A Posteriori Openable Public Key Encryption*

Xavier Bultel[1,2] and Pascal Lafourcade[1,2]

[1] CNRS, UMR 6158, LIMOS, F-63173 Aubière, France
[2] Université Clermont Auvergne, LIMOS, BP 10448, 63000 Clermont-Ferrand, France

Abstract. We present a public key encryption primitive called *A Posteriori Openable Public Key Encryption* (APO-PKE). In addition to conventional properties of public key cryptosystems, our primitive allows each user, who has encrypted messages using different public keys, to create a special decryption key. A user can give this key to a judge to open all messages that have been encrypted in a chosen time interval with the public keys of the receivers. We provide a generic efficient construction, in the sense that the complexity of the special key generation algorithm and this key size are independent of the number of ciphertexts. We give security models for our primitive against chosen plaintext attack and analyze its security in the random oracle model.

Keywords: Public-Key Encryption, Openable Encryption, ROM, CPA.

1 Introduction

Since the emergence of the Internet, email communication is accessible to anyone. Email privacy is an important computer security topic. Without public key encryption schemes, plaintext messages are sent and stored by the mail server without any protection. Fortunately, there exist many straightforward to use softwares that allow everyone to encrypt and sign emails using public key cryptography, such as the well known GnuPG[3] tool. Unfortunately, these softwares are rarely used [27], consequently encrypted emails may be considered as a suspect behavior. Hence as P. Zimmermann, the designer of PGP, said:"*If privacy is outlawed, only outlaws will have privacy*". We hope that in a near future everybody can privately exchange emails. Then our motivation is based on the following scenario, where Alice is implied in a court case. To find some clues, the judge needs to read emails that Alice has sent during a specified time period. The judge uses his power to obtain from Alice's email server all emails sent by Alice (including dates of dispatch and receiver identities). If the messages are not encrypted then the judge can read emails without relation to the investigation, which is a privacy violation. On the other hand, if messages are encrypted with

* This research was conducted with the support of the "Digital Trust" Chair from the University of Auvergne Foundation.

[3] `https://www.gnupg.org`

the receiver public key then the judge can suspect Alice to hide crucial information for the investigation. Moreover, without the receivers' private keys, Alice has no solution to prove her innocence and cannot reveal his correspondence to the judge.

To solve this problem, Alice needs a mechanism to give to the judge a possibility to open all messages sent during a specified time period. Using our solution Alice can construct such a special key called an *interval-key*. With this key, the judge can only read the encrypted messages sent during this specific interval of time, because this key does not allow him to open other encrypted messages stored on the email server. Nowadays, to the best of our knowledge, there is no efficient cryptographic solution that offers such functionality to the users. The goal of this paper is to propose a practical and efficient solution to this problem.

In many public key cryptosystems, when a ciphertext is generated, it is possible to create a special key that allows a person to decrypt it, without knowing the corresponding secret key. For example, in ElGamal [13], $C = (C_1, C_2) = (g^r, g^{x \cdot r} \cdot m)$ is the ciphertext of the message m with the public key g^x and a random element r (for g a generator of G a group of prime order). Knowing the random element r, the public key of Bob g^x and the ciphertext C a third party can compute $C_2/(g^x)^r = m$ to recover the plaintext. Using this property it is possible to construct a naïve solution by giving n random elements to a third party to decrypt n ciphertexts. However, this method presents an inherent limitation when the number n is large and the user has to store all the random elements used to encrypt all the messages during an interval of time. The aim of this paper is to allow a user to construct an interval-key to decrypt several consecutive messages in a time interval where the size of the key, the stored information and the key generation complexity are constant and do not increase with the number of ciphertexts.

Contributions: We first present the notion of *Random Coin Decryptable Public Key Encryption* (RCD-PKE). The idea of RCD-PKE is that one can open a ciphertext with the secret key and also use the random coin used during the encryption to open a cipher. We show that several existing schemes in the literature satisfy this notion, *e.g.* [1, 10, 14]. We use the RCD-PKE property to construct a scheme that allows a user to generate an interval-key for a judge to open all the messages he sent during a period of time. This scheme, called *A Posteriori Openable Public Key Encryption* (APO-PKE), allows the judge to open all messages sent between two given dates. The number of ciphertexts is potentially infinite but the judge decryption capability is limited to the a posteriori chosen interval. It contains, like a standard public key encryption, a key generation function, an encryption function and a decryption function. It also has an extraction function that, given two ciphertexts and a secret value, generates an interval-key for the judge. Using this interval-key he can then open all messages encrypted by different public keys between the two ciphertexts for which the key has been created. Our scheme is generic since it only relies on any IND-CPA secure RCD-PKE and hash functions.

Performances: Our scheme has reasonable encryption and decryption execution time overhead comparing to the PKE we use, because the size of ciphertexts generated by our scheme is approximately the double of the size of the PKE encryption. Moreover the generation of the interval-key, its size and the stored information are also independent of the number of messages contained in the interval of time. Finally, there is no restriction neither about the total number of generated ciphertexts nor about the number of ciphertexts in a time interval.

Security: We provide the security models to prove the security of our schemes in the Random Oracle Model (ROM). We prove that the judge colluding with some users cannot learn more than the messages for which he received the interval-key. We also show that several users cannot collude in order to learn information about plaintexts contained in an interval of ciphertexts with the judge interval-key. We also demonstrate that the judge gets the same plaintext as the one received by the owners of the secret keys. This means that it is not possible to forge fake messages that the judge can open and not the owners of the secret keys, and *vice-versa*.

Our construction allows us to use the extraction algorithm only once per judge (or per set of encrypted mails). Our security model captures this situation. It is not going against our motivation as long as we consider that two judges having an interval key in two different court cases (for the same set of mails) do not collude. To avoid this drawback, we need to reinitialize the secret values stored by a user after the generation of an interval-key, in order to be able to produce new interval-key on the next encrypted data. We leave the construction of an APO-PKE with constant interval key generation complexity and constant interval key size allowing several interval key generations for the same judge and the same set of encrypted mails as an open problem.

Related work: Functional encryption [26] is a public-key encryption primitive that allows a user to evaluate a function on the plaintext message using a key and a ciphertext. This cryptographic primitive was formalized in [5]. It generalizes many well know cryptographic primitives such identity based encryption [4] or attribute based encryption [26]. Moreover, some schemes that evaluate an arbitrary function have been proposed in [17,18]. *A posteriori* openable encryption can be seen as a functional encryption, where all ciphertexts (resp. plaintexts) that are encrypted by one user correspond to a unique large ciphertext (resp. plaintext). Then the interval-keys allow a user to find only some parts of the corresponding plaintext. Our proposal scheme is an efficient solution for this kind of functional encryption.

Deniable encryption [7, 22] is an encryption system that allows to encrypt two messages (original and hidden messages) in the same ciphertext. Using his secret key, the receiver can retrieve the original message. Using another shared secret key, the receiver can also decrypt the hidden message. It is not possible for the sender to prove that his encryption does not contain an hidden encrypted message. In our *a posteriori* openable encryption, the judge is only convinced that the plaintext that he decrypts is the same message that the plaintext decrypted by the secret key of the receiver. This notion differs from undeniability

since the judge is convinced that a message he decrypts using interval key has actually been sent and received, but does not deal with message from another channel that the given encryption system (including different way to encrypt or decrypt a message in the same ciphertext).

Some cryptographic primitives deal with time in decryption mechanism or rights delegation. *Timed-Release Encryption* (TRE), first proposed in [24], is a public key encryption where encrypted messages cannot be opened before a *release-time* chosen by the person who encrypted the messages. In this primitive, it is generally a time server that allows the receiver to decrypt the message *in the future* at a given date. Several TRE with diverse security properties have been proposed [3,8,9]. More recently, an extension of TRE, called *Time-Specific Encryption* (TSE), has been proposed in [25] and deals with time intervals. Somehow these primitive are close to our because APO-PKE allows somebody to give decryption capabilities *in the future*, after that encrypted messages has been sent. However, TRE and TSE cannot be used to achieve APO-PKE, because TRE ciphertext are intended to only one user and decryption capabilities cannot be delegated to another party. Moreover, in TRE, time of decryption capability must be chosen during the encryption phase, while in our primitive it can be chosen at any time (*a posteriori*).

It is interesting to note that some TRE possess a pre-open mechanism [21] that allows the sender to give decryption capabilities before the pre-specified release-time. In this case, a security requirement (called *binding* property) ensures that the decrypted message from the pre-open mechanism is the message decrypted by the receiver after the release-time [11]. For our primitive, we define a similar property, called *integrity*, since we require that decrypted messages using an interval key must be equal to the messages decrypted by the legitimate receivers.

Finally, *Key-Insulated Encryption* (KIE) [12,20,23] is a public key encryption primitive where messages are encrypted from a tag corresponding to a time period and a public key. At each time period corresponds a partial secret key computed from a master key and the previous partial secret key. Moreover, the public key is never changed. The motivation of this primitive is to provide secret keys that can be stored in an untrusted device without compromising the master key. Indeed, the leakage of a secret key compromises only messages received in a specified time interval, and future encryptions remain secure. In the motivation of [12], the authors give another interesting use of this primitive based on [16]. They provide a secure delegation of decryption rights in a time period. However, this type of delegation allows them to delegate decryption rights only on predefined time period. For example, if the time period corresponds to one month then right delegation cannot be restricted to the last week of a month and the first week of the following month without revealing all messages of these two months. Moreover, delegator must give a different secret key to each time period, so the decryption keys are proportional to the number of time periods contained in the interval. Our goal is to propose decryption delegation capabilities to the

sender, while KIE only focuses on receiver decryption right delegation. Thus this primitive cannot solve our problem.

Outline: In the next section, we introduce some cryptographic tools and define the notion of RCD-PKE. In Section 3, we present a generic *A Posteriori Openable Public Key Encryption*. Then in Section 4, we provide security models and analyze the security of our scheme before concluding in the last section. All the proofs of our security results are given in the full version of this paper [6].

2 Random Coin Decryptable Public Key Encryption

We first recall the definition of probabilistic public key encryption.

Definition 1 (Probabilistic Public Key Encryption (PKE)). *A* probabilistic PKE *is a triplet of polynomial time algorithms* $(\mathit{Gen}, \mathit{Enc}, \mathit{Dec})$ *such that* $\mathsf{Gen}(1^k)$ *returns a public/private key pair* $(\mathit{pk}, \mathit{sk})$*,* $\mathsf{Enc}_{\mathsf{pk}}(m;\sigma)$ *returns a ciphertext* c *from the public key* pk*, the message* m *and the random coin* σ*, and* $\mathsf{Dec}_{\mathsf{sk}}(c)$ *returns a plaintext* m *or a bottom symbol* $\perp$ *from a secret key* sk *and a ciphertext* c*. Moreover the following equation holds:* $\mathit{Dec}_{\mathit{sk}}(\mathit{Enc}_{\mathit{pk}}(m;\sigma)) = m$*.*

A PKE scheme Π is said indistinguishable under chosen-plaintext attack (IND-CPA) [19] if for any polynomial time adversary $\mathcal{A}$, the difference between $\frac{1}{2}$ and the probability that $\mathcal{A}$ wins the IND-CPA experiment described in Fig. 1 is negligible.

$\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$:
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_0(1^k, \mathsf{pk})$
$c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b;\sigma)$
$b' \leftarrow \mathcal{A}_1(\mathsf{st}, \mathsf{pk}, c)$
return $(b = b')$

Fig. 1. IND-CPA experiment.

We introduce the notion of *Random Coin Decryptable PKE* (RCD-PKE). A public key encryption scheme is said RCD-PKE, if there exists a second way to decrypt the ciphertext with the random coin used to construct the ciphertext. This primitive is a kind of PKE with *double decryption* mechanism (DD-PKE) which is defined in [15]. Actually RCD-PKE is a DD-PKE where the second secret key is the random coin and is used once.

Definition 2 (Random Coin Decryptable PKE (RCD-PKE)). *A probabilistic PKE is* Random Coin Decryptable *if there exists a polynomial time algorithm* CDec *such that for any public key* pk*, any message* m*, and any coin* σ*, the following equation holds:* $\mathit{CDec}_{\sigma}(\mathit{Enc}_{\mathit{pk}}(m;\sigma), \mathit{pk}) = m$*.*

For instance, ElGamal encryption scheme is RCD-PKE. It is possible, from a ciphertext $c = \mathsf{Enc}_{\mathsf{pk}}(m;\sigma) = (c_0, c_1) = (g^{\sigma}, \mathsf{pk}^{\sigma} \cdot m)$ to use the algorithm $\mathsf{CDec}_{\sigma}(c, \mathsf{pk})$ that computes c_1/pk^{σ} to retrieve the plaintext message m. Many probabilistic encryption schemes in the literature are RCD-PKE, *e.g.* [1, 10, 14]. Algorithms CDec of these two cryptosystems PKE are given in the full version of this paper [6]. We also introduce the concepts of *valid key pair* and of *verifiable key PKE*.

Definition 3 (Verifiable Key PKE (VK-PKE)). *We say that a key pair* $(\mathsf{pk}, \mathsf{sk})$ *is* valid *for* $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *when for any message* m *and any random coin* σ *the equation* $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m;\sigma)) = m$ *holds. We say that a probabilistic* PKE *is* verifiable-key *(VK) when there exists an algorithm* Ver *such that* $\mathsf{Ver}(\mathsf{pk}, \mathsf{sk}) = 1$ *if and only if* $(\mathsf{pk}, \mathsf{sk})$ *is valid for* PKE.

