Provably Secure Timed-Release Public Key Encryption

JUNG HEE CHEON
Seoul National University, Korea
and
NICHOLAS HOPPER, YONGDAE KIM and IVAN OSIPKOV
University of Minnesota - Twin Cities

A timed-release cryptosystem allows a sender to encrypt a message so that only the intended recipient can read it only after a specified time. We formalize the concept of a secure timed-release public-key cryptosystem and show that, if a third party is relied upon to guarantee decryption after the specified date, this concept is equivalent to identity-based encryption; this explains the observation that all known constructions use identity-based encryption to achieve timed-release security. We then give several provably-secure constructions of timed-release encryption: a generic scheme based on any identity-based encryption scheme, and two more efficient schemes based on the existence of cryptographically admissible bilinear mappings. The first of these is essentially as efficient as the Boneh-Franklin Identity-Based encryption scheme, and is provably secure and authenticated in the random oracle model; the final scheme is not authenticated but is provably secure in the standard model (i.e., without random oracles).

Categories and Subject Descriptors: E.3 [**Data**]: Data Encryption—*Public Key Cryptosystems*

General Terms: Security, Theory

Additional Key Words and Phrases: timed-release, authenticated encryption, key-insulated encryption

ACM Reference Format:
Cheon, J. H., Hopper, N., Kim, Y., and Osipkov, I. 2008. Provably secure timed-release public key encryption. ACM Trans. Inf. Syst. Secur. 11, 2, Article 8 (May 2008), 44 pages. DOI = 10.1145/1330332.1330336. http://doi.acm.org/10.1145/1330332.1330336.

Authors' addresses: J.H. Cheon, Seoul National University, Korea; email: jhcheon@shu.4c.kr; N. Hopper, Y. Kim, University of Minnesota - Twin Cities; email: {hopper, kyd}@cs.umn.edu; I. Osipkov, Microsoft Corp.; email: ivan.osipkov@microsoft.com.
Y. Kim and I. Osipkov were partly supported by NSF Career Grant CNS-0448423 and by The Intelligent Storage Consortium at the Digital Technology Center, University of Minnesota. J. H. Cheon was supported by Korea Telecom.

1. INTRODUCTION

The goal of timed-release cryptography is to "send a message into the future." One way to do this is to encrypt a message such that the receiver cannot decrypt the ciphertext until a specific time in the future. Such a primitive would have many practical applications; a few examples include preventing a dishonest auctioneer from prior opening of bids in a sealed-bid auction [Rivest et al. 1996], preventing early opening of votes in e-voting schemes, fair exchange, release of classified information, and delayed verification of a signed document, such as electronic lotteries [Syverson 1998] and check cashing. The problem of timed-release cryptography was first mentioned by [May 1993] and then discussed in detail by [Rivest et al. 1996].

Let us assume that Alice wants to send a message to Bob such that Bob will not be able to open it until a certain time. Previous solutions fall into two categories:

—*Time-lock puzzles:* Alice encrypts her message so that Bob needs to perform non-parallelizable computation without stopping for the required time to decrypt it. If Alice accurately predicts Bob's computing resources between now and the desired time, then Bob recovers the message.

—*Trusted decryption agents:* Alice encrypts a message such that Bob needs some secret value, published by a trusted agent on the required date, in order to decrypt the message. Once the agent releases the information, Bob can decrypt the message.

The first approach puts considerable computational overhead on the message receiver, which makes it undesirable for real-life scenarios. In addition, knowing the computational complexity of decryption, while giving us a lower bound on the computing time Bob may need to decrypt the message, does not guarantee that the plaintext will be available at a certain date. Still, this approach is widely used for specific applications [Boneh and Naor 2000; Belare and Goldwasser 1996; Syverson 1998; Garay and Pomerance 2002, 2003]. The agent-based approach, on the other hand, relieves Bob from performing nonstop computation, sets the date of decryption precisely and does not require Alice to have information on Bob's capabilities. This comes at a price, though: the agents have to be trusted and they have to be available at the designated time.

In this article we concentrate on schemes that use such "decryption agents." We formalize this notion of a secure timed-release encryption scheme and show that it is equivalent to the notion of strongly key-insulated encryption [Dodis et al. 2002]; when there is no a priori bound on the number of time periods, this notion is, in turn, known to be equivalent to identity-based encryption, or IBE [Bellare and Palacio 2002]. We also give several provably-secure constructions of timed-release public-key encryption, including the first provably-secure generic construction in the literature, and the first efficient scheme that is provably secure in the standard model, that is, without random oracles.

Our results also cast new light on several previous schemes that appear in the literature: each can be seen as an adaptation of a known key-insulated encryption scheme. For example, Rivest et al. [1996] propose that the agent could encrypt messages on request with a secret key which will be published on a designated date by the agent, or the agent can precompute pairs of public/private keys, publish all public keys and release the private keys on the required days; these exactly fit known key-insulated schemes appearing in the literature. The scheme of Crescenzo et al. [1999] essentially replaces publication of the key with publication of the message, requiring the receiver to engage in a conditional oblivious transfer protocol with the agent to decrypt the message. In Chen et al. [2002], the authors proposed to use Boneh and Franklin's IBE scheme [Boneh and Franklin 2003] for timed-release encryption: essentially, the scheme replaces the identity in an IBE scheme with the time of decryption. Similar proposals appear in Marco Casassa Mont and Sadler [2003] and Blake and Chan [2005].

While some of the above proposals contain informal proofs of security, none of them consider and/or give a formal treatment of the security properties of timed-release public key encryption (or TR-PKE). The first formal treatments of TR-PKE security were displayed in Cheon et al. [2004] and then strengthened in Cheon et al. [2006]. Independently, Cathalo et al. [2005] introduce another notion of timed-release security and argue that it is not implied by key-insulated encryption; however, this seems to be a side effect of an overly-restrictive model in which a user must commit to a specific decryption agent before choosing his public key.

Authentication for Timed-Release Encryption. Many of the applications of timed-release cryptography mentioned above require some form of authentication as well. For example, if there is no authentication of bids in a sealed-bid auction, any bidder may be able to forge bids for others, or force the auction to fail by submitting an unreasonably high bid. In this article, we consider the security properties required by these applications and develop formal security conditions for a Timed-Release Public Key Authenticated Encryption (TR-PKAE) scheme.

One avenue for developing a TR-PKAE scheme would be composing an unauthenticated TR-PKE scheme with either a signature scheme or a (non-timed-release) PKAE scheme. Although such constructions are possible, we note that the details of this composition are not trivial; examples from An [2001] and Dodis and Katz [2005] illustrate that naive constructions can fail to provide the expected security properties. Additionally, we note that such schemes are likely to suffer a performance penalty relative to a scheme based on a single primitive. Thus, besides introducing a generic construction, we also introduce a provably secure construction of a TR-PKAE scheme that is essentially as efficient as previous constructions of *non-authenticated* TR-PKE schemes [Chen et al. 2002; Marco Casassa Mont and Sadler 2003; Blake and Chan 2005].

2. DEFINITIONS

In this section we review security definitions that will be used in the article. In addition, we introduce new definitions, namely, those of timed-release public key encryption (TR-PKE) and authenticated TR-PKE (TR-PKAE).

Identity Based Encryption. Formally, we define an IBE scheme IBES to be a tuple of four randomized algorithms:

—$\text{Setup}_{\text{IBE}}(1^k)$, which given input 1^k (the security parameter), produces public parameters π_{IBE}, which include hash functions, message and ciphertext spaces among others. In addition, master secret δ_{IBE} is generated which is kept confidential by the central authority.

—$\text{Extract}_{\text{IBE}}(\pi_{\text{IBE}}, \delta_{\text{IBE}}, I)$, given public parameters π_{IBE}, master secret δ_{IBE} and identity $I \in \{0,1\}^*$, outputs a secret key sk_I. The I (together with π_{IBE}) serves as the public key corresponding to identity I.

—$\text{Encrypt}_{\text{IBE}}(\pi_{\text{IBE}}, I, m)$ computes the ciphertext c denoting the encryption for identity I of message m with public parameters π_{IBE}.

—$\text{Decrypt}_{\text{IBE}}(\pi_{\text{IBE}}, sk_I, \widehat{c})$ outputs the plaintext corresponding to $\widehat{c}$ if decryption is successful or the special symbol "fail" otherwise.

For consistency, we require that $\text{Decrypt}_{\text{IBE}}(\pi_{\text{IBE}}, sk_I, \text{Encrypt}_{\text{IBE}}(\pi_{\text{IBE}}, I, m)) = m$, for all valid (I, sk_I), $(\pi_{\text{IBE}}, \delta_{\text{IBE}})$, and m.

We use the IND-ID-CCA notion of security for and IBE scheme [Boneh and Franklin 2003]. Briefly, in this case, an adversary may adaptively ask for secret keys corresponding to arbitrary identities, and may also ask for decryption of any ciphertext using any identity. Eventually the adversary presents a "challenge identity" and a pair of "challenge plaintexts" and is given the encryption of one of these plaintexts under the challenge identity. The adversary may then continue to ask for secret keys and decryptions, except that it cannot query for the secret key of the challenge identity or for decryption of the challenge ciphertext under the challenge identity. The adversary wins if it can correctly guess which of the challenge ciphertexts was encrypted by the challenger, and the scheme is secure if no polynomial time adversary wins with an advantage non-negligibly greater than one half.

Public Key Encryption. A public key encryption system PKE consists of three algorithms:

—$\text{KeyGen}_{\text{PKE}}$, which on input 1^k, outputs public/private key pair (pk, sk). The public key also includes public parameters needed for encryption/decryption.

—$\text{Encrypt}_{\text{PKE}}$, which on input of pk and message m, outputs ciphertext c.

—$\text{Decrypt}_{\text{PKE}}$, which on input of ciphertext $\widehat{c}$ and private key sk, outputs either some message $\widehat{m}$ or failure symbol.

For consistency, it is required that $\text{Decrypt}_{\text{PKE}}(sk, \text{Encrypt}_{\text{PKE}}(pk, m)) = m$, for all valid (pk, sk) and m.

We make use of a PKE that is IND-CCA2 secure against adaptive adversary as described in Bellare et al. [1998]. Briefly, the challenger generates a public/private key pair and gives the public key to the adversary. The adversary is allowed to query for the decryption of any ciphertext using the private key. In the challenge step, the adversary produces a pair of challenge plaintexts and is given the encryption of one of the pair. The adversary wins if, given the ability to query the decryption of any message but the challenge ciphertext, it can correctly guess which of the two plaintexts was encrypted in the challenge step.

We note that given a secure IBES, we can easily obtain a secure PKE. For that purpose, each user simply runs IBES's $\text{Setup}_{\text{IBE}}$ and $\text{Extract}_{\text{IBE}}$, using an arbitrary identity I, to obtain its public key and private key (i.e., the master secret key in IBES). The identity I along with IBES's public parameters serves as user's public key, while the master secret key serves as the private key. A straightforward argument shows that if IBES is IND-ID-CCA secure then the corresponding PKE is IND-CCA2 secure. However, since in practical applications we expect one to use more efficient PKE constructions, we make use of separate IBE and PKE schemes in this article.

Digital Signatures and Labels. In addition to the above primitives, we will also use signature schemes. We start with review of standard signatures first. A signature scheme DS consists of three algorithms:

—SigGen, which on input 1^k, outputs signing/verification key pair (SK, VK). The VK also includes also public parameters such as the message space among others.

—Sig, which on input SK and message m, outputs signature σ.

—Ver, which on input message m, signature σ and VK, outputs either true or false.

For consistency, it is required that for every valid pair (SK, VK) and message m, $\text{Ver}_{VK}(m, \text{Sig}_{SK}(m)) = \textit{true}$. We will use the notion of strong unforgeability under adaptive chosen plaintext attacks (SUF-CMA). Briefly, the challenger generates a (SK, VK) pair and gives the VK to the adversary. The adversary is given signatures $\sigma_1, \sigma_2, \ldots, \sigma_q$ on adaptively chosen messages $m_1, m_2, \ldots, m_q$ and outputs a pair (m, σ). The adversary wins if (m, σ) is a valid message signature pair and is not equal to any pair (m_i, σ_i).

Beside standard signatures, we will also use *one-time signatures* which are defined analogously, except that in SUF-CMA the adversary is allowed to make only one query. Any public-key signature that is SUF-CMA secure is also a secure one-time signature. However, the opposite is obviously not true and one-time signatures are generally much more efficient.

We can also add public labels to IBE and PKE encryption/decryption mechanisms, which are bound in a nonmalleable way to the ciphertext [Shoup 2004] while preserving security. In effect, ciphertext generation additionally takes as input a label, which becomes part of the ciphertext. When decrypting, one applies not only the decryption key but also the public label. The IND-ID-CCA and IND-CCA2 games can be modified in a natural way to take labels into account.

