# Updatable Public Key Encryption with Strong CCA Security: Security Analysis and Efficient Generic Construction

Kyoichi Asano[1] and Yohei Watanabe[1,2]

[1] The University of Electro-Communications, Tokyo, Japan.
[2] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan.
{k.asano, watanabe}@uec.ac.jp

March 21, 2024

**Abstract**

With applications in secure messaging, Updatable Public Key Encryption (UPKE) was proposed by Jost et al. (EUROCRYPT '19) and Alwen et al. (CRYPTO '20). It is a natural relaxation of forward-secure public-key encryption. In UPKE, we can update secret keys by using update ciphertexts which any sender can generate. The UPKE schemes proposed so far that satisfy the strong CCA security are Haidar et al.'s concrete construction (CCS '22) and Dodis et al's generic construction that use Non-Interactive Zero-Knowledge (NIZK) arguments. Yet, even despite the aid of random oracles, their concrete efficiency is quite far from the most efficient CPA-secure scheme. In this paper, we first demonstrate a simple and efficient attack against Dodis et al.'s strongly CCA-secure scheme, and show how to fix it. Then, based on the observation from the attack and fix, we propose a new strongly CCA-secure generic construction for a UPKE scheme with random oracles and show that its instantiation is almost as concretely efficient as the most efficient CPA-secure one.

## 1 Introduction

*Forward security* (or *forward secrecy*), which is practically crucial for secure communication protocols such as Transport Layer Security (TLS) and secure-messaging protocols such as Signal, guarantees that even if users' secret keys are compromised, messages previously exchanged before the compromise remain secure. In the symmetric-key setting, forward security can be easily achieved with Pseudo-Random Generator (PRG) [BY03]; one has an initial seed $s_0$ and iteratively runs $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ to generate $(k_i, s_i)$ from $s_{i-1}$, where $k_i$ is a one-time key and $s_i$ is a seed for the next update. However, the symmetric-key approach does not scale since it is hard to efficiently and dynamically add or remove users to or from a group. Although an interactive group key-agreement protocol (e.g., [KLL04]) allows users to join and leave the group dynamically, it could take longer to run, especially when some users are offline.[1] The requirement that all users are always active and behave faithfully is unrealistic, particularly in the context of secure messaging.

Forward-Secure Public-Key Encryption (FS-PKE) [CHK03] provides efficient key-evolving functionality in the public-key setting; any sender can update a public key "$\cdots \to \mathsf{pk}_i \to \mathsf{pk}_{i+1} \to \cdots$"

[1] A trivial solution in the public-key setting, i.e., refresh all users' pairs of public and secret keys, has the same problem and requires all users to be online for each update.

consistently with the corresponding secret key "$\cdots \rightarrow \mathsf{sk}_i \rightarrow \mathsf{sk}_{i+1} \rightarrow \cdots$," which is only updated by a receiver.[2] Any ciphertexts encrypted under $\mathsf{pk}_i$ can be decrypted with $\mathsf{sk}_i$, and the key chain guarantees the one-wayness of secret keys, i.e., forward security; even if $\mathsf{sk}_i$ is exposed, no ciphertexts encrypted under $\mathsf{pk}_j$ for any $j < i$ are compromised. However, a major drawback of FS-PKE is efficiency. Canetti et al. [CHK03] showed that Hierarchical Identity-Based Encryption (HIBE) [GS02, HL02] implies an asymptotically efficient FS-PKE scheme. Although HIBE can be instantiated from various assumptions over bilinear groups [BB04, BBG05], lattices [CHK+12], and even pairing-free groups [DG17], these concrete HIBE schemes (and, therefore, the resulting FS-PKE schemes) either require relatively expensive techniques such as pairings or are considerably less efficient than many simple PKE schemes.

Another important security requirement for modern secure messaging protocols is *backward security* (or *post-compromise security*), which guarantees that once the exposure of secret keys ends, security can be restored by updating secret keys. Unfortunately, FS-PKE never supports backward security since the public-key update procedure is deterministic; the adversary who has an exposed secret key $\mathsf{sk}_i$ can obtain a sequence of the future secret keys $\mathsf{sk}_{i+1}, \mathsf{sk}_{i+2}, \ldots$. Although several key-evolving PKE schemes support backward security [DKX+02, DFK+03, DFK+04], they seem to essentially require computational costs at least as expensive as HIBE, as in FS-PKE.

**Updatable Public-Key Encryption.** Motivated by practically efficient forward and backward security in secure messaging protocols, Jost et al. [JMM19] and Alwen et al. [ACD+20] recently put forward a notion of Updatable Public-Key Encryption (UPKE), a mild variant of the above key-evolving PKE schemes.[3] UPKE allows any sender to initiate key updates, whereas senders are forced to synchronize key-update intervals with the receiver, who has a secret key in FS-PKE. Specifically, any sender can update $\mathsf{pk}_i$ to $\mathsf{pk}_{i+1}$ and generate (encrypted) update information $\mathsf{up}_{i+1}$, called an *update ciphertext*. The receiver updates $\mathsf{sk}_i$ to $\mathsf{sk}_{i+1}$ with $\mathsf{up}_{i+1}$ so as to be consistent with $\mathsf{pk}_{i+1}$. Indeed, UPKE meets both forward and backward security (in the above sense) and can close the gap in the efficiency between the existing key-evolving PKE and standard PKE schemes; Jost et al. [JMM19] and Alwen et al. [ACD+20] showed a UPKE scheme based on the hashed ElGamal PKE scheme, i.e., from the computational Diffie–Hellman (CDH) assumption in the Random Oracle Model (ROM). Following these seminal works, Dodis et al. [DKW21] got rid of the random oracles and proposed two concrete UPKE schemes from Decisional Diffie–Hellman (DDH) and the Learning with Errors (LWE) assumptions in the standard model, respectively. Recently, Haidar et al. [HLP22] showed a UPKE scheme from the Decisional Composite Residuosity (DCR) assumption without random oracles, and Haidar et al. [HPS23] proposed a new UPKE scheme from the LWE assumption in the standard model, which is more efficient than Dodis et al.'s scheme.

**Chosen-Randomness Security vs. Chosen-Update Security.** In addition to standard security notions against Chosen-Plaintext Attacks (CPA) and Chosen-Ciphertext Attacks (CCA), there are two kinds of security notions for update operations in UPKE: *Chosen-Randomness* (CR) and *Chosen-Update* (CU) security. The seminal works [JMM19, ACD+20] originally considered the former in the context of secure messaging protocols; it captures the exposure of intermediate values of computations (including the randomness $r'$ used for key updates) during sessions, and therefore an adversary can obtain vulnerable secret keys $\mathsf{sk}'_i$ updated by the exposed randomness $r'$. For that

[2] In the seminal work [CHK03], FS-PKE allows receivers to update their secret keys $\mathsf{sk}_i$ without updating the corresponding public keys $\mathsf{pk}$. Nevertheless, FS-PKE can be viewed as a key-evolving PKE by setting $\mathsf{pk}_i = (\mathsf{pk}, i)$.

[3] To be precise, UPKE does not always support backward security, though it meets forward security in any case. Nevertheless, UPKE clearly differs in the *possibility* of backward security from FS-PKE (see [JMM19, Sec. 4] for the detailed discussion).

matter, the adversary is even allowed to arbitrarily choose "bad" randomness to have the target receiver update secret keys with it. The UPKE constructions listed in the last paragraph all satisfy CR-CPA security.

Dodis et al. [DKW21] introduced CU security; it allows the adversary to provide a (maliciously-generated) public key $\mathsf{pk}_i'$ and update ciphertext $\mathsf{up}_i'$ to the target receiver, instead of the exposed randomness $r'$. It is clear that CU security is stronger than CR since the CU adversary does not necessarily follow the update procedure and choice of randomness. Taking into account the presence of such malicious senders who attempt to impersonate honest ones, CU security is much closer to reality. Thus, we mainly focus on CU security in this paper. To date, there are only two approaches to achieving CU security. The first is generic transformations, as proposed by Dodis et al. [DKW21]. Specifically, they showed two generic transformations that lift CPA to CCA security and CR to CU security, respectively. Hence, any CR-CPA-secure UPKE scheme can be transformed into a CU-CCA-secure one. Both transformations employ one-time, strong, true-simulation $f$-extractable Non-Interactive Zero-Knowledge (NIZK) arguments. Although the NIZK arguments with those properties can be instantiated by, e.g., Groth–Sahai proof [GS08], its practically efficient instantiation is unclear. The second one is direct constructions in the ROM [HLP22, HPS23]. CR-CPA-secure schemes are constructed in the standard model as the first step, and then lifted to CU-CCA-secure ones without using an inefficient NIZK argument; instead, those schemes rely on efficient NIZK arguments for specific languages. Therefore, though they require random oracles to lift CR to CU security, the constructions are more practical than Dodis et al.'s transformation. Yet, despite the aid of random oracles, the concrete efficiency of the CU-CCA-secure schemes are quite far from Jost et al.'s CR-CPA-secure UPKE scheme (see Table 1).

Going back to the original motivation of UPKE [JMM19, ACD+20], it was introduced and investigated in the hope of being used for secure messaging protocols. Therefore, in that sense, the concrete rather than asymptotic efficiency of UPKE is a crucial issue; to be put to practical use, i.e., used as a critical primitive in secure messaging protocols, UPKE has to be comparable to the existing primitives, such as the X3DH protocol [MP] and the double ratchet protocol [PM] in the Singal [Sig]. Thus, the question we ask in this paper is:

*How efficiently can we make a CU-CCA-secure UPKE scheme? In particular, is it possible to realize a CU-CCA-secure UPKE scheme almost as concretely efficient as Jost et al.'s scheme?*

## 1.1 Our Contributions

In this paper, we give an affirmative answer to the above question; we show a CU-CCA-secure UPKE scheme almost as concretely efficient as the most-efficient-ever UPKE scheme, i.e., Jost et al.'s scheme. Indeed, Jost et al.'s UPKE scheme is just a variant of hashed ElGamal PKE, and therefore, our results show that UPKE could reach the same efficiency level as standard PKE. Specifically, our contributions are two-fold.

First, along the way to our main result above, we show a practical attack breaking CR/CU-CCA-security of Dodis et al.'s CPA-to-CCA transformation [DKW21] and how to modify it. Specifically, we formalize a special property of UPKE, called *non-influential randomness*, and show that it contradicts CCA security but *not* CPA. Indeed, all the existing CR-CPA-secure UPKE schemes [JMM19, ACD+20, DKW21, HLP22] satisfy this property. We then show that our attack is effective and efficient against CCA-secure UPKE schemes obtained via Dodis et al.'s CPA-to-CCA transformation (even in a CR setting) if the underlying CPA-secure UPKE scheme meets the non-influential randomness. To put it differently, the property is carried over to the resulting CCA-secure scheme. We also show how to modify their transformation, namely, how not to take over the special property in the transformation.

**Table 1:** Comparison for existing UPKE schemes with 128-bit security.

| Scheme | Secret key | Public key | Ciphertext | Update ciphertext | Security | ROM |
|---|---|---|---|---|---|---|
| Jost et al. [JMM19] | 256b | 512b | 512b | 512b | CR-CPA | yes |
| Dodis et al. [DKW21] | 1.3kb | 328kb | 328kb | 419Mb | CR-CPA | no |
| Haidar et al. [HLP22] | 4.6kb | 6.1kb | 12kb | 12kb | CR-CPA | no |
| Ours (with [JMM19]) | 256b | 512b | 512b | 1kb | CU-CPA | yes |
| Haidar et al. [HLP22] | 4.6kb | 6.1kb | 66kb | 12kb | CR-CCA | yes |
| Haidar et al. [HLP22] | 4.6kb | 9.2kb | 91kb | 105kb | CU-CCA | yes |
| Ours (with [JMM19]) | 256b | 640b | 768b | 512b | CR-CCA | yes |
| Ours (with [JMM19]) | 256b | 640b | 768b | 1kb | CU-CCA | yes |

Second, as our main result, we propose a new generic transformation from CR-CPA to CU-CCA security in the ROM. Unlike the existing transformation, our generic construction no longer requires NIZK arguments and is almost as concretely efficient as the underlying CR-CPA-secure UPKE scheme (see Table 1). Namely, as in the seminal works [JMM19, ACD+20], we use random oracles to circumvent a circular-security issue and make the resulting scheme efficient, and we differently employ random oracles for the lifting of CR to CU security; we apply the *double Fujisaki-Okamoto (FO) transformation* for that aim, whereas the Haidar's CU-CCA-secure schemes [HLP22, HPS23] employ the specific NIZK arguments in the ROM. In that sense, we only require the *one-wayness* of CR-CPA security, i.e., OW-CR-CPA security, not indistinguishability (IND-CR-CPA security).

**Efficiency Comparison.** Table 1 compares the efficiency of the CR/CU-CCA-secure UPKE schemes among existing schemes and our schemes instantiated with Jost et al.'s CR-CPA-secure scheme [JMM19]. We here omit lattice-based schemes [DKW21, HPS23] since it is hard to estimate concrete parameters, and they obviously require larger parameters than Jost et al.'s scheme. As a reference, we also show a comparison between Jost et al.'s CR-CPA-secure scheme in the ROM and Haidar et al.'s CR-CPA secure scheme in the standard model. Specifically, we compare the sizes of secret keys, public keys, ciphertexts, and update ciphertexts when satisfying 128-bit security. Based on this comparison, we can say that our CR/CU-CCA-secure UPKE schemes are more efficient than Haidar et al.'s CR/CU-CCA-secure UPKE schemes. Specifically, it can be seen that our schemes are efficient by an order of magnitude (2 to 100 times) for each parameter size. We note that the comparison in Table 1 does not take into account the security loss caused by security proofs, so it is necessary to consider the effect of the security loss when conducting a rigorous parameter evaluation. Although, as can be seen above, our transformation is quite efficient, the resulting CU-CPA/CCA secure scheme does not achieve public verifiability, which is an important property for CU security; we leave it as an open problem.

## 1.2 Technical Overview

In this section, we give an overview of our results.