In many probabilistic public key cryptosystems, the public key is generated from the secret key by a deterministic algorithm. For example, the ElGamal public key is the value g^x computed from the secret key x. In this case, it suffices to check that $g^{\mathsf{sk}} = \mathsf{pk}$ in order to be convinced that a key pair $(\mathsf{pk}, \mathsf{sk})$ is valid. It is easy to see that [1, 10] are also VK-PKE.

3 A Posteriori Openable Public Key Encryption

An APO-PKE is a public key encryption scheme, where Alice can use receiver public keys to send them encrypted messages that can be opened thanks to the corresponding secret keys. The goal of an APO-PKE is to allow Alice to keep enough information to be able to construct a key to *a posteriori* open a sequence of messages that she had encrypted during an interval of time. We do not consider real time but a sequence of n successive ciphertexts $\{C_x\}_{1 \leq x \leq n}$ that have been encrypted by Alice with possibly different public keys. Then with an APO-PKE, it is possible for Alice to *extract* a key for a judge that opens all ciphertexts between the message C_i and the message C_j where $1 \leq i < j \leq n$. We call this key an *interval-key* denoted by $K_{i \to j}^{\mathsf{pko}}$ where pko is the public key of the opener (here the judge). Moreover before encrypting her first message with a public key, Alice needs to *initialize* a *secret global state* denoted st. The goal of st is to keep all required information to generate an interval-key and to encrypt a new message. Naturally each time Alice encrypts a message with a public key, st is updated (but has a constant size). Finally an APO-PKE, formally described in Definition 4, contains an algorithm that *opens* all ciphertexts in a given interval of time thanks to the interval-key forged by Alice.

Note that all key pairs come from the same algorithm APOgen. However, for the sake of clarity, we denote by pko and sko (for *opener public key* and *opener secret key*) the keys of an interval-key recipient, *e.g.* a judge that can open some messages, denoted by O (for opener) in the rest of the paper.

Definition 4 (A Posteriori Openable Public Key Encryption (APO-PKE)). *An APO-PKE is defined by:*

$\mathsf{APOgen}(1^k)$**:** *This algorithm generates a key pair for a user. It returns a public/private key pair* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{APOini}(1^k)$**:** *This algorithm initializes a global state* st *and returns it.*

$\mathsf{APOenc}_{\mathsf{pk}}^{\mathsf{st}}(m)$**:** *This algorithm encrypts a plain-text* m *using a public key* pk *and a global state* st*. It returns a ciphertext* C *and* st *updated.*

$\mathsf{APOdec}_{\mathsf{sk}}(C)$**:** *This algorithm decrypts a ciphertext* C *using the secret key* sk*. It returns a plaintext* m *or* $\perp$ *in case of error.*

$\mathsf{APOext}^{\mathsf{st}}_{\mathsf{pko}}(C_i, C_j)$**:** *This algorithm generates an interval-key* $K^{pko}_{i\to j}$ *that allows the owner* O *of the public key* pko *to decrypt all messages* $\{C_x\}_{i\le x\le j}$ *using algorithm* APOpen*.*

$\mathsf{APOpen}_{\mathsf{sko}}(K^{\mathsf{pko}}_{i\to j}, \{C_x\}_{i\le x\le j}, \{\mathsf{pk}_x\}_{i\le x\le j})$**:** *Inputs of this algorithm contain a ciphertext set* $\{C_x\}_{i\le x\le j}$ *and all the associated public keys* $\{\mathsf{pk}_x\}_{i\le x\le j}$*. This algorithm allows a user to decrypt all encrypted messages sent during an interval using his secret key* sk *and the corresponding interval-key* $K^{pko}_{i\to j}$*.It returns a set of plaintexts* $\{m_x\}_{i\le x\le j}$ *or* $\perp$ *in case of error.*

In Scheme 1, we give a generic construction of APO-PKE based on an IND-CPA secure RCD-PKE and three hash functions.

Scheme 1 (Generic APO-PKE (G-APO)) *Let* k *be a security parameter,* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a RCD and VK PKE scheme,* $\mathcal{R}$ *be the set of possible random coins of* $\mathcal{E}$ *and* $\mathsf{F} : \{0,1\}^* \to \{0,1\}^k$*,* $\mathsf{G} : \{0,1\}^* \to \mathcal{R}$ *and* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2k}$ *be three universal hash functions. Our generic APO-PKE is defined by the following six algorithms where* $\oplus$ *denotes the exclusive-or,* $|x|$ *denotes the bit size of message* x *and* $y||z$ *the concatenation of* y *with* z*:*

$\mathsf{APOgen}(1^k)$**:** *This algorithm generates* $(\mathsf{pk}, \mathsf{sk})$ *with* Gen *and returns it.*

$\mathsf{APOini}(1^k)$**:** *This algorithm picks three random values* $\widehat{\sigma} \xleftarrow{\$} \{0,1\}^k$*,* $\widetilde{\sigma} \xleftarrow{\$} \{0,1\}^k$ *and* $K \xleftarrow{\$} \{0,1\}^k$ *of the same size, and returns the state* $\mathsf{st} = (K||\widehat{\sigma}||\widetilde{\sigma})$*.*

$\mathsf{APOenc}^{\mathsf{st}}_{\mathsf{pk}}(m)$**:** *We note that* $\mathsf{st} = (K||\widehat{\sigma}_N||\widetilde{\sigma}_N)$*. This algorithm picks a random* $\widehat{m}$ *such that* $|\widehat{m}| = |m|$ *and computes* $\widetilde{m} = \widehat{m} \oplus m$*. Let* $\widehat{\sigma} \xleftarrow{\$} \{0,1\}^k$ *and* $\widetilde{\sigma} \xleftarrow{\$} \{0,1\}^k$ *be two random values of size* $|\widehat{\sigma}_N|$*. This algorithm computes* $\widehat{C} = \mathsf{Enc}_{pk}(\widehat{m}||(\widehat{\sigma} \oplus \mathsf{F}(\widehat{\sigma}_N)); \mathsf{G}(\widehat{\sigma}_N))$ *and* $\widetilde{C} = \mathsf{Enc}_{pk}(\widetilde{m}||(\widetilde{\sigma}_N \oplus \mathsf{F}(\widetilde{\sigma})); \mathsf{G}(\widetilde{\sigma}))$*. It also computes* $D = (\widehat{\sigma}_N||\widetilde{\sigma}) \oplus \mathsf{H}(K||\widehat{C}||\widetilde{C})$*. Finally it updates the state* st *with* $(K||\widehat{\sigma}||\widetilde{\sigma})$ *and returns* $C = (\widehat{C}||\widetilde{C}||D)$*.*

$\mathsf{APOdec}_{\mathsf{sk}}(C)$**:** *The decryption algorithm computes the decryption of* $\widehat{m}||\widehat{\sigma} = \mathsf{Dec}_{sk}(\widehat{C})$ *and the decryption of* $\widetilde{m}||\widetilde{\sigma} = \mathsf{Dec}_{sk}(\widetilde{C})$*, where* $C = (\widehat{C}||\widetilde{C}||D)$*. It returns* $m = \widehat{m} \oplus \widetilde{m}$*.*

$\mathsf{APOext}^{\mathsf{st}}_{\mathsf{pko}}(C_i, C_j)$**:** *Using the state* $\mathsf{st} = (K||\widehat{\sigma}_N||\widetilde{\sigma}_N)$*,* $C_i = (\widehat{C}_i||\widetilde{C}_i||D_i)$ *and* $C_j = (\widehat{C}_j||\widetilde{C}_j||D_j)$*, this algorithm computes* $\widehat{\sigma}_{i-1}||\widetilde{\sigma}_i = D_i \oplus \mathsf{H}(K||\widehat{C}_i||\widetilde{C}_i)$ *and* $\widehat{\sigma}_{j-1}||\widetilde{\sigma}_j = D_j \oplus \mathsf{H}(K||\widehat{C}_j||\widetilde{C}_j)$*. It picks* $r \xleftarrow{\$} \mathcal{R}$ *and returns* $K^{pko}_{i\to j} = \mathsf{Enc}_{pko}((\widehat{\sigma}_{i-1}||\widetilde{\sigma}_j); r)$*.*

$\mathsf{APOpen}_{\mathsf{sko}}(K^{\mathsf{pko}}_{i\to j}, \{(\widehat{C}_x||\widetilde{C}_x||D_x)\}_{i\le x\le j}, \{\mathsf{pk}_x\}_{i\le x\le j})$**:** *This algorithm begins to recovering values* $\widehat{\sigma}_{i-1}||\widetilde{\sigma}_j = \mathsf{Dec}_{sko}(K^{pko}_{i\to j})$*.*

- *For all* x *in* $\{i, i+1, \ldots, j\}$*, it computes* $\widehat{R} = \mathsf{G}(\widehat{\sigma}_{x-1})$ *and opens* $\widehat{C}_x$ *as follows* $\widehat{m}_x||\widehat{\sigma}^*_x = \mathsf{CDec}_{\widehat{R}}(\widehat{C}_x, \mathsf{pk}_x)$*. It computes the next* $\widehat{\sigma}_x = \widehat{\sigma}^*_x \oplus \mathsf{F}(\widehat{\sigma}_{x-1})$*. If* $\mathsf{Enc}_{\mathsf{pk}_x}((\widehat{m}_x||\widehat{\sigma}^*_x); \mathsf{G}(\widehat{\sigma}_{x-1})) \neq \widehat{C}_x$ *then it returns* $\perp$*.*
- *For all* x *in* $\{j, j-1, \ldots, i\}$*, it computes* $\widetilde{R} = \mathsf{G}(\widetilde{\sigma}_x)$ *and opens* $\widetilde{C}_x$ *as follows* $\widetilde{m}_x||\widetilde{\sigma}^*_{x-1} = \mathsf{CDec}_{\widetilde{R}}(\widetilde{C}_x, \mathsf{pk}_x)$*. It computes the previous* $\widetilde{\sigma}_{x-1} = \widetilde{\sigma}^*_{x-1} \oplus \mathsf{F}(\widetilde{\sigma}_x)$*. If* $\mathsf{Enc}_{\mathsf{pk}_x}((\widetilde{m}_x||\widetilde{\sigma}^*_{x-1}); \mathsf{G}(\widetilde{\sigma}_x)) \neq \widetilde{C}_x$ *then it returns* $\perp$*.*

Finally, it returns $\{\widehat{m}_x \oplus \widetilde{m}_x\}_{i\le x\le j}$*.*

The encryption algorithm APOenc separates the plaintext m in two parts using xor operation such that $m = \widehat{m} \oplus \widetilde{m}$. We generate two random coins $\widehat{\sigma}$ and $\widetilde{\sigma}$. Using the two previous coins $\widehat{\sigma}_N$ and $\widetilde{\sigma}_N$ in the state st, we encrypt into two different ciphertexts $\widehat{C}$ and $\widetilde{C}$ the following two messages $\widehat{m}||(\widehat{\sigma} \oplus \mathsf{F}(\widehat{\sigma}_N))$ and $\widetilde{m}||(\widetilde{\sigma}_N \oplus \mathsf{F}(\widetilde{\sigma}))$. Finally we hide the usefull random elements with $\mathsf{H}(K||\widehat{C}||\widetilde{C})$.

Knowing the secret key it is possible to recover $\widehat{m}$ and $\widetilde{m}$ and then to obtain the plaintext m thanks to the algorithm APOdec.

An interval-key for the owner O of a public key pko is constructed using the algorithm APOext. It is simply the encryption with pko of $\widehat{\sigma}_N$ and $\widetilde{\sigma}$. At each encryption, the values $\widehat{\sigma}_{i-1}$ and $\widetilde{\sigma}_i$ are masked by a *"one time pad"* with the digest $\mathsf{H}(K||\widehat{C}_i||\widetilde{C}_i)$ in D_i. Then with the ciphertexts C_i, C_j and the secret value K we can construct an interval-key that contains these values $\widehat{\sigma}_{i-1}$ and $\widetilde{\sigma}_j$.

Using an interval-key $K^{\mathsf{pko}}_{i \to j}$ it is possible to open all ciphertexts encrypted during an interval of time with the algorithm APOpen: thanks to the RCD property, someone who knows values $\widehat{\sigma}_N$ and $\widetilde{\sigma}$ for one ciphertext can open each part $\widehat{C}$ and $\widetilde{C}$ of it in order to recover $\widehat{\sigma}$ and $\widetilde{\sigma}_N$, and $\widehat{m}$ and $\widetilde{m}$, hence m. We also notice that with $\widehat{\sigma}_i$ it is possible to decrypt all ciphertexts in $\{\widehat{C}_x\}_{(i+1) \leq x \leq N}$. In the other hand, with $\widetilde{\sigma}_j$ it is possible to decrypt all ciphertexts in $\{\widetilde{C}_x\}_{1 \leq x \leq j}$. Then it is possible to recover all messages between C_i and C_j. Thus, it is possible to decrypt all messages between C_i and C_j with the knowledge of $\widehat{\sigma}_{i-1}$ and $\widetilde{\sigma}_j$.

If the interval always contains the first message, we give a more efficient algorithm. The idea is to only keep one part of the ciphertext, by consequence we do not need to split into two the message m. Hence the size of the ciphertext is smaller. Similarly if the algorithm always ends with the last encrypted message, we can also drop one half of the ciphertext and the tag value following the same idea. These simpler schemes are given in the full version of this paper [6].

