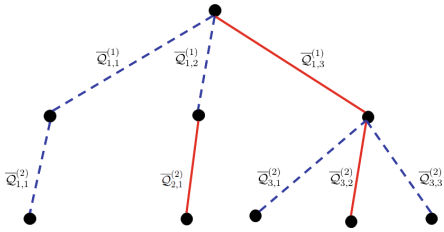

Fig. 2. Example 2

Probability Analysis. Based on the above instance generation and query selection, we can prove there must exist an integer $i^* \in [1, n]$ satisfying the minimum probability $1/q^{\frac{1}{n}}$. Precisely, for those queries with iteration time i^*, the success probability of picking a valid query from candidate queries is $1/q^{\frac{1}{n}}$. The integer i^* is adaptively decided by the adversary in query set generation, while the integer d is randomly chosen by the simulator. When $d = i^*$ (i.e. the simulator happens to embed the solution in this level), all useless queries with iteration time i^* will be removed and the corresponding success probability is $1/q^{\frac{1}{n}}$. Therefore, we yield the success probability result by

$$\Pr[suc] = \sum_{i=1}^{n} \Pr[suc|d = i] \Pr[d = i] \geq \Pr[suc|d = i^*] \Pr[d = i^*] = \frac{1}{nq^{\frac{1}{n}}}.$$

We now give four simple examples where $n = 2$ and $q = 8$ to analyze the above result. The corresponding success probability of $\Pr[suc|d = i^*]$ for some i^* should be at least $1/\sqrt{8}$. We use a solid line to denote a query at the level i if it has a valid weight equal to $g^{a_i b}$. Otherwise, we denote the query with a dashed line. In this arbitrary tree, $\overline{\mathcal{Q}}^{(i)}$ denotes a query at the level i. Notice that all queries from the same node have at most one query with a valid weight, but all queries at the same level i could have more than one valid query whose weights are all valid and equal to $g^{a_i b}$.

In these examples, if the adversary only makes two queries at the first level, we immediately have $\Pr[suc|d = 1] = \frac{1}{2} \geq \frac{1}{\sqrt{8}}$ when $d = 1$. Therefore, in the following examples, the adversary is assumed to make three queries at the first level.

- Suppose the query set can be depicted as the tree in Fig. 1. When $d = 1$, the two queries $\overline{\mathcal{Q}}^{(1)}_{1,1}, \overline{\mathcal{Q}}^{(1)}_{1,2}$ will be removed because their nodes do not have a valid path such that only one query is remained at this level. Therefore, we have $\Pr[suc|d = 1] = 1 \geq \frac{1}{\sqrt{8}}$.
- Suppose the query set can be depicted as the tree in Fig. 2. When $d = 1$, the query $\overline{\mathcal{Q}}^{(1)}_{1,1}$ will be removed because this node does not have a valid path such that two queries are remained at this level. Therefore, we have $\Pr[suc|d = 1] = \frac{1}{2} \geq \frac{1}{\sqrt{8}}$.

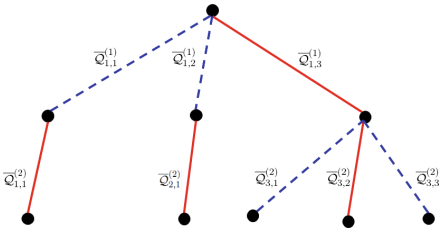

Fig. 3. Example 3

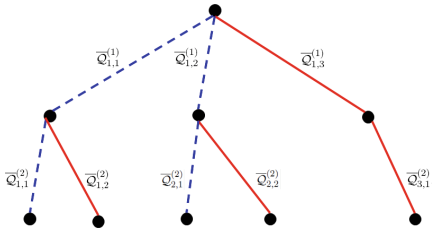

Fig. 4. Example 4

– Suppose the query set can be depicted as the tree in Fig. 3. When $d = 2$, it is easy to see that $\Pr[suc|d = 2] = \frac{3}{5} \geq \frac{1}{\sqrt{8}}$.
– Suppose the query set can be depicted as the tree in Fig. 4. The result is exactly the same as Fig. 3, where $\Pr[suc|d = 2] = \frac{3}{5} \geq \frac{1}{\sqrt{8}}$.

1.4 Other Related Work

The UF-CHP security reduction with a tight reduction for digital signatures has been studied in [1,2,6,13,14,24–26]. A tight reduction requires no abortion in signature simulation and enables to solve a hard problem from the forged signature. With the help of random oracles, it seems easier to achieve a tight reduction by adding a random bit after the message to be signed. In this reduction, the simulator uses the bit to control the hash values of messages to be signed and to be forged, such that the probability of abortion is very small.

The IND-DHP security reduction with a tight reduction for encryption has been studied in [4,7,9,15,16,22,23,26–28]. To achieve a tight reduction, the simulator must be able to simulate decryption queries for CCA security and private key queries for identity-based encryption and its variants. It also requires the simulator to program the challenge ciphertext into a one-time pad or an indistinguishable ciphertext depending on the given instance. We note that the approaches for tight reduction are different. This is because there is no general technique enabling a tight reduction for encryption, especially without random oracles.

The IND-CHP security reduction is a special reduction requiring the help of random oracles, where the simulator solves a hard problem using the adversary's queries instead of its direct attack. How to find the correct solution from the adversary's query set is necessary to achieve a tight reduction. The problem of finding loss only exists in this reduction type especially when the decisional variant is also hard. The traditional approach for finding loss is via a random pick, which results in a huge finding loss. The first non-trivial approach was introduced by Cash, Kiltz and Shoup [10] in EUROCRYPT 2008. The proposed trapdoor test can be used to solve finding loss during the corresponding IND-CHP security reductions. They had shown that the proposed approach can be applied to Diffie-Hellman key exchange [17], Cramer-Shoup encryption [16], BF-IBE [8] and password-authenticated key exchange [3] to achieve the tightness

of security reduction. This approach, however, requires that the computational hard problem can be embedded with a trapdoor test on solutions to a given instance and a created instance. This work has been extended and applied in [11,12] but they still have the same restriction. There is no efficient approach for finding loss in the IND-CHP security reduction without any restriction on the adopted computational hard assumptions.

The rest of this paper is organized as follows. We use an example to introduce how the IND-CHP security reduction works in Sect. 2. The generalization of computational hard problems is also given and discussed. In Sect. 3, we prove the correctness of Theory 3. Then, we show how to apply the iterated random oracle for encryption in Sect. 4 and key exchange towards tight(er) security reduction in Sect. 5.

2 IND-CHP Security Reduction and Generalized Problems

2.1 An Example of IND-CHP Security Reduction

Let $\mathbb{G}$ be a cyclic group of the prime order p and g be a generator. Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be a one-way hash function. Considering the following bare ciphertext CT without a public/secret key pair, where $x, y \in \mathbb{Z}_p$ and $coin \in \{0,1\}$ are chosen randomly and secretly.

$$CT = (c_1, c_2, c_3) = \Big(g^x,\ g^y,\ H(g^{xy}) \oplus m_{coin}\Big)$$

Suppose there exists an adversary who can distinguish the message $m_{coin} \in \{m_0, m_1\}$ in CT with a non-negligible advantage ϵ in a polynomial time, where the two messages $\{m_0, m_1\} \in \{0,1\}^n$ are adaptively chosen by the adversary. We can construct a simulator to solve the CDH problem in the random oracle model, where H is set as a random oracle controlled by the simulator.

Before we introduce how to program the security reduction, we first introduce the nice feature of using random oracle in security reduction. In the random oracle model, the message is encrypted with $H(g^{xy})$, which is a random string from $\{0,1\}^n$ and is independent of its hash input g^{xy} and (g^x, g^y). Without making a query on g^{xy} to the random oracle, the ciphertext CT is a one-time pad encryption on m_{coin} because $H(g^{xy})$ is random and independent of (g^x, g^y) in the ciphertext. Then, the success probability of guessing the encrypted message is $\frac{1}{2}$ only. According to the assumption, the adversary can distinguish the encrypted message with probability $\frac{1}{2} + \epsilon$. This assumption indicates that the adversary ever queried g^{xy} to the random oracle with probability 2ϵ [8]. That is, one of queries in the adversary's query set is equal to g^{xy}. This query is called challenge query, which is used to break the security of cryptosystem.

The security reduction works as follows. Given (g, g^a, g^b), the simulator aims to compute g^{ab}. Upon receiving $m_0, m_1 \in \{0,1\}^n$ from the adversary, the simulator creates the challenge ciphertext as $CT = (c_1, c_2, c_3) = (g^a,\ g^b,\ R)$, where R is a random string from $\{0,1\}^n$. What the simulator will do is to wait for

queries from the adversary. Notice that if the adversary does not make a query on g^{ab} to the simulator, the adversary cannot either distinguish the message with a non-negligible advantage or distinguish the simulation ciphertext from the real ciphertext. According to the assumption, the group element g^{ab} will appear in one of queries with probability 2ϵ. Suppose the adversary made q queries to the random oracle in total. The simulator randomly picks one of queries as the solution to the CDH problem. We have the randomly picked element is equal to g^{ab} with probability $\frac{2\epsilon}{q}$. That is, the simulator will solve the hard problem with probability $\frac{2\epsilon}{q}$ in the corresponding security reduction. This completes the description of security reduction. This reduction has a finding loss whose corresponding success probability is in the linear of hash query number q.

We note that the above bare ciphertext cannot be decrypted by anyone when the CDH problem is hard. However, in the real encryption scheme, the encryptor and the decryptor know more information than the bare ciphertext. When treating g^x as the public key and y is the chosen random number by the encryptor, we have that the bare ciphertext is equivalent to the hashed ElGamal encryption scheme, where the encryptor knows y and the decryptor knows the secret key x such that the ciphertext can be created and decrypted respectively. Roughly speaking, a secure encryption scheme is constructed in the way that a computational hard problem can be easily solved by the encryptor and decryptor with an additional secret, while outsiders (adversaries) without knowing a secret must solve the computational hard problem in order to break the scheme.

2.2 Generalized Computational Hard Problems

We generalize all computational hard problems into the following description.

- I: The input arbitrary string (also known as instance)
- P: The computational problem
- $\mathcal{C}[I, P]$: The solution to the instance I under the computational problem P.