**Updatable Public-Key Encryption.** We start describing the syntax of UPKE: it consists of standard PKE algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and the following update algorithms $(\mathsf{UpdPk}, \mathsf{UpdSk})$ for public and secret keys:

$$(\mathsf{pk}_i, \mathsf{up}_i) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{i-1}; r_i), \quad \mathsf{sk}_i \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{i-1}, \mathsf{sk}_{i-1}, (\mathsf{pk}_i, \mathsf{up}_i)),$$

where $r_i$ is update randomness used in $\mathsf{UpdPk}$, $\mathsf{up}_i$ is an update ciphertext, and it holds $\mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M})) = \mathsf{M}$ for any epoch $i$. Any sender can initiate the update procedure and execute $\mathsf{UpdPk}$ to update $\mathsf{pk}_{i-1}$ to $\mathsf{pk}_i$ and generate an update ciphertext $\mathsf{up}_i$. Receiving $(\mathsf{pk}_i, \mathsf{up}_i)$, a receiver updates the corresponding secret key $\mathsf{sk}_{i-1}$ to $\mathsf{sk}_i$.

As described earlier, there are two kinds of security notions related to the update procedures of UPKE: CR and CU security. Loosely speaking, CR security captures "leakage and malicious modifications to update randomness," whereas CU security captures "leakage and malicious modifications to whole update information (including update randomness)."

- *CR Security*: A CR adversary is allowed to arbitrarily choose update randomness $r_i^*$ and force the target receiver to run the $\mathsf{UpdPk}$ algorithm with $r_i^*$. It is worth noting that the CR adversary may choose the update randomness over arbitrary probability distribution. If chosen uniformly at random, it captures that the CR adversary observes the leakage of update randomness.
- *CU Security*: A CU adversary can create maliciously-generated public keys $\mathsf{pk}_i^*$ and update ciphertexts $\mathsf{up}_i^*$ and force the target receiver to use them for the $\mathsf{UpdSk}$ algorithms. Note that the CU adversary may not follow the protocol, whereas $\mathsf{pk}_i^*$ and $\mathsf{up}_i^*$ are honestly generated except for the choice of the update randomness in CR security. Hence, CU security is a stronger security notion than CR.

Combined with CPA and CCA security, there are four security notions for UPKE, CR-CPA, CU-CPA, CR-CCA, and CU-CCA, where the first is the weakest, and the last is the strongest.

**Existing Construction Approaches for CU-CCA-Secure UPKE.** All related works [JMM19, ACD+20, DKW21, HLP22, HPS23] have begun working on constructing CR-CPA-secure schemes directly, and then some of the works extended them to strongly secure ones, e.g., CU-CCA-secure schemes. In particular, only two existing ways to enhance CR-CPA security to CU-CCA.

- *CU-CCA Security in the ROM* [HLP22, HPS23]: Haidar et al. [HLP22] and Haidar et al. [HPS23] extended their CR-CPA-secure UPKE constructions from the DCR and LWE assumptions, respectively, to CU-CCA-secure schemes in the ROM by using specific and efficient NIZK arguments that support limited languages (but compatible with their CR-CPA-secure schemes). In particular, in the DCR-based construction [HLP22], they employed the variant of Naor–Yung transformation [NY90] with the Fiat–Shamir heuristic [FS86] to construct such an efficient NIZK argument. The downside of this approach is that this specific NIZK argument cannot be applied to other CR-CPA-secure UPKE schemes.
- *CU-CCA Security in the Standard Model* [DKW21]: Dodis et al. proposed two transformations for UPKE: the one lifts CPA security to CCA, and the other lifts CR security to CU, which we call the *CPA-to-CCA* and *CR-to-CU* transformations. These transformations also employ NIZK arguments. In particular, since Dodis et al. viewed (CR-CPA-secure) UPKE schemes as a kind of PKE schemes with circular security and leakage resilience, they required strong properties of NIZK arguments to lift CPA security to CCA *even in the leakage-resilient setting*. It is worth noting that the two transformations do not require random oracles, though it is unclear how to efficiently instantiate such a strong NIZK argument.

As seen above, both approaches are based on NIZK arguments, which seem to be an efficiency bottleneck: if one wants to realize CU-CCA-secure schemes without random oracles, inefficient NIZK arguments seem necessary; even with random oracles, one has to narrow down the language the NIZK supports to keep efficiency. Although Haidar et al.'s CU-CCA-secure scheme requires random oracles, their CR-CPA-secure UPKE scheme does not require any random oracles, so its parameter sizes are relatively large.

**A Simple Attack against Dodis et al.'s CPA-to-CCA Transformation.** Along the way to pursuing efficient CU-CCA-secure UPKE schemes, we find a simple and efficient attack against Dodis et al.'s CPA-to-CCA transformation. To be precise, this attack works if the underlying CPA-secure UPKE scheme satisfies a certain property, however, all the existing CPA-secure UPKE schemes [JMM19, ACD+20, DKW21, HLP22, HPS23] indeed meet it.

Let us briefly explain here with an instantiation of the CPA-to-CCA transformation from Jost et al.'s scheme [JMM19]. A public and secret keys are given by:

$$\mathsf{pk}_i := (g, h := g^x, \mathsf{CRS}) \text{ and } \mathsf{sk}_i := x,$$

where $g$ is a generator of a cyclic group $\mathbb{G}$ of order $q$, $x \in \mathbb{Z}_q$, and $\mathsf{CRS}$ is a common reference string of the underlying NIZK argument. The $\mathsf{UpdPk}$ and $\mathsf{UpdSk}$ algorithms are defined as follows.

- $\mathsf{UpdPk}(\mathsf{pk}_i)$: Choose update randomness $r \xleftarrow{\$} \mathbb{Z}_q$ and output $\mathsf{pk}_{i+1} := (g, h \cdot g^r, \mathsf{CRS})$ and $\mathsf{up}_{i+1} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, r)$.
- $\mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}))$: Run $r \leftarrow \mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{up}_{i+1})$ and output $\mathsf{sk}_{i+1} := x + r \bmod q$.

Here, let us briefly explain the CU-CCA-security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The adversary $\mathcal{A}$ can make either update or challenge queries: upon an update query on the update randomness $r_i$, where $i$ denotes the current epoch, the challenger $\mathcal{C}$ generates updated public and secret keys such that $\mathsf{pk}_{i+1} := (g, h \cdot g^{r_i}, \mathsf{CRS})$ and $\mathsf{sk}_{i+1} := x + r_i \bmod q$; upon the challenge query $(\mathsf{M}_0^*, \mathsf{M}_1^*)$, $\mathcal{A}$ receives the challenge ciphertext $\mathsf{ct}^*$, which is encrypted by $\mathsf{pk}_{i^*}$, where $i^*$ is the challenge epoch. Although $\mathcal{A}$ is not allowed to make a decryption query on $\mathsf{ct}^*$ at the challenge epoch $i^*$ to prevent a trivial attack, $\mathcal{A}$ can make a decryption query on $\mathsf{ct}^*$ once the public and secret keys are updated, i.e., the epoch goes by.

Now, $\mathcal{A}$ tries to break the CU-CCA security as follows. At the challenge epoch $i^*$, $\mathcal{A}$ sets update randomness $r_{i^*} := 0$ and makes an updated query on $r_{i^*}$. Then, $\mathcal{A}$ forces $\mathcal{C}$ to compute $(\mathsf{pk}_{i^*+1}, \mathsf{sk}_{i^*+1})$ such that $(\mathsf{pk}_{i^*+1}, \mathsf{sk}_{i^*+1}) = (\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*})$, since $\mathsf{pk}_{i^*+1} := (g, h \cdot g^0, \mathsf{CRS})$ and $\mathsf{sk}_{i^*+1} := \mathsf{sk}_{i^*} + 0 \bmod q$. As described above, $\mathcal{A}$ can now make a decryption query on the challenge ciphertext. Since $(\mathsf{pk}_{i^*+1}, \mathsf{sk}_{i^*+1}) = (\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*})$, $\mathcal{A}$ obtains the decryption result of the challenge ciphertext and breaks the CU-CCA security.

**Why Does the Attack Succeed?** The attack succeeds since if the underlying CPA-secure UPKE scheme meets the property that there exists update randomness $r^*$ such that $(\mathsf{pk}_{i^*+1}, \mathsf{sk}_{i^*+1}) = (\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*})$, the CPA-to-CCA transformation inherits it. In the main part, we formally reveal and define the property as *non-influential randomness*, and demonstrate a generalized version of the above attack. We want to emphasize that the property of non-influential randomness contradicts CCA security but does not contradict CPA, and all the existing CPA-secure schemes meet it.

**How to Fix the Flaw.** The above attack enables the adversary to get the decryption result of the challenge ciphertext since the adversary can create a situation for generating the same public and secret keys as those at the challenge epoch. To prevent this attack, they must have essentially different key pairs of public and secret keys in different epochs. We give an overview of our modification of Dodis et al.'s CPA-to-CCA transformation below.

1. Change the public key $\mathsf{pk}_i = (g, h, \mathsf{CRS})$ (and the corresponding relation) to $(g, h, \boxed{i}, \mathsf{CRS})$ to make it explicit that the public key has an epoch.

2. Change the $\mathsf{Enc}$ and $\mathsf{Dec}$ algorithms so that an epoch is embedded into a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \boxed{[i \| \mathsf{M}]})$, instead of $\mathsf{Enc}(\mathsf{pk}_i, \mathsf{M})$.

**Our Efficient Generic Construction.** Going back to our main goal: *to construct an efficient CU-CCA-secure UPKE scheme*. As mentioned earlier, towards our goal, it is preferable to avoid using NIZK arguments to achieve CU-CCA security. Therefore, we employ the Fujisaki-Okamoto (FO) transformation [FO99, FO13] *twice* to lift both CPA and CR security to CCA and CU, respectively.

First, let us see the encryption procedure of the original FO transformation below.

$$\mathsf{ct} := (\mathsf{Enc}(\mathsf{pk}, \sigma; \mathsf{H}(\sigma, \mathsf{SKE.Enc}(\mathsf{G}(\sigma), \mathsf{M})), \ \mathsf{SKE.Enc}(\mathsf{G}(\sigma), \mathsf{M}))\,,$$

where

- $\mathsf{SKE.Enc}$ is an encryption algorithm of symmetric-key encryption that takes as input a common secret key and a plaintext $\mathsf{M}$.
- $\sigma$ is a random value chosen from the plaintext space of PKE,
- $\mathsf{G}$ and $\mathsf{H}$ are random oracles.

The above encryption procedure is sufficient to achieve CCA security of traditional (i.e., non-updatable) PKE, but it is actually *insufficient* for the CCA security of UPKE. As discussed in the last paragraph, we need to embed the epochs into the public keys and ciphertexts not to have the property of non-influential randomness. We fix the above for UPKE by embedding the epoch $i$ into $\mathsf{H}$ as follows.

$$\mathsf{ct} := (\mathsf{Enc}(\mathsf{pk}_i, \sigma; \mathsf{H}(i, \sigma, \mathsf{SKE.Enc}(\mathsf{G}(\sigma), \mathsf{M}))), \ \mathsf{SKE.Enc}(\mathsf{G}(\sigma), \mathsf{M})).$$

Thanks to the above modification, the challenger in the CCA security game can easily reject the ciphertexts which generated at the different epoch $j$. Indeed, our attack described earlier no longer works. This is our CPA-to-CCA transformation without NIZK arguments; it yields better efficiency than the existing ones and is applicable to all the existing CPA-secure UPKE schemes.

Although our CR-to-CU transformation can be realized similarly to the above, we can fine-tune our variant of the FO transformation since, roughly speaking, there is no challenge query in the sense of update queries. Specifically, we do not need to embed an epoch into update ciphertexts since the challenger does not return the update information to the adversary, unlike the challenge query.

## 2 Preliminaries

**Notation.** Throughout the paper, $\lambda$ denotes a security parameter. For an $i$-bit binary string $\mathsf{s}_1 \in \{0,1\}^i$ and a $j$-bit binary string $\mathsf{s}_2 \in \{0,1\}^j$, let $[\mathsf{s}_1 \| \mathsf{s}_2] \in \{0,1\}^{i+j}$ denote an $(i+j)$-bit concatenation of $\mathsf{s}_1$ and $\mathsf{s}_2$. For a finite set $S$, $s \overset{\$}{\leftarrow} S$ denotes a sampling of an element of $s$ from $S$ uniformly at random and let$|S|$ denotes a cardinality of $S$. Probabilistic polynomial time is abbreviated as PPT.

**Lemma 1** (Difference Lemma [Sho04])**.** Let $A$, $B$, and $F$ be events defined in some probability distribution, and suppose that $A \wedge B \Leftrightarrow B \wedge \neg F$. Then, $|\Pr[A] - \Pr[B]| \leq \Pr[F]$ holds.

### 2.1 Updatable Public Key Encryption

**Syntax.** Let $\mathcal{M}$ and $\mathcal{R}$ be the message and the randomness spaces determined only by the security parameter, respectively. A UPKE scheme $\Pi$ consists of the five algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$ as follows:

$\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{pk}_0, \mathsf{sk}_0)$**:** On input the security parameter $\lambda$, it outputs an initial public key $\mathsf{pk}_0$ and an initial secret key $\mathsf{sk}_0$.

$\mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}) \rightarrow \mathsf{ct}$**:** On input a public key $\mathsf{pk}_i$ for epoch $i$ and a plaintext $\mathsf{M}$, it outputs a ciphertext $\mathsf{ct}$.

$\mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct}) \rightarrow \mathsf{M}$ or $\perp$**:** On input a public key $\mathsf{pk}_i$, a secret key for epoch $i$, and a ciphertext $\mathsf{ct}$, it outputs the plaintext $\mathsf{M}$ or $\perp$, which indicates the failure of decryption.

$\mathsf{UpdPk}(\mathsf{pk}_i) \rightarrow (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$**:** On input a public key $\mathsf{pk}_i$ for epoch $i$, it outputs a new public key $\mathsf{pk}_{i+1}$ and an update ciphertext $\mathsf{up}_{i+1}$ for next epoch $i+1$.