2.1 Timed-Release Public Key Encryption (TR-PKE)

In this section we formalize the functionality and security requirements for a timed-release public key encryption system. These requirements are meant to capture the implicit security requirements not addressed in previous work [May 1993; Rivest et al. 1996; Chen et al. 2002; Marco Casassa Mont and Sadler 2003; Blake and Chan 2005]; in particular they do not address the authentication requirements, which we add in Section 2.3. Informally, we can think of any principal in a TR-PKE system as filling one or more of three roles. The *timed-release agent*—or TiPuS (TImed-release PUblic Server)—publishes a timed-release public key and releases "tokens" that allow decryption of messages encrypted for the current time at regular intervals. The *receiver* publishes a public key that allows others to encrypt messages so that only he can decrypt them, using a secret key that he keeps private, and the appropriate timed-release token. The *sender* uses the receiver's public key and the TiPuS public key to encrypt messages that can later be decrypted at the time of his choice.

2.1.1 *Functional Requirements.* Formally, we define a timed-release public-key encryption system Γ to be a tuple of five randomized algorithms:

—Setup, which given input 1^k (the security parameter), produces public parameters π_g, which include hash functions, message, and ciphertext spaces among others.

—TRSetup, which on input π_g, produces a pair (δ, π_{tr}) where δ is a *master secret* and π_{tr} the corresponding timed-release public parameters. This setup is carried out by TiPuS which keeps the master secret key confidential, while all other parameters are public. We denote the combined public parameters of π_g and π_{tr} by π.

—KeyGen, given public parameters π_g, outputs a pair of secret key and public key (sk, pk).

—TG(π, δ, T) computes the token tkn_T corresponding to time T using (δ, π). This functionality is performed by TiPuS which publishes tkn_T at time T.

—Encrypt(π, pk, m, T) computes the timed-release ciphertext c denoting the encryption of message m using public key pk, public parameters π and time encoding T.

—Decrypt$(\pi, sk, \widehat{c}, tkn_T)$ outputs the plaintext corresponding to $\widehat{c}$ if decryption is successful or the special symbol "fail" otherwise.

For consistency, we require that Decrypt$(\pi, sk,$ Encrypt$(\pi, pk, m, T),$ TG $(\pi, \delta, T)) = m$, for all valid (pk, sk), (π, δ), T, and m. Unlike the functional requirements specified in Cathalo et al. [2005], we explicitly separate the functions TRSetup and KeyGen, allowing a user to generate keys independent of any timed-release server. This allows the sender to choose which servers to trust during encryption.

Algorithm 2.1: $\text{Exp}_{A,\Gamma}^{\text{IND-CCA2}}(k)$

```
$\pi_g \leftarrow \text{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \text{TRSetup}(1^k)$
$(pk, sk) \leftarrow \text{KeyGen}(\pi_g)$
$(m_0, m_1, T^*) \leftarrow A^{\text{Decrypt}(\pi, sk, \cdot, \cdot)}(\pi, \delta, pk)$
$\beta \leftarrow_R \{0, 1\}$
$c^* \leftarrow \text{Encrypt}(\pi, pk, m_\beta, T^*)$
$\beta' \leftarrow A^{\text{Decrypt}(\pi, sk, \cdot, \cdot)}(\pi, \delta, pk, c^*)$
if ($A$ queried $\text{Decrypt}(\pi, sk, c^*, tkn_{T^*})$)
  then return (false)
  else return $(\beta' = \beta)$
```

Algorithm 2.2: $\text{Exp}_{A,\Gamma}^{\text{IND-RTR-CCA2}}(k)$

```
$\pi_g \leftarrow \text{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \text{TRSetup}(1^k)$
$(m_0, m_1, pk^*, T^*)$
    $\leftarrow A^{\text{TG}(\pi, \delta, \cdot), \text{Decrypt}^*(\pi, \delta, \cdot, \cdot, \cdot)}(\pi)$
$\beta \leftarrow_R \{0, 1\}$
$c^* \leftarrow \text{Encrypt}(\pi, pk^*, m_\beta, T^*)$
$\beta' \leftarrow A^{\text{TG}(\pi, \delta, \cdot), \text{Decrypt}^*(\pi, \delta, \cdot, \cdot, \cdot)}(\pi, c^*)$
if ($A$ queried $\text{Decrypt}^*(\pi, sk^*, c^*, T^*)$,
where $sk^*$ corresponds to $pk^*$,
or $A$ queried $\text{TG}(\pi, \delta, T^*)$)
  then return (false)
  else return $(\beta' = \beta)$
```

$$\text{Adv}_{A,\Gamma}^{\text{IND-CCA2}}(k) = \Pr[\text{Exp}_{A,\Gamma}^{\text{IND-CCA}}(k) = \text{true}] - \tfrac{1}{2}$$
$$\text{Adv}_{A,\Gamma}^{\text{IND-RTR-CCA2}}(k) = \Pr[\text{Exp}_{A,\Gamma}^{\text{IND-RTR-CCA2}}(k) = \text{true}] - \tfrac{1}{2}$$

Fig. 1. TR-PKE security experiments for the IND-CCA2 and IND-RTR-CCA2 games.

2.1.2 *Security*. It is standard to require that the PKE cryptosystem be secure against adaptive chosen-ciphertext (IND-CCA2) attack [Rackoff and Simon 1991; Bellare et al. 1998; An 2001]. Ideally, in a TR-PKE, the timed-release agent should not be able to read messages intended for third-party recipients. To that effect, we require that IND-CCA2 security against a third party is provided even when the master secret is given to the adversary. We model this attack by a slightly modified IND-CCA2 game, shown in Figure 1. Here, in addition to adaptively choosing two "challenge plaintexts" that the adversary will need to distinguish between, he also adaptively chooses a "challenge time" for which his challenge ciphertext will be decrypted; he wins when he can tell whether his challenge ciphertext is an encryption of his first or second plaintext for the challenge time, given access to a decryption oracle and the master secret key of the TiPuS.

The timed-release functionality is provided by the token-generating infrastructure (i.e., TiPuS). Not knowing the corresponding token is what keeps the receiver from decrypting ciphertext until a designated time. To effect secure timed-release, any TR-PKE cryptosystem must provide confidentiality against the receiver itself until the corresponding token is made available. We model this property by the IND-RTR-CCA2 game, shown in Figure 1; in this game, we modify the basic IND-CCA2 game by allowing the adversary to adaptively choose the receiver's public key pk^* and time T^* for the challenge. Instead of access to the timed-release secret, the adversary is given access to arbitrary tokens tkn_T, where $T \neq T^*$, and a decryption oracle $\text{Decrypt}^*(\pi, \delta, \cdot, \cdot, \cdot)$ which computes $\text{Decrypt}(\pi, \cdot, \cdot, \text{TG}(\pi, \delta, \cdot))$. The adversary may thus compute the decryption of any ciphertext for any time, except the challenge ciphertext in the challenge time T^* with chosen public key pk^*. We say a timed-release public-key cryptosystem Γ is secure if every polynomial time adversary A has negligible advantages $\text{Adv}_{\text{A},\Gamma}^{\text{IND-CCA2}}(k)$ and $\text{Adv}_{\text{A},\Gamma}^{\text{IND-RTR-CCA2}}(k)$.

2.2 Strongly Key-Insulated Public Encryption

Key-insulated public key encryption was introduced by Dodis et al. [2002, 2003] and Bellare and Palacio [2002] to address the problem of computer intrusion. The key idea is to break up the lifetime of a public key into periods, and split the decryption key between the user (say, a mobile device) and a trusted "helper" (say, a desktop server) to satisfy the following properties:

—*Sequential Key Updates*: At the beginning of each time period, the helper securely transmits a "helper secret key" hsk_i to the user, which he combines with his previous key, usk_{i-1}, to obtain a secret key usk_i that will decrypt messages encrypted during time period i.

—*Random Access Key Updates*: Given any usk_i and hsk_j, the user can compute usk_j. This is useful for error recovery and it also allows the user to decrypt old messages.

—*User Compromise*: An adversary who is given access to (usk_i, hsk_i) for several time periods i cannot break the encryption for a new time period.

—*Helper Compromise*: An adversary given only hsk cannot break the encryption scheme.

Combining the results of Bellare and Palacio [2002] and Dodis and Katz [2005], one obtains that the existence of secure SKIE-OTRU is a necessary and sufficient condition for the existence of secure IBE. Briefly, a SKIE-OTRU scheme consists of the following algorithms: KG, which generates a triple (pk, usk_0, hsk) of public key, initial user secret key, and master helper key; HKU, which computes a *stage i helper secret key* hsk_i given (pk, hsk, i); UKU, which computes the *stage i user secret key* usk_i given i, pk, hsk_i, usk_{i-1}; RUKU, which computes the *stage i user secret key* usk_i given $i, j, pk, hsk_i, usk_j, \forall i \geq 1, j \geq 0$; Enc, which produces a ciphertext corresponding to m to be decrypted in stage i, given (pk, m, i); and Dec, which, given (i, pk, usk_i, c) attempts to decrypt a ciphertext for stage i. Intuitively, hsk is given to a "helper," who will securely transmit, at the beginning of each stage i, the secret hsk_i to the user. The user can then compute usk_i, delete any old usk's in his possession, and use usk_i to decrypt messages sent to him during stage i. The RUKU algorithm facilitates error recovery and allows for decryption of old ciphertexts.

A SKIE (and SKIE-OTRU) scheme is considered CCA-secure with optimal threshold if two conditions hold: (1) (IND-KIE-CCA2) given access to pk, a decryption oracle, and pairs (hsk_i, usk_i) of his choosing, an adversary cannot break the IND-CCA2 security of the encryption scheme for a stage j for which he has not been given hsk_j; and (2) (IND-S-CCA2) given pk, hsk, and a decryption oracle, an adversary cannot break the IND-CCA2 security of the encryption scheme for any stage [Dodis et al. 2002, 2003; Bellare and Palacio 2002]. The idea of separation of the timed-release master and user secrets in a TR-PKE very closely parallels the notions of helper and user secrets in a key-insulated cryptosystem; and both involve a "time period" parameter for encryption and decryption. Furthermore, the two security conditions for a SKIE scheme, in which either user keys or helper keys are assumed to be compromised, closely

```
Algorithm 2.3: $\mathrm{Exp}^{\text{IND-KC-CCA2}}_{A,\Gamma}(k)$

$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_a, sk_a) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(pk_b, sk_b) \leftarrow \mathsf{KeyGen}(\pi_g)$
$\vec{\kappa} \leftarrow (\pi, \delta, pk_a, sk_a, pk_b)$
$(m_0, m_1, T^*)$
    $\leftarrow A^{\mathsf{Decrypt}(\pi, pk_a, sk_b, \cdot, \cdot)}(\vec{\kappa})$
$\beta \leftarrow_R \{0, 1\}$
$c^* \leftarrow \mathsf{Encrypt}(\pi, sk_a, pk_b, m_\beta, T^*)$
$\beta' \leftarrow A^{\mathsf{Decrypt}(\pi, pk_a, sk_b, \cdot, \cdot)}(\vec{\kappa}, c^*)$
if ($A$ queried
        $\mathsf{Decrypt}(\pi, pk_a, sk_b, c^*, tkn_{T^*})$)
  then return (false)
  else return ($\beta' = \beta$)
```

```
Algorithm 2.4: $\mathrm{Exp}^{\text{IND-RTR-KC-CCA2}}_{A,\Gamma}(k)$

$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_a, sk_a) \leftarrow \mathsf{KeyGen}(\pi_g)$
$\vec{\kappa} \leftarrow (\pi, pk_a, sk_a)$
$(m_0, m_1, pk_b^*, T^*)$
    $\leftarrow A^{\mathsf{TG}(\pi,\delta,\cdot), \mathsf{Decrypt}^*(\pi,\delta,pk_a,\cdot,\cdot,\cdot)}(\vec{\kappa})$
$\beta \leftarrow_R \{0, 1\}$
$c^* \leftarrow \mathsf{Encrypt}(\pi, sk_a, pk_b^*, m_b, T^*)$
$\beta' \leftarrow A^{\mathsf{TG}(\pi,\delta,\cdot), \mathsf{Decrypt}^*(\pi,\delta,pk_a,\cdot,\cdot,\cdot)}(\vec{\kappa}, c^*)$
if ($A$ queried $\mathsf{Decrypt}^*(\pi, pk_a, sk_b^*, c^*, T^*)$
 or $\mathsf{TG}(\pi, \delta, T^*)$)
  then return (false)
  else return ($\beta' = \beta$)
```

$$\mathrm{Adv}^{\text{IND-KC-CCA2}}_{A,\Gamma}(k) = \Pr[\mathrm{Exp}^{\text{IND-KC-CCA2}}_{A,\Gamma}(k) = \mathsf{true}] - \tfrac{1}{2}$$
$$\mathrm{Adv}^{\text{IND-RTR-KC-CCA2}}_{A,\Gamma}(k) = \Pr[\mathrm{Exp}^{\text{IND-RTR-KC-CCA2}}_{A,\Gamma}(k) = \mathsf{true}] - \tfrac{1}{2}$$

Fig. 2. TR-PKAE experiments for the IND-KC-CCA2 and IND-RTR-KC-CCA2 games.

resemble the TR-PKE conditions IND-CCA2 and IND-RTR-CCA2 developed here.