For example, given an instance $I = (g, g^a, g^b) \in \mathbb{G}$, based on different problems P, the solution can be

$$\mathcal{C}[I, P_1] = g^{ab}, \qquad \mathcal{C}[I, P_2] = g^{\frac{b}{a}}.$$

The generalized computational hard problem is defined as

$$\Pr\Big[\mathcal{A}(I, P) = \mathcal{C}[I, P]\Big] \leq \epsilon,$$

where no adversary who is given (I, P) can find a solution $\mathcal{C}[I, P]$ with a non-negligible advantage ϵ. Here, ϵ is a function of the security parameter in the generation of the instance I.

For the computational hard problem (I, P), anyone can verify whether a solution is correct or not if the decisional variant of this problem is easy. However, if the decisional variant is also hard, it seems no one can verify the correctness of a solution. However, this observation is not correct because the instance generator, who generates the instance, can generate the instance in the way that

it knows its correct solution. Taking the CDH problem in a cyclic group as an example where the DDH problem is also hard. The instance generator can randomly choose $a, b \in \mathbb{Z}_p$ and set the instance to be (g, g^a, g^b), where the solution g^{ab} is computable by the instance generator. Hence, for the computational hard problem P, we assume the instance generator enables to generate an instance I such that $\mathcal{C}[I, P]$ can be efficiently computed. This assumption is necessary to support the definition of computational hard problems whose decisional variants are also hard. We emphasize the importance of this property here because the simulator in the iterated random oracle requires generating some instances indistinguishable from the challenge instance, such that the simulator can compute solutions to all self-generated instances under the challenge hard problem P.

3 Iterated Random Oracle and Its Proof

In the iterated random oracle, each query will be programmed using iterations, and hence it will be called as an iterated query. An iterated query is composed of an oracle response, a weight (the solution to a hard problem will appear here) and an iteration time. They are put together using a concatenation symbol "||". Given a hash list recording all iterated queries and their responses, we can depict all queries in the hash list using an arbitrary tree. The height of this arbitrary tree is n, where n is the maximum time of iteration. The details are described in the following subsections.

3.1 Iterated Query and Tree Representation

Iterated Query. We define an iterated query $\overline{\mathcal{Q}}$ to the random oracle as

$$\overline{\mathcal{Q}} = \text{Response} \,||\, \text{Weight} \,||\, \text{Iteration Time} = \overline{\mathcal{R}} \,||\, Q \,||\, i,$$

where $\overline{\mathcal{R}}$ is a response on a query from the random oracle H (an empty string 0_ϵ is assumed as the initialized response), Q is a weight (any arbitrary string) chosen by the adversary and i is the iteration time. The iteration time denotes the minimum time for making such an iterated query. If $i = 1$, it means the adversary can immediately make such a query. Otherwise, for example, given $\overline{\mathcal{Q}}_1 = 0_\epsilon||Q_1||1$ and $\overline{\mathcal{Q}}_2 = H(\overline{\mathcal{Q}}_1)||Q_2||2$, it requires the adversary to query $\overline{\mathcal{Q}}_1$ first before $\overline{\mathcal{Q}}_2$. We will use the following symbols associated with queries and responses in the following representations.

- $\overline{\mathcal{Q}}^{(i)}$ is an iterated query with the iteration time i.
- $Q_{j,k}^{(i)}$ is the weight in the iterated query $\overline{\mathcal{Q}}_{j,k}^{(i)}$.
- $\mathbb{Q}$ is the set of all queries made by the adversary.
- $\mathbb{Q}^{(i)}$ is the set of all iterated queries whose iteration time are all equal to i.
- $H(\overline{\mathcal{Q}}^{(i)})$ is the response from the random oracle on the query $\overline{\mathcal{Q}}^{(i)}$.

Tree Representation. Suppose the adversary only makes the above iterated queries to the random oracle, and an empty hash list $\mathcal{L}$ is used to record all queries and responses. We can depict all queries and corresponding responses using an artitrary tree (such as Fig. 5), where the root is the empty string 0_ϵ.

- All edges denote iterated queries and their end nodes denote their corresponding responses.
- The query $\overline{\mathcal{Q}}_{j,k}^{(i)} = H(\overline{\mathcal{Q}}^{(i-1)}) \,||\, Q_{j,k}^{(i)} \,||\, i$ is the edge with connection between the node $H(\overline{\mathcal{Q}}^{(i-1)})$ and the node $H(\overline{\mathcal{Q}}_{j,k}^{(i)})$ at the level i. Here, j in this query represents that $\overline{\mathcal{Q}}^{(i-1)}$ is the j-th query at the level $i-1$ counted from left to right, and k in this query represents that $H(\overline{\mathcal{Q}}_{j,k}^{(i)})$ is the k-th child of $H(\overline{\mathcal{Q}}^{(i-1)})$ counted from left to right.
- The height of the arbitrary tree is the maximum time of iteration in all iterated queries.

The hash list and the tree representation have the following connections. First, this is an arbitrary tree because the adversary can make any number of iterated queries $\overline{\mathcal{Q}} = \overline{\mathcal{R}} \,||\, Q \,||\, i$ with the same $\overline{\mathcal{R}}$ and i. Second, all edges starting from the same node are the depiction of queries with the same $\overline{\mathcal{R}}$ and i but distinct weights Q. Third, all iterated queries are different such that all nodes are distinct, but the weights in those queries (edges) from different nodes could be the same. For example, the weight $Q_{3,1}^{(2)}$ must be different from $Q_{3,2}^{(2)}$ because the queries $\overline{\mathcal{Q}}_{3,1}^{(2)}, \overline{\mathcal{Q}}_{3,2}^{(2)}$ already have the same oracle response and iteration time. However, $Q_{3,2}^{(2)}$ could be equal to $Q_{1,1}^{(2)}$ in Fig. 5. This observation is very important in the analysis of success probability for the iterated random oracle. Finally, the total query number is equal to the total number of edges in this arbitrary tree, if all queries are iterated queries.

In the random oracle model, the adversary can make any arbitrary string as a query chosen by itself. However, we focus on the defined iterated queries only. We emphasize that our focus does not compromise any problem because all other queries that cannot be described in this arbitrary tree must be not the challenge query and will be removed from the query set before selection.

3.2 Proof of Theory 3

It is complicated to prove this theory directly especially the analysis of success probability. We split the proof for this theory into the following steps.

Simulator Construction. Given as input an instance I and the problem P, the simulator aims to compute $\mathcal{C}[I, P]$. The simulator generates $(I_1, I_2, \cdots, I_n)$ for the adversary as follows.

- Randomly choose $d \in [1, n]$ and set $I_d = I$. We have $\mathcal{C}[I_d, P] = \mathcal{C}[I, P]$.
- Choose random instances $I_1, I_2, \cdots, I_{d-1}, I_{d+1}, \cdots, I_n$ under the problem P such that $\mathcal{C}[I_i, P]$ for all $i \in [1, n]/\{d\}$ are known by the simulator.

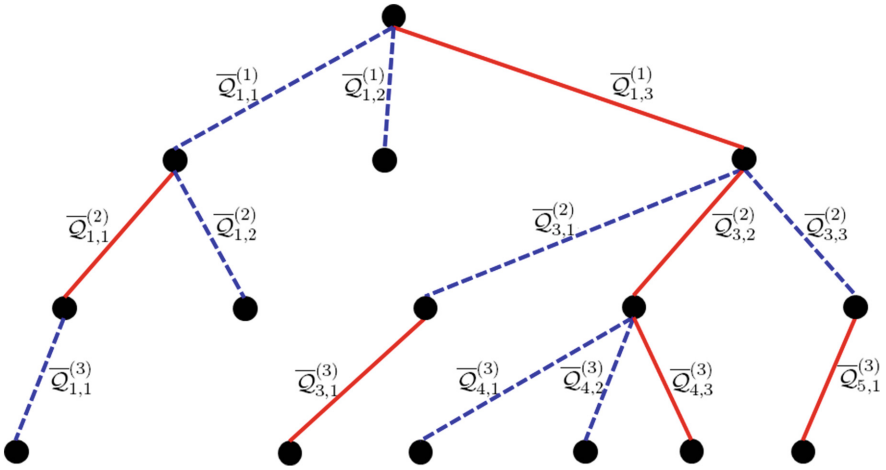

Fig. 5. An example of arbitrary tree generated from iterated queries and responses.

– Set and give $(I_1, I_2, \cdots, I_n)$ to the adversary.

According to the assumption, the adversary will make a query set $\mathbb{Q}$ to the random oracle including a challenge query $\overline{\mathcal{Q}}^* \in \mathbb{Q}$, where $\overline{\mathcal{Q}}^* = \overline{\mathcal{Q}}^{(n)}_*$. According to the definition of $\overline{\mathcal{Q}}^{(i)}$ and the property of random oracles, the adversary must ever make all challenge queries $\overline{\mathcal{Q}}^{(1)}_*, \overline{\mathcal{Q}}^{(2)}_*, \cdots, \overline{\mathcal{Q}}^{(n)}_*$ to the random oracle. Otherwise, the adversary cannot generate $\overline{\mathcal{Q}}^* \in \mathbb{Q}$. Notice that $\mathcal{C}[I, P]$ exists in $\overline{\mathcal{Q}}^{(d)}_* \in \mathbb{Q}^{(d)}$. The simulator will solve the hard problem by removing all useless queries in $\mathbb{Q}^{(d)}$, picking a random query from the remaining set $\mathbb{Q}^{(d)}$ and extracting the weight from the picked query as the solution to the hard problem. The success probability of finding the correct solution will be the one given in our theory.

Further Tree Representation. We further define queries and weights in order to clarify how to remove all useless queries from $\mathbb{Q}^{(d)}$.