$\mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1})) \rightarrow \mathsf{sk}_{i+1}$ or $\perp$**:** On input a pubic key $\mathsf{pk}_i$, a secret key $\mathsf{sk}_i$ for epoch $i$, and an update ciphertext $\mathsf{up}_{i+1}$, it outputs a new secret key $\mathsf{sk}_{i+1}$ for next epoch $i+1$.

**Correctness.** For all $\lambda \in \mathbb{N}$, all $\ell \in \mathbb{N}$, all $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$, all $\mathsf{M} \in \mathcal{M}$, all $i \in \{0, \dots, \ell\}$, it is required that $\mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M})) = \mathsf{M}$ holds with overwhelming probability, where $(\mathsf{pk}_j, \mathsf{up}_j) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{j-1})$, $\mathsf{sk}_j \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{j-1}, \mathsf{sk}_{j-1}, (\mathsf{pk}_j, \mathsf{up}_j))$ for $j = 1, \dots, \ell$.

**Security.** In this paper, we almost follow the security notions defined by Dodis et al. [DKW21].

**Definition 1** (IND-CR-CPA Security [DKW21])**.** The IND-CR-CPA security of a UPKE scheme $\Pi$ is defined by a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:

**Init:** $\mathcal{C}$ runs $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives $\mathsf{pk}_0$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ is allowed to make the following update queries to $\mathcal{C}$.

> **Update query:** $\mathcal{A}$ is allowed to make the query on $r_i \in \mathcal{R}$ where $i$ is the current epoch. Upon the query, $\mathcal{C}$ runs $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_i; r_i)$ and $\mathsf{sk}_{i+1} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}))$.

**Challenge query:** $\mathcal{A}$ is allowed to make the query only once. Upon $\mathcal{A}$'s query on $(\mathsf{M}_0^*, \mathsf{M}_1^*) \in \mathcal{M}^2$, where $\mathsf{M}_0^*$ and $\mathsf{M}_1^*$ have the same length. Then $\mathcal{C}$ flips a coin $\mathsf{coin}^* \xleftarrow{\$} \{0,1\}$ and runs $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_c}, \mathsf{M}^*_{\mathsf{coin}^*})$ where $t_c$ is the current epoch. Finally, $\mathcal{C}$ returns $\mathsf{ct}^*$ to $\mathcal{A}$.

**Phase 2:** $\mathcal{A}$ is allowed to make update queries as in Phase 1.

**Reveal query:** For the query $\perp$ from $\mathcal{A}$, $\mathcal{C}$ chooses a randomness $r^* \xleftarrow{\$} \mathcal{R}$, and runs $(\mathsf{pk}^*_{t_r}, \mathsf{up}^*) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{t_r-1}; r^*)$ and $\mathsf{sk}^*_{t_r} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{t_r-1}, \mathsf{sk}_{t_r-1}, (\mathsf{pk}^*_{t_r}, \mathsf{up}^*_{t_r}))$ where $t_r$ is the current epoch. Then, $\mathcal{C}$ returns $(\mathsf{pk}^*_{t_r}, \mathsf{up}^*_{t_r}, \mathsf{sk}^*_{t_r})$ to $\mathcal{A}$.

**Guess:** At the end of the game, $\mathcal{A}$ returns $\widehat{\mathsf{coin}} \in \{0,1\}$ as a guess of $\mathsf{coin}$.

The adversary $\mathcal{A}$ wins in the above game if $\widehat{\mathsf{coin}} = \mathsf{coin}^*$ and the advantage is defined to

$$\mathsf{Adv}^{\text{IND-CR-CPA}}_{\Pi,\mathcal{A}}(\lambda) := \left| \Pr\left[\widehat{\mathsf{coin}} = \mathsf{coin}^*\right] - \frac{1}{2} \right|.$$

If $\mathsf{Adv}^{\text{IND-CR-CPA}}_{\Pi,\mathcal{A}}(\lambda)$ is negligible in the security parameter $\lambda$ for all PPT adversaries $\mathcal{A}$, a UPKE scheme $\Pi$ is said to satisfy IND-CR-CPA security.

Next, we define IND-CU-CCA security, the strongest security notion in UPKE, based on Dodis et al.'s one [DKW21, DKW22]. We suppose $\mathsf{UpdSk}$ also performs the consistency check in our definition, whereas Dodis et al. separately defined $\mathsf{UpdSk}$ and $\mathsf{VerifyUpd}$ algorithms.[4]

**Definition 2** (IND-CU-CCA Security)**.** The IND-CU-CCA security of a UPKE scheme $\Pi$ is defined by a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:

**Init:** $\mathcal{C}$ runs $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives $\mathsf{pk}_0$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ is allowed to make the following two types of queries to $\mathcal{C}$.

**Update query:** $\mathcal{A}$ is allowed to make the query on $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$ where $i$ is the current epoch. Upon the query, $\mathcal{C}$ runs $\mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}))$ and rejects update if it outputs $\perp$.

**Decryption query:** $\mathcal{A}$ is allowed to make the query on $\mathsf{ct}$. Upon the query, $\mathcal{C}$ runs $\mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct})$ and returns the result, where $i$ is the current epoch.

**Challenge query:** $\mathcal{A}$ is allowed to make the query only once. Upon $\mathcal{A}$'s query on $(\mathsf{M}_0^*, \mathsf{M}_1^*) \in \mathcal{M}^2$, where $\mathsf{M}_0^*$ and $\mathsf{M}_1^*$ have the same length. Then $\mathcal{C}$ flips a coin $\mathsf{coin}^* \xleftarrow{\$} \{0,1\}$ and runs $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_c}, \mathsf{M}^*_{\mathsf{coin}^*})$ where $t_c$ is the current epoch. Finally, $\mathcal{C}$ returns $\mathsf{ct}^*$ to $\mathcal{A}$.

**Phase 2:** $\mathcal{A}$ is allowed to make the queries as in Phase 1 with the following exceptions:

**Decryption query:** $\mathcal{A}$ is not allowed to make the query on $\mathsf{ct}^*$ without making an update query. In other words, $\mathcal{A}$'s query $\mathsf{ct}$ on a epoch $i$ satisfies $(i, \mathsf{ct}) \neq (t_c, \mathsf{ct}^*)$.

**Reveal query:** For the query $\perp$ from $\mathcal{A}$, $\mathcal{C}$ chooses a randomness $r^* \xleftarrow{\$} \mathcal{R}$, and runs $(\mathsf{pk}^*_{t_r}, \mathsf{up}^*_{t_r}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{t_r}; r^*)$ and $\mathsf{sk}^*_{t_r} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{t_r-1}, \mathsf{sk}_{t_r-1}, (\mathsf{pk}^*_{t_r}, \mathsf{up}^*_{t_r}))$ where $t_r$ is the current epoch. Then, $\mathcal{C}$ returns $(\mathsf{pk}^*_{t_r}, \mathsf{up}^*_{t_r}, \mathsf{sk}^*_{t_r})$ to $\mathcal{A}$.

**Guess:** At the end of the game, $\mathcal{A}$ returns $\widehat{\mathsf{coin}} \in \{0,1\}$ as a guess of $\mathsf{coin}$.

The adversary $\mathcal{A}$ wins in the above game if $\widehat{\mathsf{coin}} \in \{0,1\}$ as a guess of $\mathsf{coin}$.

$$\mathsf{Adv}^{\text{IND-CU-CCA}}_{\Pi,\mathcal{A}}(\lambda) := \left| \Pr\left[\widehat{\mathsf{coin}} = \mathsf{coin}^*\right] - \frac{1}{2} \right|.$$

If $\mathsf{Adv}^{\text{IND-CU-CCA}}_{\Pi,\mathcal{A}}(\lambda)$ is negligible in the security parameter $\lambda$ for all PPT adversaries $\mathcal{A}$, a UPKE scheme $\Pi$ is said to satisfy IND-CU-CCA security.

## 2.2 Symmetric Key Encryption

**Syntax.** Let $\mathcal{M}$ and $\mathcal{SK}$ be the message and the secret key spaces determined only by the security parameter, respectively. An SKE scheme $\Gamma$ consists of the two algorithms $(\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ as follows:

$\mathsf{SKE.Enc}(k, \mathsf{M}) \rightarrow \mathsf{ct}$**:** On input a secret key $k$ and a plaintext $\mathsf{M}$, it outputs a ciphertext.

$\mathsf{SKE.Dec}(k, \mathsf{ct}) \rightarrow \mathsf{M}$ or $\perp$**:** On input a secret key and a ciphertext $\mathsf{ct}$, it outputs the plaintext $\mathsf{M}$.

[4] This implies that our syntax does not support public verifiablity, and hence, fully offline updates in the CU setting. We leave it as an open problem to how we realize an efficient CR-to-CU transformation in the publicly verifiable setting.

**Correctness.** For all $\lambda \in \mathbb{N}$, all $k \in \mathcal{SK}$, all $\mathsf{M} \in \mathcal{M}$, it is required that $\mathsf{Dec}(k, \mathsf{Enc}(k, \mathsf{M})) = \mathsf{M}$ holds.

**Definition 3** (OT-CPA security)**.** The OT-CPA security of an SKE scheme $\Gamma$ is defined by a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The game is parameterized by the security parameter $\lambda$. The game proceeds as follows: $\mathcal{A}$ sends $(\mathsf{M}_0^*, \mathsf{M}_1^*) \in \mathcal{M}^2$ to $\mathcal{C}$. $\mathcal{C}$ chooses $\mathsf{coin}^* \xleftarrow{\$} \{0,1\}$, $k \xleftarrow{\$} \mathcal{SK}$, runs $\mathsf{ct}^* \leftarrow \mathsf{SKE.Enc}(k, \mathsf{M}_{\mathsf{coin}^*}^*)$, and sends $\mathsf{ct}^*$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs $\widehat{\mathsf{coin}}$ as a guess of $\mathsf{coin}^*$ and terminates the game. In this game, $\mathcal{A}$'s advantage is defined by

$$\mathsf{Adv}_{\Gamma,\mathcal{A}}^{\text{OT-CPA}}(\lambda) := \left| \Pr\left[ \widehat{\mathsf{coin}} = \mathsf{coin}^* \right] \right| .$$

If $\mathsf{Adv}_{\Gamma,\mathcal{A}}^{\text{OT-CPA}}(\lambda)$ is negligible in the security parameter $\lambda$ for all PPT adversaries $\mathcal{A}$, an SKE scheme $\Gamma$ is said to satisfy OT-CPA security.

### 2.3 Non-Interactive Zero-Knowledge

**Syntax.** A NIZK argument for a polynomial relation $R$ consists of the three algorithms as follows:

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{CRS}, \mathsf{tk}, \mathsf{ek})$**:** On input the security parameter $\lambda$, it outputs a common reference string (CRS), a trapdoor key $\mathsf{tk}$, and an extraction key $\mathsf{ek}$.[5]

$\mathsf{Prove}(\mathsf{CRS}, x, w) \rightarrow \pi$**:** On input a $\mathsf{CRS}$, a statement $x$, and a witness $w$ with $R(x, w) = 1$, it outputs a proof $\pi$.

$\mathsf{Vrfy}(\mathsf{CRS}, x, \pi) \rightarrow 1$ or $0$**:** On input a $\mathsf{CRS}$, a statement $x$, and a proof $\pi$, it outputs 1 or 0.

In the generic constructions by Dodis et al. [DKW21, DKW22], a NIZK is used that satisfies completeness, soundness, zero knowledge, and strong extractability. In this paper, we only introduce the definition of completeness because we only use completeness for the attack.

**Definition 4** (Completeness)**.** For all $\lambda \in \mathbb{N}$, all $(x, w) \in R$, and all $(\mathsf{CRS}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda)$, it is required that $\mathsf{Vrfy}(\mathsf{CRS}, x, \mathsf{Prove}(\mathsf{CRS}, x, w)) = 1$ holds.

## 3 Security Analysis of Dodis et al.'s UPKE Scheme

There are only two existing ways to achieve CU-CCA-secure UPKE schemes.

- Haidar et al. [HLP22] and Haidar et al. [HPS23] provided direct CR-CPA-secure UPKE constructions from the DCR and LWE assumptions, respectively, and extended them to CU-CCA-secure schemes with the aid of random oracles.

- Dodis et al. [DKW21] proposed two transformations for UPKE: the one converts CR/CU-CPA security into CR/CU-CCA, called the *CPA-to-CCA transformation*, and the other converts CR-CPA/CCA security into CU-CPA/CCA, called the *CR-to-CU transformation*. Since CR-CPA-secure UPKE schemes from the DDH and LWE assumptions, respectively, in the standard model, one obtains CU-CCA-secure UPKE schemes from those assumptions. It is worth noting that the two transformations do not require random oracles, though they employ NIZK arguments with strong properties instead.

[5] $\mathsf{tk}$ and $\mathsf{ek}$ are used to define security notions such as zero knowledge and strong extractability, although we omit their definitions.

In this section, we point out that Dodis et al.'s CPA-to-CCA transformation has a fatal flaw when the underlying CPA-secure UPKE scheme satisfies the property of *non-influential randomness*, which is introduced in Sec. 3.1. We can break the CCA security of the CPA-to-CCA transformation with our simple and efficient attack demonstrated in Sec. 3.2. Indeed, all existing CR-CPA-secure schemes [JMM19, ACD+20, DKW21, HLP22, HPS23] satisfy the particular property; therefore, *our security analysis shows that the CPA-to-CCA transformation does not work with any existing CPA-secure UPKE scheme*. Nevertheless, we also show that the transformation can be fixed with a slight modification in Sec. 3.3.

## 3.1 UPKE with Non-Influential Randomness

We reveal and formalize the property that all the existing CPA-secure UPKE schemes have, called *non-influential randomness*. Roughly speaking, we say a UPKE scheme has non-influential randomness if there exists efficiently computable randomness $r^*$ such that it holds $(\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}) = (\mathsf{pk}_i, \mathsf{sk}_i)$ for some epoch $i$, where $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_i; r^*)$, $\mathsf{sk}_{i+1} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}))$. Namely, the randomness $r^*$ has no influence on key updates. Note that such non-influential randomness can be a sequence; namely, if there exists a sequence of randomness $(r^*_i, r^*_{i+1}, \ldots, r^*_{i+\ell})$ such that it holds $(\mathsf{pk}_{i+\ell}, \mathsf{sk}_{i+\ell}) = (\mathsf{pk}_i, \mathsf{sk}_i)$ for some epoch $i$, where $(\mathsf{pk}_{i+j}, \mathsf{up}_{i+j}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{i+j-1}; r^*)$, $\mathsf{sk}_{i+j} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{i+j-1}, \mathsf{sk}_{i+j-1}, (\mathsf{pk}_{i+j}, \mathsf{up}_{i+j}))$ for every $j = 1, \ldots, \ell$.