2.3 Authenticated TR-PKE (TR-PKAE)

The notion of authenticated encryption has been explored in depth in An [2001] and Abdalla et al. [2001]. In this section we adapt these definitions to give formal security and functionality requirements for a TR-PKAE scheme.

2.3.1 *Basic Cryptosystem.* The syntactic definition of a TR-PKAE scheme is essentially the same as that of a TR-PKE scheme with the addition of the sender's public and secret key. Namely, the types of Setup, TRSetup, KeyGen and TG stay the same, but Encrypt and Decrypt are modified to take into account sender's keys:

—Encrypt(π, sk_a, pk_b, m, T) returns an authenticated timed-release ciphertext c denoting the encryption from sender A to receiver B of m for time T.

—Decrypt(π, pk_a, sk_b, $\widehat{c}$, tkn_T) outputs plaintext $\widehat{m}$ if both decryption and authentication are successful and the special symbol "fail" otherwise.

The consistency requirement is modified to require that, for all valid (pk_a, sk_a), (pk_b, sk_b), (π, δ), T, and m, Decrypt(π, pk_a, sk_b, Encrypt(π, sk_a, pk_b, m, T), TG(π, δ, T))=m.

2.3.2 *Security.*

Confidentiality. The confidentiality requirements of a TR-PKAE are essentially the same as the confidentiality requirements of a TR-PKE; except that we make the conservative assumption that the third party (in the case of IND-CCA2) or the receiver (in the case of IND-RTR-CCA2) has compromised the sender's secret key. This results in two new notions, IND-KC-CCA2 and

IND-RTR-KC-CCA2, which we define formally in Figure 2. As before, we say that a TR-PKAE scheme provides confidentiality if every polynomial time adversary has negligible advantage, as defined in Figure 2.

As in the case of TR-PKE, the difference between IND-KC-CCA2 and IND-RTR-KC-CCA2 is in reversal of adversary roles. In IND-RTR-KC-CCA2, the goal is to ensure security against the receiver itself prior to the designated time.

Ciphertext (Plaintext) Forgery. For authentication properties of TR-PKAE, we concentrate on ciphertext forgery (plaintext forgery is defined analogously). We consider two types of ciphertext forgery: *third-party forgery* (TUF-CTXT), by an adversary that does not know the sender's and receiver's private keys but knows the master secret; and *forgery by the ciphertext receiver* (RUF-CTXT) [An 2001]. If the TR-PKAE is not secure against TUF-CTXT then the scheme cannot claim authentication properties since a third party may be able to forge new (perhaps decrypting to junk) ciphertexts between two users. If a TR-PKAE is not secure against RUF-CTXT, then the scheme does not provide non-repudiation[1] and furthermore, if the receiver's private key is compromised, the attacker can impersonate any sender to this receiver. We introduce the following games to model unforgeability (see Figure 3).

Receiver Unforgeability (RUF-CTXT and RUF-TR-CTXT). We introduce two notions of receiver unforgeability: RUF-CTXT in which the receiver cannot forge ciphertext to himself for any time and a weaker timed-release notion of RUF-CTXT, called RUF-TR-CTXT, which requires that the receiver should not be able to forge ciphertext to himself for a future date. The notion RUF-TR-CTXT has two important implications: (1) the receiver should discard any ciphertexts received past decryption dates if his private key may be compromised; and (2) the receiver may be able to prove to a third party that a ciphertext was generated by the alleged sender if he can produce a proof of ciphertext existence prior to the decryption date. The game in Figure 3 is an enhancement of the RUF-CTXT condition proposed by An [2001] to allow adaptive adversarial behavior: The receiver is not given access to the token for a single, adaptively-chosen *challenge* time period; in addition, the adversary can choose any receiver public key in the encryption queries. We say that a TR-PKAE encryption is secure against RUF-TR-CTXT, if every polynomial-time adversary A has negligible advantage, $\mathrm{Adv}_{\mathrm{A},\Gamma}^{\mathrm{RUF\text{-}TR\text{-}CTXT}}(k)$, against the challenger in the RUF-TR-CTXT game. The game for RUF-CTXT is a natural simplification of RUF-TR-CTXT in which the receiver obtains the master secret (and thus the token queries are no longer required).

Third-Party Unforgeability (TUF-CTXT). In addition to timed-release receiver unforgeability, we also require a time-independent third-party unforgeability (TUF-CTXT) condition, which allows us to separate timed-release functionality from PKAE. Thus, in the TUF-CTXT game defined in Figure 3, the

[1]Since the receiver can generate the ciphertext allegedly coming from another user to himself, the receiver will not be able to prove to anybody that ciphertext was generated by the alleged sender even if all secret information is disclosed.

```
Algorithm 2.5: $\mathrm{Exp}^{\text{TUF-CTXT}}_{\mathcal{A},\Gamma}(k)$

$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_a, sk_a) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(pk_b, sk_b) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(c^*, T^*)$
    $\leftarrow \mathcal{A}^{\mathsf{Encrypt}^*(\pi, sk_a, pk_b, \cdot, \cdot)}(\pi, \delta, pk_a, pk_b)$
if $(\mathsf{Decrypt}^*(\pi, \delta, pk_a, sk_b, c^*, T^*) = \mathsf{fail}$
 or
$\mathsf{Encrypt}^*(\pi, sk_a, pk_b, \cdot, T^*)$ returned $c^*)$
  then return (false)
  else return (true)
```

```
Algorithm 2.6: $\mathrm{Exp}^{\text{RUF-TR-CTXT}}_{\mathcal{A},\Gamma}(k)$

$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_a, sk_a) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(c^*, T^*, pk_b^*, sk_b^*)$
    $\leftarrow \mathcal{A}^{\mathsf{TG}(\pi, \delta, \cdot), \mathsf{Encrypt}^*(\pi, \mathsf{sk_a}, \cdot, \cdot, \cdot)}(\pi, pk_a)$
if $(\mathsf{Decrypt}^*(\pi, \delta, pk_a, sk_b^*, c^*, T^*) = \mathsf{fail}$
 or $\mathsf{Encrypt}^*(\pi, sk_a, pk_b^*, \cdot, T^*)$ returned $c^*$
 or $(pk_b^*, sk_b^*) \notin [\mathsf{KeyGen}(1^k)]$
 or $\mathcal{A}$ queried $\mathrm{TG}(\mathrm{T}^*))$
  then return (false)
  else return (true)
```

$$\mathrm{Adv}^{\text{TUF-CTXT}}_{\mathcal{A},\Gamma}(k) = \Pr[\mathrm{Exp}^{\text{TUF-CTXT}}_{\mathcal{A},\Gamma}(k) = \mathsf{true}]\ .$$
$$\mathrm{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{A},\Gamma}(k) = \Pr[\mathrm{Exp}^{\text{RUF-TR-CTXT}}_{\mathcal{A},\Gamma}(k) = \mathsf{true}]\ .$$

Fig. 3. TR-PKAE security experiments for the TUF-CTXT and RUF-TR-CTXT games.

master key is given to the adversary. We say that a TR-PKAE scheme Γ is secure against TUF-CTXT if every polynomial time adversary A has negligible advantage, $\mathrm{Adv}^{\text{TUF-CTXT}}_{\mathsf{A},\Gamma}(k)$, in k.

3. STRONGLY KEY-INSULATED PUBLIC ENCRYPTION AND TIMED RELEASE

Despite similarities between SKIE-OTRU and TR-PKE notions mentioned before, there is a key difference between them. In the SKIE-OTRU setting, a helper is associated with at most one user, and cooperates exclusively with that user, whereas in the TR-PKE setting, it is assumed that many users may use the services of the TiPuS server, but the interaction between each user and the server will be minimal. This results in several operational differences: 1) *User and Master Key Generation*: in a TR-PKE scheme, they are generated independently, whereas in a SKIE-OTRU they are generated jointly; 2) *Dissemination of secrets per time period*: a SKIE scheme must use a secure channel to send the hsk_i to only one user, whereas the tokens generated by a TiPuS are assumed to be publicly disseminated; 3) *Security notion of "user compromise"*: a SKIE scheme's notion of "user compromise" is limited to chosen time periods and the keys are generated by the victim, whereas in TR-PKE's notion the attacker is the user herself and she can generate her public key adaptively (perhaps without necessarily knowing the corresponding secret key) in order to break timed-release confidentiality. The following theorem shows that despite these differences, these notions are essentially equivalent. Below we provide a sketch of the proof and refer the reader to Appendix 7.2 for more details.

THEOREM 3.1. *There exists a (chosen-ciphertext) secure timed-release public key encryption scheme if and only if there exists a secure strongly key-insulated public-key encryption scheme with optimal threshold that allows random-access key updates.*

More precisely, given a SKIE-OTRU, PKE[2] *and one-time signature DS_{one}, we can construct a TR-PKE with the following properties:*

—*Given an IND-CCA2 adversary* A *against TR-PKE, we can construct algorithms* B_1 *and* B_2 *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{IND\text{-}CCA2}_{\mathsf{A},TR\text{-}PKE}(k) \leq \frac{1}{2}Adv^{SUF\text{-}CMA}_{\mathsf{B}_1,DS_{one}}(k) + Adv^{IND\text{-}CCA2}_{\mathsf{B}_2,PKE}(k)$.

—*Given an IND-RTR-CCA2 adversary* A *against TR-PKE, we can construct algorithms* B_1 *and* B_2 *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{IND\text{-}RTR\text{-}CCA2}_{\mathsf{A},TR-PKE}(k) \leq \frac{1}{2}Adv^{SUF\text{-}CMA}_{\mathsf{B}_1,DS_{one}}(k) + Adv^{IND\text{-}KIE\text{-}CCA2}_{\mathsf{B}_2,SKIE\text{-}OTRU}(k)$.

Conversely, given a TR-PKE scheme, we can construct a SKIE-OTRU with the following properties:

—*Given an IND-S-CCA2 adversary* A *against SKIE-OTRU, we can construct an algorithm* B *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{IND\text{-}S\text{-}CCA2}_{\mathsf{A},SKIE\text{-}OTRU}(k) \leq Adv^{IND\text{-}CCA2}_{\mathsf{B},TR\text{-}PKE}(k)$.

—*Given an IND-KIE-CCA2 adversary* A *against SKIE-OTRU, we can construct an algorithm* B *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{IND\text{-}KIE\text{-}CCA2}_{\mathsf{A},SKIE\text{-}OTRU}(k) \leq Adv^{IND\text{-}RTR\text{-}CCA2}_{\mathsf{B},TR\text{-}PKE}(k)$.

PROOF. (Sketch) Suppose we have a secure TR-PKE scheme Γ = (Setup, TRSetup, TG, Encrypt, Decrypt). We construct a SKIE-OTRU scheme from Γ as follows. Set $\text{KG}(1^k) = ((\pi, pk), sk, \delta)$, where $(\pi, \delta) \leftarrow \text{TRSetup}(1^k)$ and $(pk, sk) \leftarrow \text{KeyGen}(\pi)$; $\text{HKU}((\pi, pk), \delta, i) = tkn_i$, where $tkn_i \leftarrow \text{TG}(\pi, \delta, i)$; $\text{UKU}(i, (\pi, pk), tkn_i, (sk, tkn_{i-1})) = (sk, tkn_i)$; $\text{RUKU}(i, j, (\pi, pk), tkn_i, (sk, tkn_j)) = (sk, tkn_i)$; $\text{Enc}((\pi, pk), m, i) = c$, where $c \leftarrow \text{Encrypt}(\pi, pk, m, i)$; and set $\text{Dec}(i, (\pi, pk), (sk, tkn_i), c) = \text{Decrypt}(\pi, sk, c, tkn_i)$. This scheme essentially makes the TiPuS server in TR-PKE scheme Γ into a helper for an SKIE-OTRU scheme.

It is easy to see that this scheme must be a secure SKIE-OTRU scheme. Suppose an IND-S-CCA2 attacker given access to $spk = (\pi, pk)$, $hsk = \delta$ and a decryption oracle can break the scheme; then it is easy to see that such an adversary can also be used to mount an IND-CCA2 attack on Γ, since these are exactly the resources given to an adversary in the IND-CCA2 game. Likewise, an IND-KIE-CCA2 adversary who can break the scheme given access to $spk = (\pi, pk)$, selected $(usk_i, hsk_i) = (sk, tkn_i)$ pairs, and a decryption oracle can easily be used to mount an IND-RTR-CCA2 attack on Γ: when the SKIE adversary makes a corruption request for stage i, the corresponding IND-RTR-CCA2 adversary queries its TG oracle for tkn_i and can forward (sk, tkn_i) to the SKIE adversary since the IND-RTR-CCA2 adversary gets sk as an input; all other queries made by the SKIE adversary can be passed directly to the corresponding oracles of the IND-RTR-CCA2 adversary. The security reduction statements follow trivially.