4 Model and Security

We present the security properties of an APO-PKE scheme and we analyze the security of our G-APO scheme. The first security property corresponds to a chosen-plaintext attack scenario where the adversary has access to interval-keys on intervals that do not contain the challenge. We next introduce the notion of *indistinguishability under chosen sequence of plaintext attack* security (IND-CSPA) that corresponds to a chosen-plaintext attack scenario where the challenge is an interval of ciphertexts and the corresponding interval-key generated for a given judge public key. The last property is *integrity*, and captures the integrity of messages decrypted by APOpen algorithm. All security proofs are detailed in [6].

4.1 IND-CPA security

It concerns the resistance of an APO-PKE against a collusion of adversaries that have access to interval-keys in a chosen-plaintext attack scenario. For example, if

we consider a judge who receives an interval-key to open a sequence of ciphertexts and who colludes with ciphertext recipients; then it ensures that they cannot deduce any information about messages that are not in the sequence. Indeed, he cannot request an interval-key for an interval containing the challenge. We define the OT-IND-CPA security when only one interval-key can be asked during the experiment. Our scheme is proved secure in this model.

Definition 5 (OT-IND-CPA experiment). *Let Π be an APO-PKE, let k be a security parameter, and let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a pair of polynomial time algorithms. We define the* one-time indistinguishability under interval opener chosen-plaintext attack *(OT-IND-CPA) experiment as follows:*

$\mathbf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$*:*
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{pk}_*, \mathsf{sk}_*) \leftarrow \mathsf{APOgen}(1^k)$
$\mathsf{st}_* \leftarrow \mathsf{APOini}(1^k)$
$(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_0(1^k, \mathsf{pk}_*)$
$C_* \leftarrow \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}_*}(m_b)$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, C_*)$
If $b = b'$ *return* 1*, else* 0

The adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$ have access to the following oracles:

$\mathcal{O}^{\mathsf{CPA}}_{\mathsf{enc}}$**:** *On the first call to this oracle, it initializes the following values $l = 1$ and $n = 1$. This oracle takes as input a public key pk and a message m. It returns $C_l = \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}}(m)$. It increments the counter l. Only in the first phase, it increments the value n that counts the number of calls to the encryption oracle before the generation of the challenge.*

$\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$**:** *The adversary can ask this oracle only one time during the experiment. This oracle takes a public key pko and two ciphertexts C'_a and C'_b. In the second phase, if there exists $C_i = C'_a$ and $C_j = C'_b$ such that $i \leq n \leq j$ then the oracle rejects the query. Else, if $C'_a = C_n$ or $C'_b = C_n$, it rejects the query. Else it returns $\mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}}(C'_a, C'_b)$.*

We also define the IND-CPA experiment as the same as the OT-IND-CPA experiment except that the adversary can ask the oracle APOext several times.

Definition 6 (OT-IND-CPA advantage). *The advantage of the adversary $\mathcal{A}$ against OT-IND-CPA is defined by:*

$$\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k) = |Pr[\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k) = 1] - \frac{1}{2}|$$

We define the advantage on OT-IND-CPA experiment by:

$$\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi}(k) = \max\{\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)\}$$

for all $\mathcal{A} \in \mathrm{POLY}(k)$. The advantages on IND-CPA experiment are similar to those of OT-IND-CPA. We say that a APO-PKE scheme Π is OT-IND-CPA (resp. IND-CPA) secure when $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi}(k)$ (resp. $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi}(k)$) is negligible.

Our construction is not IND-CPA since if a judge has two interval-keys for two different intervals of time given by the same user and computed with the same secret value then he can open all messages between the two extreme dates.

Theorem 1. *Let E be an IND-CPA secure RCD-PKE, then G-APO based on E is OT-IND-CPA secure in the random oracle model.*

Proof idea: To prove the OT-IND-CPA security, we show first that no polynomial adversary wins the experiment with non negligible probability using the oracle $\mathcal{O}_{\mathsf{ext}}^{\mathsf{CPA}}$ in an interval of previous ciphertexts of the challenge. The interval-key allows to open the part $\widehat{C}_*$ of the challenge C_*, but since the PKE is IND-CPA then the interval-key gives no information about the part of the challenge encrypted in the part $\widetilde{C}_*$. Similarly, we then prove that no adversary can win using the oracle in an interval of next ciphertexts of the challenge. Finally, using this two results, we show that our scheme is OT-IND-CPA in any case. □

4.2 IND-CSPA security

A sequence of ciphertexts coupled with an interval-key can be seen as an unique ciphertext that encrypts a sequence of plaintexts because the open algorithm allows a judge to decrypt all the messages of the sequence with the knowledge of any secret key. Thus, we define a security model where the adversary must distinguish the sequence of plaintexts used to produce a challenge sequence of ciphertexts associated to an interval-key. The IND-CSPA security captures this security property. In this model, the adversary is a collusion of users that must distinguish the sequence of plaintexts used to produce a sequence of ciphertexts given the corresponding interval-key generated for the judge.

Definition 7 (IND-CSPA$_\phi$ experiment). *Let Π be an APO-PKE, let k be a security parameter, and let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a pair of polynomial time algorithms. We define the* indistinguishability under chosen sequence of plaintext attack *(IND-CSPA$_\phi$) experiment as follows, where n denotes the number of calls to the encryption oracle during the first phase and ϕ denotes the number of calls to the generation oracle:*

$\mathbf{Exp}^{\mathsf{IND\text{-}CSPA}_\phi}_{\Pi,\mathcal{A}}(k)$:
$b, d \xleftarrow{\$} \{0,1\}$
$(\mathsf{pko}_*, \mathsf{sko}_*) \leftarrow \mathsf{APOgen}(1^k)$
$\mathsf{st}_* \leftarrow \mathsf{APOini}(1^k)$
$(q, \{m_x^0\}_{n<x\leq n+q}, \{m_x^1\}_{n<x\leq n+q}, \{\mathsf{pk}_x\}_{n<x\leq n+q}, \mathsf{state}) \leftarrow \mathcal{A}_0(1^k, \mathsf{pko}_*)$
$\forall\ x \in \{n+1, n+2, ..., n+q\}$:
  if pk_x *comes from* $\mathcal{O}_{\mathsf{gen}}^{\mathsf{CSPA}}$ *then* $C_x^* = \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}_x}(m_x^b)$
  else, $C_x^* = \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}_x}(m_x^d)$
$K^{\mathsf{pko}_*}_{(n+1)\to(n+q)} \leftarrow \mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}_*}(C_{n+1}, C_{n+q})$
$b' \leftarrow \mathcal{A}_1(\mathsf{state}, \{C_x^*\}_{n<x\leq n+q}, K^{\mathsf{pko}_*}_{(n+1)\to(n+q)})$
If $b = b'$ *return* 1, *else* 0

The adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$ have access to the following oracles:

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{gen}}$**:** *At the first call, the oracle creates a keys' list* K *that contains* $(\mathsf{pko}_*, \mathsf{sko}_*)$. *At each call, it generates values* $(\mathsf{pk}, \mathsf{sk})$ *from* $\mathsf{APOgen}(1^k)$ *and adds it to* K. *Then it returns* pk. *This oracle can be called only* ϕ *times.*

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$**:** *This oracle takes as inputs a public key* pk *and a message* m. *Only in the first phase, it increments the value* n *that counts the number of calls to the encryption oracle before the generation of the challenge.In the two phases, it returns* $\mathsf{APOenc}^{\mathsf{st}_*}_{pk}(m)$.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$**:** *This oracle takes as input two ciphertexts* C_i *and* C_j. *It returns the interval-key* $K^{pko_*}_{i\to j} = \mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}_*}(C_i, C_j)$.

In the first phase The challenger generates $(\mathsf{pko}_*, \mathsf{sko}_*)$ from $\mathsf{APOgen}(1^k)$ and a state st_* from $\mathsf{APOini}(1^k)$. He sends the public key pko_* to the adversary. The challenger initializes a counter n that counts number of calls to the oracle $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$ during this phase. Finally, the adversary sends to the challenger values $(q, \{m^0_x\}_{n<x\leq(n+q)}, \{m^1_x\}_{n<x\leq(n+q)}, \{\mathsf{pk}_x\}_{n<x\leq n+q}, \mathsf{state})$.

In second phase, the challenger computes a sequence of ciphertexts from the adversary's output. He encrypts messages of one of the two sequences. The sequence of produced ciphertexts forms the challenge. More formally, the challenger picks two random bits b and d. Then, $\forall\ x \in \{n+1, n+2, ..., n+q\}$, if pk_x corresponds to an honest user (*i.e.* pk_x comes from oracle $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{gen}}$) then he computes $C^*_x = \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}_x}(m^b_x)$ else if pk_x corresponds to a dishonest user (*i.e.* pk_x comes from the adversary), he computes $C^*_x = \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}_x}(m^d_x)$. Finally, he computes $K^{\mathsf{pko}_*}_{(n+1)\to(n+q)} = \mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}_*}(C_{n+1}, C_{n+q})$ and he sends $(\mathsf{state}, \{C^*_x\}_{n<x\leq(n+q)}, K^{\mathsf{pko}_*}_{(n+1)\to(n+q)})$ to the adversary $\mathcal{A}_1$. During the guess phase, the adversary returns the bit b'. If $b' = b$ then $\mathcal{A}$ wins.

Definition 8 (IND-CSPA advantage). *We define the advantage of* $\mathcal{A}$ *against* IND-CSPA *by:*

$$\mathsf{Adv}^{\mathit{IND\text{-}CSPA}_\phi}_{\Pi,\mathcal{A}}(k) = |Pr[\mathit{Exp}^{\mathit{IND\text{-}CSPA}_\phi}_{\Pi,\mathcal{A}}(k) = 1] - \frac{1}{2}|$$

We define by:

$$\mathsf{Adv}^{\mathit{IND\text{-}CSPA}_\phi}_{\Pi}(k) = \max\{\mathsf{Adv}^{\mathit{IND\text{-}CSPA}_\phi}_{\Pi,\mathcal{A}}(k)\}$$

for all $\mathcal{A} \in \text{POLY}(k)$ *the advantage on* IND-CSPA. *We say that an* APO-PKE *scheme* Π *is* IND-CSPA *secure when the advantage* $\mathsf{Adv}^{\mathit{IND\text{-}CSPA}_\phi}_{\Pi}(k)$ *is negligible for any polynomial* ϕ.

Theorem 2. *Let* E *be a* PKE *that is* RCD, *then* G-APO *using* E *is* IND-CSPA *secure in the random oracle model.*

Proof idea: In [2] authors prove that any IND-CPA PKE is still secure in multi-user setting, *i.e.* where the adversary can ask several challenges for several different public keys. Without interval-key oracle, the IND-CSPA security of our scheme can be reduced to the IND-CPA of the PKE in multi-user setting since the challenge corresponds to ciphertexts of several messages from several public keys. Moreover, since the interval-keys from the oracle are encrypted, then the adversary must break the IND-CPA security of PKE to use it. It is possible to prove that no adversary can efficiently break the IND-CSPA of our scheme using these two arguments. □

4.3 Integrity

The last security property for APO-PKE is the *integrity*. This property is similar to *binding* property of TRE defined in [11]. The judge must be sure that the messages he decrypts with APOpen algorithm are the sent messages.

Definition 9 (Integrity experiment). *Let Π a APO-PKE, let k be a security parameter, and let $\mathcal{A}$ a polynomial time algorithm. We define the* integrity *experiment as follows:*

$\mathbf{Exp}^{\mathsf{Integrity}}_{\Pi,\mathcal{A}}(k)$*:*
$(pko_*, sko_*) \leftarrow APOgen(1^k)$
$(N, \{C_x\}_{1\leq x\leq N}, \{pk_x\}_{1\leq x\leq N}, l, sk_l, i, j, K^{pko_*}_{i\to j}) \leftarrow \mathcal{A}(1^k, pk_*)$
if (pk_l, sk_l) *is not a valid key pair then return* 0
$\{m_x\}_{i\leq x\leq j} \leftarrow APOpen_{sko_*}(K^{pko_*}_{i\to j}, \{C_x\}_{i\leq x\leq j}, \{pk_x\}_{i\leq x\leq j})$
if $m_l \neq APOdec_{sk_l}(C_l)$ *then return* 1*, else* 0*.*

The challenger generates $(\mathsf{pko}_*, \mathsf{sko}_*)$ from $\mathsf{APOgen}(1^k)$ and sends the public key pko_* to the adversary. The adversary $\mathcal{A}$ sends to the challenger an integer N, an ordered set of N ciphertexts $\{C_x\}_{1\leq x\leq N}$ and an ordered set of N public keys $\{\mathsf{pk}_x\}_{1\leq x\leq N}$. The adversary then sends two integers i and j and the corresponding interval-key $K^{\mathsf{pko}_*}_{i\to j}$. He finally sends the integer l and the secret key sk_l corresponding to pk_l. If $(\mathsf{pk}_l, \mathsf{sk}_l)$ is not a valid key pair then the challenger aborts and returns 0. The challenger then computes $\{m_x\}_{i\leq x\leq j} \leftarrow \mathsf{APOpen}_{\mathsf{sko}_*}(K^{\mathsf{pko}_*}_{i\to j}, \{C_x\}_{i\leq x\leq j}, \{\mathsf{pk}_x\}_{i\leq x\leq j})$. If $m_l \neq \mathsf{APOdec}_{\mathsf{sk}_l}(C_l)$ then the challenger returns 1, else he returns 0.