**Definition 5** (UPKE with Non-Influential Randomness)**.** Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$ be a UPKE scheme. Consider a game between a PPT adversary $\mathcal{A}^*$ and a challenger $\mathcal{C}$ as follows:

**Init:** $\mathcal{C}$ sets a counter $\mathsf{ctr} = 0$ and runs $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives $\mathsf{pk}_0$ to $\mathcal{A}^*$.

**Update query:** $\mathcal{A}^*$ is allowed to iteratively make the update query. Upon the query, $\mathcal{C}$ sets $\mathsf{ctr} = \mathsf{ctr} + 1$, and runs $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{up}_{\mathsf{ctr}}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{\mathsf{ctr}-1})$ and $\mathsf{sk}_{\mathsf{ctr}} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{\mathsf{ctr}-1}, \mathsf{sk}_{\mathsf{ctr}-1}, (\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}))$. $\mathcal{C}$ returns $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{up}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$.

**Challenge query:** Upon $\mathcal{A}^*$'s query on $(r^*_1, r^*_2, \ldots, r^*_\ell)$, which is issued only once, where $\ell$ is polynomial in $\lambda$. For every $j = 1, 2, \ldots, \ell$, $\mathcal{C}$ runs $(\mathsf{pk}_{\mathsf{ctr}+j}, \mathsf{up}_{\mathsf{ctr}+j}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{\mathsf{ctr}+j-1}; r^*_j)$ and $\mathsf{sk}_{\mathsf{ctr}+j} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{\mathsf{ctr}+j-1}, \mathsf{sk}_{\mathsf{ctr}+j-1}, (\mathsf{pk}_{\mathsf{ctr}+j}, \mathsf{sk}_{\mathsf{ctr}+j}))$. $\mathcal{C}$ returns $(\mathsf{pk}_{\mathsf{ctr}+\ell}, \mathsf{sk}_{\mathsf{ctr}+\ell})$ and $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ as the output of the game and halts the game.

We say that the UPKE scheme has non-influential randomness if for all PPT adversaries $\mathcal{A}^*$, it holds

$$(\mathsf{pk}_{\mathsf{ctr}+\ell}, \mathsf{sk}_{\mathsf{ctr}+\ell}) = (\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}),$$

with overwhelming probability in $\lambda$.

It is worth noting that the above property does not contradict CR/CU-CPA security since the property brings the CPA adversary no benefit; even if the CPA adversary has the challenger produce $(\mathsf{pk}_{t_c+\ell}, \mathsf{sk}_{t_c+\ell})$ such that $(\mathsf{pk}_{t_c+\ell}, \mathsf{sk}_{t_c+\ell}) = (\mathsf{pk}_{t_c}, \mathsf{sk}_{t_c})$ for some $\ell \in \mathbb{N}$ and $t_c$ is the challenge epoch, all information the CPA adversary can obtain after the challenge query is the response to the reveal query $(\mathsf{pk}^*_{t_r}, \mathsf{up}^*_{t_r}, \mathsf{sk}^*_{t_r})$, which is randomized with fresh randomness chosen by the challenger.

Indeed, all the existing CR-CPA secure UPKE schemes [JMM19, ACD+20, DKW21, HLP22, HPS23] satisfy the property of non-influential randomness. For simplicity, we give a concrete example of non-influential randomness with a *variant* of Jost et al.'s CR-CPA secure scheme, which is the simplest UPKE scheme.[6] Let $\mathcal{M}$ and $\mathsf{H} : \{0,1\}^* \rightarrow \mathcal{M}$ denote a plaintext space and random oracle, respectively.

$\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{pk}_0, \mathsf{sk}_0)$**:** Choose a cyclic group $\mathbb{G}$ of order $q$ with a generator $g$, which is determined by the security parameter $\lambda$, and choose $x \xleftarrow{\$} \mathbb{Z}_q$. Output $\mathsf{pk}_0 := (g, g^x)$ and $\mathsf{sk}_0 := x$.

$\mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}) \rightarrow \mathsf{ct}$**:** Parse $\mathsf{pk}_i = (g, h)$ and output $\mathsf{ct} = (g^s, h^s \cdot \mathsf{M})$, where $s \xleftarrow{\$} \mathbb{Z}_q$.

$\mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct}) \rightarrow \mathsf{M}$**:** Parse $\mathsf{sk}_i = x$ and $\mathsf{ct} = (a, b)$, and output $b/a^x$.

$\mathsf{UpdPk}(\mathsf{pk}_i) \rightarrow (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$**:** Parse $\mathsf{pk}_i = (g, h)$, choose $r \xleftarrow{\$} \mathbb{Z}_q$, and output $\mathsf{pk}_{i+1} := (g, h \cdot g^r)$ and $\mathsf{up}_{i+1} := (g^s, \mathsf{H}(h^s) \oplus r)$, where $s \xleftarrow{\$} \mathbb{Z}_q$.

$\mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1})) \rightarrow \mathsf{sk}_{i+1}$**:** Parse $\mathsf{sk}_i = x$ and $\mathsf{up}_{i+1} = (a, b)$, calculate $r := \mathsf{H}(a^x) \oplus b$, and output $\mathsf{sk}_{i+1} := x + r \bmod q$.

We can construct the adversary $\mathcal{A}^*$ against the game in Def. 5 of the above scheme as follows: the challenger $\mathcal{C}$ generates $(\mathsf{pk}_0, \mathsf{sk}_0) = ((g, g^x), x)$ and gives $\mathsf{pk}_0$ to $\mathcal{A}^*$. $\mathcal{A}^*$ makes a challenge query $r^* := 0$ and the challenger runs the $\mathsf{UpdPk}$ and the $\mathsf{UpdSk}$ algorithms. Then, the public and secret keys at the epoch 1 is $(\mathsf{pk}_1, \mathsf{sk}_1) = ((g, g^x \cdot g^0), x + 0) = ((g, g^x), x)$, which is the same as $(\mathsf{pk}_0, \mathsf{sk}_0)$. Therefore, Jost et al.'s UPKE scheme indeed has non-influential randomness.

**Remark 1.** In the above attack against the variant of Jost et al.'s scheme, we demonstrated the simplest case; we use the fact that 0 is the identity element of the secret key space $\mathbb{Z}_q$ and $g^0 = 1$ is also the identity element of the public key space $\mathbb{G}$. A naive countermeasure is to modify $\mathsf{UpdPk}$ and $\mathsf{UpdSk}$ so that they halt if the randomness $r$ is the identity element. However, as stated in Def. 5, the adversary is allowed to make multiple update queries, and we can improve the above attack to circumvent the countermeasure: Suppose the adversary randomly chooses $r_1, r_2, \ldots, r_i \in \mathbb{Z}_q$ and use them as $i$ update queries. Now the challenger has a correctly generated key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. the adversary arbitrarily chooses $i$ and $\ell$, and randomly chooses $r'_1, r'_2, \ldots, r'_\ell \in \mathbb{Z}_q$ such that $\sum_{j=1}^{\ell} r'_j = 0 \bmod q$. After the challenge query on $(r'_1, r'_2, \ldots, r'_\ell)$, it obviously holds $(\mathsf{pk}_{i+\ell}, \mathsf{sk}_{i+\ell}) = (\mathsf{pk}_i, \mathsf{sk}_i)$. Although one can check all the previous keys to detect the above attack, it seems unrealistic and inefficient.

Although the property of non-influential randomness may coexist with CPA security by definition, in contrast, it hinders CCA security due to decryption queries. Specifically, the CCA adversary is allowed to make decryption queries on even the challenge ciphertext after the epoch when the challenge query is made. Therefore, the adversary easily breaks CCA security by issuing a decryption query on the challenge ciphertext at the epoch $\ell$ such that $(\mathsf{pk}_{t_c+\ell}, \mathsf{sk}_{t_c+\ell}) = (\mathsf{pk}_{t_c}, \mathsf{sk}_{t_c})$, where $t_c$ is the epoch when the adversary makes the challenge query.

### 3.2 Security Analysis of Dodis et al's CPA-to-CCA Transformation

We show a simple and efficient attack against the CPA-to-CCA transformation proposed by Dodis et al. [DKW21]. It clearly stems from the non-influential randomness property of the CCA-secure UPKE scheme obtained by the transformation. Put differently, if the underlying CPA-secure scheme

[6] Since Dodis et al.'s CPA-to-CCA transformation requires a NIZK for a relation $R = \{((\mathsf{pk}_i, \mathsf{ct}), (\mathsf{M}, r)) \mid \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}; r)\}$, we need to get rid of a RO from the $\mathsf{Enc}$ algorithm of Jost et al.'s scheme to apply the transformation. Therefore, we here show the variant of Jost et al.'s CR-CPA-secure scheme, which is secure under the DDH assumption in the ROM; the $\mathsf{Enc}$ algorithm, which is the same as that of the hashed ElGamal PKE, is replaced with the standard ElGamal PKE's one.

has non-influential randomness, the CPA-to-CCA transformation takes over the property. In that sense, our attack does not work if the underlying CPA-secure UPKE scheme does not meet the property; however, as mentioned above, all the known CPA-secure schemes do.

We describe the CPA-to-CCA transformation from any CR/CU-CPA-secure UPKE scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$ and any NIZK argument $\Omega = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ for the relation $R = \big\{((\mathsf{pk}_i, \mathsf{ct}), (\mathsf{M}, r)) \mid \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}; r)\big\}$ to a CR/CU-CCA-secure UPKE scheme $\Sigma = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{UpdPk}', \mathsf{UpdSk}')$ as follows.

**$\mathsf{Gen}'(1^\lambda) \to (\mathsf{pk}'_0, \mathsf{sk}'_0)$:** Generate an initial key pair as follows.

- $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$,
- $(\mathsf{CRS}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda)$.

Output $\mathsf{pk}'_0 := (\mathsf{pk}_0, \mathsf{CRS})$ and $\mathsf{sk}'_0 := \mathsf{sk}_0$.

**$\mathsf{Enc}'(\mathsf{pk}'_i, \mathsf{M}) \to \mathsf{ct}'$:** Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, \mathsf{CRS})$ and run

- $r \xleftarrow{\$} \mathcal{R}$,
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}; r)$,
- $\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, (\mathsf{pk}_i, \mathsf{ct}), (\mathsf{M}, r))$.

Output $\mathsf{ct}' = (\mathsf{ct}, \pi)$.

**$\mathsf{Dec}'(\mathsf{pk}'_i, \mathsf{sk}'_i, \mathsf{ct}') \to \mathsf{M}$ or $\bot$:** Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, \mathsf{CRS})$, $\mathsf{sk}'_i = \mathsf{sk}_i$, and $\mathsf{ct}' = (\mathsf{ct}, \pi)$. If $1 \leftarrow \mathsf{Vrfy}(\mathsf{CRS}, (\mathsf{pk}_i, \mathsf{ct}), \pi)$ holds, then run and output $\mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct})$. Otherwise, output $\bot$.

**$\mathsf{UpdPk}'(\mathsf{pk}'_i) \to (\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1})$:** Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, \mathsf{CRS})$ and run $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_i)$. Then, output $\mathsf{pk}'_{i+1} = (\mathsf{pk}_{i+1}, \mathsf{CRS})$ and $\mathsf{up}'_{i+1} = \mathsf{up}_{i+1}$.

**$\mathsf{UpdSk}'(\mathsf{pk}'_i, \mathsf{sk}'_i, (\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1})) \to \mathsf{sk}'_{i+1}$:** Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, \mathsf{CRS})$, $\mathsf{sk}'_i = \mathsf{sk}_i$, $\mathsf{pk}'_{i+1} = (\mathsf{pk}_{i+1}, \mathsf{CRS})$, and $\mathsf{up}'_{i+1} = \mathsf{up}_{i+1}$. Run $\mathsf{sk}_{i+1} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}))$ and output $\mathsf{sk}'_{i+1} := \mathsf{sk}_{i+1}$.

**Theorem 1.** Let $\Sigma = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{UpdPk}', \mathsf{UpdSk}')$ be the UPKE scheme constructed above. Then, if the underlying IND-CU-CPA-secure (resp., IND-CR-CPA-secure) UPKE scheme has non-influential randomness, there exists a PPT algorithm $\mathcal{A}$ that breaks IND-CU-CCA security (resp., IND-CR-CCA security).