– The query $\overline{\mathcal{Q}}^{(i)}_{j,k}$ is a **challenge query** if $\overline{\mathcal{Q}}^{(i)}_{j,k} = \overline{\mathcal{Q}}^{(i)}_*$.
– The weight $Q^{(i)}_{j,k}$ is a **valid weight** if $Q^{(i)}_{j,k} = \mathcal{C}[I_i, P]$.
– The query $\overline{\mathcal{Q}}^{(i)}_{j,k}$ is a **valid query** if it has a valid weight.
– A path from a node to a leaf is a **valid path** if all edges in this path are valid queries.
– The query $\overline{\mathcal{Q}}^{(i)}_{j,k}$ is a **child query** of $\overline{\mathcal{Q}}^{(i-1)}$ if $\overline{\mathcal{Q}}^{(i)}_{j,k} = H(\overline{\mathcal{Q}}^{(i-1)})||Q^{(i)}_{j,k}||i$.
– The query $\overline{\mathcal{Q}}^{(i)}_{j,k}$ is a **candidate query** if there exists a valid path from the node $H(\overline{\mathcal{Q}}^{(i)}_{j,k})$ to a leaf node at the level n. All queries in $\mathbb{Q}^{(n)}$ are defined as candidate queries.

– The query $\overline{\mathcal{Q}}_{j,k}^{(i)}$ is a **useless query** if there is no valid path from the node $H(\overline{\mathcal{Q}}_{j,k}^{(i)})$ to a leaf node at the level n.

We note that all queries that cannot be depicted in this arbitrary tree or can only be depicted outside this arbitrary tree are useless queries. The maximum number of edges in this tree is q. About the relationship among valid query, challenge query and candidate query, we have a challenge query must be both a valid query and a candidate query. The definition of valid query and candidate query are independent. There must exist one valid path only from the root to a leaf at the level n because all queries from the root have only one valid query. There could exist more than one valid query in $\mathbb{Q}^{(i)}$ for any $i \geq 2$, but each query has one valid child query at most. In Fig. 5, we use a solid edge to denote a valid query and a dashed edge to denote an invalid query.

We have two important observations in the following two claims.

Claim 1. *If $\overline{\mathcal{Q}}^{(i)}$ is a candidate query, it must have a valid child query.*

According to the definition of candidate query, there exists a valid path from the node $H(\overline{\mathcal{Q}}^{(i)})$ to a leaf node at the level n. The first edge in this valid path is a valid query comprising of the response $H(\overline{\mathcal{Q}}^{(i)})$. This is the valid child query of $\overline{\mathcal{Q}}^{(i)}$.

Claim 2. *If $\overline{\mathcal{Q}}^{(i)}$ is a candidate query and its child query denoted by $\overline{\mathcal{Q}}^{(i+1)}$ is a valid query, we have that $\overline{\mathcal{Q}}^{(i+1)}$ is also a candidate query.*

We prove by contradiction. According to the first claim and the tree representation, there exists only one valid child query of $\overline{\mathcal{Q}}^{(i)}$ denoted by $\overline{\mathcal{Q}}^{(i+1)}$. All paths starting from the node $H(\overline{\mathcal{Q}}^{(i)})$ through invalid child queries of $\overline{\mathcal{Q}}^{(i)}$ must be invalid paths. If all paths starting from the node $H(\overline{\mathcal{Q}}^{(i)})$ through the edge $\overline{\mathcal{Q}}^{(i+1)}$ are invalid paths either, there is no valid path from the node $H(\overline{\mathcal{Q}}^{(i)})$ to a leaf node. Hence $\overline{\mathcal{Q}}^{(i)}$ is not a candidate query. Therefore, the assumption is incorrect and there should exist a valid path starting from the node $H(\overline{\mathcal{Q}}^{(i)})$ through the edge $\overline{\mathcal{Q}}^{(i+1)}$, which implies that $\overline{\mathcal{Q}}^{(i+1)}$ is also a candidate query.

Lemma 1. *If the following rate*

$$R^{(i)} = \frac{\textit{The number of valid queries in } \mathbb{Q}^{(i)}}{\textit{The number of candidate queries in } \mathbb{Q}^{(i)}} < \frac{1}{q^{\frac{1}{n}}}$$

holds for all $i \in [1, n]$, the adversary must make more than q candidate queries.

Proof. Let $N = q^{\frac{1}{n}}$. All queries in $\mathbb{Q}^{(i)}$ must be either valid or invalid. Let VQ_i denote the number of valid queries at the level i of tree. Let IQ_i denote the number of invalid queries at the level i of tree. If the rate $R^{(i)}$ holds for all $i \in [1, n]$, we have the following deduction from the first level to the last level based on the above two claims, where only candidate queries are counted.

- **Level 1.** All queries are from the root and there is one valid query only, which is also a candidate query. That is, $VQ_1 = 1$. To make sure the rate is less than $1/N$, the adversary must make $IQ_1 \geq (N-1) \cdot VQ_1 + 1$ invalid queries that are also candidate queries. The total number of candidate queries in this level therefore is $VQ_1 + IQ_1$. Hence, according to the Claim 1, the total number of valid queries in the next level is $VQ_1 + IQ_1$.
- **Level 2.** According to the result in the level 1, the number of valid queries is $VQ_2 = VQ_1 + IQ_1$. According to Claim 2, these valid queries are also candidate queries. To make sure the rate is less than $1/N$, the adversary must make $IQ_2 \geq (N-1) \cdot VQ_2 + 1$ invalid queries that are also candidate queries. The total number of candidate queries in this level therefore is $VQ_2 + IQ_2$. Hence, according to Claim 1, the total number of valid queries in the next level is $VQ_2 + IQ_2$.
- **Level 3.** According to the result in the level 2, the number of valid queries is $VQ_3 = VQ_2 + IQ_2$. According to Claim 2, these valid queries are also candidate queries. To make sure the rate is less than $1/N$, the adversary must make $IQ_3 \geq (N-1) \cdot VQ_3 + 1$ invalid queries that are also candidate queries. The total number of candidate queries in this level therefore is $VQ_3 + IQ_3$. Hence, according to Claim 1, the total number of valid queries in the next level is $VQ_3 + IQ_3$.
- The result in the level i is the same as the previous analysis.
- **Level $n-1$.** According to the result in the level $n-2$, the number of valid queries is $VQ_{n-1} = VQ_{n-2} + IQ_{n-2}$. According to Claim 2, these valid queries are also candidate queries. To make sure the rate is less than $1/N$, the adversary must make $IQ_{n-1} \geq (N-1) \cdot VQ_{n-1} + 1$ invalid queries that are also candidate queries. The total number of candidate queries in this level therefore is $VQ_{n-1} + IQ_{n-1}$. Hence, according to Claim 1, the total number of valid queries in the next level is $VQ_{n-1} + IQ_{n-1}$.
- **Level n.** According to the result in the level $n-1$, the number of valid queries is $VQ_n = VQ_{n-1} + IQ_{n-1}$. To make sure the rate is less than $1/N$, the adversary must make $IQ_n \geq (N-1) \cdot VQ_n + 1$ invalid queries. The total query number in this level therefore is $VQ_n + IQ_n$. All queries are treated as candidate queries.

From the above analysis, we obtain the following results for all $i \in [1, n]$.

$$\begin{aligned} VQ_1 + IQ_1 &= N + 1 \\ VQ_i + IQ_i &\geq VQ_i + (N-1) \cdot VQ_i + 1 \\ &= N \cdot VQ_i + 1 \\ &> N \cdot VQ_i \\ &= N \cdot (VQ_{i-1} + IQ_{i-1}). \end{aligned}$$

Then, we yield

$$\sum_{i=1}^{n} (VQ_i + IQ_i) > (VQ_n + IQ_n) > N^{n-1}(VQ_1 + IQ_1) > N^n = q.$$

This completes the proof of Lemma 1. □

Based on the above definitions and explanations, we are ready to give the proof of Theory 3.

Proof of Theory 3. In the simulation, the number d is randomly chosen by the simulator and all instances $(I_1, I_2, \cdots, I_n)$ are indistinguishable. The adversary therefore does not know d. The query set $\mathbb{Q}$ generated by the adversary hence is independent of d.

According to Lemma 1, if the adversary makes q queries at most, there must exist an integer $i^* \in [1, n]$ satisfying

$$R^{(i^*)} = \frac{\text{The number of valid queries in } \mathbb{Q}^{(i^*)}}{\text{The number of candidate queries in } \mathbb{Q}^{(i^*)}} \geq \frac{1}{q^{\frac{1}{n}}}.$$

When $d = i^*$, the simulator can remove all useless queries in $\mathbb{Q}^{(i^*)}$ because $\mathcal{C}[I_i, P]$ for all $i \in [d+1, n]$ are computable by the simulator. Then, the success probability of picking a valid query from all candidate queries is at least $1/q^{\frac{1}{n}}$. The success probability $\Pr[suc]$ given in Theory 3 holds because

$$\begin{aligned}\Pr[suc] &= \sum_{i=1}^{n} \Pr[suc|d=i] \Pr[d=i] \\ &\geq \Pr[suc|d=i^*] \Pr[d=i^*] \\ &= \frac{1}{q^{\frac{1}{n}}} \cdot \frac{1}{n}.\end{aligned}$$

This completes the proof of Theory 3. □

3.3 Variant

The success probability given in Theory 3 is the lower bound probability because the probability $\Pr[suc|d=i] > 0$ holds for all $i \neq i^*$. We can repeat hash operations in the iterated random oracle to obtain a larger lower bound success probability.

Theory 4 (Improved Iterated Random Oracle). *Let H be a random oracle. Suppose an adversary, who is given instances $(I_1, I_2, \cdots, I_n)$ generated by the simulator, must make a set of queries $\mathbb{Q}$ ($|\mathbb{Q}| = q$) including a challenge query $\overline{\mathcal{Q}}^* = H^{k-1}(\overline{\mathcal{Q}}_*^{(n)})$ to the random oracle, where $\overline{\mathcal{Q}}_*^{(n)}$ is defined as*

$$\overline{\mathcal{Q}}_*^{(i)} = H^k(\overline{\mathcal{Q}}_*^{(i-1)}) \,||\, \mathcal{C}[I_i, P] \,||\, i: \quad i \in [1, n], \quad H(\overline{\mathcal{Q}}_*^{(0)}) = 0_\epsilon \text{ is an empty string.}$$

We can construct a simulator who controls the random oracle to solve the hard problem P using the query set $\mathbb{Q}$ with success probability at least $k/(nq^{\frac{1}{n}})$. Here, $H^i(\overline{\mathcal{Q}})$ is to repeat hash operation on $\overline{\mathcal{Q}}$ for i times. $H^i(\overline{\mathcal{Q}}) = H\left(H^{i-1}(\overline{\mathcal{Q}})\right)$ and $H^0(\overline{\mathcal{Q}}) = \overline{\mathcal{Q}}$.