Now suppose we have a secure SKIE-OTRU scheme Σ. If Σ has the additional property that KG can be implemented as two independent keying algorithms that generate (pk_h, hsk) and (pk_u, usk), then it is straightforward to

[2]The PKE can be constructed from the SKIE-OTRU.

transform Σ into a TR-PKE scheme. Since we would not expect this property to hold in general, we work around this problem as follows. We know that by the existence of Σ there also exists an ordinary chosen-ciphertext secure PKC $\Pi = (\text{PKGen}, \text{PKEnc}, \text{PKDec})$.

The idea behind our construction is that TRSetup will sample $(spk, hsk, usk_0) \leftarrow \Sigma.\text{KG}(1^k)$ and set $\pi = spk$ and $\delta = (hsk, usk_0)$; KeyGen will sample $(pk, sk) \leftarrow \Pi.\text{PKGen}(1^k)$ and output (pk, sk). $\text{TG}(\pi, \delta, i)$ will compute $hsk_i = \text{HKU}(spk, hsk, i)$ and then use usk_0 and hsk_i to compute $tkn_i = usk_i = \text{RUKU}(i, 0, spk, usk_0, hsk_i)$. Encryption and decryption will use the multiple-encryption technique of Dodis and Katz [2005] with one-time signature scheme DS_{one}[3]. Applying the results of Dodis and Katz [2005], an IND-CCA2 attack on this scheme reduces to an IND-CCA2 attack on Π, while an IND-RTR-CCA2 attack (even when receiver chooses its public key adaptively) on this scheme reduces to an IND-KIE-CCA2 attack on Σ. □

4. GENERIC CONSTRUCTIONS OF TR-PKE AND TR-PKAE

We note that the previous theorem essentially gives a generic construction of TR-PKE based on a SKIE-OTRU scheme. Note that since, as mentioned previously, SKIE-OTRU and IBE have been shown to be equivalent, this gives a generic construction using any IBE scheme as well. Here we elaborate on this construction and show how to turn it into TR-PKAE.

The main idea of the generic TR-PKE ($\text{TR-PKE}_{\text{gen}}$) construction is to combine a PKE scheme[4] and IBE using multiple encryption. Note that naive multiple encryption fails to provide adaptive chosen-ciphertext security as was shown in Dodis and Katz [2005]. More specifically, suppose that messages are encrypted first for the receiver and then for the time. Then the time server, in the IND-CCA2 game, can win by removing the outer layer of encryption on the challenge ciphertext, re-encrypting for another time, and querying the decryption of this ciphertext. Similarly, if messages are first encrypted for the time and then for the receiver, the receiver can win in the IND-RTR-CCA2 game with a similar strategy.

Thus we need to be careful when combining encryptions and will use the approach to IND-CCA2 multiple encryption proposed by Dodis and Katz [2005]. As in the proof of Theorem 3.1, the resulting encryption scheme will be secure against IND-CCA2 and IND-RTR-CCA2 attacks. One can extend the $\text{TR-PKE}_{\text{gen}}$ to obtain $\text{TR-PKAE}_{\text{gen}}$, the generic TR-PKAE. For that purpose, one can use the Encrypt-then-Sign approach [An 2001], where the $\text{TR-PKE}_{\text{gen}}$ ciphertext is signed by the sender using SUF-CMA secure digital signature scheme DS.

[3]Specifically, to encrypt message m for time T, we: (1) pick $s_1 \leftarrow U_{|m|}$, and set $s_2 = m \oplus s_1$, (2) pick signing and verification keys (SK, VK) for the one-time signature scheme DS_{one}, (3) let $c_1 = \Sigma.\text{Enc}^{VK}(spk, s_1, T)$, $c_2 = \Pi.\text{PKEnc}^{VK}(pk, s_2)$, and (4) output $(VK, c_1, c_2, \text{Sig}(VK, (T, c_1, c_2)))$. Decryption follows the scheme of Dodis and Katz [2005], except that c_1 is decrypted using $tkn_T = usk_T$.

[4]Which may or may not be derived from the IBE scheme used.

Setup/TRSetup: Given security parameter $1^{k_{\text{IBE}}}$ for IBE, we run $\text{Setup}_{\text{IBE}}$ producing public parameters π_{IBE} and master secret δ_{IBE} which is kept secret by the TiPuS.

KeyGen: Each user u first runs $\text{KeyGen}_{\text{PKE}}$ on input of security parameter 1^k chosen by u to obtain public/private key pair $(pk_{\text{PKE}}, sk_{\text{PKE}})$, and then runs $\text{SigGen}_{\text{DS}}$ with (another) input 1^k to obtain the signing/verification key pair $(SK_{\text{DS}}, VK_{\text{DS}})$. The public key u in TR-PKAE is $pk = (pk_{\text{PKE}}, VK_{\text{DS}})$ and the private one is $sk = (sk_{\text{PKE}}, SK_{\text{DS}})$. During PKE operations, the pk_{PKE} and sk_{PKE} will be used, while during DS operations the SK_{DS} and VK_{DS} are used.

TG: On input the time encoding T, the central server TiPuS outputs secret key sk_T corresponding to identity T under the IBE.

Encrypt: Given the public key pk_b of receiver and secret key sk_a of the sender, message m and time encoding T, 1) pick a random string s_1 of the same size as m and set $s_2 = m \oplus s_1$. 2) Then pick signing and verification key pair (SK, VK) for one-time signature DS_{one}. 3) Compute $c_1 = \text{Encrypt}_{\text{IBE}}^{VK}(\pi_{\text{IBE}}, T, s_1)$ using label VK, compute $c_2 = \text{Encrypt}_{\text{PKE}}^{VK}(pk_b, s_2)$ using label VK. 4) Compute sender's signature $c_3 = \text{Sig}_{sk_a}(T, c_1, c_2, VK, pk_b)$. 5) The resulting ciphertext is $c = (VK, T, c_1, c_2, c_3, \text{Sig}_{SK}(T, c_1, c_2, c_3))$.

Decrypt: Given ciphertext $c = (VK, T, c_1, c_2, c_3, \text{Sig}_{SK}(T, c_1, c_2, c_3))$ encrypted using pk_b, sk_a and time T, one decrypts it as follows: 1) Verify one-time signature $\text{Sig}_{SK}(T, c_1, c_2, c_3)$ using VK; 2) Verify signature of the sender c_3 using pk_a; 3) obtain $tkn_T = sk_T$; 4) $\widehat{s_1} = \text{Decrypt}_{\text{IBE}}^{VK}(\pi_{\text{IBE}}, sk_T, c_1)$; 5) $\widehat{s_2} = \text{Decrypt}_{\text{PKE}}^{VK}(sk_b, c_2)$, 6) construct $\widehat{m} = \widehat{s_1} \oplus \widehat{s_2}$. If any of the steps fail, output "fail".

Fig. 4. The TR-PKAE$_{\text{gen}}$ scheme.

Due to similarity of TR-PKE$_{\text{gen}}$ and TR-PKAE$_{\text{gen}}$ constructions, below we immediately jump to the TR-PKAE$_{\text{gen}}$ construction. By removing sender's signature from the TR-PKAE$_{\text{gen}}$ mechanism, we immediately obtain the corresponding TR-PKE$_{\text{gen}}$.

4.1 TR-PKAE$_{\text{gen}}$: Generic TR-PKAE

The generic construction is shown in Figure 4. The approach is to use IBE for construction of timed-release encryption (TRE) and then encrypt the message using multiple encryption that combines TRE and PKE, which results in non-authenticated version. To obtain TR-PKAE, we remark that requiring the sender to simply sign the ciphertext will not produce an IND-KC-CCA2 or IND-RTR-KC-CCA2 secure scheme since in these games the adversary has access to the sender's secret key and thus may be able to generate a new signature on the challenge ciphertext and submit this modified challenge ciphertext to the decryption oracle; in the end, the adversary is able to decrypt the challenge ciphertext. To deal with this slight complication, prior to generation of the one-time signature, the sender signs the concatenation of the intermediate ciphertext, the one-time verification key and the receiver's public key. The one-time signature is then computed on the intermediate ciphertext and the sender's signature—this ensures that the adversary will need to compute another one-time signature with the same verification key in the previous attack.

THEOREM 4.1.1. *The generic TR-PKAE$_{gen}$ scheme is secure against IND-KC-CCA2, IND-RTR-KC-CCA2, TUF-CTXT and RUF-CTXT provided that the* *one-time signature scheme is SUF-CMA-secure, PKE is IND-CCA2-secure, IBE is IND-ID-CCA-secure and DS is SUF-CMA-secure.*

More precisely, given an IBE, PKE[5]*, signature mechanism DS and one-time signature DS_{one}, we can construct a TR-PKAE with the following properties:*

—*Given an IND-KC-CCA2 adversary* A *against TR-PKAE, we can construct algorithms* B_1 *and* B_2 *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{IND\text{-}KC\text{-}CCA2}_{\mathsf{A},TR\text{-}PKAE}(k) \leq \frac{1}{2}Adv^{SUF\text{-}CMA}_{\mathsf{B}_1,DS_{one}}(k) + Adv^{IND\text{-}CCA2}_{\mathsf{B}_2,PKE}(k)$.

—*Given an IND-RTR-KC-CCA2 adversary* A *against TR-PKAE, we can construct algorithms* B_1 *and* B_2 *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{IND\text{-}RTR\text{-}KC\text{-}CCA2}_{\mathsf{A},TR\text{-}PKAE}(k) \leq \frac{1}{2}Adv^{SUF\text{-}CMA}_{\mathsf{B}_1,DS_{one}}(k) + Adv^{IND\text{-}ID\text{-}CCA}_{\mathsf{B}_2,IBE}(k)$.

—*Given a RUF-CTXT adversary* A *against TR-PKAE that makes* q_e *encryption queries, we can construct algorithms* B_1 *and* B_2 *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{RUF\text{-}CTXT}_{\mathsf{A},TR\text{-}PKAE}(k) \leq q_e \cdot Adv^{SUF\text{-}CMA}_{\mathsf{B}_1,DS_{one}}(k) + Adv^{SUF\text{-}CMA}_{\mathsf{B}_2,DS}(k)$.

—*Given a TUF-CTXT adversary* A *against TR-PKAE that makes* q_e *encryption queries, we can construct algorithms* B_1 *and* B_2 *with run-time* $\mathsf{O}(Time(\mathsf{A}))$ *such that* $Adv^{TUF\text{-}CTXT}_{\mathsf{A},TR\text{-}PKAE}(k) \leq q_e \cdot Adv^{SUF\text{-}CMA}_{\mathsf{B}_1,DS_{one}}(k) + Adv^{SUF\text{-}CMA}_{\mathsf{B}_2,DS}(k)$.

PROOF. The proofs of IND-KC-CCA2 and IND-RTR-KC-CCA2 are very similar to the [SKIE-OTRU $\Rightarrow$ TR-PKE] proof of Theorem 3.1 (found in full in Appendix 7.2) and, thus, here we concentrate on the main ideas.

To show that the scheme is secure against IND-KC-CCA2, whenever adversary A makes a decryption query, we decrypt s_1 using the IBE master secret (after having verified the signatures) and forward c_2 for decryption to the PKE oracle. During the challenge phase, we pick s_1 at random, compute c_1 and submit $s_{2,1} = m_1 \oplus s_1$ and $s_{2,2} = m_2 \oplus s_1$ along with VK as the challenge parameters to the PKE challenger which encrypts one of $s_{2,i}$. We return the resulting complete ciphertext (with all required signatures) to A. Now suppose that the adversary submits a ciphertext (different from the challenge one) for decryption after the challenge. If the VK in the submitted ciphertext is the same as in the challenge, then either the adversary submits the challenge ciphertext (which is an invalid query) or he breaks SUF-CMA of the one-time signature scheme (in which case we return a random bit to PKE challenger). Otherwise, since VK is different from the challenge one, we can use the PKE decryption oracle without being forced to submit the challenge ciphertext that was returned by the PKE challenger. If A can guess which message was encrypted, then we automatically guess which message was encrypted by the PKE during the challenge, thus breaking IND-CCA2 security of PKE. Noting that, in case of SUF-CMA break, we win against the PKE with probability 1/2, the stated reductions follow.

To show that the scheme is secure against IND-RTR-KC-CCA2, note that 1) the token queries correspond to Extract queries in underlying IBE, and 2) for decryption queries we can use (to decrypt s_1) the IBE decryption oracle (where s_2 can be decrypted using the private key provided by the adversary). Also, during the challenge, we submit corresponding challenge to IBE similarly to the

[5]The PKE can be constructed from the IBE.