*Proof.* It is sufficient to show how to break the IND-CU-CCA security of the above UPKE scheme if the underlying UPKE scheme has non-influential randomness, so we omit the case of IND-CR-CCA security. For this purpose, we use a PPT adversary $\mathcal{A}^*$, which can efficiently find non-influential randomness via the game in Def. 5, to construct a PPT adversary $\mathcal{A}$ that breaks IND-CU-CR-CCA security as follows. The challenger $\mathcal{C}$ begins the IND-CU-CCA security game, runs $(\mathsf{pk}'_0 = (\mathsf{pk}_0, \mathsf{CRS}), \mathsf{sk}'_0 = \mathsf{sk}_0) \leftarrow \mathsf{Gen}'(1^\lambda)$, and gives $\mathsf{pk}'_0$ to $\mathcal{A}$. $\mathcal{A}$ then gives $\mathsf{pk}_0$ to $\mathcal{A}^*$. $\mathcal{A}^*$ iteratively makes update queries to $\mathcal{A}$; for an $i$-th update query from $\mathcal{A}^*$, $\mathcal{A}$ runs $\mathsf{UpdPk}(\mathsf{pk}_{i-1})$ to get $(\mathsf{pk}_i, \mathsf{up}_i)$, and gives $(\mathsf{pk}'_i, \mathsf{up}'_i) := ((\mathsf{pk}_i, \mathsf{CRS}), \mathsf{up}_i)$ to $\mathcal{C}$. At some point (suppose that the current epoch is $i^*$), $\mathcal{A}$ receives a challenge query $(r^*_1, r^*_2, \ldots, r^*_\ell)$ for some $\ell \in \mathbb{N}$ from $\mathcal{A}^*$. Then, $\mathcal{A}$ chooses $\mathsf{M}^*_0, \mathsf{M}^*_1 \xleftarrow{\$} \mathcal{M}$ such that $|\mathsf{M}^*_0| = |\mathsf{M}^*_1| \wedge \mathsf{M}^*_0 \neq \mathsf{M}^*_1$ and makes the challenge query on $(\mathsf{M}^*_0, \mathsf{M}^*_1)$. Upon $\mathcal{A}$'s challenge query, $\mathcal{C}$ chooses $\mathsf{coin}^* \leftarrow \{0,1\}$, runs $\mathsf{ct}^* = (\mathsf{ct}, \pi) \leftarrow \mathsf{Enc}'(\mathsf{pk}'_{i^*}, \mathsf{M}^*_{\mathsf{coin}^*})$, and returns $\mathsf{ct}^*$ to $\mathcal{A}$. $\mathcal{A}$ then makes an update query on $(\mathsf{pk}'_{i^*+j}, \mathsf{up}'_{i^*+j})$ for $j = 1, 2, \ldots, \ell$ by executing $\mathsf{UpdPk}(\mathsf{pk}_{i^*+j-1}; r_j) \to (\mathsf{pk}_{i^*+j}, \mathsf{up}_{i^*+j})$ and setting $(\mathsf{pk}'_{i^*+j}, \mathsf{up}'_{i^*+j}) := ((\mathsf{pk}_{i^*+j}, \mathsf{CRS}), \mathsf{up}_{i^*+j})$. Upon the update query, $\mathcal{C}$ runs $\mathsf{UpdSk}(\mathsf{pk}_{i^*+j-1}, \mathsf{up}_{i^*+j-1}, (\mathsf{pk}_{i^*+j}, \mathsf{up}_{i^*+j}))$ and gets $\mathsf{sk}'_{i^*+j}$. Then, due to the non-influential randomness property, it holds that $(\mathsf{pk}_{i^*+\ell}, \mathsf{sk}_{i^*+\ell}) = (\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*})$ with overwhelming probability.

Finally, $\mathcal{A}$ makes a decryption query on the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}, \pi)$. Since the current secret key $\mathsf{sk}_{i^*+\ell}$ is the same as the secret key $\mathsf{sk}_{i^*}$ at the challenge epoch, $\mathcal{A}$ obtains the decryption result of the challenge ciphertext, which clearly breaks the IND-CU-CCA security. □

Note that, in the above attack, we only use the non-influential randomness and correctness properties of the CR/CU-CPA-secure UPKE scheme $\Pi$, and the completeness of the NIZK argument $\Omega$.

### 3.3 Fixing the Transformation

The reason why the attack in the previous section succeeds is that the transformation inherits the property of non-influential randomness that the underlying CPA secure UPKE scheme has. The non-influential randomness allows the adversary to get the decryption result of the challenge ciphertext since the adversary can have the challenger generate the same public and secret keys as those at the challenge epoch. To prevent this attack, they must have essentially different key pairs of public and secret keys in different epochs.

One may come up with the following naive approach: just appending epoch information to public and secret keys, i.e., $(\mathsf{pk}'_i, \mathsf{sk}'_i) := ((\mathsf{pk}_i, i, \mathsf{CRS}), (\mathsf{sk}_i, i))$, instead of $(\mathsf{pk}'_i, \mathsf{sk}'_i) := ((\mathsf{pk}_i, \mathsf{CRS}), \mathsf{sk}_i)$. Unfortunately, this approach does not work since the key pairs are not *essentially* different; although it holds $(\mathsf{pk}'_i, \mathsf{sk}'_i) \neq (\mathsf{pk}'_{i+\ell}, \mathsf{sk}'_{i+\ell})$ for any $i, \ell$, the parts of the secret keys $\mathsf{sk}_i$ and $\mathsf{sk}_{i+\ell}$ could still be equivalent due to the non-influential randomness property, and therefore $\mathsf{sk}_{i+\ell}$ might be able to decrypt the challenge ciphertext encrypted at the epoch $i$.

Based on the above observation, we show two simple modifications of Dodis et al.'s CPA-to-CCA transformation not to inherit the property by embedding epochs in not only public keys but also ciphertexts. One is changing the relation from $R = \{((\mathsf{pk}_i, \mathsf{ct}), (\mathsf{M}, r)) \mid \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}; r)\}$ to $R' = \{((\mathsf{pk}_i, i, \mathsf{ct}, j), (\mathsf{M}, r)) \mid \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}_i, \mathsf{M}; r) \wedge i = j\}$, where $j$ is an epoch $\mathsf{ct}$ was made. The other is also changing the relation but only changing a witness, not a statement. To this end, we make two minor changes as follows. First, we change the public key $\mathsf{pk}'_i = (\mathsf{pk}_i, \mathsf{CRS})$ to $(\mathsf{pk}_i, i, \mathsf{CRS})$ to make it explicit that the public key has an epoch, and the corresponding relation to $R' = \{((\mathsf{pk}_i, i, \mathsf{ct}), (\mathsf{M}, r)) \mid \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}_i, [i\|\mathsf{M}]; r)\}$. Second, accordingly, we also change the $\mathsf{Enc}'$ and $\mathsf{Dec}'$ algorithms as follows:

$\mathsf{Enc}'(\mathsf{pk}'_i, \mathsf{M}) \rightarrow \mathsf{ct}'$: Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, i, \mathsf{CRS})$ and run

- $r \xleftarrow{\$} \mathcal{R}$,
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, [i\|\mathsf{M}]; r)$,
- $\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, (\mathsf{pk}_i, i, \mathsf{ct}), (\mathsf{M}, r))$.

Output $\mathsf{ct}' = (\mathsf{ct}, \pi)$.

$\mathsf{Dec}'(\mathsf{pk}'_i, \mathsf{sk}'_i, \mathsf{ct}') \rightarrow \mathsf{M}$ or $\perp$: Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, i, \mathsf{CRS})$, $\mathsf{sk}'_i = \mathsf{sk}_i$, and $\mathsf{ct}' = (\mathsf{ct}, \pi)$. If $1 \leftarrow \mathsf{Vrfy}(\mathsf{CRS}, (\mathsf{pk}_i, i, \mathsf{ct}), \pi)$, then run $[i\|\mathsf{M}] \leftarrow \mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct})$ and output $\mathsf{M}$. Ohterwise, output $\perp$.

By making the above change, even if the adversary makes a decryption query on a challenge ciphertext $\mathsf{ct}^*$ after the update queries, the challenger can return $\perp$ thanks to the $\mathsf{Vrfy}$ algorithm.

**Theorem 2.** If $\Pi$ is an IND-CU-CPA-secure (resp., IND-CR-CPA-secure) UPKE scheme and $\Omega$ is a strong one-time true-simulation $f$-extractable NIZK argument,[7] the UPKE scheme $\Sigma = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{UpdPk}', \mathsf{UpdSk}')$ constructed above meets IND-CU-CCA security (resp., IND-CR-CCA security).

[7] See the original paper [DKW21] for the detailed definition of such a NIZK argument.

We omit the proof since it is almost the same as in the original transformation [DKW21].

# 4 Efficient Generic Construction of CU-CCA-Secure UPKE

As seen above, fortunately, although Dodis et al.'s CPA-to-CCA transformation is fixable, it still employs NIZK arguments with strong requirements. Their original motivation to use such a strong NIZK argument is to lift CPA security to CCA in the context of UPKE, which is regarded as PKE with circular security and leakage resilience by Dodis et al. Their technique can rule out random oracles; however, it is unclear how the strong NIZK arguments can be efficiently instantiated.

Haidar et al. [HLP22] showed another method to achieving CCA security of UPKE is to apply a specific variant of the Naor–Yung transformation [NY90] to their CPA-secure UPKE scheme from the DCR assumption. They proposed a concrete $\Sigma$-protocol to prove plaintext equality so that it is compatible to their specific CPA-secure UPKE scheme, and applied the Fiat–Shamir heuristic [FS86] to make the $\Sigma$-protocol non-interactive in the ROM. Although this approach preserves the efficiency of the underlying CPA-secure scheme, there are two issues with applicability and concrete efficiency: first, the $\Sigma$-protocol (and the NIZK via the Naor–Yung transformation) supports a specific language, and therefore it cannot be combined with other CPA-secure schemes; second, though the variant of the Naor–Yung transformation does not lose the efficiency of the underlying UPKE scheme that much, the CPA security of the underlying UPKE scheme was proved without random oracles, and hence it (and the resulting CCA-secure scheme) requires relatively larger concrete parameters compared to Jost et al.'s scheme [JMM19]. Haidar et al. [HPS23] also took a similar approach and proposed a CCA-secure UPKE scheme from the LWE assumption with an efficient NIZK argument for a specific language. The above NIZK issues seem to be an unavoidable dilemma; even with random oracles, one has to narrow down the language the NIZK supports to keep efficiency.

Towards concretely efficient CCA-secure UPKE schemes, we take an alternative approach: applying random oracles to convert *any* CPA-secure UPKE scheme to a CCA-secure scheme without using NIZK. To this end, we employ the FO transformation [FO99, FO13] *twice* to lift both CPA and CR to CCA and CU, respectively. Therefore, the one-wayness of CR-CPA security suffices to build our construction, which will be introduced in Sec. 4.1.

## 4.1 Definitions for UPKE

We define the security notion considering one-wayness called the OW-CR-CPA security, as in OW-CPA security in PKE.

**Definition 6** (OW-CR-CPA Security)**.** The OW-CR-CPA security of a UPKE scheme $\Pi$ is defined by a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, which is the same as the IND-CR-CPA security game defined in Def. 1 except for the following change:

**Challenge query:** $\mathcal{A}$ is allowed to make the query only once. Upon $\mathcal{A}$'s query on $\perp$, $\mathcal{C}$ chooses $\mathsf{M}^* \leftarrow \mathcal{M}$ and runs $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_c}, \mathsf{M}^*)$ where $t_c$ is the current epoch. Finally, $\mathcal{C}$ returns $\mathsf{ct}^*$ to $\mathcal{A}$.

**Guess:** At the end of the game, $\mathcal{A}$ returns $\widehat{\mathsf{M}}$ as a guess of $\mathsf{M}^*$.

The relationship between OW-CR-CPA and IND-CR-CPA security can be easily derived below. We omit the proofs, which can be proved similarly to traditional PKE.

**Proposition 1.** If a UPKE scheme $\Pi$ satisfies IND-CR-CPA security, it also satisfies OW-CR-CPA security.

**Proposition 2.** There exists a UPKE scheme $\Pi$ that satisfies OW-CR-CPA security but does not meet IND-CR-CPA security.

We next define $\gamma$-spread for UPKE based on the same property of PKE [FO99, FO13].

**Definition 7** ($\gamma$-spread)**.** A UPKE scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$ is $\gamma$-spread if for all $\lambda, \ell \in \mathbb{N}$ all $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$, all $\mathsf{M} \in \mathcal{M}$, all $i \in \{0, \ldots, \ell\}$, we have

$$-\log\left(\max_{\mathsf{ct}\in\{0,1\}^*} \Pr\left[h \xleftarrow{\$} \mathcal{R} : \mathsf{ct} = \mathsf{Enc}((\mathsf{pk}_i, \mathsf{M}; h)\right]\right) \geq \gamma,$$

where $(\mathsf{pk}_j, \mathsf{up}_j) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{j-1})$, $\mathsf{sk}_j \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{j-1}, sk_{j-1}, (\mathsf{pk}_j, \mathsf{up}_j))$ for $j = 1, \ldots, \ell$.

Our construction requires UPKE schemes to meet $\gamma$-spread, where $\gamma$ is at least $O(\lambda)$. Indeed, all the existing CR-CPA-secure UPKE schemes satisfy this requirement and can be used to instantiate our construction. Even if CR-CPA-secure UPKE schemes with insufficient $\gamma$-spread are proposed in the near future, as in [FO99, FO13], one can strengthen the $\gamma$-spread to the $(\gamma + \gamma')$-spread by appending a random value $r \xleftarrow{\$} \{0,1\}^{\gamma'}$ to the end of ciphertexts.

## 4.2 Our Construction

We combine the two FO transformations and fine-tune them to achieve an efficient CU-CCA-secure UPKE scheme. Specifically, the second FO transformation could be a weaker form since CU security requires the consistency check but no (kind of) challenge queries.

Formally, we construct an IND-CU-CCA-secure UPKE scheme $\Sigma = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{UpdPk}', \mathsf{UpdSk}')$ from an OW-CR-CPA-secure UPKE scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$, an OT-CPA-secure SKE scheme $\Gamma = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$, and four random oracles $\mathsf{G} : \{0,1\}^* \to \mathcal{SK}_{\mathsf{sym}}$, $\mathsf{H} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \mathcal{R}_{\mathsf{asy}}$, $\widehat{\mathsf{G}} : \{0,1\}^* \to \mathcal{SK}_{\mathsf{sym}}$, and $\widehat{\mathsf{H}} : \{0,1\}^* \times \{0,1\}^* \to \mathcal{R}_{\mathsf{asy}}$,[8] where $\mathcal{M}_{\mathsf{asy}}$, $\mathcal{R}_{\mathsf{asy}}$, and $\mathcal{SK}_{\mathsf{sym}}$ are the spaces of plaintexts of $\Pi$, randomness of $\Pi$, and secret keys of $\Gamma$, respectively.

$\mathsf{Gen}'(1^\lambda) \to (\mathsf{pk}'_0, \mathsf{sk}'_0)$ : Run $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$ and output $\mathsf{pk}'_0 := (\mathsf{pk}_0, 0)$ and $\mathsf{sk}'_0 := \mathsf{sk}_0$.