$H^{k-1}(\overline{\mathcal{Q}}_*^{(n)})$	$H^{k-2}(\overline{\mathcal{Q}}_*^{(n)})$	$\cdots$	$H(\overline{\mathcal{Q}}_*^{(n)})$	$\overline{\mathcal{Q}}_*^{(n)}$
$H^{k-1}(\overline{\mathcal{Q}}_*^{(n-1)})$	$H^{k-2}(\overline{\mathcal{Q}}_*^{(n-1)})$	$\cdots$	$H(\overline{\mathcal{Q}}_*^{(n-1)})$	$\overline{\mathcal{Q}}_*^{(n-1)}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$H^{k-1}(\overline{\mathcal{Q}}_*^{(2)})$	$H^{k-2}(\overline{\mathcal{Q}}_*^{(2)})$	$\cdots$	$H(\overline{\mathcal{Q}}_*^{(2)})$	$\overline{\mathcal{Q}}_*^{(2)}$
$H^{k-1}(\overline{\mathcal{Q}}_*^{(1)})$	$H^{k-2}(\overline{\mathcal{Q}}_*^{(1)})$	$\cdots$	$H(\overline{\mathcal{Q}}_*^{(1)})$	$\overline{\mathcal{Q}}_*^{(1)}$

In this theory, the adversary must make $k \cdot n$ queries to obtain the challenge query $\overline{\mathcal{Q}}^* = H^{k-1}(\overline{\mathcal{Q}}_*^{(n)}) \in \mathbb{Q}$.

The query on $H^i(\overline{\mathcal{Q}})$ requires the adversary to make a query on $H^{i-1}(\overline{\mathcal{Q}})$ first. In particular, the query on $\overline{\mathcal{Q}}_*^{(j)}$ requires the adversary to make a query on $H^{k-1}(\overline{\mathcal{Q}}_*^{(j-1)})$ to obtain $H^k(\overline{\mathcal{Q}}_*^{(j-1)})$ to compose $\overline{\mathcal{Q}}_*^{(j)}$. The proof of this theory is based on a slightly different lemma where the rate is $k/q^{\frac{1}{n}}$. This is because the total number of queries in each level is $k \cdot (VQ_i + IQ_i)$ instead of $(VQ_i + IQ_i)$. The other analysis is similar and we omit them here without redundancy. Therefore we have the success probability shown in the theory.

3.4 Comparison of Success Probability

We compare the success probability of finding the solution from the query set among the traditional approach, the Cash-Kiltz-Shoup approach and the iterated random oracle, where concrete integers $n = 10$ and $k = 10$ are chosen. The result is given in Table 2. It shows that the iterated random oracle has a very small finding loss compared to the traditional approach even the iteration time n is very small. With a proper hash repeating time k, it further improves the success probability. Notice that the Cash-Kiltz-Shoup's approach is the most efficient approach, but it is not a universal approach for any computational hard problem.

Table 2. Comparison of success probability.

	$q = 2^{40}$	$q = 2^{50}$	$q = 2^{60}$
Traditional approach	$\frac{1}{2^{40}}$	$\frac{1}{2^{50}}$	$\frac{1}{2^{60}}$
Cash-Kiltz-Shoup [10,11]	1	1	1
Iterated random oracle with $n = 10, k = 1$	$\frac{1}{160}$	$\frac{1}{320}$	$\frac{1}{640}$
Iterated random oracle with $n = 10, k = 10$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$

3.5 Comparison of Query Efficiency and Finding Efficiency

The price to pay for small finding loss from the iterated random oracle is the efficiency loss in the generation of challenge query. Recall that the challenge

query is associated with one instance computation in the traditional approach (Theory 1) and two instance computations in the Cash-Kiltz-Shoup [10,11] approach (Theory 2). The challenge query in the iterated random oracle is associated with n instance computations and n queries or $n \cdot k$ queries. The efficiency loss is in the linear of n. Fortunately, n can be as small as 10 in the iteration. Furthermore, when the efficiency is mainly dominated by the computation of $\mathcal{C}[I_i, P]$, they can be performed in parallel because all computations are independent.

In the iterated random oracle, the simulator needs to compute $\mathcal{C}[I_i, P]$ for all $i \in [d+1, n]$, where d is randomly chosen from $[1, n]$[1], in order to remove all useless queries. Then, the simulator randomly picks one query from all candidate queries. Hence, the time cost of finding a solution is mainly dominated by instance computations and the time complexity is $O(n)$. In comparison with the other two approaches, the simulator in the traditional approach (Theory 1) directly picks one solution in a random way and the time complexity is $O(1)$. The simulator in the Cash-Kiltz-Shoup [10,11] approach (Theory 2) has to test each query until it finds the correct solution. Therefore, their time complexity is (q) more expensive than the iterated random oracle.

3.6 Remarks of Simulation Based on Theories

The introduced three theories for finding loss can be described as follows in a general summary. Suppose an adversary, who is given an instance $I_{\mathcal{A}}$ generated by the simulator, must make a set of queries $\mathbb{Q}$ ($|\mathbb{Q}| = q$) including a challenge query $\overline{\mathcal{Q}}^* = \mathcal{C}[I_{\mathcal{A}}, P_{\mathcal{A}}]$ to the random oracle. Here, $\mathcal{C}[I_{\mathcal{A}}, P_{\mathcal{A}}]$ is the solution to the instance $I_{\mathcal{A}}$ under the computational hard problem $P_{\mathcal{A}}$ which is defined by the simulator. We aim to construct a simulator to solve a hard problem P using the query set $\mathbb{Q}$.

In the corresponding simulator construction, the simulator is given an instance I under the hard problem P and aims to solve it with the help of the adversary. The simulator should construct an instance $I_{\mathcal{A}}$ for the adversary using the given instance I and define the hard problem $P_{\mathcal{A}}$ such that $\mathcal{C}[I, P]$ will appear in the query set. The resulting results $(I_{\mathcal{A}}, P_{\mathcal{A}})$ from the traditional approach, Cash-Kiltz-Shoup's approach and our approach are different. We remark that the successful construction of such a simulator is not the end of simulation. It merely introduces the approach of how to find the correct solution from the adversary's query set. To complete the reduction, the simulator must enable to use the created instance $I_{\mathcal{A}}$ to simulate the proposed cryptosystem and make sure the challenge query including $\mathcal{C}[I_{\mathcal{A}}, P_{\mathcal{A}}]$ will appear in the query set. This is required in the security reduction because the adversary is not to solve a hard problem for the simulator but is going to break a cryptosystem.

[1] When $d = n$, the simulator does not need to compute any $\mathcal{C}[I, P]$.

4 Tight Security in Security Transformation for Encryption

The principle application of the iterated random oracle is the security transformation from a key encapsulation mechanism with one-way security to an encryption with indistinguishability security, whose reduction is tight. In this section, we show how to achieve such a security transformation without expanding ciphertext size.

A key encapsulation mechanism (KEM) is an asymmetric encryption whose encryption algorithm will generate a random key (a.k.a. the encapsulation key), together with a corresponding ciphertext (a.k.a. the encapsulation). The random key is then used for symmetric encryption while the encapsulation forms part of the message ciphertext to deliver the random key in an asymmetric manner. Any receiver who owns a valid secret key can decapsulate the random key from the encapsulation. In the definition of one-way security for KEM, the challenger generates a challenge ciphertext CT^* for the adversary and the aim of the adversary is to return the corresponding challenge random key.

We observe that any KEM with one-way security does not have a security loss in finding a correct solution, if the random key is the solution to a computational hard problem in security reduction. This is because the adversary only returns one answer to the simulator, which is the correction solution to a hard problem. However, in the IND-CHP security reduction with the help of random oracles, the correct solution is hidden in a large query set made by the adversary. In this section, we show how to fill this gap by using the iterated random oracle.

Our security transformation is based on the KEM of functional encryption, namely functional key encapsulation mechanism (FKEM). The functional encryption can be seen as a generalized asymmetric encryption including public key encryption, identity-based encryption and attribute-based encryption. We adopt the FKEM because the iterated random oracle is a general approach fitting for all asymmetric encryptions.

Our security transformation can be applied to any FKEM. However, this generic transformation could be accompanied with a long ciphertext under iterated random oracles. This is because the challenge ciphertext must be associated with n different instances using the iterated random oracle, where the adversary is required to compute n solutions to different instances. To obtain a short ciphertext after transformation, these n instances must have shared input parameters. We extract one special type of FKEM from all FKEM with the following two properties.

– Firstly, global system parameters Param will be defined for FKEM, where many master key pairs $(\widetilde{\mathsf{mpk}}_1, \widetilde{\mathsf{msk}}_1), (\widetilde{\mathsf{mpk}}_2, \widetilde{\mathsf{msk}}_2), \cdots, (\widetilde{\mathsf{mpk}}_n, \widetilde{\mathsf{msk}}_n)$ can be generated with this global parameters. We note that these global system parameters are very common in an asymmetric encryption. It could include the definitions of pairing group, chosen generator and hash functions. All of these parameters are shared and used by different users or authorities.

– Secondly, the ciphertext encapsulation is computed without the input of master public keys, which will be the shared input parameters for all generated master key pairs. We note that many asymmetric encryptions fall into this type, such as the ElGamal public-key encryption scheme [21], the Boneh-Franklin identity-based encryption scheme [8] and the Waters identity-based encryption scheme [30]. One instantiation is given at the end of this section.

In the remaining of this section, we first give the definition of FKEM under our chosen type, and then show how to transform the FKEM with one-way security to a functional encryption with indistinguishability security against a chosen-plaintext attack (CPA) and a chosen-ciphertext attack (CCA).

4.1 Functional Key Encapsulation Mechanism

The functional key encapsulation mechanism (FKEM) is defined as follows.