IND-KC-CCA2 approach. Once again, if after the challenge the (valid) decryption query submitted by the adversary has the same VK as in the challenge, then we break the SUF-CMA of the one-time signature (in which case we return a random bit). Otherwise, we can use the IBE decryption oracle even after the challenge. If the adversary manages to guess correctly which message was encrypted in the challenge, we automatically guess which message was encrypted by the IBE challenger, thus breaking IND-ID-CCA security of IBE. Reduction analysis stays the same as in the previous case.

The proofs of RUF-CTXT and TUF-CTXT are straightforward. Since RUF-CTXT security automatically implies TUF-CTXT [An 2001], we are left with a proof of RUF-CTXT security, where the adversary knows the timed-release secret but no longer knows the sender's secret key. The adversary has access to the encryption oracle where he can submit any m, T and pk_b. In the end, he returns the secret receiver key sk_b^*, time T^*, and ciphertext c^*, which should contain the sender's authentication. If the adversary manages to generate a new sender's signature in the returned ciphertext (either with the new input or a different signature with one of the inputs used during encryption queries), then we break the SUF-CMA security of the DS. Otherwise, the VK^*, T^*, c_1^*, c_2^*, c_3^*, and pk_b^* in the returned ciphertext should be the same as in one of the ciphertexts returned by the encryption oracle. Thus, if the returned ciphertext is different from the ones returned by the encryption oracle, the adversary has to break SUF-CMA of the one-time signature in order to win. The stated reductions follow easily. □

4.2 TR-PKE$_{\text{gen}}$: Generic TR-PKE

To obtain generic TR-PKE$_{\text{gen}}$, we can simply remove the parts in the TR-PKAE$_{\text{gen}}$ description where the sender signs with his secret key, i.e., users no longer have to generate their signing/verification keys, and we remove c_3 from the construction while leaving everything else intact. The proofs and the reductions stay the same as in TR-PKAE$_{\text{gen}}$, with obvious modifications.

THEOREM 4.2.1. *The generic TR-PKE$_{gen}$ construction is secure against IND-CCA2 and IND-RTR-CCA2 attacker provided that the IBE is IND-ID-CCA-secure, PKE is IND-CCA2-secure and the underlying one-time signature is SUF-CMA-secure.*

5. TR-PKAE$_{\text{BF}}$: TR-PKAE BASED ON A SINGLE PRIMITIVE

The generic construction TR-PKAE$_{\text{gen}}$ provides TR-PKAE with all required security. However, below we show how to construct a TR-PKAE that satisfies all of the above security requirements with the exception that RUF-CTXT is replaced by RUF-TR-CTXT, which is based on a single primitive and is nearly as efficient as BF-IBE scheme [Boneh and Franklin 2003]. We argue that in practical applications RUF-TR-CTXT is sufficient since the ciphertexts are submitted before designated time. Moreover, it is desirable for modern authenticated encryption to have one primitive that achieves the desired security properties [Boyen 2003]: such solutions generally allow for a more efficient scheme, tighter security bounds and more stringent security. We start with a review of the Bilinear Diffie-Hellman Problem.

5.1 Bilinear Diffie-Hellman Problem

Let G_1 and G_2 be two abelian groups of prime order q. We will use additive notation for the group operation in G_1 (where aP denotes P added a times for $P \in \mathsf{G}_1, a \in \mathsf{Z}_q$) and multiplicative notation for G_2 (g^a denotes the g multiplied a times for $g \in \mathsf{G}_2, a \in \mathsf{Z}_q$). Let $e : \mathsf{G}_1 \times \mathsf{G}_1 \to \mathsf{G}_2$ be an admissible bilinear map [Boneh and Franklin 2003]. The properties of the groups and constructions of e are explained in detail in [Boneh and Franklin 2003].

Let G be a *Bilinear Diffie-Hellman* (BDH) *Parameter Generator* [Boneh and Franklin 2003], i.e. a randomized algorithm that takes positive integer input k, runs in polynomial time in k and outputs prime q, descriptions of G_1, G_2 of order q, description of admissible bilinear map $e : \mathsf{G}_1 \times \mathsf{G}_1 \to \mathsf{G}_2$ along with polynomial deterministic algorithms for group operations and e computations. The advantage of algorithm A in solving the *computational* BDH Problem (BDHP) for G is defined as follows:

$$\begin{aligned}\mathrm{Adv}^{\mathrm{cbdh}}_{\mathsf{A},\mathsf{G}}(k) = \Pr[&\langle q, \mathsf{G}_1, \mathsf{G}_2, e\rangle \leftarrow \mathsf{G}(1^k), P \leftarrow_R \mathsf{G}_1, a, b, c \leftarrow_R \mathsf{Z}_q^* : \\ &\mathsf{A}(q, \mathsf{G}_1, \mathsf{G}_2, e, P, aP, bP, cP) = e(P, P)^{abc}]\end{aligned} \tag{1}$$

We say that G satisfies the *computational* BDH Assumption if for any randomized polynomial-time algorithm A and any polynomial $f \in \mathsf{Z}[x]$ we have $\mathrm{Adv}^{\mathrm{cbdh}}_{\mathsf{A},\mathsf{G}}(k) < 1/f(k)$ for sufficiently large k.

The advantage of algorithm A in solving the *decisional* BDHP [Boneh and Boyen 2004] for G is defined as follows:

$$\begin{aligned}\mathrm{Adv}^{\mathrm{dbdh}}_{\mathsf{A},\mathsf{G}}(k) = |&\Pr[\mathsf{A}(q, \mathsf{G}_1, \mathsf{G}_2, e, P, aP, bP, cP, e(P, P)^{abc}) = 0] \\ &- \Pr[\mathsf{A}(q, \mathsf{G}_1, \mathsf{G}_2, e, P, aP, bP, cP, T) = 0]|\end{aligned} \tag{2}$$

where the probabilities are taken over the experiment that draws $\langle q, \mathsf{G}_1, \mathsf{G}_2, e\rangle \leftarrow \mathsf{G}(1^k)$; $P \leftarrow_R \mathsf{G}_1$; $a, b, c \leftarrow_R \mathsf{Z}_q^*$; and $T \leftarrow_R \mathsf{G}_2$.

We say that G satisfies the *decisional* BDH (DBDH) Assumption if for any randomized polynomial-time algorithm A and any polynomial $f \in \mathsf{Z}[x]$ we have $\mathrm{Adv}^{\mathrm{dbdh}}_{\mathsf{A},\mathsf{G}}(k) < 1/f(k)$ for sufficiently large k.

We also introduce a decisional tripartite Diffie-Hellman (decisional TDHP) problem. A similar problem in the asymmetric setting was introduced in Laguillaumie et al. [2005]. In this problem the adversary is given once again the parameters generated by G, random $aP, bP, cP \in \mathsf{G}_1$ and $T \in \mathsf{G}_1$. The adversary has to decide if $T = abcP$. Even though decisional Diffie-Hellman is easy in G_1[6], it is still hard to compute abP and the bilinear map appears to be of little help in making the decision in TDHP. The advantage of algorithm A in solving the *decisional* TDHP for G is defined as follows:

$$\begin{aligned}\mathrm{Adv}^{\mathrm{dtdh}}_{\mathsf{A},\mathsf{G}}(k) = |&\Pr[\mathsf{A}(q, \mathsf{G}_1, \mathsf{G}_2, e, P, aP, bP, Q, abQ) = 0] \\ &- \Pr[\mathsf{A}(q, \mathsf{G}_1, \mathsf{G}_2, e, P, aP, bP, Q, T) = 0]|\end{aligned} \tag{3}$$

[6]Given random $aP, bP, Q \in \mathsf{G}_1$, one can check if $Q = abP$ by verifying equality $e(aP, bP) = e(Q, P)$.

where the probabilities are taken over the experiment that draws $(q, \mathsf{G}_1, \mathsf{G}_2, e) \leftarrow \mathsf{G}(1^k)$; $P, Q \leftarrow_R \mathsf{G}_1$; $a, b \leftarrow_R \mathsf{Z}_q^*$; and $T \leftarrow_R \mathsf{G}_1$.

We say that G satisfies the *decisional* TDH (DTDH) Assumption if for any randomized polynomial-time algorithm A and any polynomial $f \in \mathsf{Z}[x]$ we have $\mathrm{Adv}^{\mathrm{dtdh}}_{\mathsf{A},\mathsf{G}}(k) < 1/f(k)$ for sufficiently large k.

Note that DTDH assumption is stronger than DBDH. Namely, hardness of DTDH problem easily implies hardness of DBDH problem. The converse, however, is unknown.

Before we move on to the constructions, we must make a few final remarks. First, computational BDHP is harder than decisional BDHP. Second, hardness of computational BDHP automatically implies hardness of the computational Diffie-Hellman problem (CDHP) in G_1 and G_2. Also, hardness of the discrete logarithm problem (DLP) in G_1 implies hardness of DLP in in G_2 [Menezes et al. 1993]. However, we must remind the reader that the decisional Diffie-Hellman problem is easy in G_1.

5.2 Description of the Scheme

Let G be a *BDH Parameter Generator*. Figure 5 gives a complete description of our construction.[7] The symmetric encryption scheme used is a straightforward adaptation of the Fujisaki-Okamoto scheme [Fujisaki and Okamoto 1999]. We briefly demonstrate the consistency of the scheme before moving on to security considerations. Given ciphertext $c = \langle Q_1, Q_2, \sigma \oplus K, m \oplus H_5(\sigma)\rangle$ computed using sk_a, pk_b and T, we note that in the corresponding Decrypt computations we have 1) $\widehat{K} = K$ since $e(Q_2 + pk_a, sP_T + sk_b \cdot Q_1) = e(r_2P + sk_aP, sP_T + sk_b \cdot r_1P_T)$ $= e([r_2 + sk_a]P, [s + r_1 \cdot sk_b]P_T) = e([s + r_1 \cdot sk_b]P, [r_2 + sk_a]P_T) = e(P_{pub} + r_1 \cdot pk_b, [r_2 + sk_a]P_T)$, 3) as in Fujisaki-Okamoto, it follows that $\widehat{\sigma} = \sigma$, $\widehat{m} = m$ and 4) $Q_1 = H_3(\widehat{\sigma}, \widehat{m})P_T$ and $Q_2 = H_4(\widehat{\sigma}, \widehat{m})P$. Thus the original plaintext is retrieved.

5.3 Security of the Scheme

The following security results apply to TR-PKAE$_{\mathrm{bm}}$. The hash functions are modeled as random oracles [Bellare and Rogaway 1995]. Below we sketch the main ideas used in the proofs of IND-RTR-KC-CCA2 (receiver timed-release confidentiality) and RUF-TR-CTXT (receiver timed-release unforgeability), and refer the reader to Appendix A for full details. The proofs of IND-KC-CCA2 and TUF-CTXT are more straightforward and are given in Appendix B. First, we note the confidentiality properties of the proposed scheme.

THEOREM 5.3.1. *[IND-RTR-KC-CCA2] Let* A *be an IND-RTR-KC-CCA2 adversary that makes* q_d *decryption queries,* q_2 *queries to* H_2 *and* q_{tok} *queries to TG. Assume that* $Adv^{IND\text{-}RTR\text{-}KC\text{-}CCA2}_{\mathsf{A},TR\text{-}PKAE_{bm}}(k) \geq \epsilon$. *Then there exists an al-*

[7] As in Boneh and Franklin [2003], we can weaken the surjectivity assumption on hash function H_1. The security proofs and results will hold true with minor modifications. We skip the details and refer the reader to Boneh and Franklin [2003].

Setup: Given security parameter $k \in \mathbb{Z}^+$, the following steps are followed:

1: $\mathcal{G}$ takes k and generates a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ and arbitrary generator $P \in \mathbb{G}_1$.

2: The following cryptographic hash functions are chosen: 1) $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$, 2) $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some n, 3) $H_3, H_4 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and 4) $H_5 : \{0,1\}^n \to \{0,1\}^n$. These functions will be treated as random oracles in security considerations.

3: The message space is chosen to be $\mathcal{M} = \{0,1\}^n$ and the ciphertext space is $\mathcal{C} = \mathbb{G}_1^* \times \mathbb{G}_1^* \times \{0,1\}^n \times \{0,1\}^n$. The general system parameters are $\pi_g = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, H_i, i = 1...5\rangle$

TRSetup:

1: Choose $s \in_R \mathbb{Z}_q^*$ and set $P_{pub} = sP$.

2: The timed-release public system parameter is $\pi_{tr} = P_{pub}$ and the *master key* δ is $s \in \mathbb{Z}_q^*$. The combined public parameters are $\pi = \pi_g || \pi_{tr} = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_{pub}, H_i, i = 1...5\rangle$

KeyGen: Uniformly choose private key $sk = a \in \mathbb{Z}_q^*$, and compute the corresponding public key pk as $0 \neq aP \in \mathbb{G}_1^*$.