$\mathsf{Enc}'(\mathsf{pk}'_i, \mathsf{M}) \to \mathsf{ct}$ : Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, i)$. Run

- $\sigma \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$,
- $k := \mathsf{G}(\sigma)$,
- $\mathsf{ct}_{\mathsf{sym}} \leftarrow \mathsf{SKE.Enc}(k, \mathsf{M})$,
- $h := \mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}})$,
- $\mathsf{ct}_{\mathsf{asy}} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \sigma; h)$.

Output $\mathsf{ct} := (\mathsf{ct}_{\mathsf{asy}}, \mathsf{ct}_{\mathsf{sym}})$.

$\mathsf{Dec}'(\mathsf{pk}'_i, \mathsf{sk}'_i, \mathsf{ct}) \to \mathsf{M}$ or $\perp$ : Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, i)$, $\mathsf{sk}'_i = \mathsf{sk}_i$, and $\mathsf{ct} = (\mathsf{ct}_{\mathsf{asy}}, \mathsf{ct}_{\mathsf{sym}})$. Run

- $\sigma \leftarrow \mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct}_{\mathsf{asy}})$.

[8] One may merge $\mathsf{G}$ and $\widehat{\mathsf{G}}$ (and also $\mathsf{H}$ and $\widehat{\mathsf{H}}$) into one by using a bit flag, though we use the random oracles separately to make the security proof simple. For instance, one may set $\mathsf{G}(0\|\cdot)$ and $\mathsf{G}(1\|\cdot)$ instead of $\mathsf{G}(\cdot)$ and $\widehat{\mathsf{G}}(\cdot)$, where '$\|$' denotes concatenation.

Output $\perp$ if $\sigma \notin \mathcal{M}_{\mathsf{asy}}$. Otherwise, run

- $h := \mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}})$.

Output $\perp$ if $\mathsf{ct}_{\mathsf{asy}} \neq \mathsf{Enc}(\mathsf{pk}_i, \sigma; h)$. Otherwise, run

- $k := \mathsf{G}(\sigma)$.

Finally, run and output $\mathsf{SKE.Dec}(k, \mathsf{ct}_{\mathsf{sym}})$.

$\mathsf{UpdPk}'(\mathsf{pk}'_i) \to (\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1})$ : Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, i)$. Run

- $r \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$,
- $s := \widehat{\mathsf{G}}(r)$
- $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_i; s)$,
- $h := \widehat{\mathsf{H}}(r, \mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$,
- $\mathsf{ct}_{\mathsf{aux}} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, r; h)$.

Output $\mathsf{pk}'_{i+1} := (\mathsf{pk}_{i+1}, i+1)$ and $\mathsf{up}'_{i+1} := (\mathsf{up}_{i+1}, \mathsf{ct}_{\mathsf{aux}})$.

$\mathsf{UpdSk}'(\mathsf{pk}'_i, \mathsf{sk}'_i, (\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1})) \to \mathsf{sk}'_{i+1}$ or $\perp$ : Parse $\mathsf{pk}'_i = (\mathsf{pk}_i, i)$, $\mathsf{sk}'_i = \mathsf{sk}_i$, $\mathsf{pk}'_{i+1} = (\mathsf{pk}_{i+1}, i+1)$, and $\mathsf{up}'_{i+1} = (\mathsf{up}_{i+1}, \mathsf{ct}_{\mathsf{aux}})$.[9] Run

- $r \leftarrow \mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct}_{\mathsf{aux}})$.

Output $\perp$ if $r \notin \mathcal{M}_{\mathsf{asy}}$ holds. Otherwise, run

- $h := \widehat{\mathsf{H}}(r, \mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$,
- $s := \widehat{\mathsf{G}}(r)$.

Output $\perp$ if at least one of the following holds:

- $\mathsf{ct}_{\mathsf{aux}} \neq \mathsf{Enc}(\mathsf{pk}_i, r; h)$,
- $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \neq \mathsf{UpdPk}(\mathsf{pk}_i; s)$.

Otherwise, run $\mathsf{sk}_{i+1} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}))$ and output $\mathsf{sk}'_{i+1} := \mathsf{sk}_{i+1}$.

### 4.3 Correctness

We prove the correctness of our UPKE construction as follows.

**Theorem 3.** Our UPKE scheme $\Sigma$ satisfies correctness if the underlying UPKE scheme $\Pi$ and SKE scheme $\Gamma$ satisfy correctness.

*Proof.* First, we prove that the $\mathsf{UpdPk}'$ and $\mathsf{UpdSk}'$ algorithms work in the same way as the $\mathsf{UpdPk}$ and $\mathsf{UpdSk}$ algorithms. Suppose that $\lambda \in \mathbb{N}$, $\ell \in \mathbb{N}$, $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$, $r \in \mathcal{M}_{\mathsf{asy}}$, $i \in \{0, \ldots, \ell-1\}$ are arbitrarily fixed. The $\mathsf{UpdPk}'$ algorithm computes

- $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_i; \widehat{\mathsf{G}}(r))$,
- $h := \widehat{\mathsf{H}}(r, \mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$,
- $\mathsf{ct}_{\mathsf{aux}} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, r; h)$,

[9] To be precise, one has to confirm whether the second element of $\mathsf{pk}'_{i+1} = (\mathsf{pk}_{i+1}, j)$ denotes the next epoch of $i$, i.e., $j = i + 1$; we omit the procedure since one can easily check it by just incrementing the second element of $\mathsf{pk}'_i = (\mathsf{pk}_i, i)$ and comparing it with $j$.

and outputs $(\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1}) = ((\mathsf{pk}_{i+1}, i{+}1), (\mathsf{up}_{i+1}, \mathsf{ct}_{\mathsf{aux}}))$. On the other hand, the $\mathsf{UpdSk}'$ algorithm computes

- $r' \leftarrow \mathsf{Dec}(\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}, \mathsf{ct}_{\mathsf{aux}})$,
- $h' := \widehat{\mathsf{H}}(r', \mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$,
- $s' := \widehat{\mathsf{G}}(r')$,

and check whether it hold (a) $\mathsf{ct}_{\mathsf{aux}} = \mathsf{Enc}(\mathsf{pk}_i, r'; h')$ and (b) $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) = \mathsf{UpdPk}(\mathsf{pk}_i; s')$. If so, $\mathsf{UpdSk}'$ outputs $\mathsf{sk}_{i+1} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_i, \mathsf{sk}_i, (\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}))$; Otherwise, it outputs $\perp$.

First, $r = r'$ holds with overwhelming probability due to the correctness of the UPKE scheme $\Pi$. It then holds (b) $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) = \mathsf{UpdPk}(\mathsf{pk}_i; s')$ since $s' = \widehat{\mathsf{G}}(r') = \widehat{\mathsf{G}}(r) = s$. Finally, it holds (a) $\mathsf{ct}_{\mathsf{aux}} = \mathsf{Enc}(\mathsf{pk}_i, r'; h')$ since it hold $r' = r$ and $h' = \widehat{\mathsf{H}}(r', \mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) = \widehat{\mathsf{H}}(r, \mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) = h$. Therefore, the $\mathsf{UpdPk}'$ and $\mathsf{UpdSk}'$ algorithms work the same as in the $\mathsf{UpdPk}$ and $\mathsf{UpdSk}$ algorithms with overwhelming probability.

Next, we prove that the $\mathsf{Dec}'$ and $\mathsf{Enc}'$ algorithms satisfy the correctness. Suppose that $\lambda \in \mathbb{N}$, $\ell \in \mathbb{N}$, $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$, $\mathsf{M} \in \mathcal{M}_{\mathsf{sym}}$, and $i \in \{0, \ldots, \ell\}$ are arbitrarily fixed. The $\mathsf{Enc}'$ algorithm computes

- $\sigma \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$,
- $\mathsf{ct}_{\mathsf{sym}} \leftarrow \mathsf{SKE.Enc}(\mathsf{G}(\sigma), \mathsf{M})$,
- $h := \mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}})$,
- $\mathsf{ct}_{\mathsf{asy}} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \sigma; h)$,

and outputs $\mathsf{ct} := (\mathsf{ct}_{\mathsf{asy}}, \mathsf{ct}_{\mathsf{sym}})$. On the other hand, the $\mathsf{Dec}'$ algorithm computes

- $\sigma' \leftarrow \mathsf{Dec}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct}_{\mathsf{asy}})$,
- $h' := \mathsf{H}(i, \sigma', \mathsf{ct}_{\mathsf{sym}})$,
- $\mathsf{M}' := \mathsf{SKE.Dec}(\mathsf{G}(\sigma'), \mathsf{ct}_{\mathsf{sym}})$,

and outputs $\mathsf{M}'$ if it holds (c) $\mathsf{ct}_{\mathsf{asy}} = \mathsf{Enc}(\mathsf{pk}_i, \sigma'; h')$; it outputs $\perp$ otherwise.

First, $\sigma' = \sigma$ holds with overwhelming probability due to the correctness of the UPKE scheme $\Pi$. Then, we have $\mathsf{M}' = \mathsf{M}$ from the fact that $\mathsf{G}(\sigma') = \mathsf{G}(\sigma)$ and the correctness of the SKE scheme $\Gamma$. Since $h' = \mathsf{H}(i, \sigma', \mathsf{ct}_{\mathsf{sym}}) = \mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}}) = h$ and $\sigma' = \sigma$, it holds (c) $\mathsf{ct}_{\mathsf{asy}} = \mathsf{Enc}(\mathsf{pk}_i, \sigma; h) = \mathsf{Enc}(\mathsf{pk}_i, \sigma'; h')$.

It completes the proof. □ □

## 4.4 Security

**Theorem 4.** If the underlying UPKE scheme $\Pi$ is $\gamma$-spread and satisfies OW-CR-CPA security, SKE scheme $\Gamma$ satisfies OT-CPA security, and $\mathsf{G}$, $\mathsf{H}$, $\widehat{\mathsf{G}}$, and $\widehat{\mathsf{H}}$ are random oracles, then our proposed UPKE scheme $\Sigma$ satisfies IND-CU-CCA security.

*Proof.* Let $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{asy}}, \mathsf{ct}^*_{\mathsf{sym}})$ be the challenge ciphertext, and $(\mathsf{pk}^*_{t_r}, \mathsf{sk}^*_{t_r}, \mathsf{up}^*_{t_r}) = ((\mathsf{pk}_{t_r}, t_r), \mathsf{sk}_{t_r}, (\mathsf{up}_{t_r}, \mathsf{ct}^*_{\mathsf{aux}}))$ be the response of the reveal query (at epoch $t_r$), i.e., the tuple of the public key, the secret key, and the update ciphertext revealed at the reveal query. We arbitrarily fix a PPT adversary $\mathcal{A}$ and consider a game sequence $\mathbf{Game}_0, \ldots, \mathbf{Game}_4$. Let $W_i$ denote an event that $\mathcal{A}$ wins in $\mathbf{Game}_i$ for $i \in \{0, 1, \ldots, 4\}$.

$\mathbf{Game}_0$: This game is the same as the original IND-CU-CCA security game in Def. 2 between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

**Game**$_1$: This game is the same as **Game**$_0$ except that $\mathcal{C}$ handles update queries without the secret keys as follows: $\mathcal{C}$ simulates $\widehat{\mathsf{H}}$ and records the queries and the answers in $\widehat{\mathcal{H}}$, which is an empty list initially. Upon $\mathcal{A}$'s update query on $(\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1}) = ((\mathsf{pk}_{i+1}, i+1), (\mathsf{up}_{i+1}, \mathsf{ct}_{\mathsf{aux}}))$, $\mathcal{C}$ searches for a tuple $(r, \mathsf{pk}_{i+1}, \mathsf{up}_{i+1}, h)$ in $\widehat{\mathcal{H}}$ such that it holds

$$r \in \mathcal{M}_{\mathsf{asy}} \ \wedge \ \mathsf{ct}_{\mathsf{aux}} = \mathsf{Enc}(\mathsf{pk}_i, r; h). \tag{1}$$

We say that the update query $(\mathsf{pk}'_{i+1}, \mathsf{up}'_{i+1})$ is valid if Eq. (1) holds. If $\mathcal{C}$ finds such a tuple and it holds $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) = \mathsf{UpdPk}(\mathsf{pk}_i; \widehat{\mathsf{G}}(r))$, $\mathcal{C}$ accepts the update and increments the current epoch $i$ to $i+1$. Otherwise, i.e., there is no tuple satsfying Eq. (1) in $\widehat{\mathcal{H}}$ or it holds $(\mathsf{pk}_{i+1}, \mathsf{up}_{i+1}) \neq \mathsf{UpdPk}(\mathsf{pk}_i; \widehat{\mathsf{G}}(r))$, $\mathcal{C}$ rejects the update and returns $\perp$ to $\mathcal{A}$. Note that $\mathcal{C}$ can perform the above without the secret key $\mathsf{sk}_i$.

We show that **Game**$_0$ and **Game**$_1$ are computationally indistinguishable from $\mathcal{A}$'s view if $\Pi$ is $\gamma$-spread. Briefly speaking, the difference between **Game**$_0$ and **Game**$_1$ is that $\mathcal{C}$ in **Game**$_1$ outputs $\perp$ if $\mathcal{A}$'s query is *not* accepted, while $\mathcal{C}$ in **Game**$_0$ might not. Let $\mathsf{Bad}_1$ be an event where $\mathcal{A}$ issues a *valid* update query *without* querying $\hat{\mathsf{H}}$. **Game**$_0$ and **Game**$_1$ proceed identically unless $\mathsf{Bad}_1$ occurs. From Lemma 1, we have

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[\mathsf{Bad}_1].$$

We then estimate the probability that $\mathsf{Bad}_1$ occurs by focusing on $\widehat{\mathsf{H}}$, which is used only for update queries and a reveal query. Since $\widehat{\mathsf{H}}$ is a random oracle, $\widehat{\mathsf{H}}(r^*, \mathsf{pk}_{t_r}, \mathsf{up}_{t_r})$ is chosen uniformly at random, where $r^* \in \mathcal{M}_{\mathsf{asy}}$ is randomly chosen for the reveal query, and independent of $\widehat{\mathsf{H}}(r_i, \mathsf{pk}_{i+1}, \mathsf{up}_{i+1})$ for every $i \in \{0, \ldots, t_r - 1\}$. Therefore, $\mathcal{A}$ cannot obtain any information on $\widehat{\mathsf{H}}(r^*, \mathsf{pk}_{t_r}, \mathsf{up}_{t_r})$. Due to the $\gamma$-spread property of $\Pi$, $\mathcal{A}$ can make an update query which $\mathsf{Bad}_1$ occurs with probability at most $2^{-\gamma}$. Then, from the union bound, we have

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[\mathsf{Bad}_1] \leq 2^{-\gamma} q_{\mathsf{Upd}}, \tag{2}$$

where $q_{\mathsf{Upd}}$ is (the upper bound of) the number of update queries issued by $\mathcal{A}$.