Functional Key Encapsulation Mechanism

- $\mathsf{Param} \xleftarrow{\$} \mathsf{SysGen}(1^\lambda)$. This algorithm takes as input the security parameter λ and outputs the global system parameters Param.
- $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(\mathsf{Param})$. This algorithm takes as input Param and outputs the master key pair $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{usk} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{Param}, \mathsf{mpk}, \mathsf{msk}, \mathsf{upk})$. This algorithm takes as input Param, the master key pair $(\mathsf{mpk}, \mathsf{msk})$ and upk (*will be explained later*) and outputs the user secret key usk.
- $(\mathsf{C}, \mathsf{K}) \xleftarrow{\$} \mathsf{Encap}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}, \mathsf{r})$. This algorithm consists of two sub-algorithms.
  - $\mathsf{C} \xleftarrow{\$} \mathsf{Encap_c}(\mathsf{Param}, \mathsf{str}, \mathsf{r})$. This sub-algorithm takes as input Param, a string str (*will be explained later*), a randomness r and outputs the encapsulation C. The encapsulation generation is independent of mpk.
  - $\mathsf{K} \xleftarrow{\$} \mathsf{Encap_k}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}, \mathsf{r})$. This sub-algorithm takes as input $(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}, \mathsf{r})$ and outputs the encapsulation key K.
- $\mathsf{K} \xleftarrow{\$} \mathsf{Decap}(\mathsf{Param}, \mathsf{mpk}, \mathsf{upk}, \mathsf{usk}, \mathsf{C})$. This algorithm takes as input $(\mathsf{Param}, \mathsf{mpk}, \mathsf{upk}, \mathsf{usk})$ and the encapsulation C, and outputs the encapsulation key K or $\perp$.

Definition 1 (Correctness). *For any* $(\mathsf{C}, \mathsf{K}) \xleftarrow{\$} \mathsf{Encap}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}, \mathsf{r})$ *and* $\mathsf{usk} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{Param}, \mathsf{mpk}, \mathsf{msk}, \mathsf{upk})$*, we have that,*

$$\mathsf{Decap}(\mathsf{Param}, \mathsf{mpk}, \mathsf{upk}, \mathsf{usk}, \mathsf{C}) = \begin{cases} \mathsf{K} & F(\mathsf{upk}, \mathsf{str}) = 1, \\ \perp & \textit{otherwise}, \end{cases}$$

where $\mathsf{Param} \xleftarrow{\$} \mathsf{SysGen}(1^\lambda)$ *and* $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(\mathsf{Param})$. *The function* F *evaluates the relationship between the* upk *and the string* str.

The key pair $(\mathsf{mpk}, \mathsf{msk}), (\mathsf{upk}, \mathsf{usk})$, the string str and the function F have different representations in specified asymmetric encryptions. For example,

- In a public-key encryption, $\mathsf{mpk} = \mathsf{upk}$ is the public key while $\mathsf{msk} = \mathsf{usk}$ is the corresponding secret. str is also a public key and the function $F(\mathsf{upk}, \mathsf{str}) = 1$ if and only if $\mathsf{str} = \mathsf{mpk} = \mathsf{upk}$.
- In an identity-based encryption, upk is the identity of user and str is the identity of receiver. The function $F(\mathsf{upk}, \mathsf{str}) = 1$ if and only if $\mathsf{str} = \mathsf{upk}$.
- In an identity-based broadcast encryption, upk is the identity of user and str is the identity set of receivers. The function $F(\mathsf{upk}, \mathsf{str}) = 1$ if and only if upk is one of identities in the identity set str.
- In a ciphertext-policy attribute-based encryption, upk is an attribute set of a user while str is an access policy. The function $F(\mathsf{upk}, \mathsf{str}) = 1$ if and only if the access policy str accepts the attribute set upk.
- In a key-policy attribute-based encryption, upk is an access policy for a user while str is an attribute set. $F(\mathsf{upk}, \mathsf{str}) = 1$ if and only if the access policy upk accepts the attribute set str.
- In an inner-product encryption, both upk and str are vectors. $F(\mathsf{upk}, \mathsf{str}) = 1$ if and only if the inner product $\mathsf{upk} \cdot \mathsf{str} = 0$.

Definition 2 (One-Way FKEM). *A functional key encapsulation mechanism (*SysGen, Setup, KeyGen, Encap, Decap*) is one-way secure if for any PPT adversary* $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{OW}}_{\mathcal{A},\mathsf{FKEM}}(\lambda) = \Pr\left[\mathsf{K}' = \mathsf{K}^* : \begin{array}{l} \mathsf{Param} \xleftarrow{\$} \mathsf{SysGen}(1^\lambda); \\ (\mathsf{mpk}^*, \mathsf{msk}^*) \xleftarrow{\$} \mathsf{Setup}(\mathsf{Param}); \\ \mathsf{str}^* \leftarrow \mathcal{A}^{\mathcal{O}_K(\cdot)}(\mathsf{Param}, \mathsf{mpk}^*); \\ (\mathsf{C}^*, \mathsf{K}^*) \xleftarrow{\$} \mathsf{Encap}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}^*, \mathsf{r}^*); \\ \mathsf{K}' \leftarrow \mathcal{A}^{\mathcal{O}_K(\cdot)}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}^*, \mathsf{C}^*) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{O}_K(\cdot)$ *is a key generation oracle that on input of any* upk, *returns* $\mathsf{usk} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{Param}, \mathsf{mpk}, \mathsf{msk}, \mathsf{upk})$ *on the condition that* $F(\mathsf{upk}, \mathsf{str}^*) \neq 1$.

The definition of function encryption is similar with the FKEM except the encryption algorithm and the decryption algorithm. The encryption algorithm additionally takes as input a message and returns a ciphertext for the message directly. While the decryption algorithm directly returns the message or outputs failure. The corresponding security model under indistinguishability against a chosen-plaintext attack and a chosen-ciphertext attack is also similar except that the adversary outputs $\mathsf{str}^*, \mathsf{m}_0, \mathsf{m}_1$ for challenge and the challenge ciphertext is encrypted with a random message from $\{m_0, m_1\}$ chosen by the simulator. We define IND-CCA for FE in the following definition. The definition of IND-CPA is the same as IND-CCA except that the adversary cannot access the decryption oracle in the security model.

Definition 3 (IND-CCA FE). *A functional encryption (*SysGen, Setup, *KeyGen,* Encrypt, Decrypt*) is IND-CCA secure if for any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{IND-CCA}}_{\mathcal{A},\mathsf{FKEM}}(\lambda) = \Pr\left[\begin{array}{l} \mathsf{coin}' \\ = \\ \mathsf{coin} \end{array} : \begin{array}{l} \mathsf{Param} \xleftarrow{\$} \mathsf{SysGen}(1^\lambda); \\ (\mathsf{mpk}^*, \mathsf{msk}^*) \xleftarrow{\$} \mathsf{Setup}(\mathsf{Param}); \\ (\mathsf{str}^*, \mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{K}(\cdot),\mathcal{O}_\mathsf{D}(\cdot)}(\mathsf{Param}, \mathsf{mpk}^*); \\ \mathsf{coin} \xleftarrow{R} \{0,1\} \\ \mathsf{CT}^* \xleftarrow{\$} \mathsf{Encypt}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}^*, \mathsf{r}^*, \mathsf{m}_{\mathsf{coin}}); \\ \mathsf{coin}' \leftarrow \mathcal{A}^{\mathcal{O}_K(\cdot),\mathcal{O}_D(\cdot)}(\mathsf{Param}, \mathsf{mpk}, \mathsf{str}^*, \mathsf{CT}^*) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{O}_K(\cdot)$ *is a key generation oracle that on input of any* upk*, returns* $\mathsf{usk} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{Param}, \mathsf{mpk}, \mathsf{msk}, \mathsf{upk})$ *on the condition that* $F(\mathsf{upk}, \mathsf{str}^*) \neq 1$ *and* $\mathcal{O}_D(\cdot)$ *is a decryption oracle that on input of any* str, CT*, returns* $\{\mathsf{m}, \perp\} \xleftarrow{\$} \mathsf{Decrypt}(\mathsf{Param}, \mathsf{mpk}, \mathsf{upk}, \mathsf{usk}, \mathsf{CT})$ *on the condition that* $\mathsf{str} \neq \mathsf{str}^*$ *or* $\mathsf{CT} \neq \mathsf{CT}^*$*.*

4.2 Generic Conversion from OW-FKEM to IND-CPA-FE with Tight Reduction

Let $\mathsf{Param}_{\mathsf{OW}}$ be the global system parameters of FKEM with one-way security. Let $(\widetilde{\mathsf{mpk}}_i, \widetilde{\mathsf{msk}}_i)$ for all $i \in [1, n]$ be n master key pairs of FKEM and $\widetilde{\mathsf{usk}}_i$ be the secret key of upk generated from $(\widetilde{\mathsf{mpk}}_i, \widetilde{\mathsf{msk}}_i)$. Here, n can be as small as $n = 10$ depending on the choice of security loss. We choose n pairs in order to compute a different encapsulation key under each key pair, such that all n encapsulation keys can be iterated together following the iterated random oracle approach to generate the final encapsulation key. The functional encryption with IND-CPA security is constructed as follows.