Definition 10. *The advantage of $\mathcal{A}$ against integrity is defined by:*

$$\mathsf{Adv}^{Integrity}_{\Pi,\mathcal{A}}(k) = Pr[Exp^{Integrity}_{\Pi,\mathcal{A}}(k) = 1]$$

The advantage against integrity by:

$$\mathsf{Adv}^{Integrity}_{\Pi}(k) = \max\{\mathsf{Adv}^{Integrity}_{\Pi,\mathcal{A}}(k)\}$$

for all $\mathcal{A} \in \text{POLY}(k)$. We say that a APO-PKE scheme Π satisfies the integrity property $\mathsf{Adv}^{Integrity}_{\Pi}(k)$ is negligible.

Theorem 3. *Let E be a RCD and VK PKE that is IND-CPA secure, then G-APO using this PKE satisfies the integrity property.*

Proof idea: Since the judge has all the random coins and all the public keys used to encrypt all the opened messages, he can use them to re-encrypt these messages. Thus, if the ciphertexts that he opens correspond to the ciphertexts that he encrypts by himself, then he can conclude that the opened messages are the same as the messages decrypted by the recipient secret keys. □

5 Conclusion

We introduce the notion of RCD-PKE. Based on this notion, we propose an *a posteriori* openable PKE (APO-PKE) scheme. Our scheme allows a user to prove his innocence by showing to a judge the content of his encrypted communication with several PKE during a period of time. Our construction preserves the privacy of the others communications, meaning that the judge cannot learn any information concerning the other encrypted messages. Moreover the receivers of the encrypted messages cannot collude in order to learn more information that is contained in the received messages. Our construction is proven secure in the Random Oracle Model and is generic because it only requires RCD-PKE and hash functions.

In the future, we aim at proving that is not possible to have a secure construction that supports several generations of interval key with constant size interval-key and stored data (state). Another future work is to design a security model for chosen-ciphertext security of APO-PKE and to provide a generic construction that achieves this higher security. Finally, it may be interesting to design such a scheme in the standard model.

References

1. M. Abdalla, M. Bellare, and P. Rogaway. DHIES: an encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a, Sept. 1998.
2. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, May 2000.
3. I. F. Blake and A. C.-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. ICDS, IEEE Computer Society Press, 2005.
4. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.
5. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Mar. 2011.
6. X. Bultel and P. Lafourcade. A posteriori openable public key encryption. Technical report, University Clermont Auvergne, LIMOS, 2015. `http://sancy.univ-bpclermont.fr/~lafourcade/APOPKE.pdf`.
7. R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *CRYPTO'97*, volume 1294 of *LNCS*, pages 90–104. Springer, 1997.

8. J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In *ICICS 05*, volume 3783 of *LNCS*, pages 291–303. Springer, 2005.
9. J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Timed-release and key-insulated public key encryption. In G. Di Crescenzo and A. Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 191–205. Springer, Feb. / Mar. 2006.
10. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
11. A. W. Dent and Q. Tang. Revisiting the security model for timed-release encryption with pre-open capability. In J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, editors, *ISC 2007*, volume 4779 of *LNCS*, pages 158–174. Springer, Oct. 2007.
12. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 65–82. Springer, 2002.
13. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.
14. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, 2013.
15. D. Galindoa and J. Herranz. On the security of public key cryptosystems with a double decryption mechanism. *Information Processing Letters*, 108(5):279–283, Nov. 2008.
16. O. Goldreich, B. Pfitzmann, and R. L. Rivest. Self-delegation with controlled propagation - or - what if you lose your laptop. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 153–168. Springer, Aug. 1998.
17. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In *CRYPTO 2013*, volume 8043 of *LNCS*, pages 536–553. Springer, 2013.
18. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *45th ACM STOC*, pages 555–564. ACM Press, 2013.
19. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
20. G. Hanaoka and J. Weng. Generic constructions of parallel key-insulated encryption. In *SCN'10*, volume 6280, pages 36–53, 2010.
21. Y. H. Hwang, D. H. Yum, and P. J. Lee. Timed-release encryption with pre-open capability and its application to certified e-mail system. In *ISC 2005*, volume 3650 of *LNCS*, pages 344–358. Springer, 2005.
22. M. Klonowski, P. Kubiak, and M. Kutyłowski. Practical deniable encryption. In *SOFSEM 2008: Theory and Practice of Computer Science*, volume 5805 of *LNCS*, pages 599–609. Springer, 2008.
23. B. Libert, J.-J. Quisquater, and M. Yung. Parallel key-insulated public key encryption without random oracles. In T. Okamoto and X. Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 298–314. Springer, Apr. 2007.
24. T. May. Time-release crypto. Manuscript, 1993.
25. K. G. Paterson and E. A. Quaglia. Time-specific encryption. In *SCN'10*, volume 6280, pages 1–16, 2010.
26. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, May 2005.
27. A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

A Proof of Theorem 1

Theorem 1. *Let E be an IND-CPA secure RCD-PKE, then G-APO based on E is OT-IND-CPA secure in the random oracle model.*

Lemma 1. *Let F be a hash function. We define the following experiment:*

$\mathbf{Exp}^{\mathbf{exp0}}_{\mathcal{A}}(k)$*:*
$q \leftarrow \mathcal{A}_0(1^k)$
$\forall\ i \in \{0, 1, 2, ..., q\}$:
$\quad \sigma_i \xleftarrow{\$} \{0,1\}^k$
$(n, \{\theta_i\}_{0<i\leq n}) \leftarrow \mathcal{A}_1(\mathsf{state}, \{\sigma_i \oplus \mathsf{F}(\sigma_{i-1})\}_{0<i\leq q})$
return 1 *if* $\exists\ i \leq n$ *such that* $\sigma_q = \theta_i$

Then $Pr[Exp^{exp0}_{\mathcal{A}}(k) = 1] \leq n/2^k$ *in the random oracle model.*

Proof (Lemma 1). We perform a proof by recurrence.

- Suppose that $q = 1$, the adversary must find σ_1 from $\sigma_1 \oplus \mathsf{F}(\sigma_0)$. Without knowledge of the value σ_0, the value $\mathsf{F}(\sigma_0)$ seems to be a random value from the adversary point of view. $\mathsf{F}(\sigma_0)$ is a one time pad used to hide σ_q, and $\mathcal{A}$ has no better solution that picks a random value in $\{0,1\}^k$. The adversary tries n values, so the probability that he wins the experiment is $n/2^k$.
- Assume that $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{exp0}}_{\mathcal{A}}(k) = 1] \leq n/2^k$ is true when $q \leftarrow \mathcal{A}_0(1^k)$. If adversary chooses the value $q + 1$ during the first phase, the challenger gives it the set $\{\sigma_i \oplus \mathsf{F}(\sigma_{i-1})\}_{0<i\leq q}$ plus the value $\sigma_{q+1} \oplus \mathsf{F}(\sigma_q)$. We know that $\{\sigma_i \oplus \mathsf{F}(\sigma_{i-1})\}_{0<i\leq q}$ gives no information on the value σ_q, so $\mathsf{F}(\sigma_q)$ is a one time pad for the value σ_{q+1}. Without information about σ_{q+1}, the adversary has no better choice that to try a random value to win the experiment and $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{exp0}}_{\mathcal{A}}(k) = 1] \leq n/2^k$. We conclude that for all polynomial $q \in \mathbb{N}$, $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{exp0}}_{\mathcal{A}}(k) = 1] \leq n/2^k$.

□

Lemma 2. *Let F be a hash function. We define the following experiment:*

$\mathbf{Exp}^{\mathbf{exp1}}_{\mathcal{A}}(k)$*:*
$q \leftarrow \mathcal{A}_0(1^k)$
$\forall\ i \in \{0, 1, 2, ..., q\}$:
$\quad \sigma_i \xleftarrow{\$} \{0,1\}^k$
$(n, \{\theta_i\}_{0<i\leq n}) \leftarrow \mathcal{A}_1(\mathsf{state}, \{\sigma_{i-1} \oplus \mathsf{F}(\sigma_i)\}_{0<i\leq q})$
return 1 *if* $\exists\ i \leq n$ *such that* $\sigma_0 = \theta_i$

Then $Pr[Exp^{exp0}_{\mathcal{A}}(k) = 1] \leq n/2^k$ *in the random oracle model.*

Proof (Lemma 2). The proof is done by recurrence.

– Suppose that $q = 1$, the adversary must find σ_0 from $\sigma_0 \oplus \mathsf{F}(\sigma_1)$. Without knowledge of the value σ_1, the value $\mathsf{F}(\sigma_1)$ seems to be a random value from the adversary point of view. $\mathsf{F}(\sigma_1)$ is a one time pad used to hide σ_0, and $\mathcal{A}$ has no better solution that picks a random value in $\{0,1\}^k$. The adversary tries n values, so the probability that he wins the experiment is $n/2^k$.
– Assume that $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{exp1}}_{\mathcal{A}}(k) = 1] \leq n/2^k$ is true when $q \leftarrow \mathcal{A}_0(1^k)$. If the adversary chooses the value $q+1$ during the first phase, the challenger gives it the set $\{\sigma_{i-1} \oplus \mathsf{F}(\sigma_i)\}_{0<i\leq q}$ plus the value $\sigma_q \oplus \mathsf{F}(\sigma_{q+1})$.Without information about σ_{q+1}, $\sigma_q \oplus \mathsf{F}(\sigma_{q+1})$ seems to be a random value and is useless for $\mathcal{A}$. The adversary has no better choice that to try a random value to win the experiment and $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{exp1}}_{\mathcal{A}}(k) = 1] \leq n/2^k$. We conclude that for all $q \in \mathbb{N}$, $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{exp1}}_{\mathcal{A}}(k) = 1] \leq n/2^k$.

□

Lemma 3. *Let* $\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}_0}_{\Pi,\mathcal{A}}(k)$ *be the same experiment that* $\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$ *except that* $\mathcal{A}_1$ *does not access to the oracle* $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$*. Then* $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}_0}_{\mathsf{G\text{-}APO}}(k)$ *is negligible when* E *is* IND-CPA*.*

Proof (Lemma 3). We assume there exists a polynomial time adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}_0}_{\mathsf{G\text{-}APO},\mathcal{A}}(k)$ is non negligible. We show then how to construct an adversary $\mathcal{B}$ with a similar running time such that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k)$ is also non negligible.

Description of the adversary $\mathcal{B}$:

Phase 1: $\mathcal{B}$ receives the key pk_* and sends it to $\mathcal{A}$. It picks three random values K, $\widehat{\sigma}_0$ and $\widetilde{\sigma}_0$ from $\{0,1\}^k$ and initializes two counters $n = 1$ and $l = 1$. It computes $\mathsf{st}_* = (K||\widehat{\sigma}_0||\widetilde{\sigma}_0)$ and creates three empty lists $\mathsf{F}_{\text{LIST}}, \mathsf{G}_{\text{LIST}}$ and H_{LIST}. It then simulates the oracles as follows:

- $\mathsf{F}(.)$**:** It takes an input value f. If there exists $(f', F) \in \mathsf{F}_{\text{LIST}}$ such that $f = f'$ then $\mathcal{B}$ returns F. Else it picks a random value F, adds (f, F) to F_{LIST} and returns F.
- $\mathsf{G}(.)$**:** It takes an input value g. If there exists $(g', G) \in \mathsf{G}_{\text{LIST}}$ such that $g = g'$ then $\mathcal{B}$ returns G. Else it picks a random value G, adds (g, G) to G_{LIST} and returns G.
- $\mathsf{H}(.)$**:** It takes an input value h. If there exists $(h', G) \in \mathsf{H}_{\text{LIST}}$ such that $h = h'$ then $\mathcal{B}$ returns H. Else it picks a random value H, adds (h, H) to H_{LIST} and returns H.
- $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{enc}}$**:** Using the input (pk, m) and the oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathsf{H}(.)$, $\mathcal{B}$ runs the algorithm $\mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}}(m)$ to compute C_l. It increments l and n and returns C_l.
- $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$**:** It returns $\mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}}(C'_i, C'_j)$ using the input $(\mathsf{pko}, C'_i, C'_j)$ and the oracle $\mathsf{H}(.)$.

Finally, $\mathcal{A}$ returns (m_0, m_1).

Phase 2: $\mathcal{B}$ picks $\widehat{\sigma}_n, \widetilde{\sigma}_n \xleftarrow{\$} \{0,1\}^k$ and $\widehat{m} \xleftarrow{\$} \{0,1\}^{|m_0|}$. It then computes $\widetilde{m}_0 = m_0 \oplus \widehat{m}$ and $\widetilde{m}_1 = m_1 \oplus \widehat{m}$. Using $\mathsf{st}_* = (K||\widehat{\sigma}_{(n-1)}||\widetilde{\sigma}_{(n-1)})$, it

sets $M_0 = \widetilde{m}_0||(\widetilde{\sigma}_{(n-1)} \oplus \mathsf{F}(\widetilde{\sigma}_n))$ and $M_1 = \widetilde{m}_1||(\widetilde{\sigma}_{(n-1)} \oplus \mathsf{F}(\widetilde{\sigma}_n))$. It computes $\widehat{C}_n = \mathsf{Enc}_{\mathsf{pk}_*}(\widehat{m}||(\widehat{\sigma}_n \oplus \mathsf{F}(\widehat{\sigma}_{(n-1)})); \mathsf{G}(\widehat{\sigma}_{(n-1)}))$ and sends (M_0, M_1) to the challenger. $\mathcal{B}$ receives the challenge $\widetilde{C}_n$. Finally, it computes $D_n = (\widehat{\sigma}_{(n-1)}||\widetilde{\sigma}_n) \oplus \mathsf{H}(K||\widehat{C}_n||\widetilde{C}_n)$ and sends $C_n = (\widehat{C}_n||\widetilde{C}_n||D_n)$ to $\mathcal{A}$. It increments l.
It then simulates the oracles as follows:

$\mathsf{F}(.)$**:** It is as in the first phase.
$\mathsf{G}(.)$**:** If input g is $\widetilde{\sigma}_n$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.
$\mathsf{H}(.)$**:** If $\exists\ (\alpha, \beta)$ such that $h = (K||\alpha||\beta)$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.
$\mathcal{O}^{\mathsf{CPA}}_{\mathsf{enc}}$**:** It is as in the first phase excepts that it does not increment n.