**Game**$_2$: This game is the same as **Game**$_1$ except that $\mathcal{C}$ answers to decryption queries without the secret keys as follows: $\mathcal{C}$ simulates $\mathsf{H}$ and records the queries and the answers in $\mathcal{H}$, which is an empty list initially. Upon $\mathcal{A}$'s query on $\mathsf{ct} = (\mathsf{ct}_{\mathsf{asy}}, \mathsf{ct}_{\mathsf{sym}})$, $\mathcal{C}$ searches for a tuple $(i, \sigma, \mathsf{ct}_{\mathsf{sym}}, h)$ in $\widehat{\mathsf{H}}$ such that it holds

$$\sigma \in \mathcal{M}_{\mathsf{asy}} \ \wedge \ \mathsf{ct}_{\mathsf{asy}} = \mathsf{Enc}(\mathsf{pk}_i, \sigma; h). \tag{3}$$

We say that the decryption query $\mathsf{ct}$ is valid if Eq. (3) holds. If $\mathcal{C}$ finds such a tuple, $\mathcal{C}$ runs $\mathsf{M} \leftarrow \mathsf{SKE.Dec}(\mathsf{G}(\sigma), \mathsf{ct}_{\mathsf{sym}})$ and returns $\mathsf{M}$. Otherwise, i.e., there is no tuple satisfying Eq. (3) in $\mathcal{H}$, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Note that $\mathcal{C}$ can perform the above without the secret key $\mathsf{sk}_i$.

We show that **Game**$_1$ and **Game**$_2$ are computationally indistinguishable from $\mathcal{A}$'s view if $\Pi$ is $\gamma$-spread. Briefly speaking, the difference between **Game**$_1$ and **Game**$_2$ is that $\mathcal{C}$ in **Game**$_2$ outputs $\perp$ if $\mathcal{A}$'s query is *not* accepted, while $\mathcal{C}$ in **Game**$_1$ might not. Let $\mathsf{Bad}_2$ be an event where $\mathcal{A}$ issues a *valid* decryption query *without* querying $\mathsf{H}$. **Game**$_1$ and **Game**$_2$ proceed identically unless $\mathsf{Bad}_2$ occurs. From Lemma 1, we have

$$|\Pr[W_1] - \Pr[W_2]| \leq \Pr[\mathsf{Bad}_2].$$

We then estimate the probability that $\mathsf{Bad}_2$ occurs by focusing on $\mathsf{H}$, which is used only for decryption queries and a challange query. In the epoch $i$, $\mathcal{A}$'s decryption query on $\mathsf{ct} = (\mathsf{ct}_{\mathsf{sym}}, \mathsf{ct}_{\mathsf{asy}})$ should satisfy $(i, \mathsf{ct}_{\mathsf{asy}}, \mathsf{ct}_{\mathsf{sym}}) \neq (t_c, \mathsf{ct}^*_{\mathsf{asy}}, \mathsf{ct}^*_{\mathsf{sym}})$. This condition can be written as $(i, \sigma, \mathsf{ct}_{\mathsf{sym}}) \neq$

$(t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}})$ where $\sigma$ is the plaintext of $\mathsf{ct}_{\mathsf{asy}}$ and $\sigma^*$ is the plaintext of $\mathsf{ct}^*_{\mathsf{asy}}$. If $(i, \mathsf{ct}_{\mathsf{sym}}) = (t_c, \mathsf{ct}^*_{\mathsf{sym}})$ holds when $\mathsf{ct}_{\mathsf{asy}} = \mathsf{ct}^*_{\mathsf{asy}}$, then this implies that $\sigma = \sigma^*$. If $(i, \sigma, \mathsf{ct}_{\mathsf{sym}}) = (t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}})$ holds when $\mathsf{ct}_{\mathsf{asy}} \neq \mathsf{ct}^*_{\mathsf{asy}}$, from $\mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}}) = \mathsf{H}(t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}})$, $\mathsf{ct}_{\mathsf{asy}} = \mathsf{ct}^*_{\mathsf{asy}} = \mathsf{Enc}(\mathsf{pk}_i, \sigma; \mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}})) = \mathsf{Enc}(\mathsf{pk}_{t^*}, \sigma^*, \mathsf{H}(t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}}))$ holds. This is a contradiction. Since $\mathsf{H}$ is a random oracle, $\mathsf{H}(t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}})$ is chosen uniformly at random, where $\sigma^* \in \mathcal{M}_{\mathsf{asy}}$ is randomly chosen for the challenge query and independent of $\mathsf{H}(i, \sigma, \mathsf{ct}_{\mathsf{sym}})$. Therefore, $\mathcal{A}$ cannot obtain any information on $\mathsf{H}(t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}})$. Due to the $\gamma$-spread property of $\Pi$, $\mathcal{A}$ can make an update query which $\mathsf{Bad}_2$ occurs with probability at most $2^{-\gamma}$. Then, from the union bound, we have

$$|\Pr[W_1] - \Pr[W_2]| \leq \Pr[\mathsf{Bad}_2] \leq 2^{-\gamma} q_{\mathsf{Dec}}, \tag{4}$$

where $q_{\mathsf{Dec}}$ is (the upper bound of) the number of decryption queries issued by $\mathcal{A}$.

**Game**$_3$: This game is the same as **Game**$_2$ except that $\mathcal{C}$ generates the update ciphertext at the reveal query without $\widehat{\mathsf{G}}$ and $\widehat{\mathsf{H}}$ as follows:

1. $r^* \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$,
2. $s^* \xleftarrow{\$} \mathcal{R}_{\mathsf{asy}}$ (while using $\widehat{\mathsf{G}}$ to compute $s^* := \widehat{\mathsf{G}}(r^*)$ in **Game**$_2$),
3. $(\mathsf{pk}_{t_r}, \mathsf{up}_{t_r}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{t_r-1}; s^*)$,
4. $h^* \xleftarrow{\$} \mathcal{R}_{\mathsf{asy}}$ (while using $\widehat{\mathsf{H}}$ to compute $h^* := \widehat{\mathsf{H}}(r^*, \mathsf{pk}_{t_r}, \mathsf{up}_{t_r})$ in **Game**$_2$),
5. $\mathsf{ct}^*_{\mathsf{aux}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_r-1}, r^*; h^*)$.

Let $\mathsf{Ask}_1$ be an event that $\mathcal{A}$ queries $r^*$ or $(r^*, \cdot, \cdot)$ to $\widehat{\mathsf{G}}$ or $\widehat{\mathsf{H}}$, respectively. Obviously, **Game**$_2$ and **Game**$_3$ proceed identically unless $\mathsf{Ask}_1$ occurs. Therefore, from Lemma 1, we have

$$|\Pr[W_2] - \Pr[W_3]| \leq \Pr[\mathsf{Ask}_1]. \tag{5}$$

We show that **Game**$_2$ and **Game**$_3$ are computationally indistinguishable from $\mathcal{A}$'s view if $\Pi$ is OW-CR-CPA secure. For this purpose, we use $\mathcal{A}$ to construct a PPT adversary $\mathcal{B}_1$ that breaks OW-CR-CPA security of $\Pi$. Let $\mathcal{C}_1$ denote a challenger of the OW-CR-CPA security game of $\Pi$. $\mathcal{C}_1$ begins the OW-CR-CPA security game and gives $\mathsf{pk}_0$ to $\mathcal{B}$. Then, $\mathcal{B}_1$ begins the IND-CU-CCA security game and gives $\mathsf{pk}'_0 = (\mathsf{pk}_0, 0)$ to $\mathcal{A}$. In Phase 1, $\mathcal{B}_1$ can answer decryption queries and update queries without the secret keys thanks to the changes in **Game**$_1$ and **Game**$_2$. Note that to answer each update query on $(\mathsf{pk}'_i, \mathsf{up}'_i)$, $\mathcal{B}_1$ retrieves the tuple $(r, \cdot, \cdot, \cdot)$ satisfying Eq. (1) from $\widehat{\mathcal{H}}$ and makes an update query on $\widehat{\mathsf{G}}(r)$ to $\mathcal{C}_1$. This is important to synchronize epochs and corresponding public and secret keys in both OW-CR-CPA and IND-CU-CCA games.

Upon $\mathcal{A}$'s challenge query on $(\mathsf{M}^*_0, \mathsf{M}^*_1)$, $\mathcal{B}_1$ chooses $\mathsf{coin}^* \xleftarrow{\$} \{0,1\}$, runs $\mathsf{ct}^* \leftarrow \mathsf{Enc}'(\mathsf{pk}_{t_c}, \mathsf{M}^*_{\mathsf{coin}^*})$, and gives $\mathsf{ct}^*$ to $\mathcal{A}$. $\mathcal{B}_1$ can simulate Phase 2 in the same way as Phase 1.

Upon $\mathcal{A}$'s reveal query, $\mathcal{B}_1$ makes the challenge query to $\mathcal{C}_1$. Then, $\mathcal{C}_1$ chooses $r^* \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$, runs $\mathsf{ct}^*_{\mathsf{aux}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_r-1}, r^*)$, and gives it to $\mathcal{B}_1$. After that, $\mathcal{B}_1$ makes the reveal query to $\mathcal{C}_1$. $\mathcal{C}_1$ samples $s^* \leftarrow \mathcal{R}_{\mathsf{asy}}$, runs $(\mathsf{pk}_{t_r}, \mathsf{up}_{t_r}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{t_r-1}; s^*)$ and $\mathsf{sk}_{t_r} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{t_r-1}, \mathsf{sk}_{t_r-1}, (\mathsf{pk}_{t_r}, \mathsf{up}_{t_r}))$, and returns $(\mathsf{pk}_{t_r}, \mathsf{sk}_{t_r}, \mathsf{up}_{t_r})$ to $\mathcal{B}_1$. Then, $\mathcal{B}_1$ returns $(\mathsf{pk}^*_{t_r}, \mathsf{sk}^*_{t_r}, \mathsf{up}^*_{t_r}) = ((\mathsf{pk}_{t_r}, t_r), \mathsf{sk}_{t_r}, (\mathsf{up}_{t_r}, \mathsf{ct}^*_{\mathsf{aux}}))$ to $\mathcal{A}$. After $\mathcal{A}$ outputs $\widehat{\mathsf{coin}}$ as a guess of $\mathsf{coin}$, $\mathcal{B}_1$ randomly chooses $j \xleftarrow{\$} \{1, \ldots, q_{\widehat{\mathsf{Hash}}}\}$ where $q_{\widehat{\mathsf{Hash}}}$ is the number of queries $\mathcal{A}$ makes to $\widehat{\mathsf{G}}$ and $\widehat{\mathsf{H}}$. $\mathcal{B}_1$ then retrieves $\widehat{r}$ from the $j$-th query to the random oracles $\widehat{\mathsf{G}}$ and $\widehat{\mathsf{H}}$, where $\widehat{r}$ is stored in $\widehat{\mathsf{G}}$ in the form of $\widehat{r}$ or in $\widehat{\mathsf{H}}$ in the form of $(\widehat{r}, \cdot, \cdot)$, and outputs it as a guess of $r^*$.

Since we consider the situation that $\mathsf{Ask}_1$ occurs, i.e., $\mathcal{A}$ queries $r^*$ to $\widehat{\mathsf{G}}$ or $\widehat{\mathsf{H}}$, the probability that $\mathcal{B}_1$ outputs $r^*$ is $q^{-1}_{\widehat{\mathsf{Hash}}}$. Therefore, we have

$$|\Pr[W_2] - \Pr[W_3]| \leq \Pr[\mathsf{Ask}_1] \leq q_{\widehat{\mathsf{Hash}}} \mathsf{Adv}^{\text{OW-CR-CPA}}_{\Pi, \mathcal{B}_1}(\lambda). \tag{6}$$

**Game**$_4$: This game is the same as **Game**$_3$ except that $\mathcal{C}$ generates the challenge ciphertext without $\mathsf{G}$ and $\mathsf{H}$ as follows:

1. $\sigma^* \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$,
2. $k^* \xleftarrow{\$} \mathcal{SK}_{\mathsf{sym}}$ (while using $\mathsf{G}$ to compute $k^* := \mathsf{G}(\sigma^*)$ in **Game**$_3$),
3. $\mathsf{ct}^*_{\mathsf{sym}} \leftarrow \mathsf{SKE.Enc}(k^*, \mathsf{M}^*)$,
4. $h^* \xleftarrow{\$} \mathcal{R}_{\mathsf{asy}}$ (while using $\mathsf{H}$ to compute $h^* := \mathsf{H}(t_c, \sigma^*, \mathsf{ct}^*_{\mathsf{sym}})$ in **Game**$_3$),
5. $\mathsf{ct}^*_{\mathsf{asy}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_c}, \sigma^*; h^*)$.

Let $\mathsf{Ask}_2$ be an event that $\mathcal{A}$ queries $\sigma^*$ or $(\cdot, \sigma^*, \cdot)$ to $\mathsf{G}$ or $\mathsf{H}$, respectively. Obviously, **Game**$_3$ and **Game**$_4$ proceed identically unless $\mathsf{Ask}_2$ occurs. From the same discussion in Eq. (5), we have

$$|\Pr[W_3] - \Pr[W_4]| \leq \Pr[\mathsf{Ask}_2].$$

We show that **Game**$_3$ and **Game**$_4$ are computationally indistinguishable from $\mathcal{A}$'s view if $\Pi$ is OW-CR-CPA secure. For this purpose, we use $\mathcal{A}$ to construct a PPT adversary $\mathcal{B}_2$ that breaks OW-CR-CPA security of $\Pi$. Let $\mathcal{C}_2$ denote a challenger of the OW-CR-CPA security game of $\Pi$. $\mathcal{C}_2$ begins the OW-CR-CPA security game and gives $\mathsf{pk}_0$ to $\mathcal{B}_2$. Then, $\mathcal{B}_2$ begins the IND-CU-CCA security game and gives $\mathsf{pk}'_0 = (\mathsf{pk}_0, 0)$ to $\mathcal{A}$. From the changes in **Game**$_1$ and **Game**$_2$, $\mathcal{B}_2$ can answer decryption and update queries without the secret keys in Phase 1 in the same way as $\mathcal{B}_1$.