SysGen: Choose a secure one-way hash function $H : \{0,1\}^* \to \{0,1\}^\ell$, where the message space is $\{0,1\}^\ell$. The global system parameters of FE are

$$\mathsf{Parama} = (\mathsf{Param}_{\mathsf{OW}}, H).$$

Setup: Set the master public key mpk and the master secret key msk of FE as

$$\mathsf{mpk} = (\widetilde{\mathsf{mpk}}_1, \widetilde{\mathsf{mpk}}_2, \cdots, \widetilde{\mathsf{mpk}}_\mathsf{n}), \quad \mathsf{msk} = (\widetilde{\mathsf{msk}}_1, \widetilde{\mathsf{msk}}_2, \cdots, \widetilde{\mathsf{msk}}_\mathsf{n}).$$

KeyGen: Taking as input Param, mpk, msk, upk, run the key generation algorithm $\mathsf{KeyGen}(\mathsf{Param}, \widetilde{\mathsf{mpk}}_\mathsf{i}, \widetilde{\mathsf{msk}}_\mathsf{i}, \mathsf{upk}) \xrightarrow{\$} \widetilde{\mathsf{usk}}_\mathsf{i}$ for all $i \in [1, n]$ and output the private key usk for upk as

$$\mathsf{usk} = (\widetilde{\mathsf{usk}}_1, \widetilde{\mathsf{usk}}_2, ..., \widetilde{\mathsf{usk}}_\mathsf{n}).$$

Encrypt: Taking as input Param, mpk, str and a message $\mathsf{m} \in \{0,1\}^\ell$, create the ciphertext CT for upk as follows

– Choose a random r for the Encap algorithm.
– Compute $\mathsf{C}_1 = \mathsf{Encap}_\mathsf{c}(\mathsf{Param}_\mathsf{OW}, \mathsf{str}, \mathsf{r})$.
– Compute $\mathsf{K}_\mathsf{i} = \mathsf{Encap}_\mathsf{k}(\mathsf{Param}_\mathsf{OW}, \widetilde{\mathsf{mpk}}_\mathsf{i}, \mathsf{str}, \mathsf{r})$ for all $i \in [1, n]$.
– Compute the iteration as

$$A_i = H(A_{i-1}) \;||\; \mathsf{K}_\mathsf{i} \;||\; i: \quad i \in [1, n], \quad \text{where } H(A_0) = 0_\epsilon.$$

– Set $\mathsf{C}_2 = H(A_n) \oplus \mathsf{m}$.

The output ciphertext is $\mathsf{CT} = (\mathsf{C}_1, \mathsf{C}_2)$.

Decrypt: Taking as input $\mathsf{Param}, \mathsf{mpk}, \mathsf{str}, \mathsf{upk}, \mathsf{usk}$ and a ciphertext $\mathsf{CT} = (\mathsf{C}_1, \mathsf{C}_2)$, decrypt the message as

– Compute $\mathsf{K}_\mathsf{i} = \mathsf{Decap}(\mathsf{Param}_\mathsf{OW}, \widetilde{\mathsf{mpk}}_\mathsf{i}, \mathsf{upk}, \widetilde{\mathsf{usk}}_\mathsf{i}, \mathsf{C}_1)$ for all $i \in [1, n]$.
– Compute the iteration as

$$B_i = H(B_{i-1})||\mathsf{K}_\mathsf{i}||i: \quad i \in [1, n], \quad \text{where } H(B_0) = 0_\epsilon.$$

– Compute $\mathsf{m} = \mathsf{C}_2 \oplus H(B_n)$.

This completes the description of FE construction. Without counting the size of the encrypted message, the ciphertext size is the same as FKEM. That is, the generic conversion from OW-FKEM to IND-CPA-FE does not expand the ciphertext size. This conversion without expanding ciphertext requires that the encapsulation is independent of the master public key $\widetilde{\mathsf{mpk}}$. Otherwise, the ciphertext is composed of n number of distinct C_1 generated under a different $\widetilde{\mathsf{mpk}}_i$. In the following theorem, we prove that the IND-CPA security of FE can be tightly reduced to one-way security of an FKEM.

Theorem 1. *Let H be a random oracle. If there exists an adversary $\mathcal{A}$ who makes q queries to H has an advantage ϵ in the IND-CPA security model against the constructed encryption scheme, then we can construct a simulator $\mathcal{B}$ that has advantage $\mathsf{Adv}^{\mathsf{OW}}_{\mathcal{B},\mathsf{FKEM}}(\lambda) = \frac{2\epsilon}{nq^{\frac{1}{n}}}$ in breaking the underlying FKEM in the one-way security model.*

Proof. Suppose there exists an adversary $\mathcal{A}$ who can break the above encryption scheme with an advantage ϵ. We construct a simulator $\mathcal{B}$ to break the one-way security of the underlying FKEM. The reduction works as follows.

Setup: $\mathcal{B}$ first obtains $(\mathsf{Param}_\mathsf{OW}, \widetilde{\mathsf{mpk}}^*)$ from FKEM. It then picks a random $d \in [1, n]$, and runs the setup algorithm $\mathsf{Setup}(\mathsf{Param}_\mathsf{OW}) \xrightarrow{\$} (\widetilde{\mathsf{mpk}}_\mathsf{i}, \widetilde{\mathsf{msk}}_\mathsf{i})$ for all $i \in [1, n] \backslash d$ to generate master key pairs. Finally, it sets $\mathsf{mpk} = (\widetilde{\mathsf{mpk}}_1, ..., \widetilde{\mathsf{mpk}}_\mathsf{n})$ where

$$\widetilde{\mathsf{mpk}}_\mathsf{i} = \begin{cases} \widetilde{\mathsf{mpk}}_\mathsf{i} & \text{if } i \neq d, \\ \widetilde{\mathsf{mpk}}^* & \text{otherwise}, \end{cases}$$

and $\mathsf{Param} = \mathsf{Param}_{\mathsf{OW}}$ where H is treated as a random oracle controlled by the simulator. Finally, the simulator returns $(\mathsf{Param}, \mathsf{mpk})$ to $\mathcal{A}$.

H-Query: $\mathcal{B}$ maintains a hash list L to record all queries to the random oracle H. If a query $\overline{\mathcal{Q}}$ has been made and appears in the list $(\overline{\mathcal{Q}}, \overline{\mathcal{R}})$, $\mathcal{B}$ responds with the same response $\overline{\mathcal{R}}$. Otherwise, the simulator randomly chooses $\overline{\mathcal{R}}$ from $\{0,1\}^{\ell}$ as the response $\overline{\mathcal{R}} = H(\overline{\mathcal{Q}})$ and adds $(\overline{\mathcal{Q}}, \overline{\mathcal{R}})$ into the list.

Phase 1: $\mathcal{A}$ requests the secret key of upk in this phase, which is adaptively chosen by the adversary. The simulator $\mathcal{B}$ first queries upk to the key generation oracle $\mathcal{O}_K(\cdot)$ which returns $\widetilde{\mathsf{usk}}$ and sets $\widetilde{\mathsf{usk}}_{\mathsf{d}} = \widetilde{\mathsf{usk}}$. For all other $i \in [1,n]\backslash d$, $\mathcal{B}$ runs $\mathsf{KeyGen}(\mathsf{Param}, \mathsf{mpk}_{\mathsf{i}}, \mathsf{msk}_{\mathsf{i}}, \mathsf{upk}) \xrightarrow{\$} \widetilde{\mathsf{usk}}_{\mathsf{i}}$ by itself to compute $\widetilde{\mathsf{usk}}_i$. Finally, it sets $\mathsf{usk} = (\widetilde{\mathsf{usk}}_1, \widetilde{\mathsf{usk}}_2, ..., \widetilde{\mathsf{usk}}_{\mathsf{n}})$ and returns usk to $\mathcal{A}$ as the query response.

Challenge: $\mathcal{A}$ outputs two distinct challenge messages m_0, m_1 from $\{0,1\}^n$ and a challenge string str^* with the restriction that for any upk queried in the **Phase 1**, $F(\mathsf{upk}, \mathsf{str}^*) \neq 1$. $\mathcal{B}$ then forwards str^* to FKEM and obtains the challenge encapsulation ciphertext C^*. Finally, $\mathcal{B}$ randomly chooses $\mathsf{R} \in \{0,1\}^{\ell}$ and sets the challenge ciphertext as

$$\mathsf{CT}^* = (\mathsf{C}^*, \mathsf{R}).$$

Phase 2: $\mathcal{A}$ issues more secret key queries on any chosen upk such that $F(\mathsf{upk}, \mathsf{str}^*) \neq 1$. $\mathcal{B}$ responds the same as in the **Phase 1**.

Output: Finally, $\mathcal{A}$ outputs its guess $coin' \in \{0,1\}$. $\mathcal{B}$ then follows the approach in Theory 3 to find the underlying key K^* from the recorded hash list L to break the FKEM.

This completes the description of simulation and solution. All master key pairs are generated from the setup algorithm of FKEM. They are therefore indistinguishable from the view of the adversary, such that the adversary has no advantage in guessing d. The random oracle is simulated using truly random string, and hence the simulator performs a correct simulation on the random oracle. Let $\mathsf{C}^* = \mathsf{Encap}_{\mathsf{c}}(\mathsf{Param}_{\mathsf{OW}}, \mathsf{str}^*, \mathsf{r}^*)$. Since C^* is generated from the encapsulation algorithm and R^* is randomly chosen, the challenge ciphertext is a one-time pad unless the adversary queries A_n^*, which is defined as

$$A_i^* = H(A_{i-1}^*) \,||\, \mathsf{Encap}_{\mathsf{k}}(\mathsf{Param}_{\mathsf{OW}}, \widetilde{\mathsf{mpk}}_{\mathsf{i}}, \mathsf{str}^*, \mathsf{r}^*) \,||\, i: \quad i \in [1,n], \quad H(A_0^*) = 0_{\epsilon}.$$

We have

$$K^* = \mathsf{Encap}_{\mathsf{k}}(\mathsf{Param}_{\mathsf{OW}}, \widetilde{\mathsf{mpk}}_{\mathsf{d}}, \mathsf{str}^*, \mathsf{r}^*) = \mathsf{Encap}_{\mathsf{k}}(\mathsf{Param}_{\mathsf{OW}}, \widetilde{\mathsf{mpk}}^*, \mathsf{str}^*, \mathsf{r}^*),$$

in A_d^* is the solution to the FKEM. The approach of finding the correct encapsulation key exactly falls into Theory 3 where the simulator can successfully pick a valid query with probability $1/(nq^{\frac{1}{n}})$. According to the definition of advantage, the adversary will make such a query with probability 2ϵ to the random oracle. We therefore yield Theorem 1. □

4.3 Generic Conversion from OW-FKEM to IND-CCA-FE

Given an FKEM composed of

$$\mathsf{C} = \mathsf{Encap_c}(\mathsf{Param_{OW}}, \mathsf{str}, \mathsf{r}),$$
$$\mathsf{K_i} = \mathsf{Encap_k}(\mathsf{Param_{OW}}, \widetilde{\mathsf{mpk_i}}, \mathsf{str}, \mathsf{r}) : \ i \in [1, n],$$

we have shown how to construct an IND-CPA FE via

$$\mathsf{CT} = \Big(\mathsf{Encap_c}(\mathsf{Param_{OW}}, \mathsf{str}, \mathsf{r}), \quad H(A_n) \oplus \mathsf{m}\Big),$$

where $A_i = H(A_{i-1}) \mid\mid \mathsf{K_i} \mid\mid i : \quad i \in [1, n]$, and $H(A_0) = 0_\epsilon$.