TG: On input the time encoding $T \in \{0,1\}^n$, output sP_T where $P_T = H_1(T)$

Encrypt: Given the private key sk_a of the sender, public key pk_b of receiver, plaintext $m \in \mathcal{M}$ and time encoding T, encryption is done as follows: 1) sample $\sigma \in_R \{0,1\}^n$, compute $r_1 = H_3(\sigma, m)$ and $r_2 = H_4(\sigma, m)$; set $Q_1 = r_1 P_T$ and $Q_2 = r_2 P$; 2) compute $\mathcal{L} = e(P_{pub} + r_1 \cdot pk_b, (r_2 + sk_a)P_T)$ and symmetric key $K = H_2(\mathcal{L})$ and 3) the ciphertext c is set to be $c = \langle Q_1, Q_2, \sigma \oplus K, m \oplus H_5(\sigma)\rangle$

Decrypt: Given ciphertext $c = \langle Q_1, Q_2, c_1, c_2\rangle$ encrypted using sk_a, pk_b and time T, one decrypts it as follows: (1) obtain $tkn_T = sP_T$; (2) $\widehat{K} = H_2(e(Q_2 + pk_a, sP_T + sk_b \cdot Q_1))$; 3) retrieve $\widehat{\sigma} = c_1 \oplus \widehat{K}$ and compute $\widehat{m} = c_2 \oplus H_5(\widehat{\sigma})$ and 4) verify that $Q_1 = H_3(\widehat{\sigma}, \widehat{m})P_T$ and $Q_2 = H_4(\widehat{\sigma}, \widehat{m})P$; if so, output $\widehat{m}$, otherwise output "fail".

Fig. 5. The TR-PKAE$_{\text{bm}}$ scheme.

gorithm B *that solves computational BDHP with advantage* $Adv^{cbdh}_{\mathsf{B},\mathsf{G}}(k) \geq \frac{1}{4q_2 \cdot \max(q_2, q_d)} \left[\frac{\epsilon}{e\cdot(1+q_{tok})}\right]^3$ *and running time* $\mathsf{O}(Time(\mathsf{A}))$*, where* $e = 2.71828....$

PROOF. Below we sketch the main idea of the proof. Let $a'P, b'P, c'P$ be the BDH parameters and our goal is to compute $e(P,P)^{a'b'c'}$. Since the adversary should know the sender's private key, we set $sk_a = a \in \mathsf{Z}_q^*$ and make it public. Let us write the bilinear map in the challenge ciphertext as $e(P_{pub} + r_1 \cdot pk_b, (r_2 + a)P_T) = e(P_{pub} + r_1 \cdot pk_b, r_2 P_T) \cdot [e(P_{pub}, P_T)e(pk_b, Q_1)]^a$. Note that, given a, anyone can compute the part $[e(P_{pub}, P_T)e(pk_b, Q_1)]^a$. On the other hand, when we examine $e(P_{pub} + r_1 \cdot pk_b, r_2 P_T)$, we note that we can use the adversary to solve a useful problem only if we do not know either r_1 or r_2 during the challenge. However, even if the adversary manages to compute the bilinear map (which we could not compute), it may not be trivial to extract pk_b from the bilinear map with the goal of solving BDHP. Let us set $P_{pub} = sP = b'P$ and suppose during the challenge time T we set $P_T = c'P$ (essentially the only time for which we cannot compute token sP_T). Let us choose r_1 in the normal way, but set $Q_2 = r_2 P = a'P$. Then the interesting portion of the challenge bilinear map becomes $e(P_{pub} + r_1 \cdot pk_b, r_2 P_T) = e(b'P, a'c'P)e(pk_b, r_2 P_T)^{r_1}$. Note that even if the adversary manages to compute this value we still cannot get rid of pk_b. However, let us run the simulation once again with the same random tapes, except that 1) the simulator's random tape changes right after the challenge ciphertext has been generated, and 2) in the challenge ciphertext everything stays the same as in the previous simulation except that r_1 is a different random number. Then if the adversary manages to compute a challenge bilinear map again, we will obtain the value of $e(b'P, a'c'P)e(pk_b, r_2P_T)^{r'_1}$ where $r_1 \neq r'_1$. Using $e(b'P, a'c'P)e(pk_b, r_2P_T)^{r_1}$ and $e(b'P, a'c'P)e(pk_b, r_2P_T)^{r'_1}$, we can easily extract $e(b'P, a'c'P)$ and thus solve the BDHP. □

THEOREM 5.3.2. *[IND-KC-CCA2] Let* A *be an IND-KC-CCA2 adversary that makes* q_2 *queries to* H_2*. Assume that* $Adv^{IND\text{-}KC\text{-}CCA2}_{\mathsf{A},TR\text{-}PKAE_{bm}}(k) \geq \epsilon$*. Then there exists an algorithm* B *that solves computational BDHP with advantage* $Adv^{cbdh}_{\mathsf{B},\mathsf{G}}(k) \geq \frac{2\epsilon}{q_2}$ *and running time* $\mathsf{O}(Time(\mathsf{A}))$*.*

The proposed protocol also satisfies the authentication properties specified in the previous section, i.e., TUF-CTXT and RUF-TR-CTXT.

THEOREM 5.3.3. *[RUF-TR-CTXT] Let* A *be a RUF-TR-CTXT adversary that makes* q_e *encryption queries,* q_2 *queries to* H_2*, and* q_{tok} *queries to* TG*, and let* $Adv^{RUF\text{-}TR\text{-}CTXT}_{\mathsf{A},TR\text{-}PKAE_{bm}}(k) \geq \epsilon$*. Then there exists an algorithm* B *with computational BDHP advantage* $Adv^{cbdh}_{\mathsf{B},\mathsf{G}}(k) \geq \frac{\epsilon}{2 \cdot q_2 \cdot q_e \cdot e \cdot (1+q_{tok})}$ *and running time* $\mathsf{O}(Time(\mathsf{A}))$*, where* $e = 2.71828\ldots$

PROOF. Below we sketch the main idea of the proof. Let $a'P, b'P, c'P$ be the BDH parameters and our goal is to compute $e(P, P)^{a'b'c'}$. Suppose the adversary manages to compute correctly a bilinear map using the adaptively chosen receiver's secret key b and time T. In this case, it can compute the bilinear map $e(P_{pub} + r_1 \cdot bP, (r_2 + sk_a)P_T)$, where now the adversary no longer knows sk_a. Let us rewrite the bilinear map as $e(P_{pub} + r_1 \cdot bP, (r_2 + sk_a)P_T) = e(P_{pub}, sk_aP_T) \cdot [e(P_{pub}, r_2P_T) \cdot e(Q_2 + pk_a, Q_1)^b]$. The second part of the bilinear map is easily computed using information provided by the adversary and the actual value of r_2 (obtained from the random oracles). To solve BDHP we can use the first part $e(P_{pub}, sk_aP_T)$ to our advantage by setting $P_{pub} = b'P$, $pk_a = a'P$ and $P_T = c'P$ (as before, essentially the only time we cannot compute token sP_T). In that case, if the adversary computes the bilinear map, we automatically obtain the value $e(b'P, a'c'P)$, *i.e.*, the solution to BDHP. □

THEOREM 5.3.4. *[TUF-CTXT] Let* A *be a TUF-CTXT adversary that makes* q_e *encryption queries and* q_2 *queries to* H_2*, and let* $Adv^{TUF\text{-}CTXT}_{\mathsf{A},TR\text{-}PKAE_{bm}}(k) \geq \epsilon$*. Then there exists an algorithm* B *with computational BDHP advantage* $Adv^{cbdh}_{\mathsf{B},\mathsf{G}}(k) \geq \frac{\epsilon}{2 \cdot q_e \cdot q_2}$ *and running time* $\mathsf{O}(Time(\mathsf{A}))$*.*

6. TR-PKE$_{\text{STD}}$: TR-PKE IN THE STANDARD MODEL

The generic TR-PKE$_{\text{gen}}$ construction can be shown to be secure in the standard model provided that the underlying primitives are also secure in the standard model. Although efficient and secure (in the standard model) PKE and signature schemes do exist, until recently all efficient IBE constructions were shown to be IND-ID-CCA-secure only in the random oracle model. The first efficient and fully secure IBE was constructed by Waters [2005]: the construction used a reduction from chosen-plaintext secure 2-level HIBE [Gentry and Silverberg 2002; Horwitz and Lynn 2002; Boneh et al 2005; Boyen and Waters 2006] (constructed using a semantically secure IBE from the same paper [Waters 2005][8]) to fully secure IBE using the techniques in Boneh et al. [2006]. Following this work, Kiltz and Galindo [2006] and Kiltz [2006] directly constructed the first efficient IBE scheme secure in the standard model without use of the HIBE reduction and instead combining the basic IBE scheme of Waters [2005] with techniques first described in Cramer and Shoup [1998]. More precisely, Kiltz and Galindo [2006] and Kiltz [2006] constructed a secure identity-based key encapsulation scheme [Cramer and Shoup 2003], which, together with the hybrid construction technique proposed in Shoup [2000], can be used to construct efficient and secure IBE in the standard model.

In this section we give an example TR-PKE scheme in the standard model which uses the same approach as in TR-PKAE$_{\mathrm{bm}}$. The scheme presented here is secure against *adaptive* IND-CCA2 for TR-PKE and secure against *non-adaptive* IND-RTR-CCA,[9] given hardness of the *decisional* BDHP. The difference between IND-RTR-CCA2 and IND-RTR-CCA is that in the latter the adversary no longer has access to decryption/token oracles once the challenge ciphertext has been generated. Granted that this is a weaker attack, in many practical scenarios this notion still provides sufficient security. Moreover, there is evidence of *adaptive* IND-RTR-CCA2 security for the proposed scheme, although reducing it to well-accepted standard hardness assumptions appears to be challenging. More precisely, we also show that the scheme is secure against *adaptive* IND-RTR-CCA2 provided *decisional* TDHP is hard.

The scheme presented here is an adaptation of the scheme in Kiltz and Galindo [2006]. In addition to hardness of decisional BDHP (and decisional TDHP for adaptive IND-RTR-CCA2), we also require existence of a target collision-resistant hash function $h : \mathsf{G}_1 \to \mathsf{Z}_q$ which can be efficiently built as shown in Boyen et al. [2005]. Briefly, for any polynomial-time A, $\mathrm{Adv}^{\mathrm{tcr}}_{\mathrm{A,h}}(k) = \Pr[x \leftarrow \mathsf{G}_1, y \leftarrow A(x) \; : \; h(y) = h(x) \wedge y \neq x]$ should be negligible, i.e., given random $x \in \mathsf{G}_1$ it should be hard to find $y \neq x$ such that $h(x) = h(y)$. Note that if h is injective (which is possible since both G_1 and Z_q are of order q) then it trivially satisfies this requirement. However, in practice standard cryptographic hash functions can also be used. We approach construction of TR-PKE$_{\mathrm{std}}$ as follows: first we construct a timed-release key encapsulation scheme and then we use the approach given in Shoup [2000] to provide fully functional encryption.

6.1 Description of the Scheme

Let G be a *BDH Parameter Generator*. Figure 6 gives a complete description of the key encapsulation scheme for TR-PKE$_{\mathrm{std}}$.[10] The encapsulation scheme is

[8]The construction was later improved in Naccache [2005] and Chatterjee and Sarkar [2005].

[9]i.e., timed-release security against the receiver under "lunchtime" chosen-ciphertext attacks.

[10]If only IND-RTR-CCA and IND-CCA2 are required, then one can simplify the encapsulation scheme further as follows: 1) generator P_3 is no longer required and the public key now is simply

very similar to that given in Kiltz and Galindo [2006] and therefore we refer the reader to the previous work for a consistency proof. Note that if any part of encapsulation is inconsistent, then decapsulation will produce a random result [Cramer and Shoup 2003; Kiltz and Galindo 2006]. More precisely, the following are the modifications to Kiltz and Galindo [2006]:

—The public key of the timed-release server is sP as in Waters [2005] (with private key sP_2), which is the same as Kiltz and Galindo [2006] except that they slightly simplify it. In addition to that, each receiver has a public/private key pair constructed in the same way, i.e., a receiver has public key (bP, bP_3) and private key bP_2 where bP_3 is used only for key verification. We denote $P_T = \mathbf{H}(T) = U' + \sum_i T_i V_i$ to be the hash of T rather than the identity, constructed in the same way as in Waters [2005].

—To encrypt for time T, we simply add the public keys sP of timed-release server and bP of receiver and encapsulate for time T the same way as Kiltz and Galindo [2006] encapsulates for identity $P_T = \mathbf{H}(T)$ with the public key $sP + bP$.