Finally, $\mathcal{A}$ returns b and $\mathcal{B}$ returns the same b.

Analyze: While $\mathcal{B}$ does not abort the experiment, it is perfectly simulated for $\mathcal{A}$. In this case, if $\mathcal{A}$ wins his experiment then $\mathcal{B}$ also wins his experiment. We claims that $\mathsf{Pr}[\mathcal{B}$ aborts] is negligible: Let E_1 and E_2 be two events such that:

- E_1 = "$\mathcal{B}$ *aborts during oracle* $\mathsf{H}(.)$ *simulation*"
- E_2 = "$\mathcal{B}$ *aborts during oracle* $\mathsf{G}(.)$ *simulation*"

We remark that $\mathcal{A}$ has no knowledge about the value K. Let λ_{H} be the number of queries to $\mathsf{H}(.)$ asked by $\mathcal{A}$, then $\mathsf{Pr}[E_1] \leq \lambda_{\mathsf{H}}/2^k$ is negligible.

Note that $E_2 \Rightarrow \neg E_1$ since if $\mathcal{B}$ aborts on $H(.)$ he cannot abort on $G(.)$. $\neg E_1$ implies that all D_i for all $i < l$ seems to be randoms elements of the uniform distribution on $\{0,1\}^k$. In this case, informations about $\widetilde{\sigma}_n$, that $\mathcal{A}$ has, are the following values $\{\widetilde{\sigma}_{i-1} \oplus \mathsf{F}(\widetilde{\sigma}_i)\}_{n<i\leq l}$. However, Lemma 1 claims that these elements give no information about $\widetilde{\sigma}_n$ in the random oracle model. We can deduce that $\mathsf{Pr}[E_2] \leq \lambda_{\mathsf{G}}/2^k$ for λ_{G} the number of queries to $\mathsf{G}(.)$ asked by $\mathcal{A}$

We note E_0 = "$\mathcal{B}$ does not abort" $= \neg E_1 \wedge \neg E_2$. From previous results, we have that $\mathsf{Pr}[E_0]$ is non negligible and $\mathsf{Pr}[\neg E_0]$ is non negligible. Finally, as $\mathsf{Pr}[\mathcal{B}\text{ wins}|E_1] = \mathsf{Pr}[\mathcal{B}\text{ wins}|E_2] = 0$,

$$\begin{aligned}\mathsf{Pr}[\mathcal{B}\text{ wins}] &= \mathsf{Pr}[\mathcal{B}\text{ wins}|E_0].\mathsf{Pr}[E_0] \\ &= \mathsf{Pr}[\mathcal{A}\text{ wins}].\mathsf{Pr}[E_0]\end{aligned}$$

On the other hand:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k) &= \left|(\mathsf{Pr}[\mathcal{A}\text{ wins}] - \frac{1}{2}).\mathsf{Pr}[E_0] - \frac{1}{2}\mathsf{Pr}[\neg E_0]\right| \\ &\geq \left|\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}_0}_{\mathsf{G\text{-}APO}}(k).\mathsf{Pr}[E_0] - \frac{1}{2}\mathsf{Pr}[\neg E_0]\right|\end{aligned}$$

We can conclude that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k)$ is non negligible, which is a contradiction since E is IND-CPA. □

Lemma 4. *Let* $\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA_1}}_{\Pi,\mathcal{A}}(k)$ *be the same experiment that* $\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$ *except that* $\mathcal{A}_0$ *does not access to the oracle* $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$*. Then* $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA_1}}_{\mathsf{G\text{-}APO}}(k)$ *is negligible when* E *is* IND-CPA*.*

Proof (Lemma 4). We assume there exists a polynomial time adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA_1}}_{\mathsf{G\text{-}APO},\mathcal{A}}(k)$ is non negligible. We show then how to construct an adversary $\mathcal{B}$ with a similar running time such that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k)$ is also non negligible.

$\mathcal{B}$ description:

Phase 1: $\mathcal{B}$ receives the key pk_* and sends it to $\mathcal{A}$. It picks three random values K, $\widehat{\sigma}_0$ and $\widetilde{\sigma}_0$ from $\{0,1\}^k$ and initializes two counters $n = 1$ and $l = 1$. It computes $\mathsf{st}_* = (K||\widehat{\sigma}_0||\widetilde{\sigma}_0)$ and creates three empty lists $\mathsf{F}_{\text{LIST}}, \mathsf{G}_{\text{LIST}}$ and H_{LIST}. It then simulates the oracles as follows:

- $\mathsf{F}(.)$**:** It takes an input value f. If there exists $(f', F) \in \mathsf{F}_{\text{LIST}}$ such that $f = f'$ then $\mathcal{B}$ returns F. Else it picks a random value F, adds (f, F) to F_{LIST} and returns F.
- $\mathsf{G}(.)$**:** It takes an input value g. If there exists $(g', G) \in \mathsf{G}_{\text{LIST}}$ such that $g = g'$ then $\mathcal{B}$ returns G. Else it picks a random value G, adds (g, G) to G_{LIST} and returns G.
- $\mathsf{H}(.)$**:** It takes an input value h. If there exists $(h', G) \in \mathsf{H}_{\text{LIST}}$ such that $h = h'$ then $\mathcal{B}$ returns H. Else it picks a random value H, adds (h, H) to H_{LIST} and returns H.
- $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{enc}}$**:** Using the input (pk, m) and the oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathsf{H}(.)$, $\mathcal{B}$ runs the algorithm $\mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}}(m)$ to compute C_l. It increments l and n and returns C_l.

Finally, $\mathcal{A}$ returns (m_0, m_1).

Phase 2: $\mathcal{B}$ picks $\widehat{\sigma}_n, \widetilde{\sigma}_n \xleftarrow{\$} \{0,1\}^k$ and $\widetilde{m} \xleftarrow{\$} \{0,1\}^{|m_0|}$. It then computes $\widehat{m}_0 = m_0 \oplus \widetilde{m}$ and $\widehat{m}_1 = m_1 \oplus \widetilde{m}$. Using $\mathsf{st}_* = (K||\widehat{\sigma}_{(n-1)}||\widetilde{\sigma}_{(n-1)})$, it sets $M_0 = \widehat{m}_0||(\widehat{\sigma}_n \oplus \mathsf{F}(\widehat{\sigma}_{(n-1)}))$ and $M_1 = \widehat{m}_1||(\widehat{\sigma}_n \oplus \mathsf{F}(\widehat{\sigma}_{(n-1)}))$. It computes $\widetilde{C}_n = \mathsf{Enc}_{\mathsf{pk}_*}(\widetilde{m}||(\widetilde{\sigma}_{(n-1)} \oplus \mathsf{F}(\widetilde{\sigma}_n)); \mathsf{G}(\widetilde{\sigma}_n))$ and sends (M_0, M_1) to the challenger. $\mathcal{B}$ receives the challenge $\widehat{C}_n$. Finally, it computes $D_n = (\widehat{\sigma}_{(n-1)}||\widetilde{\sigma}_n) \oplus \mathsf{H}(K||\widehat{C}_n||\widetilde{C}_n)$ and sends $C_n = (\widehat{C}_n||\widetilde{C}_n||D_n)$ to $\mathcal{A}$. It increments l.
It then simulates the oracles as follows:

- $\mathsf{F}(.)$**:** It is as in the first phase.
- $\mathsf{G}(.)$**:** If input g is $\widehat{\sigma}_{(n-1)}$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.
- $\mathsf{H}(.)$**:** If $\exists\ (\alpha, \beta)$ such that $h = (K||\alpha||\beta)$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.
- $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{enc}}$**:** It is as in the first phase excepts that it does not increment n.
- $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$**:** It receives $(\mathsf{pko}, C'_a, C'_b)$ If it exists $C_i = C'_a$ and $C_j = C'_b$ such that $i \leq n \leq j$ then the oracle rejects the query. Else, if $C'_a = C_n$ or $C'_b = C_n$, it rejects the query. Else it returns $\mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}}(C'_a, C'_b)$.

Finally, $\mathcal{A}$ returns b and $\mathcal{B}$ returns the same b.

Analyze: While $\mathcal{B}$ does not abort the experiment, it is perfectly simulated for $\mathcal{A}$. In this case, if $\mathcal{A}$ wins his experiment then $\mathcal{B}$ also wins his experiment. We claims that $\mathsf{Pr}[\mathcal{B}$ aborts$]$ is negligible: Let E_1 and E_2 be two events such that:

- E_1 = "$\mathcal{B}$ *aborts during oracle* $\mathsf{H}(.)$ *simulation*"
- E_2 = "$\mathcal{B}$ *aborts during oracle* $\mathsf{G}(.)$ *simulation*"

We remark that $\mathcal{A}$ has no knowledge about the value K. Let λ_{H} be the number of queries to $\mathsf{H}(.)$ asked by $\mathcal{A}$, then $\mathsf{Pr}[E_1] \leq \lambda_{\mathsf{H}}/2^k$ is negligible.

Note that $E_2 \Rightarrow \neg E_1$ since if $\mathcal{B}$ aborts on $H(.)$ he cannot abort on $G(.)$. $\neg E_1$ implies that all D_i for all $i < l$ seems to be randoms elements of the uniform distribution on $\{0,1\}^k$. In this case, informations about $\widehat{\sigma}_{(n-1)}$, that $\mathcal{A}$ has, are the following values $\{\widehat{\sigma}_i \oplus \mathsf{F}(\widehat{\sigma}_{i-1})\}_{1 \leq i \leq n}$. However, Lemma 2 claims that these elements give no information about $\widehat{\sigma}_{(n-1)}$ in the random oracle model. We can deduce that $\mathsf{Pr}[E_2] \leq \lambda_{\mathsf{G}}/2^k$ for λ_{G} the number of queries to $\mathsf{G}(.)$ asked by $\mathcal{A}$

We note E_0 = "$\mathcal{B}$ does not abort" $= \neg E_1 \wedge \neg E_2$. From previous results, we have that $\mathsf{Pr}[E_0]$ is non negligible and $\mathsf{Pr}[\neg E_0]$ is non negligible. Finally, as $\mathsf{Pr}[\mathcal{B} \text{ wins}|E_1] = \mathsf{Pr}[\mathcal{B} \text{ wins}|E_2] = 0$,

$$\begin{aligned}\mathsf{Pr}[\mathcal{B} \text{ wins}] &= \mathsf{Pr}[\mathcal{B} \text{ wins}|E_0].\mathsf{Pr}[E_0] \\ &= \mathsf{Pr}[\mathcal{A} \text{ wins}].\mathsf{Pr}[E_0]\end{aligned}$$

On the other hand:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k) &= \left|(\mathsf{Pr}[\mathcal{A} \text{ wins}] - \frac{1}{2}).\mathsf{Pr}[E_0] - \frac{1}{2}\mathsf{Pr}[\neg E_0]\right| \\ &\geq \left|\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}_1}_{\mathsf{G\text{-}APO}}(k).\mathsf{Pr}[E_0] - \frac{1}{2}\mathsf{Pr}[\neg E_0]\right|\end{aligned}$$

We can conclude that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k)$ is non negligible, which is a contradiction since E is IND-CPA. □

Theorem 1. *Let E be an IND-CPA secure RCD-PKE, then G-APO based on E is OT-IND-CPA secure in the random oracle model.*

Proof (Theorem 1).

This theorem is a direct consequence of Lemma 3 and 4: Let $\mathcal{A}$ be an adversary against APO-PKE. During the experiment $\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\mathsf{APO\text{-}PKE},\mathcal{A}}(k)$, if $\mathcal{A}$ chooses to call oracle $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$ in the first phase then his advantage is negligible. However, if $\mathcal{A}$ chooses to call oracle $\mathcal{O}^{\mathsf{CPA}}_{\mathsf{ext}}$ in the second phase then his advantage is also negligible. Thus, we can conclude that $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\mathsf{APO\text{-}PKE},\mathcal{A}}(k)$ is negligible.