We can further transfer the conversion from FKEM to FE with IND-CCA security by applying the Fujisaki-Okamoto transformation approach [19,20]. This approach requires two more one-way secure hash functions H_1, H_2 in the global system parameters and they are also treated as random oracles in the security proof. The first hash function H_1 has the same output space as the randomness r and the second one H_2 has the same output space as H.

Taking as input $\mathsf{Param}, \mathsf{mpk}, \mathsf{str}$ and a message $\mathsf{m} \in \{0,1\}^\ell$, the encryption algorithm for IND-CCA security works as follows.

- Choose a random string $\sigma \in \{0,1\}^\ell$ and compute $\mathsf{r} = \mathsf{H_1}(\sigma, \mathsf{m})$.
- Run the IND-CPA encryption algorithm using the randomness r to encrypt σ, which returns

$$(\mathsf{C_1}, \mathsf{C_2}) = \Big(\mathsf{Encap_c}\Big(\mathsf{Param_{OW}}, \mathsf{str}, \mathsf{H_1}(\sigma, \mathsf{m})\Big), \quad H(A_n) \oplus \sigma\Big).$$

- Set $\mathsf{C_3} = H_2(\sigma) \oplus \mathsf{m}$.

The output ciphertext is

$$\mathsf{CT} = (\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}) = \Big(\mathsf{Encap_c}(\mathsf{Param_{OW}}, \mathsf{str}, \mathsf{H_1}(\sigma, \mathsf{m})), \quad \mathsf{H}(\mathsf{A_n}) \oplus \sigma, \quad \mathsf{H_2}(\sigma) \oplus \mathsf{m}\Big).$$

In the corresponding decryption algorithm, the decryptor first runs the IND-CPA decryption algorithm to obtain σ, and then it computes $H_2(\sigma) \oplus \mathsf{C_3}$ to obtain m. Finally, it outputs the message m if $\mathsf{C_1}$ is the generation using the randomness $H_1(\sigma, \mathsf{m})$. Otherwise, it simply returns $\perp$.

It is not hard to obtain the security proof based on the proposed security reduction for CPA security and the Fujisaki-Okamoto transformation. First, all key queries will be generated the same as the proof in Theorem 1; Second, all decryption queries will be responded using the Fujisaki-Okamoto transformation approach. Finally, the challenge ciphertext is simulated using $(\mathsf{C}^*, \mathsf{R_1}, \mathsf{R_2})$, where C^* is the challenge encapsulation from FKEM and $\mathsf{R_1}, \mathsf{R_2}$ are random strings. From the view of the adversary, if the adversary has an advantage in distinguishing the encrypted message, it must make a query on σ. The probability of obtaining σ is bounded by a random guess which is negligible for a large ℓ and bounded by breaking the IND-CPA construction. While the probability of breaking the IND-CPA construction is bounded by making a query on A_n. Therefore, if σ appears in the query list with probability 2ϵ, the probability of querying A_n is nearly 2ϵ. The simulator then is able to break the underlying FKEM with probability $2\epsilon/(nq^{1/n})$ by applying the approach in Theory 3.

4.4 Identity-Based Key Encapsulation Mechanism

At the end of this section, we give an instantiation using the Park-Lee identity-based encryption [28], which can be modified to a key encapsulation mechanism satisfying the requirement for short ciphertext in transformation. We choose this scheme as an example because there is no security loss during the private key simulation. By using the iterated random oracle, the corresponding encryption with indistinguishability security can be tightly reduced to solve the Bilinear Diffie-Hellman problem.

SysGen: This algorithm takes as input the security parameter λ. It selects a pairing group $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, g, p, e)$ and one secure one-way hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$. Then it randomly chooses $u \in \mathbb{G}$. The global system parameters Param are

$$\mathsf{Param} = (\mathbb{PG}, u, H_1).$$

Setup: It randomly chooses $\alpha \in \mathbb{Z}_p$ and computes $e(g,g)^\alpha$. The algorithm returns a master public/secret key pair $(\mathsf{mpk}, \mathsf{msk})$ as

$$\mathsf{mpk} = e(g,g)^\alpha, \qquad \mathsf{msk} = \alpha.$$

KeyGen: The key generation algorithm takes as input Param, an identity $ID \in \{0,1\}^*$ and the master key pair $(\mathsf{mpk}, \mathsf{msk})$. It randomly chooses $s, t_k \in \mathbb{Z}_p$ and creates the private key as

$$d_{ID} = (d_0, d_1, d_2, d_3) = \Big(t_k,\ g^s,\ \big(H_1(ID)u^{t_k}\big)^s, g^\alpha u^s\Big).$$

It requires that the random number t_k for ID is the same in the private key generation.

Encap: The encryption algorithm takes as input Param, the master public key mpk and an identity ID. It randomly chooses $r, t_c \in \mathbb{Z}_p$ and creates the ciphertext and the encapsulation key as follows.

$$\mathsf{C} = \mathsf{Encap_c}(\mathsf{Param}, \mathsf{ID}, \mathsf{r}) = \Big(t_c,\ \ (H_1(ID)u^{t_c})^r,\ \ g^r\Big),$$
$$\mathsf{K} = \mathsf{Encap_k}(\mathsf{Param}, \mathsf{mpk}, \mathsf{ID}, \mathsf{r}) = e(g,g)^{\alpha \cdot r}.$$

Decap: The decryption algorithm takes as input Param, the master public key mpk, an identity ID, a private key d_{ID} and a ciphertext $\mathsf{C} = (\mathsf{C}_0, \mathsf{C}_1, \mathsf{C}_2)$. It computes the random key as

$$\mathsf{K} = e(d_3, \mathsf{C}_2) \cdot \left(\frac{e(\mathsf{C}_1, d_1)}{e(\mathsf{C}_2, d_2)}\right)^{-\frac{1}{t_c - t_k}}.$$

The correctness of the decapsulation is showed as follows.

$$\begin{aligned}
\mathsf{K} &= e(d_3, \mathsf{C}_2) \cdot \left(\frac{e(\mathsf{C}_1, d_1)}{e(\mathsf{C}_2, d_2)}\right)^{-\frac{1}{t_c - t_k}} \\
&= e(g^{\alpha} u^s, g^r) \cdot \left(\frac{e\Big((H_1(ID)u^{t_c})^r, g^s\Big)}{e\left(g^r, \Big(H_1(ID)u^{t_k}\Big)^s\right)}\right)^{-\frac{1}{t_c - t_k}} \\
&= e(g,g)^{\alpha r} \cdot e(u,g)^{rs} \cdot \Big(e(u,g)^{rs(t_c - t_k)}\Big)^{-\frac{1}{t_c - t_k}} \\
&= e(g,g)^{\alpha r}.
\end{aligned}$$

Theorem 2. *Let H_1 be a random oracle. If there exists an adversary who can break the Park-Lee identity-based key encapsulation mechanism with (t, q_1, q_k, ϵ) in the one-way security model, where the adversary makes q_1 queries to H_1 and q_k numbers of private keys, then we can construct a simulator to solve the BDH problem with $(t + T_s, \epsilon)$ where T_s denotes the time cost of simulation.*

Proof. Suppose there exists an adversary $\mathcal{A}$ who can break the identity-based encryption scheme. We can construct a simulator $\mathcal{B}$ to solve the BDH problem. Given as input the instance (g, g^a, g^b, g^c) in the pairing group $\mathbb{PG}$, the simulator aims to compute $e(g,g)^{abc}$. $\mathcal{B}$ interacts with the adversary as follows.

Setup: $\mathcal{B}$ picks a random $z \in \mathbb{Z}_p$, sets $u = g^{z-a}, \alpha = ab$ and computes $e(g,g)^{\alpha} = e(g^a, g^b)$. Then, it gives $\mathsf{Param} = (\mathbb{PG}, u)$ and $\mathsf{mpk} = e(g,g)^{\alpha}$ except H_1 to the adversary, where H_1 is treated as a random oracle controlled by the simulator.

H-Query: $\mathcal{B}$ maintains a hash list L_1 to record all queries to the random oracle H_1. If a query ID_i has been made and $(ID_i, x_i, y_i, H_1(ID_i))$ is in the list, $\mathcal{B}$ responds with $H_1(ID_i)$. Otherwise, $\mathcal{B}$ randomly chooses $x_i, y_i \in \mathbb{Z}_p$, sets $H_1(ID_i) = g^{x_i a + y_i}$ and adds $(ID_i, x_i, y_i, H_1(ID_i))$ into the hash list.

Phase 1: $\mathcal{A}$ requests private keys of identities in this phase. For the query on ID, $\mathcal{B}$ first runs the H_1 query to get the corresponding $(ID, x, y, H_1(ID))$, randomly chooses $s \in \mathbb{Z}_p$ and computes the private key as

$$\begin{aligned}
d_0 &= t_k = x \\
d_1 &= g^{b+s} \\
d_2 &= g^{b(y+xz)+s(y+xz)} \\
d_3 &= g^{zs+zb-sa},
\end{aligned}$$

which can be computed by the simulator. Let $s' = b + s$ and $t_k = x$. We have

$$\begin{aligned}
(d_1, d_2, d_3) &= \Big(g^{s'}, \ \ (H_1(ID)u^{t_k})^{s'}, \ \ g^{\alpha} u^{s'}\Big) \\
&= \Big(g^{b+s}, \ \ (g^{xa+y} g^{x(z-a)})^{b+s}, \ \ g^{ba} g^{(z-a)(b+s)}\Big) \\
&= \Big(g^{b+s}, \ \ g^{b(y+xz)+s(y+xz)}, \ \ g^{zs+zb-sa}\Big).
\end{aligned}$$

Therefore, $d_{ID} = (d_0, d_1, d_2, d_3)$ is a valid private key of ID.