—To decapsulate, the secret information needed in Kiltz and Galindo [2006] is the identity decryption key $\{sP_2 + t\mathbf{H}(T), tP\}$. In our case, this information will be published by the Timed-Release server on date T. However, this is insufficient since the encapsulation was done using $sP + bP$ and not simply sP as the public key. This can be easily overcome by having receiver add bP_2 (his secret key) to $sP_2 + t\mathbf{H}(T)$ and obtain $\{(sP_2 + bP_2) + t\mathbf{H}(T), tP\}$ which is exactly the decryption key that would be required in Kiltz and Galindo [2006] to decapsulate information that was encapsulated using $sP + bP$.

—To provide security against adaptive IND-KEM-RTR-CCA2, (receiver timed-release confidentiality) we also slightly modify what goes inside the cryptographic hash function h. In Kiltz and Galindo [2006], the authors used $h(rP)$. In our case, we use $h(rP_T + bP)$ which makes it harder for an adversary to launch successful adaptive attacks.

6.2 Security of the Scheme

The security proofs are modifications of Waters [2005] and Kiltz and Galindo [2006] and are provided in Appendix C, where we discuss the relevant modifications and reduction analysis. The proofs are given with respect to notions IND-KEM-CCA2 and IND-KEM-RTR-CCA which are defined almost identically to IND-CCA2 and IND-RTR-CCA except that decryption queries are replaced by decapsulation queries; and during the challenge we compute an encapsulation and choose at random whether to give the adversary the encapsulated key or a random key; the adversary's goal is to decide which key was given.

$\widehat{b}P$; 2) instead of $h(rP_T + pk_1)$, one can compute $h(rP)$ with corresponding modifications in the decapsulation. One can show that the resulting encapsulation scheme is IND-KEM-RTR-CCA and IND-KEM-CCA2-secure. Using Shoup's hybrid approach [Shoup 2000], the resulting TR-PKE scheme is IND-RTR-CCA and IND-CCA2-secure. This saves two bilinear maps in the encryption, but the encapsulation scheme is demonstrably insecure against adaptive IND-KEM-RTR-CCA2.

Setup: Given security parameter $k \in \mathbb{Z}^+$, the following steps are followed

1: $\mathcal{G}$ takes k and generates a prime q, where $2^k < q < 2^{k+1}$, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ and arbitrary generators $P, P_2, P_3 \in \mathbb{G}_1$.

2: A cryptographic hash function $h : \mathbb{G}_1 \to \mathbb{Z}_q$ is chosen.

3: Random $U' \in \mathbb{G}_1$ and $V = (V_1, ..., V_n) \in \mathbb{G}_1^n$ are chosen. Also, random $L_1, L_2 \in \mathbb{G}_1$ are chosen. The general system parameters are $\pi_g = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, U', V, P, P_2, P_3, L_1, L_2, h\rangle$

TRSetup:

1: Choose $s \in_R \mathbb{Z}_q$ and set $P_{pub} = sP$. The private key is $Q = sP_2$.

2: The timed-release public system parameter is $\pi_{tr} = P_{pub}$ and the *master key* δ is Q. The combined public parameters are $\pi = \pi_g || \pi_{tr} = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, U', V, P, P_2, P_3, L_1, L_2, h, P_{pub}\rangle$.

KeyGen: Uniformly choose $\widehat{b} \in \mathbb{Z}_q^*$. The public key is $pk = \{0 \neq \widehat{b}P, \widehat{b}P_3\}$, and the private key is $\widehat{b}P_2$.

TG: On input the time encoding $T \in \{0,1\}^n$, pick random $t \in \mathbb{Z}_q^*$ and output $\{Q+tP_T, tP\}$, where $P_T = U' + \sum_i T_i \cdot V_i$ and $T_1||T_2...||T_n$ is bitwise representation of T.

Encapsulate: Given the public key $pk = \{pk_1, pk_2\}$ of the receiver and time encoding T, encapsulation is done as follows: 1) verify consistency of pk by checking $e(pk_1, P_3) = e(pk_2, P)$; 2) sample $r \in_R \mathbb{Z}_q$, compute $C_1 = r[h(rP_T + pk_1)L_1 + L_2]$, $C_2 = rP$ and $C_3 = rP_T$. The key encapsulated is $K = e(P_{pub} + pk_1, P_2)^r$, while the encapsulation is $C = \langle C_1, C_2, C_3\rangle$.

Decapsulate: Given encapsulation $C = \langle C_1, C_2, C_3\rangle$ using pk with corresponding private key $\widehat{b}P_2$ and time T, one decapsulates it as follows: (1) obtain $tkn_T = \{Q + tP_T, tP\} = \{d_1, d_2\}$; (2) pick random $r_1, r_2 \in \mathbb{Z}_q$ and compute the encapsulated key $\widehat{K} = \frac{e(C_2, d_1 + \widehat{b}P_2 + r_1[h(C_3+pk_1)L_1+L_2] + r_2 P_T)}{e(C_3, d_2 + r_2 P)\cdot e(r_1 P, C_1)}$.

Fig. 6. The TR-PKE$_{\text{std}}$ key encapsulation scheme in the standard model.

Below we sketch the proof of receiver timed-release confidentiality IND-KEM-RTR-CCA (and IND-KEM-RTR-CCA2 given hardness of decisional TDHP). One notable (but expected) property is that due to the extra binding in users' public keys, the decryption oracle in the simulation given in the proof no longer requires a receiver's private key.

THEOREM 6.2.1. *The TR-PKE$_{std}$ encapsulation scheme is IND-KEM-RTR-CCA secure, assuming that h is a target collision-resistant hash function and the decisional BDHP is hard. Moreover, the scheme is IND-KEM-RTR-CCA2-secure assuming also hardness of decisional TDHP.*

More precisely, let A_1 (resp, A_2) be a IND-KEM-RTR-CCA (resp, IND-KEM-RTR-CCA2) adversary for TR-PKE$_{std}$, with polynomial running time $t(k)$, that makes $p(k)$ decryption queries and has non-negligible advantage $\epsilon(k)$. Then there exist adversaries A_{bdh}, A_{hash}, and A_{tdh} such that

(1) $\mathrm{Adv}^{\text{IND-KEM-RTR-CCA}}_{\mathsf{A}_1,\text{TR-PKE}_{\text{std}}}(k) \leq \frac{2}{\lambda}\mathrm{Adv}^{\text{dbdh}}_{\mathsf{A}_{\text{bdh}},\mathsf{G}}(k) + \frac{4p(k)}{q\lambda} + 2\mathrm{Adv}^{\text{tcr}}_{\mathsf{A}_{\text{hash}},\text{h}}(k)$

(2) $\mathrm{Adv}^{\text{IND-KEM-RTR-CCA2}}_{\mathsf{A}_2,\text{TR-PKE}_{\text{std}}}(k) \leq \frac{2}{\lambda}\mathrm{Adv}^{\text{dbdh}}_{\mathsf{A}_{\text{bdh}},\mathsf{G}}(k) + \frac{4p(k)}{q\lambda} + 2\mathrm{Adv}^{\text{tcr}}_{\mathsf{A}_{\text{hash}},\text{h}}(k) + \frac{2}{\lambda}\mathrm{Adv}^{\text{dtdh}}_{\mathsf{A}_{\text{tdh}},\mathsf{G}}(k)$

Where $\lambda = \frac{1}{4(n+1)p(k)}$, Time$(\mathsf{A}_{\text{bdh}}) = \mathsf{O}(t(k) + \epsilon^{-2}(k)\log\epsilon\log\lambda/\lambda + p(k))$, Time$(\mathsf{A}_{\text{hash}}) = \mathsf{O}(t(k))$, and Time$(\mathsf{A}_{\text{tdh}}) = \mathsf{O}(t(k))$.

PROOF. (Sketch) Let aP, bP, cP, K be a DBDHP challenge, i.e., with equal probability K is random or $\mathsf{K} = e(P,P)^{abc}$. Due to the similarity between our scheme and that of Kiltz and Galindo [2006], most of the proof features carry over without significant changes. In particular, we set the system parameters so that $L_1 = aP$ and $L_2 = -h(cP) \cdot aP + d \cdot P$ (for random d), $Q = sP_2 = bL_1 = abP$ are constructed the same way.[11] To suit this setup, we set $P_2 = \alpha \cdot bP$ (for random α), $P_{pub} = sP = \alpha^{-1} \cdot aP$ and $P_3 = bP$. Also the distribution of U', V is constructed the same way as in the previous proof, so $\mathbf{H}(T) = P_T = x(T)P + y(T)L_1$, where the simulator knows $x(T)$ and $y(T)$.

As in Kiltz and Galindo [2006], we immediately abort (returning a random bit) when $y(T) = 0 \mod q$ during token queries, or when $y(T^*) \neq 0 \mod q$ for challenge time T^*. We follow the same technique for answering token queries as in Kiltz and Galindo [2006]. During decapsulation query after the challenge with $T \neq T^*$, we safely assume that a token query has been made for T and thus $y(T) \neq 0 \mod q$ in such cases. If the simulation did not abort, at the end we compute whether ArtAbort, or artificial abort, should be signaled using sampling as in Kiltz and Galindo [2006]. To do so, the simulator computes, by sampling, the probability that either $y(T) = 0 \mod q$ during token queries or $y(T^*) \neq 0 \mod q$ happen, keeping the adversarial view of the game the same as in the current game trace but varying other parameters. When ArtAbort is signaled, the simulator aborts and returns a random bit. The need to replicate this portion of the simulation introduces the dependence on $\epsilon(k)$ in the runtime of A_{bdh}.

The decapsulation queries are carried out similarly to Kiltz and Galindo [2006] except that now the adversary also submits receiver private key $\widehat{b}P_2$ (which is added to the token if needed). Suppose the adversary submits tuple $(T, C = \{C_1, rP, rP_T\})$ with consistent C to decapsulation oracle. There are two modifications that are introduced by our scheme, one is a new event and the other one is a technical modification:

—If $cP = rP_T + \widehat{b}P$ and $y(T) = 0 \mod q$, then we mark this event as CoAbort and abort returning a random bit.
—If $h(cP) \neq h(rP_T + \widehat{b}P)$, we return

$$K = e(C_1 - d \cdot rP, bP)^{(h(rP_T + \widehat{b}P) - h(cP))^{-1}} \cdot e(rP, \widehat{b}P_2)\,.$$

The challenge ciphertext is computed once again similarly to Kiltz and Galindo [2006], but a few technical modifications are made due to the presence of the receiver public key submitted by the adversary. Given challenge time T^* and public key $\{\widehat{b}^*P, \widehat{b}^*P_3 = \widehat{b}^* \cdot bP\}$, the challenge ciphertext is computed as follows (if $y(T^*) \neq 0 \mod q$ we abort returning a random bit). We choose $cP = r^*P_{T^*} + \widehat{b}^*P$, and create challenge

[11]One slight difference is that we use $Q = sP_2$ (with the public information $P_{pub} = sP$), the approach used by Waters. This does not affect the proof.

ciphertext $\{d \cdot \frac{cP-\widehat{b}^*P}{x(T^*)}, \frac{cP-\widehat{b}^*P}{x(T^*)}, cP - \widehat{b}^*P\}$ with corresponding session key $K^* = \mathsf{K}^{1/x(T^*)} \cdot e(\widehat{b}^*P, P_2)^{(c-\widehat{b}^*)/x(T^*)} \cdot e(-aP, \widehat{b}^* \cdot b\,P)^{1/x(T^*)}$. Note that $e(\widehat{b}^*P, P_2)^{(c-\widehat{b}^*)/x(T^*)} = e(P, \alpha \cdot \widehat{b}^*b\,P)^{(c-\widehat{b}^*)/x(T^*)} = e((c-\widehat{b}^*)P, \alpha \cdot \widehat{b}^*b\,P)^{1/x(T^*)}$.

Finally, we return to the DBDHP challenger the output of A.

We note that if we consider a non-adaptive adversary, the analysis of the CoAbort event is the same as in Kiltz and Galindo [2006], i.e., since the adversary obtains no information on cP until the challenge the probability that CoAbort occurs is less than $2p/q$. Consequently, all the analysis made in Kiltz and Galindo [2006] carries over automatically and IND-KEM-RTR-CCA security follows. The main thing to notice is that if in the DBDHP challenge, K is in fact the solution to the computational BDHP, then the challenge ciphertext is correct including the returned encapsulated key. If instead K is random then so is the returned encapsulated key. Thus, if the adversary can tell a random challenge encapsulated key from a real one, we will be able to solve decisional BDHP.