□

B Proof of Theorem 2

Theorem 2. *Let E be a PKE that is RCD, then G-APO using E is IND-CSPA secure in the random oracle model.*

Consider the following experiment:

$\mathbf{Exp}^{n-\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$:
$b \xleftarrow{\$} \{0,1\}$
For $i = 1, \cdots, n$ do $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^k)$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{LR}_b(\cdot,\cdot);\sigma),\cdots,\mathsf{Enc}_{\mathsf{pk}_n}(\mathsf{LR}_b(\cdot,\cdot);\sigma)}(\mathsf{st}, \mathsf{pk}_1, \cdots, \mathsf{pk}_n)$
return $(b = b')$

Were $\mathsf{LR}_b(\cdot,\cdot)$ returns m_b from (m_0, m_1) and $\mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{LR}_b(\cdot,\cdot);\sigma)$ is an oracle that returns $\mathsf{Enc}_{\mathsf{pk}_i}(m_b;\sigma)$ from (m_0, m_1). Let $\mathsf{Adv}^{n-\mathsf{IND\text{-}CPA}}_{\Pi}(k)$ be the advantage function associated with the previous experiment. In [2], authors prove that if Π is $\mathsf{IND\text{-}CPA}$ then $\mathsf{Adv}^{n-\mathsf{IND\text{-}CPA}}_{\Pi}(k)$ is negligible for any poly-time $\mathcal{A}$ and polynomial n. We use this result in the proof of Theorem 2.

Lemma 5. *Let $Exp^{IND\text{-}CSPA'_\phi}_{\Pi,\mathcal{A}}(k)$ be the same experiment that $Exp^{IND\text{-}CSPA_\phi}_{\Pi,\mathcal{A}}(k)$ except that in guess phase, $\mathcal{A}$ returns σ and wins the experiment if it exists $K_{i\to j}$ generated by oracle $\mathcal{O}^{CSPA}_{ext}$ such that $\sigma_1 = \sigma$ for $Dec_{sk_*}(K_{i\to j}) = (\sigma_0||\sigma_1)$. Let $\mathsf{Adv}^{IND\text{-}CSPA'_\phi}_{\Pi}(k)$ be the advantage on $Exp^{IND\text{-}CSPA'_\phi}_{\Pi,\mathcal{A}}(k)$ of $\mathcal{A}$ defined by*

$$\mathsf{Adv}^{IND\text{-}CSPA'_\phi}_{\Pi,\mathcal{A}}(k) = Pr[Exp^{IND\text{-}CSPA'_\phi}_{\Pi,\mathcal{A}}(k) = 1]$$

Then $\mathsf{Adv}^{IND\text{-}CSPA'_\phi}_{\Pi}(k)$ is negligible when $\mathsf{Adv}^{1-IND\text{-}CPA}_{\Pi}(k)$ is negligible.

Proof (Lemma 5). We assume there exists a polynomial time adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{IND\text{-}CSPA}'_\phi}_{\Pi,\mathcal{A}}(k)$ is non negligible. We show then how to construct an adversary $\mathcal{B}$ with a similar running time such that $\mathsf{Adv}^{1-\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$ is also non negligible.

Description of the adversary $\mathcal{B}$:

Phase 1: $\mathcal{B}$ receives the key pko_*. It picks four random values $K, \widetilde{\sigma}_0, \widetilde{\sigma}_0$ and $\widehat{\sigma}$ from $\{0,1\}^k$ and initializes three lists $\widehat{\Sigma}(0) = \widehat{\sigma}$, $\widetilde{\Sigma}_0(0) = \widetilde{\sigma}_0$ and $\widetilde{\Sigma}_1(0) = \widetilde{\sigma}_1$. It initializes counters $n = 0$ and $l = 1$. It creates three empty lists $\mathsf{F}_{\text{LIST}}, \mathsf{G}_{\text{LIST}}$ and H_{LIST}. It sends pk_* to $\mathcal{A}$ and then simulates the oracles as follows:

- $\mathsf{F}(.)$**:** It takes as input value f. If there exists $(f', F) \in \mathsf{F}_{\text{LIST}}$ such that $f = f'$ then $\mathcal{B}$ returns F. Else it picks a random value F, adds (f, F) to F_{LIST} and returns F.
- $\mathsf{G}(.)$**:** It takes as input value g. If there exists $(g', G) \in \mathsf{G}_{\text{LIST}}$ such that $g = g'$ then $\mathcal{B}$ returns G. Else it picks a random value G, adds (g, G) to G_{LIST} and returns G.
- $\mathsf{H}(.)$**:** It takes as input value h. If there exists $(h', G) \in \mathsf{H}_{\text{LIST}}$ such that $h = h'$ then $\mathcal{B}$ returns H. Else it picks a random value H, adds (h, H) to H_{LIST} and returns H.
- $\mathcal{O}^{\mathbf{CSPA}}_{\mathbf{gen}}$**:** $\mathcal{B}$ generates $(\overline{\mathsf{pk}}_i, \overline{\mathsf{sk}}_i)$ from $\mathsf{Gen}(k)$ and sends value $\overline{\mathsf{pk}}_i$ on the i^{th} call to this oracle.
- $\mathcal{O}^{\mathbf{CSPA}}_{\mathbf{enc}}$**:** Using the input (pk, m), the state st_* and oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathsf{H}(.)$, $\mathcal{B}$ picks a random coin g from $\mathcal{R}$, a random message $\widehat{m}$ from $\{0,1\}^{|m|}$, random elements $f, \widetilde{\sigma}_0, \widetilde{\sigma}_0$ and $\widehat{\sigma}$ from $\{0,1\}^k$ and h from $\{0,1\}^{2k}$, and

computes $\widehat{C}_l = \mathsf{Enc}_{\mathsf{pk}}(\widehat{m}||(\widehat{\sigma} \oplus \mathsf{F}(\widehat{\Sigma}(l))); \mathsf{G}(\widehat{\Sigma}(l)))$, $\widetilde{m} = m \oplus \widehat{m}$, $\widetilde{C}_l = \mathsf{Enc}_{\mathsf{pk}}(\widetilde{m}||f; g)$, $D_l = h$ and returns $C_l = (\widehat{C}_l, \widetilde{C}_l, D_l)$. It increments l and n and initializes $\widehat{\Sigma}(l) = \widehat{\sigma}$, $\widetilde{\Sigma}_0(l) = \widetilde{\sigma}_0$ and $\widetilde{\Sigma}_1(l) = \widetilde{\sigma}_1$.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$**:** Using the input $(C'_a||C'_b)$ such that $C'_a = (\widehat{C}'_a||\widetilde{C}'_a||D'_a)$ and $C'_b = (\widehat{C}'_b||\widetilde{C}'_b||D'_b)$:

- If it exists i such that $C_i = C'_a$, $\mathcal{B}$ computes $\alpha = \widehat{\Sigma}(i-1)$.
- Else, it computes $(\alpha||\alpha') = D'_a \oplus \mathsf{H}(K, \widehat{C}'_a, \widetilde{C}'_a)$.
- If it exists j such that $C_j = C'_b$, $\mathcal{B}$ computes $\beta_0 = \widetilde{\Sigma}_0(j)$ and $\beta_1 = \widetilde{\Sigma}_1(j)$ and sends $(\alpha||\beta_0)$ and $(\alpha||\beta_1)$ to the oracle $\mathsf{Enc}_{\mathsf{pko}_*}(\mathsf{LR}_b(\cdot,\cdot); r)$ and receives $K^{\mathsf{pko}_*}_{a\to b}$. $\mathcal{B}$ returns it.
- Else, it computes $(\beta||\beta') = D'_b \oplus \mathsf{H}(K||\widehat{C}'_b||\widetilde{C}'_b)$, $K^{\mathsf{pko}_*}_{a\to b} = \mathsf{Enc}_{\mathsf{pko}_*}((\alpha||\beta); r)$ and returns $K^{\mathsf{pko}_*}_{a\to b}$.

Finally, $\mathcal{A}$ returns $(q, \{m^0_x\}_{n+1\le x\le n+q}, \{m^1_x\}_{n+1\le x\le n+q}, \{\mathsf{pk}_x\}_{n+1\le x\le n+q})$.

Phase 2: For all x such that $n+1 \le x \le x+q$, $\mathcal{B}$ performs the following computations:

- If it exists $y \in \{1, 2, \cdots \phi\}$ such that $\mathsf{pk}_x = \overline{\mathsf{pk}_y}$ then uses $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$ on m^b_x to compute C_x.
- Else it uses $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$ on m^d_x to compute C_x.

$\mathcal{B}$ then computes $K^{\mathsf{pk}_*}_{(n+1)\to(n+q)}$ using $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$ on C_{n+1} and C_{n+q}, and sends $\{C_x\}_{n+1\le x\le x+q}$ and $K^{\mathsf{pk}_*}_{(n+1)\to(n+q)}$ to $\mathcal{A}$.

It then simulates the oracles as follows:

$\mathsf{F}(.)$**:** If $\mathcal{A}$ sends σ such that $\sigma = \Sigma_o(x)$ for $0 \le x \le l$, $\mathcal{B}$ aborts the experiment and returns the bit o. Else, it is as in the first phase.

$\mathsf{G}(.)$**:** If $\mathcal{A}$ sends σ such that $\sigma = \Sigma_o(x)$ for $0 \le x \le l$, $\mathcal{B}$ aborts the experiment and returns the bit o. Else, it is as in the first phase.

$\mathsf{H}(.)$**:** It takes as input value h. If $\exists\ C_x = (\widehat{C}_x||\widetilde{C}_x||D_x)$ for $1 \le x \le l$ such that $h = (K||\widehat{C}_x||\widetilde{C}_x)$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{gen}}$**,**$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$ **and** $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$**:** $\mathcal{B}$ process as in the first phase.

Finally, $\mathcal{A}$ returns σ. If it exists x such that $\sigma = \Sigma_o(x)$ for $0 \le x \le l$, $\mathcal{B}$ returns the bit o. Else, it returns a random bit.

Analyze: Let E_1 and E_2 be two events such that:

- E_1 = "$\mathcal{B}$ *aborts during oracle* $\mathsf{H}(.)$ *simulation*"
- E_2 = "$\mathcal{B}$ *aborts during oracle* $\mathsf{F}(.)$ *or* $\mathsf{G}(.)$ *simulation*"

During simulation of $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$, oracle $H(.)$ is never called on an input h such that $\exists\ C_x = (\widehat{C}_x||\widetilde{C}_x||D_x)$ for $1 \le x \le l$ and $h = (K||\widehat{C}_x||\widetilde{C}_x)$. We remark that $\mathcal{A}$ has no knowledge about the value K. Let λ_{H} be the number of query to $\mathsf{H}(.)$ asked by $\mathcal{A}$, then $\mathsf{Pr}[E_1] \le \lambda_{\mathsf{H}}/2^k$ is negligible.

We first prove that $\mathsf{Pr}[$" $\mathcal{B}$ loses "$|E_2]$ is negligible. Let λ_{G} (resp. λ_{F}) be the number of query to $\mathsf{G}(.)$ (resp. $\mathsf{F}(.)$) asked by $\mathcal{A}$. Since $\mathcal{A}$ has no information

about elements of $\widetilde{\Sigma}_{1-b}$, the probability that $\mathcal{A}$ sends σ such that $\sigma = \Sigma_{1-b}(x)$ for $0 \leq x \leq l$ is inferior to $(\lambda_{\mathsf{G}} + \lambda_{\mathsf{H}})/2^k$, that is negligible. On the other hand, we can remark that $\mathsf{Pr}[\text{" } \mathcal{B} \text{ wins"}|E_2]$ is non negligible.

We note $E_0 = \neg E_1 \wedge \neg E_2$. We remark that $\mathsf{Pr}[\mathcal{B} \text{ wins}|E_1] = 0$. If $\mathsf{Pr}[\neg E_2]$ is negligible, then:

$$\begin{aligned}
&\mathsf{Adv}_{E,\mathcal{B}}^{1-\mathsf{IND\text{-}CPA}}(k) \\
&= \left| \mathsf{Pr}[\mathcal{B} \text{ wins}] - \frac{1}{2} \right| \\
&\geq \left| \mathsf{Pr}[\mathcal{B} \text{ wins}|E_2].\mathsf{Pr}[E_2] - \frac{1}{2} + \mathsf{Pr}[\mathcal{B} \text{ wins}|E_0].\mathsf{Pr}[\neg E_1].\mathsf{Pr}[\neg E_2] \right| \\
&\geq \left| \left(\frac{1}{2} - \frac{\lambda_F + \lambda_G}{2^k} \right) \cdot \mathsf{Pr}[E_2] + \mathsf{Pr}[\mathcal{B} \text{ wins}|E_0].\mathsf{Pr}[\neg E_1].\mathsf{Pr}[\neg E_2] \right|
\end{aligned}$$

Then $\mathsf{Adv}_{E,\mathcal{B}}^{1-\mathsf{IND\text{-}CPA}}(k)$ is non negligible. On the other hand, if $\mathsf{Pr}[E_2]$ is negligible, then:

$$\begin{aligned}
&\mathsf{Adv}_{E,\mathcal{B}}^{1-\mathsf{IND\text{-}CPA}}(k) \\
&= \left| \mathsf{Pr}[\mathcal{B} \text{ wins}] - \frac{1}{2} \right| \\
&\geq \left| \mathsf{Pr}[\mathcal{B} \text{ wins}|E_2].\mathsf{Pr}[E_2] + \mathsf{Pr}[\mathcal{B} \text{ wins}|E_0].\mathsf{Pr}[\neg E_0] - \frac{1}{2} \right| \\
&\geq \left| \mathsf{Pr}[\mathcal{B} \text{ wins}|E_2].\mathsf{Pr}[E_2] + \frac{1}{2}\mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{IND\text{-}CSPA}'_{\phi}}(k) \cdot \mathsf{Pr}[\neg E_1] \cdot \mathsf{Pr}[\neg E_2] - \frac{1}{2}\mathsf{Pr}[E_1 \vee E_2] \right|
\end{aligned}$$

Then $\mathsf{Adv}_{E,\mathcal{B}}^{1-\mathsf{IND\text{-}CPA}}(k)$ is non negligible. Finally, $\mathsf{Adv}_{E,\mathcal{B}}^{1-\mathsf{IND\text{-}CPA}}(k)$ is non negligible in any cases. □

Theorem 2. *Let E be a PKE that is RCD, then G-APO using E is IND-CSPA secure in the random oracle model.*

Proof (Theorem 2).