Upon $\mathcal{A}$'s challange query on $(\mathsf{M}^*_0, \mathsf{M}^*_1)$, $\mathcal{B}_2$ chooses $\mathsf{coin}^* \xleftarrow{\$} \{0,1\}$, $k^* \xleftarrow{\$} \mathcal{SK}_{\mathsf{sym}}$, and runs $\mathsf{ct}^*_{\mathsf{sym}} \leftarrow \mathsf{SKE.Enc}(k^*, \mathsf{M}^*_{\mathsf{coin}^*})$. Then, $\mathcal{B}_2$ makes a challenge query on $\perp$ to $\mathcal{C}_2$. Upon $\mathcal{B}_2$'s challenge query, $\mathcal{C}_2$ chooses $\sigma^* \leftarrow \mathcal{M}_{\mathsf{asy}}$, runs $\mathsf{ct}^*_{\mathsf{asy}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_c}, \sigma^*)$, and returns $\mathsf{ct}^*_{\mathsf{asy}}$ to $\mathcal{B}_2$. After receiving the ciphertext from $\mathcal{C}_2$, $\mathcal{B}_2$ returns $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{asy}}, \mathsf{ct}^*_{\mathsf{sym}})$ to $\mathcal{A}$.

Upon $\mathcal{A}$'s reveal query, $\mathcal{B}_2$ makes a reveal query and receives $(\mathsf{pk}_{t_r}, \mathsf{sk}_{t_r}, \mathsf{up}_{t_r})$. Then $\mathcal{B}_2$ generates $\mathsf{ct}^*_{\mathsf{aux}}$, which is another element in $\mathsf{up}^*$, as in the changes in **Game**$_3$, and returns $(\mathsf{pk}^*_{t_r}, \mathsf{sk}^*_{t_r}, \mathsf{up}^*_{t_r}) = ((\mathsf{pk}_{t_r}, t_r), \mathsf{sk}_{t_r}, (\mathsf{up}_{t_r}, \mathsf{ct}^*_{\mathsf{aux}}))$ to $\mathcal{A}$. After $\mathcal{A}$ outputs $\widehat{\mathsf{coin}}$ as a guess of $\mathsf{coin}^*$, $\mathcal{B}_2$ randomly chooses $j \xleftarrow{\$} \{1, \ldots, q_{\mathsf{Hash}}\}$ where $q_{\mathsf{Hash}}$ is the number of queries $\mathcal{A}$ makes to $\mathsf{G}$ and $\mathsf{H}$. $\mathcal{B}_2$ then retrieves $\widehat{\sigma}$ from the $j$-th query to the random oracles $\mathsf{G}$ and $\mathsf{H}$, where $\widehat{\sigma}$ is stored in $\mathsf{G}$ in the form of $\widehat{\sigma}$ or in $\mathsf{H}$ in the form of $(\cdot, \widehat{\sigma}, \cdot)$, and outputs it as a guess of $\sigma^*$.

Since we consider the situation that $\mathsf{Ask}_2$ occurs, i.e., $\mathcal{A}$ queries $\sigma^*$ to $\mathsf{G}$ or $\mathsf{H}$, the probability that $\mathcal{B}_2$ outputs $\sigma^*$ is $q^{-1}_{\mathsf{Hash}}$. Therefore, we have

$$|\Pr[W_3] - \Pr[W_4]| \leq \Pr[\mathsf{Ask}_2] \leq q_{\mathsf{Hash}} \mathsf{Adv}^{\text{OW-CR-CPA}}_{\Pi, \mathcal{B}_2}(\lambda). \tag{7}$$

Finally, we show that it is computationally infeasible for $\mathcal{A}$ to win in **Game**$_4$ if $\Gamma$ is OT-CPA secure. To this end, we use $\mathcal{A}$ to construct $\mathcal{B}_3$ that breaks OT-CPA security of $\Gamma$. Let $\mathcal{C}_3$ denote a challenger of the OT-CPA security game of $\Gamma$. $\mathcal{B}_3$ begins the IND-CU-CCA security game and gives $\mathsf{pk}'_0$ to $\mathcal{A}$. In Phase 1, $\mathcal{B}_3$ can answer decryption and update queries in the same way as $\mathcal{B}_1$. Upon $\mathcal{A}$'s challenge query on $(\mathsf{M}^*_0, \mathsf{M}^*_1)$, $\mathcal{B}_3$ generates the challenge ciphertext as follows: $\mathcal{B}_2$ makes the challange query on $(\mathsf{M}^*_0, \mathsf{M}^*_1)$ to $\mathcal{C}_3$. $\mathcal{C}_3$ chooses $\mathsf{coin}^* \xleftarrow{\$} \{0,1\}$, $k^* \xleftarrow{\$} \mathcal{SK}_{\mathsf{sym}}$, runs $\mathsf{ct}^*_{\mathsf{sym}} \leftarrow \mathsf{SKE.Enc}(k^*, \mathsf{M}^*_{\mathsf{coin}^*})$, and gives $\mathsf{ct}^*_{\mathsf{sym}}$ to $\mathcal{B}_3$. After that, $\mathcal{B}_3$ chooses $\sigma^* \xleftarrow{\$} \mathcal{M}_{\mathsf{asy}}$, $h^* \xleftarrow{\$} \mathcal{R}_{\mathsf{asy}}$ and runs $\mathsf{ct}^*_{\mathsf{asy}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{t_c}, \sigma^*; h^*)$. Then, $\mathcal{B}_3$ returns $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{asy}}, \mathsf{ct}^*_{\mathsf{sym}})$ to $\mathcal{A}$. In Phase 2, $\mathcal{B}_2$ can answer queries in the same way as in Phase 1.

Upon $\mathcal{A}$'s reveal query, $\mathcal{B}_3$ generates the update ciphertext $\mathsf{ct}^*_{\mathsf{aux}}$ as in the changes in **Game**$_3$. Then, $\mathcal{B}_3$ samples $s^* \xleftarrow{\$} \mathcal{R}_{\mathsf{asy}}$, runs $(\mathsf{pk}_{t_r}, \mathsf{up}_{t_r}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_{t_r-1}; s^*)$ and $\mathsf{sk}_{t_r} \leftarrow \mathsf{UpdSk}(\mathsf{pk}_{t_r-1}, \mathsf{sk}_{t_r-1}, (\mathsf{pk}_{t_r}, \mathsf{up}_{t_r}))$, and gives $(\mathsf{pk}^*_{t_r}, \mathsf{sk}^*_{t_r}, \mathsf{up}^*_{t_r}) = ((\mathsf{pk}_{t_r},$

$t_r), \mathsf{sk}_{t_r}, (\mathsf{up}_{t_r}, \mathsf{ct}^*_{\mathsf{aux}}))$ to $\mathcal{A}$. Since $\mathcal{B}_3$ has the initial secret key, $\mathcal{B}_3$ can run the $\mathsf{UpdSk}$ algorithm. After $\mathcal{A}$ outputs $\widehat{\mathsf{coin}}$ as a guess of the coin in the IND-CU-CCA security game, $\mathcal{B}_3$ outputs the same $\widehat{\mathsf{coin}}$ as a guess of $\mathsf{coin}^*$ in the OT-CPA security game. Therefore, we have

$$\left|\Pr[W_4] - \frac{1}{2}\right| = \mathsf{Adv}^{\text{OT-CPA}}_{\Gamma,\mathcal{B}_3}(\lambda). \tag{8}$$

From Eqs. (2), (4), (6), (7), and (8), we have

$$\begin{aligned}\mathsf{Adv}^{\text{IND-CU-CCA}}_{\Sigma,\mathcal{A}}(\lambda) \leq 2^{-\gamma}(q_{\mathsf{Upd}} + q_{\mathsf{Dec}}) + q_{\widehat{\mathsf{Hash}}}\mathsf{Adv}^{\text{OW-CR-CPA}}_{\Pi,\mathcal{B}_1}(\lambda)\\ + q_{\mathsf{Hash}}\mathsf{Adv}^{\text{OW-CR-CPA}}_{\Pi,\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\text{OT-CPA}}_{\Gamma,\mathcal{B}_3}(\lambda).\end{aligned}$$

We conclude the proof. □ □

# 5 Conclusion

In this paper, we demonstrated that Dodis et al.'s CPA-to-CCA transformation [DKW21] does not satisfy CCA security if the underlying UPKE schemes have non-influential randomness, a special property of UPKE that we introduced in this paper. The property enables the adversary to have the challenger generate the same public and secret keys as those at the challenge epoch. Then, the adversary can get the decryption result of the challenge ciphertext by making the decryption query. To prevent the above attack, we modified their CPA-to-CCA transformation by embedding epochs in public keys and cipheretexts.

We also proposed a new generic construction of the IND-CU-CCA secure UPKE scheme from any OW-CR-CPA-secure UPKE scheme in the ROM. By employing the FO transformation [FO99, FO13] twice instead of using inefficient NIZK arguments, we can obtain the most efficient IND-CR-CCA/IND-CU-CCA secure UPKE schemes from the most efficient IND-CR-CPA-secure UPKE scheme by Jost et al. [JMM19].

Although our CU-CCA-secure scheme is quite efficient, it does not support public verifiability. It would be interesting to realize a transformation that supports public verifiability with almost the same efficiency as ours since it is an important property in the CU setting.

**Acknowledgments**. The authors would like to thank Shuichi Katsumata, Keita Xagawa, Calvin Abou Haidar, Yevgeniy Dodis, and Harish Karthikeyan for their valuable comments. The authors also thank Paul Rösler for spotting typos in the UPKE security definitions. This work was supported by JSPS KAKENHI Grant Numbers JP23H00479, JP23KJ0968, JP21H03341, 21H03395, 18H05289, and MEXT Leading Initiative for Excellent Young Researchers.

# References

[ACD+20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. "Security Analysis and Improvements for the IETF MLS Standard for Group Messaging." In: *CRYPTO*. 2020, pp. 248–277.

[BB04] Dan Boneh and Xavier Boyen. "Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles." In: *EUROCRYPT*. 2004, pp. 223–238.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. "Hierarchical Identity Based Encryption with Constant Size Ciphertext." In: *EUROCRYPT*. 2005, pp. 440–456.

[BY03] Mihir Bellare and Bennet S. Yee. "Forward-Security in Private-Key Cryptography." In: *CT-RSA*. 2003, pp. 1–18.

[CHK+12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. "Bonsai Trees, or How to Delegate a Lattice Basis." In: *J. Cryptol.* (2012), pp. 601–639.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. "A Forward-Secure Public-Key Encryption Scheme." In: *EUROCRYPT*. 2003, pp. 255–271.

[DFK+03] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. "Intrusion-Resilient Public-Key Encryption." In: *CT-RSA*. 2003, pp. 19–32.

[DFK+04] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. "A Generic Construction for Intrusion-Resilient Public-Key Encryption." In: *CT-RSA*. 2004, pp. 81–98.

[DG17] Nico Döttling and Sanjam Garg. "Identity-Based Encryption from the Diffie-Hellman Assumption." In: *CRYPTO*. 2017, pp. 537–569.

[DKW21] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. "Updatable Public Key Encryption in the Standard Model." In: *TCC*. 2021, pp. 254–285.

[DKW22] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. "Updatable Public Key Encryption in the Standard Model." In: *IACR Cryptol. ePrint Arch.* (2022), p. 68. URL: https://eprint.iacr.org/2022/068.

[DKX+02] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. "Key-Insulated Public Key Cryptosystems." In: *EUROCRYPT*. 2002, pp. 65–82.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes." In: *J. Cryptol.* (2013), pp. 80–101.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes." In: *CRYPTO*. 1999, pp. 537–554.

[FS86] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems." In: *CRYPTO*. 1986, pp. 186–194.

[GS02] Craig Gentry and Alice Silverberg. "Hierarchical ID-Based Cryptography." In: *ASIACRYPT*. 2002, pp. 548–566.

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups." In: *EUROCRYPT*. 2008, pp. 415–432.

[HL02] Jeremy Horwitz and Ben Lynn. "Toward Hierarchical Identity-Based Encryption." In: *EUROCRYPT*. 2002, pp. 466–481.

[HLP22] Calvin Abou Haidar, Benoît Libert, and Alain Passelègue. "Updatable Public Key Encryption from DCR: Efficient Constructions With Stronger Security." In: *CCS 2022*. 2022, pp. 11–22.

[HPS23] Calvin Abou Haidar, Alain Passelègue, and Damien Stehlé. "Efficient Updatable Public-Key Encryption from Lattices." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1400. URL: https://eprint.iacr.org/2023/1400.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. "Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging." In: *EUROCRYPT*. 2019, pp. 159–188.

[KLL04] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee. "Constant-Round Authenticated Group Key Exchange for Dynamic Groups." In: *ASIACRYPT*. 2004, pp. 245–259.

[MP] Moxie Marlinspike and Trevor Perrin. "The X3DH Key Agreement Protocol." URL: https://signal.org/docs/specifications/x3dh/.

[NY90] Moni Naor and Moti Yung. "Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks." In: *STOC*. 1990, pp. 427–437.

[PM] Trevor Perrin and Moxie Marlinspike. "The Double Ratchet Algorithm." URL: https://signal.org/docs/specifications/doubleratchet/.

[Sho04] Victor Shoup. "Sequences of games: a tool for taming complexity in security proofs." In: *IACR Cryptol. ePrint Arch.* (2004), p. 332. URL: http://eprint.iacr.org/2004/332.

[Sig] Signal protocol. “Signal protocol: Technical documentation.” URL: https://signal.org/doc.