Challenge: The adversary $\mathcal{A}$ outputs an identity ID^* for challenge, where the adversary never requested the private key of ID^*. Let the query response of ID in the random oracle be $(ID^*, x^*, y^*, H_1(ID^*))$. The simulator $\mathcal{B}$ sets the challenge encapsulation as

$$\mathsf{C}^* = \left(x^*,\ g^{(y+x^*z)c},\ g^c\right).$$

Ler $r = c$ and $t_c = x^*$. We have

$$\begin{aligned}\mathsf{C}^* &= \left(t_c,\ (H_1(ID^*)u^{t_c})^r,\ \ g^r\right)\\ &= \left(x^*,\ (g^{x^*a+y^*}g^{x^*(z-a)})^c,\ \ g^c\right)\\ &= \left(x^*,\ g^{(y+x^*z)c},\ \ g^c\right).\end{aligned}$$

Therefore, C^* is a valid challenge encapsulation whose corresponding key K^* is

$$e(g,g)^{\alpha r} = e(g,g)^{abc}.$$

Output: Finally, $\mathcal{A}$ outputs K^* and the simulator outputs K^* as the solution to the BDH problem.

This completes the simulation and solution. We have that a, z are chosen randomly and independently such that both Param and mpk are indistinguishable from the real scheme. x, y are chosen randomly and independently such that the random oracle simulation is correctly performed. x^*, c are chosen randomly and independently such that the challenge ciphertext is indistinguishable from the real scheme. According to the definition of advantage and the assumption, we have the adversary will output K^* with probability ϵ and the simulator will solve the BDH problem with probability ϵ. This completes the proof of Theorem 2. □

5 Tight Reduction for Key Exchange

The iterated random oracle can also be applied in the key exchange for tight(er) reduction in the IND-CHP security reduction. However, we observe that the application is a little complicated due to many different definitions of key exchange protocols. In this section, we discuss how to apply the iterated random oracle for this cryptographic primitive and what will occur during the applications.

Identity-Based Non-Interactive Key Exchange (IB-NIKE). In the Sakai-Ohgishi-Kasahara IB-NIKE protocol [29], the private key of ID is $d_{ID} = H_1(ID)^\alpha$, where $\alpha \in \mathbb{Z}_p$ is the master secret key and $H_1 : \{0,1\}^* \to \mathbb{G}$ is a

collision-resistant hash function. Here, the IB-NIKE is constructed over a pairing group. The NIKE between ID_A and ID_B is defined as

$$\begin{aligned} K &= H\Big(e(d_{ID_A}, H_1(ID_B))\Big) \\ &= H\Big(e(d_{ID_B}, H_1(ID_A))\Big) \\ &= H\Big(e\Big(H_1(ID_A), H_1(ID_B)\Big)^{\alpha}\Big), \end{aligned}$$

where $H : \{0,1\}^* \to \{0,1\}^{\ell}$ is another secure one-way hash function.

The above IB-NIKE protocol is provably secure in the random oracle model (assuming H_1, H are random oracles) under the BDH assumption. The finding loss exists because the simulator cannot decide which query in the adversary's query set is the correct solution to the BDH problem. We can apply the iterated random oracle by iterating the section keys as follows.

– Compute the private key d_{ID} of ID as

$$d_{ID} = \Big(H_1(ID,1)^{\alpha}, \quad H_1(ID,2)^{\alpha}, \quad \cdots, H_1(ID,n)^{\alpha}\Big).$$

– Compute the i-th intermediate key between ID_A and ID_B as

$$K_i = e\Big(H_1(ID_A, i), H_1(ID_B, i)\Big)^{\alpha}.$$

– The final section key between ID_A and ID_B is $H(EK_n)$ where

$$EK_i = H(EK_{i-1}) \mid\mid K_i \mid\mid i : \quad i \in [1,n], \quad \text{where } H(EK_0) = 0_{\epsilon}.$$

It is not hard to prove its security when the simulator can simulate all private keys except $H(ID_A, d)^{\alpha}$ and $H(ID_B, d)^{\alpha}$ where $e\Big(H_1(ID_A, d), H_1(ID_B, d)\Big)^{\alpha}$ is programmed as the solution to the BDH problem. By applying Theory 3, we have the final security reduction will have a very small finding loss.

In comparison with the original scheme, ours gives a tighter reduction. We admit that our scheme requires each user to store n private keys. Although n can be as small as 10, the final key length is still longer compared to the length of original scheme by expending group size for security loss. Therefore, this construction is somewhat theoretically interesting only for short length. However, when parallel computation is allowed, all pairing computations and hash group operations in our scheme can be completed in parallel within a group. Our scheme will reduce the time cost because there is no need to expand group size for security loss.

Other (Authenticated) Key Exchange. Similarly, we can utilize the above approach to solve the finding loss in other key exchange protocols by generating n keys for each user instead of one. Only the d-th sub-key can be programmed to solve a hard problem while the others can be simulated or computed by the simulator. However, it seems that we still have to resort to the help of

decision oracle [18] because the simulator cannot simulate some section keys for the adversary. Let upk_A and upk_B be the challenge public keys. In the security model for key exchange, the adversary is allowed to launch section key query between for example upk_A and a corrupted user namely upk_C. Notice that the secret key of usk_A is unknown (only the d-th sub-key is programmed as unknown). When the secret key of usk_C is also unknown, the simulator cannot simulate the section keys correctly for the adversary especially on the random oracle without the help of decision oracle. If the assumption still needs a decision oracle, there is no finding loss in security reduction because the simulator can use the decision oracle to find the correct solution.

We emphasize that there is still a benefit of applying the iterated random oracle for key exchange, whose security assumption is a strong computational assumption with a decision oracle. Notice that the iterated random oracle will exponentially consume the hash queries from the adversary if it wants to hide the challenge query. Then, the simulator can make less number of queries to the decision oracle especially when the simulator wants to simulate the section key and find the correct solution. That is, by applying the iterated random oracle, we can adopt a strong computational assumption where the access time to the decision oracle is bounded with a small number. This assumption is better than the assumption with q times access to the decision oracle.

6 Conclusion

Finding loss is a common security loss in those security reductions for indistinguishability security under computational hard assumptions, when their decisional variants are also hard. This security loss will result in a significant loose reduction by a random pick because the number of queries can be as large as 2^{60}. The novel Cash-Kiltz-Shoup's approach is efficient without any finding loss, but can only be applied to a computational hard problem with a trapdoor test. We proposed a completely new approach, namely the iterated random oracle, as a universal approach for finding loss, which can be applied to any computational hard problem without any restriction on the adopted hard problem. The finding loss in this approach is very small. The corresponding success probability is $\frac{1}{64}$ compared to $\frac{1}{2^{60}}$ by a random pick. This approach has been applied to achieve a security transformation for encryption and key exchange towards tight(er) reductions.

References

1. Abdalla, M., Ben Hamouda, F., Pointcheval, D.: Tighter reductions for forward-secure signature schemes. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 292–311. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36362-7_19

2. Abdalla, M., Fouque, P.-A., Lyubashevsky, V., Tibouchi, M.: Tightly-secure signatures from lossy identification schemes. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 572–590. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29011-4_34
3. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30574-3_14
4. Attrapadung, N., Furukawa, J., Gomi, T., Hanaoka, G., Imai, H., Zhang, R.: Efficient identity-based encryption with tight security reduction. In: Pointcheval, D., Mu, Y., Chen, K. (eds.) CANS 2006. LNCS, vol. 4301, pp. 19–36. Springer, Heidelberg (2006). doi:10.1007/11935070_2
5. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) CCS 1993, pp. 62–73. ACM (1993)
6. Blazy, O., Kakvi, S.A., Kiltz, E., Pan, J.: Tightly-secure signatures from chameleon hash functions. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 256–279. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46447-2_12
7. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24676-3_14
8. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). doi:10.1007/3-540-44647-8_13
9. Boyen, X.: Miniature CCA2 PK encryption: tight security without redundancy. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 485–501. Springer, Heidelberg (2007). doi:10.1007/978-3-540-76900-2_30
10. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78967-3_8
11. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. J. Cryptology **22**(4), 470–504 (2009)
12. Chen, L., Chen, Y.: The n-Diffie-Hellman problem and its applications. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 119–134. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24861-0_9
13. Chevallier-Mames, B.: An efficient CDH-based signature scheme with a tight security reduction. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 511–526. Springer, Heidelberg (2005). doi:10.1007/11535218_31
14. Chevallier-Mames, B., Joye, M.: A practical and tightly secure signature scheme without hash function. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 339–356. Springer, Heidelberg (2006). doi:10.1007/11967668_22
15. Coron, J.: A variant of boneh-franklin IBE with a tight reduction in the random oracle model. Des. Codes Cryptography **50**(1), 115–133 (2009)
16. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998). doi:10.1007/BFb0055717
17. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
18. Fiore, D., Gennaro, R.: Making the Diffie-Hellman protocol identity-based. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 165–178. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11925-5_12

19. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_34
20. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. J. Cryptology **26**(1), 80–101 (2013)
21. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). doi:10.1007/3-540-39568-7_2
22. Gay, R., Hofheinz, D., Kiltz, E., Wee, H.: Tightly CCA-secure encryption without pairings. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 1–27. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49890-3_1
23. Gentry, C.: Practical identity-based encryption without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006). doi:10.1007/11761679_27
24. Goh, E., Jarecki, S., Katz, J., Wang, N.: Efficient signature schemes with tight reductions to the diffie-hellman problems. J. Cryptology **20**(4), 493–514 (2007)
25. Hofheinz, D., Jager, T.: Tightly secure signatures and public-key encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 590–607. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32009-5_35
26. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) CCS 2003, pp. 155–164. ACM (2003)
27. Kurosawa, K., Takagi, T.: Some RSA-based encryption schemes with tight security reduction. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 19–36. Springer, Heidelberg (2003). doi:10.1007/978-3-540-40061-5_2
28. Park, J.H., Lee, D.H.: An efficient ibe scheme with tight security reduction in the random oracle model. Des. Codes Crypt. **79**(1), 63–85 (2016)
29. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: The 2000 Symposium on Cryptography and Information Security, vol. 45, pp. 26–28 (2000)
30. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). doi:10.1007/11426639_7