We assume there exists a polynomial time adversary $\mathcal{A}$ such that $\mathsf{Adv}_{\mathsf{G\text{-}APO},\mathcal{A}}^{\mathsf{IND\text{-}CSPA}_{\phi}}(k)$ is non negligible. We show then how to construct an adversary $\mathcal{B}$ with a similar running time such that $\mathsf{Adv}_{E,\mathcal{B}}^{\phi-\mathsf{IND\text{-}CPA}}(k)$ is also non negligible.

Description of the adversary $\mathcal{B}$:

Phase 1: $\mathcal{B}$ receives the key set $\{\overline{\mathsf{pk}}_x\}_{1 \leq x \leq \phi}$ and generates $(\mathsf{pko}_*, \mathsf{sko}_*) \xleftarrow{\$} \mathsf{Gen}(1^k)$. It picks three random values $K, \widehat{\sigma}_0$ and $\widetilde{\sigma}_0$ from $\{0,1\}^k$ and initializes $\mathsf{st}_* = (K||\widehat{\sigma}_0||\widetilde{\sigma}_0)$. It initializes counters $n = 0$ and $l = 1$. It creates three empty lists $\mathsf{F}_{\text{LIST}}, \mathsf{G}_{\text{LIST}}$ and H_{LIST}. It sends pko_* to $\mathcal{A}$ and then simulates the oracles as follows:

$\mathsf{F}(.)$**:** It takes as input value f. If there exists $(f', F) \in \mathsf{F}_{\text{LIST}}$ such that $f = f'$ then $\mathcal{B}$ returns F. Else it picks a random value F, adds (f, F) to F_{LIST} and returns F.

$\mathsf{G}(.)$**:** It takes as input value g. If there exists $(g', G) \in \mathsf{G}_{\text{LIST}}$ such that $g = g'$ then $\mathcal{B}$ returns G. Else it picks a random value G, adds (g, G) to G_{LIST} and returns G.

$\mathsf{H}(.)$**:** It takes as input value h. If there exists $(h', G) \in \mathsf{H}_{\text{LIST}}$ such that $h = h'$ then $\mathcal{B}$ returns H. Else it picks a random value H, adds (h, H) to H_{LIST} and returns H.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{gen}}$**:** $\mathcal{B}$ sends value $\overline{\mathsf{pk}}_i$ on the i^{th} call to this oracle.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$**:** Using the input (pk, m), the state st_* and the oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathsf{H}(.)$, $\mathcal{B}$ computes $C_l = \mathsf{APOenc}^{\mathsf{st}_*}_{\mathsf{pk}}(m)$ and returns it. During the encryption, st_* is updated by $(K||\widehat{\sigma}_l||\widetilde{\sigma}_l)$. $\mathcal{B}$ increments n and l.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$**:** Using the input $(C'_a||C'_b)$, the oracle $\mathsf{H}(.)$ and st_*, $\mathcal{B}$ computes and sends $\mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}_*}(C'_a, C'_b)$.

Finally, $\mathcal{A}$ returns $(q, \{m^0_x\}_{n+1\leq x\leq n+q}, \{m^1_x\}_{n+1\leq x\leq n+q}, \{\mathsf{pk}_x\}_{n+1\leq x\leq n+q})$.

Phase 2: For all x such that $n+1 \leq x \leq x+q$, $\mathcal{B}$ performs the following computations:

- It picks $\widehat{m}_x \xleftarrow{\$} \{0,1\}^{|m^0_x|}$ and computes $\widetilde{m}^0_x = m^0_x \oplus \widehat{m}_x$ and $\widetilde{m}^1_x = m^1_x \oplus \widehat{m}_x$.
- It picks $\widehat{\sigma}_x, \widetilde{\sigma}_x \xleftarrow{\$} \{0,1\}^k$.
- If it exists $y \in \{1, 2, \cdots \phi\}$ such that $\mathsf{pk}_x = \overline{\mathsf{pk}}_y$ then it computes $\widehat{C}_x = \mathsf{Enc}_{\overline{\mathsf{pk}}_y}(\widehat{m}_x||(\widehat{\sigma}_x \oplus \mathsf{F}(\widehat{\sigma}_{x-1})); \mathsf{H}(\widehat{\sigma}_{x-1}))$ using $\mathsf{st}_* = (K||\widehat{\sigma}_{x-1}||\widetilde{\sigma}_{x-1})$. $\mathcal{B}$ sends $\widetilde{m}^0_x||(\widetilde{\sigma}_{x-1} \oplus \mathsf{F}(\widetilde{\sigma}_x))$ and $\widetilde{m}^1_x||(\widetilde{\sigma}_{x-1} \oplus \mathsf{F}(\widetilde{\sigma}_x))$ to oracle $\mathsf{Enc}_{\overline{\mathsf{pk}}_y}(\mathsf{LR}_b(\cdot,\cdot); r)$. It responds by the value $\widetilde{C}_x$. It computes $D_x = (\widehat{\sigma}_{x-1}||\widetilde{\sigma}_x) \oplus \mathsf{H}(K||\widehat{C}_x||\widetilde{C}_x)$. It computes $C_x = (\widehat{C}_x||\widetilde{C}_x||D_x)$ and updates $\mathsf{st}_* = (K||\widehat{\sigma}_x||\widetilde{\sigma}_x)$ and increments l.
- Else it uses $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$ on m^d_x to compute C_x.

$\mathcal{B}$ then computes $K^{\mathsf{pko}_*}_{(n+1)\to(n+q)} = \mathsf{APOext}^{\mathsf{st}_*}_{\mathsf{pko}_*}(C_{n+1}, C_{n+q})$, and sends $\{C_x\}_{n+1\leq x\leq x+q}$ and $K^{\mathsf{pko}_*}_{(n+1)\to(n+q)}$ to $\mathcal{A}$.
It then simulates the oracles as follows:

$\mathsf{F}(.)$**:** It is as in the first phase.

$\mathsf{G}(.)$**:** It takes as input value g. If $\exists\ \widetilde{\sigma}_x$ such that $n+1 \leq x \leq n+q$ and $\widetilde{\sigma}_x = g$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.

$\mathsf{H}(.)$**:** It takes as input value h. If $\exists\ (\alpha||\beta)$ such that $h = (K||\alpha||\beta)$ then $\mathcal{B}$ aborts the experiment. Else it is as in the first phase.

$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{gen}}$**,**$\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{enc}}$ **and** $\mathcal{O}^{\mathsf{CSPA}}_{\mathsf{ext}}$**:** $\mathcal{B}$ is as in the first phase.

Finally, $\mathcal{A}$ returns b and $\mathcal{B}$ returns the same b.

Analyze: While $\mathcal{B}$ does not abort the experiment, it is perfectly simulated for $\mathcal{A}$. In this case, if $\mathcal{A}$ wins his experiment then $\mathcal{B}$ also wins his experiment. We claims that $\mathsf{Pr}[\mathcal{B}$ aborts$]$ is negligible: Let E_1 and E_2 be two events such that:

- E_1 = *"$\mathcal{B}$ aborts during oracle* $\mathsf{H}(.)$ *simulation"*
- E_2 = *"$\mathcal{B}$ aborts during oracle* $\mathsf{G}(.)$ *simulation"*

We remark that $\mathcal{A}$ has no knowledge about the value K. Let λ_{H} be the number of queries to $\mathsf{H}(.)$ asked by $\mathcal{A}$, then $\mathsf{Pr}[E_1] \leq \lambda_{\mathsf{H}}/2^k$ is negligible.

Note that $E_2 \Rightarrow \neg E_1$ since if $\mathcal{B}$ aborts on $H(.)$ then it cannot abort on $G(.)$. $\neg E_1$ implies that all D_i for all $i < l$ seems to be randoms elements of the uniform distribution on $\{0,1\}^k$. Moreover, Lemma 1 claims that values $\{\widetilde{\sigma}_{i-1} \oplus \mathsf{F}(\widetilde{\sigma}_i)\}_{0<i\leq l}$ give no information about values $\{\widetilde{\sigma}_x\}_{0<x<l}$. Finally, since interval keys are the only one information that $\mathcal{A}$ possess about values $\{\widetilde{\sigma}_x\}_{0<x<l}$, finding σ such that $\{\widetilde{\sigma}_x\}_{0<x<l}$ is equivalent to win the experiment $\mathsf{Exp}^{\mathsf{IND\text{-}CSPA}'_{\phi}}_{\Pi,\mathcal{A}}(k)$. Thus, $\mathsf{Pr}[E_2] \leq \mathsf{Adv}^{\mathsf{IND\text{-}CSPA}'_{\phi}}_{\Pi,\mathcal{A}}(k)$ that is negligible (Lemma 5)

We note $E_0 =$ "$\mathcal{B}$ does not abort" $= \neg E_1 \wedge \neg E_2$. Finally, as $\mathsf{Pr}[\mathcal{B} \text{ wins}|E_1] = \mathsf{Pr}[\mathcal{B} \text{ wins}|E_2] = 0$,

$$\begin{aligned}\mathsf{Pr}[\mathcal{B} \text{ wins}] &= \mathsf{Pr}[\mathcal{B} \text{ wins}|E_0].\mathsf{Pr}[E_0] \\ &= \mathsf{Pr}[\mathcal{A} \text{ wins}].\mathsf{Pr}[E_0]\end{aligned}$$

On the other hand:

$$\begin{aligned}\mathsf{Adv}^{\phi-\mathsf{IND\text{-}CPA}}_{E,\mathcal{B}}(k) &= \left|(\mathsf{Pr}[\mathcal{A} \text{ wins}] - \frac{1}{2}).\mathsf{Pr}[E_0] - \frac{1}{2}\mathsf{Pr}[\neg E_0]\right| \\ &\geq \left|\mathsf{Adv}^{\mathsf{IND\text{-}CSPA}_{\phi}}_{\mathsf{G\text{-}APO}}(k).\mathsf{Pr}[E_0] - \frac{1}{2}\mathsf{Pr}[\neg E_0]\right|\end{aligned}$$

Since $\mathsf{Pr}[E_0]$ is non negligible, we can conclude that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{E_0,\mathcal{B}}(k)$ is non negligible, which is a contradiction since E is $\phi - \mathsf{IND\text{-}CPA}$. □

C Proof of Theorem 3

Theorem 3. *Let E be a RCD and VK PKE that is IND-CPA secure, then G-APO using this PKE satisfies the integrity property.*

Proof. Suppose that $\mathsf{Exp}^{\mathsf{Integrity}}_{\Pi,\mathcal{A}}(k) = 1$ then there exists a tuple

$$(N, \{C_x\}_{1\leq x\leq N}, \{\mathsf{pk}_x\}_{1\leq x\leq N}, l, \mathsf{sk}_l, i, j, K^{\mathsf{pk}_*}_{i\to j})$$

such that $\mathsf{Ver}(\mathsf{pk}_l, \mathsf{sk}_l) = 1$ and $m_l \neq \mathsf{APOdec}_{\mathsf{sk}_l}(C_l)$ for $m_l \in \{m_x\}_{i\leq x\leq j} = \mathsf{APOpen}_{\mathsf{sk}_*}(K^{\mathsf{pk}_*}_{i\to j}, \{C_x\}_{i\leq x\leq j}, \{\mathsf{pk}_x\}_{i\leq x\leq j})$.

We note $C_l = (\widehat{C}_l||\widetilde{C}_l||D_l)$. During APOpen execution, algorithm computes $(\widehat{m}_l||\widehat{\sigma}_l) = \mathsf{CDec}_{\widehat{R}}(\widehat{C}_l, \mathsf{pk}_l)$ and $(\widetilde{m}_l||\widetilde{\sigma}_l) = \mathsf{CDec}_{\widetilde{R}}(\widetilde{C}_l, \mathsf{pk}_l)$ for two given keys $\widehat{R}$ and $\widetilde{R}$. Algorithm then checks that $\mathsf{Enc}_{\mathsf{pk}_l}((\widehat{m}_l||\widehat{\sigma}_l); \widehat{R}) = \widehat{C}_l$ and $\mathsf{Enc}_{\mathsf{pk}_l}((\widetilde{m}_l||\widetilde{\sigma}_l); \widetilde{R}) = \widetilde{C}_l$. It finally returns $m_l \in \{m_x\}_{i\leq x\leq j}$ such that $m_l = \widehat{m}_l \oplus \widetilde{m}_l$. By the correctness property of PKE, $\mathsf{Dec}_{\mathsf{sk}_l}(\mathsf{Enc}_{\mathsf{pk}_l}((\widehat{m}_l||\widehat{\sigma}_l); \widehat{R})) = (\widehat{m}_l||\widehat{\sigma}_l)$ and $\mathsf{Dec}_{\mathsf{sk}_l}(\mathsf{Enc}_{\mathsf{pk}_l}((\widetilde{m}_l||\widetilde{\sigma}_l); \widetilde{R})) = (\widetilde{m}_l||\widetilde{\sigma}_l)$.

However, decryption algorithm $\mathsf{APOdec}_{\mathsf{sk}_l}(C_l)$ computes two values $(\widehat{m}'_l||\widehat{\sigma}'_l) = \mathsf{Dec}_{\mathsf{sk}_l}(\widehat{C}_l)$ and $(\widetilde{m}'_l||\widetilde{\sigma}'_l) = \mathsf{Dec}_{\mathsf{sk}_l}(\widetilde{C}_l)$ and returns $m'_l = \widehat{m}'_l \oplus \widetilde{m}'_l$. From previous result, $\widehat{m}'_l = \widehat{m}_l$ and $\widetilde{m}'_l = \widetilde{m}_l$ but $m_l \neq m'_l$, which leads us to a contradiction.

We conclude that $\mathsf{Adv}^{\mathsf{Integrity}}_{\mathsf{G\text{-}APO}}(k) = 0$.

□

D Exemple of RCD-PKE

Definition 11 (Cramer-Shoup Cryptosystem [10]). *The* Cramer-Shoup cryptosystem *is a PKE defined by:*

$\mathsf{Gen}(1^k)$**:** *Choose a cyclic group G of prime order p and two generators (g_1, g_2) of G. Choose H a universal one-way hash function. Pick five random values x_1, x_2, y_1 ,y_2 and z in $\mathbb{Z}_p^*$. Compute $c = g_1^{x_1} \cdot g_2^{x_2}$, $d = g_1^{y_1} \cdot g_2^{y_2}$ and $h = g_1^z$. Return the public key $\mathsf{pk} = (G, p, g_1, g_2, \mathsf{H}, c, d, h)$ and the secret key $\mathsf{sk} = (x_1, x_2, y_1, y_2, z)$.*

$\mathsf{Enc}_{\mathsf{pk}}(m; \sigma)$**:** *Compute $u_1 = g_1^\sigma$, $u_2 = g_2^\sigma$, $e = h^\sigma.m$, $\alpha = \mathsf{H}(u_1, u_2, e)$ and $v = c^\sigma \cdot d^{\sigma\alpha}$. Return (u_1, u_2, e, v).*

$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *Compute $\alpha = \mathsf{H}(u_1, u_2, e)$. If $u_1^{x_1} \cdot u_2^{x_2} \cdot (u_1^{y_1} u_2^{y_2})^\alpha = v$ then return $m = \frac{e}{u_1^z}$, else return $\perp$.*

Cramer-Shoup cryptosystem is a RCD-PKE since we have $\mathsf{CDec}_\sigma((u_1, u_2, e, v), \mathsf{pk}) = e/h^\sigma = m$.

Definition 12 (DHIES Cryptosystem [1]). *The* DHIES cryptosystem *is a Diffie-Hellman based PKE using a IND-CPA symmetric encryption scheme $\mathsf{SYM} = (\mathcal{E}^s, \mathcal{D}^s)$ and a CMA message authentication code scheme $\mathsf{MAC} = (\mathcal{G}, \mathcal{T}, \mathcal{V})$. DHIES is defined by:*

$\mathsf{Gen}(1^k)$**:** *Choose a cyclic group G of prime order p and a generator g of G. Choose $\mathsf{H} : \{0,1\}^{\mathsf{gLen}} \to \{0,1\}^{\mathsf{mLen}+\mathsf{eLen}}$ a universal one-way hash function such that gLen is the length of g, mLen is the length of a MAC key and eLen is the length of a SYM key. Pick a random value x in $\mathbb{Z}_p^*$. Return the public key $\mathsf{pk} = (g^x, H)$ and the secret key $\mathsf{sk} = x$.*

$\mathsf{Enc}_{\mathsf{pk}}(m; \sigma)$**:** *Compute $K = \mathsf{H}((g^x)^\sigma)$ from the public key pk, forge the key K_{MAC} from the mLen first bits of K and the key K_{SYM} from the eLen last bits of K. Compute $C_0 = g^\sigma$,$C_1 = \mathcal{E}^s_{K_{\mathsf{SYM}}}(m)$ and $C_2 = \mathcal{T}_{K_{\mathsf{MAC}}}(C_1)$ and return $C = (C_0, C_1, C_2)$.*

$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *Compute $K = \mathsf{H}(C_0^{\mathsf{sk}})$, forge the key K_{MAC} from the mLen first bits of K and the key K_{SYM} from the eLen last bits of K. If $\mathcal{V}_{K_{\mathsf{MAC}}}(C_1, C_2) = 1$ then compute $m = \mathcal{E}^s_{K_{\mathsf{SYM}}}(C_1)$ and return it, else return $\perp$.*

DHIES is a RCD-PKE since we can construct the CDec algorithm as follows:

$\mathsf{CDec}_\sigma((C_0, C_1, C_2), \mathsf{pk})$: Compute $K = \mathsf{H}((g^x)^\sigma)$, forge the key K_{SYM} from the eLen last bits of K and return $m = \mathcal{E}^s_{K_{\mathsf{SYM}}}(C_1)$.

Definition 13 (Fujisaki-Okamoto generic construction [14]).
Fujisaki-Okamotto generic construction *is a PKE construction using an IND-CPA symmetric encryption scheme* $\mathsf{SYM} = (\mathcal{E}^s, \mathcal{D}^s)$*, an IND-CPA public key encryption* $\mathsf{PUB} = (\mathcal{G}^p, \mathcal{E}^p, \mathcal{D}^p)$ *and two hash functions* G *and* H*. This construction is defined as follows:*

$\mathsf{Gen}(1^k)$**:** *Generate* $(\mathsf{pk}, \mathsf{sk}) = \mathcal{G}^p(1^k)$ *and return it.*
$\mathsf{Enc}_{\mathsf{pk}}(m; \sigma)$**:** *Compute* $C_0 = \mathcal{E}^s_{\mathsf{G}(\sigma)}(m)$ $C_1 = \mathcal{E}^p_{pk}(\sigma; \mathsf{H}(\sigma, C_0))$*. Return* $C = (C_0, C_1)$*.*
$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *Compute the two values* $\sigma = \mathcal{D}^p_{sk}(C_1)$ *and* $m = \mathcal{D}^s{}_{\mathsf{G}(\sigma)}(C_0)$*. If* $C_1 = \mathcal{E}^p_{pk}(\sigma; \mathsf{H}(\sigma, C_0))$ *then return* m*, else return* $\perp$*.*

Fujisaki-Okamoto construction is a RCD-PKE since we can construct the algorithm $\mathsf{CDec}_\sigma((C_0, C_1), \mathsf{pk}) = \mathcal{D}^s{}_{\mathsf{G}(\sigma)}(C_0)$.

E Particular APO-PKE schemes

We introduce two particular cases of APO-PKE: *Last A Posteriori Openable Public Key Encryption* (LAPO-PKE) that is an APO-PKE where openable intervals always contain the last message, and *First A Posteriori Openable Public Key Encryption* (FAPO-PKE) that is an APO-PKE where openable intervals always contain the last message. We give a generic construction for theses two primitives from G-APO.

Definition 14 (Generic LAPO-PKE (G-LAPO)). *Let* k *be a security parameter,* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a RCD and VK public key encryption scheme,* $\mathcal{R}$ *be the* $\mathcal{E}$ *set of random coin and* $\mathsf{F} : \{0,1\}^* \to \{0,1\}^k$*,* $\mathsf{G} : \{0,1\}^* \to \mathcal{R}$ *and* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ *be three universal hash functions. Our generic LAPO-PKE is defined by:*

$\mathsf{APOgen}(1^k)$**:** *This algorithm generates* $(\mathsf{pk}, \mathsf{sk})$ *with* Gen *and returns it.*
$\mathsf{APOini}(1^k)$**:** *This algorithm picks two random values* σ *and* K *in* $\{0,1\}^k$*. It returns* $\mathsf{st} = (K||\sigma)$*.*
$\mathsf{APOenc}^{\mathsf{st}}_{\mathsf{pk}}(m)$**:** *Using the state* $\mathsf{st} = (K||\sigma_N)$*, this algorithm picks a random value* σ *in* $\{0,1\}^k$ *and computes* $C = \mathsf{Enc}_{pk}(m||(\sigma \oplus \mathsf{F}(\sigma_N)); \mathsf{G}(\sigma_N))$ *and* $D = \sigma_N \oplus \mathsf{H}(K||C)$*. Finally it updates the state* $\mathsf{st} := (K||\sigma)$ *and returns* $(C||D)$*.*
$\mathsf{APOdec}_{\mathsf{sk}}(C)$**:** *This algorithm computes* $(m||\sigma) = \mathsf{Dec}_{sk}(C)$ *and it returns* m*.*
$\mathsf{APOext}^{\mathsf{st}}_{\mathsf{pko}}(C_i, C_j)$**:** *We note* $\mathsf{st} = (K||\sigma_N)$ *and* $C_i = (C||D)$*. It initializes* $\sigma_{i-1} = D \oplus \mathsf{H}(K||C)$*. It picks* $r \xleftarrow{\$} \mathcal{R}$ *and returns* $K^{pko}_{i \to j} = \mathsf{Enc}_{pko}(\sigma_{i-1}; r)$*.*
$\mathsf{APOpen}_{\mathsf{sko}}(K^{\mathsf{pko}}_{i \to N}, \{(C_x||D_x)\}_{i \le x \le N}, \{\mathsf{pk}_x\}_{i \le x \le N})$**:** *This algorithm starts by recovering* $\sigma_{i-1} = \mathsf{Dec}_{sko}(K^{pko}_{i \to N})$*.*
- *For all* x *in* $\{i, i+1, \ldots, N\}$*, it computes* $R = \mathsf{G}(\sigma_{x-1})$*. It opens* C_x *as follows* $(m_x||\sigma^*_x) = \mathsf{CDec}_R(C_x, \mathsf{pk}_x)$*. Finally it computes* $\sigma_x = \sigma^*_x \oplus \mathsf{F}(\sigma_{x-1})$*. If* $\mathsf{Enc}_{\mathsf{pk}_x}((m_x||\sigma^*_x); \mathsf{G}(\sigma_{x-1})) \neq C_x$ *then it aborts and returns* $\perp$*.*

Finally, it returns $\{m_x\}_{i \le x \le N}$*.*

Definition 15 (Generic FAPO-PKE (G-FAPO)). *Let k be a security parameter, $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a RCD and VK public key encryption scheme, $\mathcal{R}$ be the $\mathcal{E}$ set of random coin and $\mathsf{F} : \{0,1\}^* \to \{0,1\}^k$, $\mathsf{G} : \{0,1\}^* \to \mathcal{R}$ and $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ be three universal hash functions. Our generic FAPO-PKE is defined by:*

$\mathsf{APOgen}(1^k)$**:** *This algorithm generates $(\mathsf{pk}, \mathsf{sk})$ with Gen and returns it.*

$\mathsf{APOini}(1^k)$**:** *This algorithm picks two random values σ and K from $\{0,1\}^k$. It returns $\mathsf{st} = (K||\sigma)$.*

$\mathsf{APOenc}^{\mathsf{st}}_{\mathsf{pk}}(m)$**:** *Using the state $\mathsf{st} = (K||\sigma_N)$, this algorithm picks a random value σ in $\{0,1\}^k$ and computes $C = \mathsf{Enc}_{\mathsf{pk}}(m||(\sigma_N \oplus \mathsf{F}(\sigma)); \mathsf{G}(\sigma))$ and $D = \sigma \oplus \mathsf{H}(K||C)$. Finally it updates the state $\mathsf{st} := (K||\sigma)$ and returns $(C||D)$.*

$\mathsf{APOdec}_{\mathsf{sk}}(C)$**:** *This algorithm computes $(m||\sigma) = \mathsf{Dec}_{\mathsf{sk}}(C)$ and it returns m.*

$\mathsf{APOext}^{\mathsf{st}}_{\mathsf{pko}}(C_i, C_j)$**:** *We note $\mathsf{st} = (K||\sigma_N)$ and $C_j = (C||D)$. It initializes $\sigma_j = D \oplus \mathsf{H}(K||C)$. It picks $r \xleftarrow{\$} \mathcal{R}$ and returns $K^{\mathsf{pko}}_{i \to j} = \mathsf{Enc}_{\mathsf{pko}}(\sigma_j; r)$.*

$\mathsf{APOpen}_{\mathsf{sko}}(K^{\mathsf{pko}}_{1 \to j}, \{(C_x||D_x)\}_{1 \le x \le j}, \{\mathsf{pk}_x\}_{1 \le x \le j})$**:** *This algorithm starts by recovering $\sigma_j = \mathsf{Dec}_{\mathsf{sko}}(K^{\mathsf{pko}}_{1 \to j})$.*

- *For all x in $\{j, j-1, \ldots, 1\}$, it computes $R = \mathsf{G}(\sigma_x)$. It opens C_x as follows $(m_x||\sigma^*_{x-1}) = \mathsf{CDec}_R(C_x, \mathsf{pk}_x)$. Finally it computes $\sigma_{x-1} = \sigma^*_{x-1} \oplus \mathsf{F}(\sigma_x)$. If $\mathsf{Enc}_{\mathsf{pk}_x}((m_x||\sigma^*_{x-1}); \mathsf{G}(\sigma_x)) \neq C_x$ then it aborts and returns $\perp$.*

Finally, it returns $\{m_x\}_{1 \le x \le j}$.