To show that the scheme is IND-KEM-RTR-CCA2 secure given hardness of decisional TDHP, we note that in the above scheme we only need to address the case when event CoAbort happens after the challenge. In this case, the adversary finds $rP, rP_{T^*}, \widehat{b}P, \widehat{b}P_2, \widehat{b}P_3$ such that $r^*P_{T^*} + \widehat{b}^*P = rP_{T^*} + \widehat{b}P$ and $\widehat{b} \neq \widehat{b}^*$.[12] Rewrite the equation as $r^*P_{T^*} + (\widehat{b}^* - \widehat{b})P = rP_{T^*}$. Let $\zeta P, \nu P, R, \mathsf{T}$ be a DTDH challenge where we have to decide if $\mathsf{T} = \zeta\nu R$. We start with the real IND-KEM-RTR-CCA2 game and change it as follows: we set U', V to random multiples of ζP and $P_3 = R$; in particular given any P_T we know κ such that $\kappa \cdot \zeta P = P_T$. Other than that the rest of the simulation proceeds exactly as in the real game. According to the above, we have $r^*P_{T^*} + (\widehat{b}^* - \widehat{b})P = rP_{T^*}$, or equivalently $\kappa \cdot r^*\zeta P + (\widehat{b}^* - \widehat{b})P = \kappa \cdot r\zeta P$ for some known κ. Dividing both sides by ζ, we obtain that we can compute $\frac{(\widehat{b}^*-\widehat{b})}{\zeta}P$. Next we compute $A_1 = e(\frac{(\widehat{b}^*-\widehat{b})}{\zeta}P, \mathsf{T})$ and $A_2 = e((\widehat{b}^* - \widehat{b})R, \nu P)$, since $R = P_3$ and $\widehat{b}^*P_3$ and $\widehat{b}P_3$ are part of the public keys. If and only if $A_1 = A_2$, then $\mathsf{T} = \zeta\nu Q$: indeed, if $\mathsf{T} = \zeta\nu Q$, then $A_1 = e((\widehat{b}^* - \widehat{b})P, \nu R) = e((\widehat{b}^* - \widehat{b})R, \nu P) = A_2$. Therefore we can solve decisional TDHP instances with exactly the probability of the event CoAbort after the challenge. □

The proof of the Theorem 6.2.2 below is more straightforward and we refer the reader to Appendix C.

THEOREM 6.2.2. *The TR-PKE$_{std}$ encapsulation scheme is IND-KEM-CCA2 secure under the assumption that h is a target-resistant collision hash function and the decisional BDHP is hard.*

[12]As a side note, one can easily show there can be only one such distinct query and rewinding the experiment, such that behavior changes only once $\mathcal{A}$ has submitted challenge parameters, will not change the value of $\widehat{b}$, for otherwise we will be able to solve the computational Diffie-Hellman problem in G_1.

Table I. Cost of Basic Operations

Function	Modulus (bits)	Exponent (bits)	Performance (msec)
RSA(Sig/Dec)	1024	1024	2.96
RSA(Ver/Enc)	1024	16 ($e = 2^{16} + 1$)	0.14
Scalar Mul in EC over F_p	160	160	2.23
MapToPoint	512	-	1.52
Pairing	512	160	18.15

More precisely, if there exists IND-KEM-CCA2 adversary A *with polynomial running time* $t(k)$ *that makes* $p(k)$ *decapsulation queries and has advantage* $\epsilon(k)$*, then there exist adversaries* A_{bdh} *and* A_{hash} *such that*

$$Adv^{IND\text{-}KEM\text{-}CCA2}_{\mathsf{A},TR\text{-}PKE_{std}}(k) \leq \frac{2}{\lambda}(Adv^{dbdh}_{\mathsf{A}_{bdh},\mathsf{G}}(k) + 2p/q) + 2Adv^{tcr}_{\mathsf{A}_{hash},h}(k) ,$$

where $\lambda = \frac{1}{4(n+1)p(k)}$, $Time(\mathsf{A}_{bdh}) = \mathsf{O}(t(k) + \epsilon^{-2}(k) \log \epsilon \log \lambda/\lambda + p(k))$ *and* $Time(\mathsf{A}_{hash}) = \mathsf{O}(t(k))$.

The following corollary is a direct application of Theorem 1 given in Shoup [2000].

COROLLARY 1. *Using Shoup's hybrid scheme [Shoup 2000] and the encapsulation scheme for TR-PKE*$_{std}$*, we obtain a TR-PKE scheme secure against IND-CCA2 and IND-RTR-CCA in the standard model, given the existence of a secure pseudo-random bit generator, target collision-resistant hash function h and the hardness of decisional BDHP. In addition, given the hardness of decisional TDHP, the resulting TR-PKE scheme is secure against IND-RTR-CCA2.*

7. DISCUSSION

In this article, we discussed several constructions of TR-PKE and TR-PKAE. Fully secure generic constructions of TR-PKE require chosen-ciphertext secure IBE, PKE and SUF-CMA-secure one-time signature scheme, while TR-PKAE requires in addition a SUF-CMA-secure digital signature scheme. The most efficient IBE schemes were constructed to be secure in the random oracle model [Boneh and Franklin 2003]. Table I shows the cost of basic operations involved in bilinear maps. The performance results were computed using Miracl library v.4.8.3 [Shamus Software Ltd.] with Tate pairing for the bilinear map and were all averaged over 10,000 runs, except that the RSA results were obtained by running OpenSSL v.0.9.8 *speed* command. The group G_1 was chosen to be a subgroup of order q in a super-singular elliptic curve E over F_p, where p is a 512 bit and q is a 160 bit prime. Group G_2 was a subgroup of a finite field of order 1024 bits. We used a P4-3.2 GHz "Northwood" (800MHz FSB) with 2GB of 400 MHz RAM desktop. Table II shows the comparison of BF-IBE, Kiltz-Galindo's IBE and single-primitive constructions of TR-PKAE$_{bm}$ and TR-PKE$_{std}$ proposed in this article. Note that TR-PKAE$_{bm}$ is only slightly

Table II. Complexity Comparison of Significant Operations

Function	# Bilinear Maps	# of Exp	# of MapToPoint
BF-IBE Enc/Dec	1/1	2/0	1/0
TR-PKAE$_{bm}$ Enc/Dec	1/1	4/3	1/0
Kiltz-Galindo IBE Enc/Dec	0/3	5/5	0/0
TR-PKE$_{std}$ Enc/Dec	1/3	5/5	0/0

more expensive than BF-IBE.[13] When we switch to IBE in the standard model, the encryption becomes less expensive while decryption is at least three times slower compared to BF-IBE. Extending Kiltz-Galindo's IBE to TR-PKE$_{\text{std}}$ adds additional bilinear map in the encryption[14], while decryption complexity stays the same. All these observations are natural since constructions in the random oracle model are generally more efficient and allow for rich functionality and extensions [Boyen 2003]. Efficient extensions of primitives in the standard model are much more challenging. Moreover, there is a security trade-off when one considers random oracles and the standard model: security in BF-IBE was shown under computational BDH assumption, while security in the standard model was under stronger decisional BDH assumption. If we make the even stronger mBDDH assumption, then one can reduce the cost of decryption down to two bilinear maps [Kiltz 2006].

7.1 Alternative Models for TR-PKE

Cathalo et al. [2005] introduced a slightly different model of timed-release public key encryption, in which a receiver's public key was bound to the public key of timed-release server. The constructed scheme (without authentication) was shown to be secure in the random oracle model, assuming hardness of Bilinear Diffie-Hellman Inversion problem [Boneh and Boyen 2004], with similar efficiency as the proposed TR-PKAE$_{\text{bm}}$.[15] However, binding the user's public key to a specific server violates our goal of separating timed-release servers from users and does not allow the sender to choose which servers will be used. In addition, Cathalo et al. [2005] introduce a stronger game for IND-RTR-CCA2 in which the receiver no longer supplies a private key during decryption queries. While this models a very strong attack, it is not clear how realistic such game

[13]However, when BF-IBE is extended to provide comparable functionality to TR-PKAE$_{\text{bm}}$, e.g., using generic constructions, we expect the resulting scheme to be at least as expensive in practice as the proposed protocol.

[14]Since the user public key in the TR-PKE$_{\text{std}}$ is independent of timed-release server, one can assume that public keys are preverified. Otherwise additional two bilinear maps are needed to verify consistency of public key. Moreover, one can precompute for a given sender and timed-release servers such that no bilinear maps are required in the encryption (this also applies to BF-IBE).

[15]Encryption requires verification of the receiver's public key which takes two bilinear map computations; provided verification can be done beforehand no bilinear map computation is needed in the encryption. Moreover, ciphertext is shorter by one point of $\mathbb{G}_1$.

is, since in a real game the decryption oracle needs a receiver's private key to decrypt ciphertext.[16]

7.2 Extensions

The proposed TR-PKAE$_{\text{bm}}$ (and the TR-PKE$_{\text{std}}$ along with generic TR-PKE/ TR-PKAE with well-known IBE constructions) allows for efficient use of timed-release encryption using multiple independent servers. In TR-PKAE$_{\text{bm}}$, given public keys $s_1P, ..., s_mP$ of m timed-release servers, the sender can simply encrypt using $sP = \sum_j s_{i_j}P$ while the receiver decrypts using $sP_T = \sum_j s_{i_j}P_T$. In this case, the receiver will be able to decrypt only when all chosen servers have published their tokens. This approach enhances security for timed-release confidentiality, since if the receiver fails to obtain even one token he will not be able to obtain any information about the encrypted plaintext. However, if a single chosen server fails, the receiver will not be able to decrypt at all. To address robustness issues, one can easily adapt Pedersen's distributed threshold protocol [Pederson 1991], in which the servers will have to perform a brief initial setup among themselves which allows them to compute their secret keys d_i such that given a threshold of d_i's one can reconstruct the master secret d of the group. After that, each server publishes d_iP and on day T publishes d_iP_T. Any user will be able to use this group of servers in a threshold manner by noting that a user can compute dP (dP_T) from any threshold number of d_iP's (d_iP_T respectively) using Lagrange multipliers. As long as at least a threshold number of servers publish their tokens, the receiver will be able to decrypt a valid ciphertext. Also, as long as fewer than the threshold of servers cooperate, the group master secret d stays secret. Note that using Lagrange coefficients, in the TR-PKAE$_{\text{bm}}$ the added complexity in decryption is a threshold number of multiplications/additions in $\mathbb{G}_1$, while encryption complexity stays the same (provided dP is published along with the description of the group of servers). Thus after the initial setup, the timed-release servers can function without interaction. To reduce complexity of decryption, the servers can take the job of computing dP_T onto themselves in which case the complexity of decryption is as in the base TR-PKAE$_{\text{bm}}$. Similar techniques can be applied to the TR-PKE$_{\text{std}}$ in which now the servers split the master secret Q, while complexity discussion stays the same as in TR-PKAE$_{\text{bm}}$.

Finally, the timed-release schemes proposed here and in the literature, require that past tokens be stored in a repository in case a user attempts to decrypt a message with designated time well in the past. As a result, the required storage for tokens grows linearly over time. Hierarchical IBE (HIBE) schemes such as the one proposed by Boneh et al. [Boneh et al 2005], when

[16]The main issue here is that of knowledge: if no one knows the private key and/or the encrypted plaintext, then decrypting without knowledge of the private key is unrealistic in the real game. If the adversary is asking the oracle to decrypt some intercepted ciphertext for which it does not know the private key, the adversarial ability to choose public keys is limited to those of existing users'. Such situations can be modeled by a simple modification of the IND-RTR-CCA2 game in which the challenger generates a random series of public/private key pairs and gives the public keys to the adversary.

used in the generic constructions, allow us to reduce the required token storage to $\mathsf{O}(\log^{3/2} T)$ at the expense of larger tokens (from which all previous tokens can be derived) and encryption/decryption complexity, where T is an upper bound on the number of time periods when tokens are published.

Appendix A. Full Proofs of Theorems 5.3.1 and 5.3.3

PROOF. [IND-RTR-KC-CCA2] Assume that we are given $\langle q, \mathsf{G}_1, \mathsf{G}_2, e\rangle$ (output by $\mathsf{G}(1^k)$) and a random instance of BDH parameters $\langle X, a'X, b'X, c'X\rangle$, where X is a generator of G_1. Below, we design an algorithm B that interacts with A by simulating a real game for the adversary in order to compute solution to BDHP $e(X, X)^{a'b'c'}$.

We choose $P = X$. When hash functions $H_1, H_2, ..., H_5$ are queried as random oracles, B will store the returned value in its database coupled with the query and repeated queries will retrieve answers from the database. The oracles $H_2, ..., H_5$ will return random answers, while the way queries to H_1 are handled will be specified later. Moreover, we also use a biased coin that with probability $\theta > 0$ returns 0 and otherwise returns 1. The optimal value of θ maximizes the probability P_θ that simulation does not quit during token queries or the challenge. We have $P_\theta \geq \theta^{q_{tok}} \cdot (1 - \theta)$, where q_{tok} is the number of token queries made by A, and the right-hand side is maximized when $\theta = 1 - 1/(q_{tok} + 1)$ with value $P_\theta \geq \frac{1}{e\cdot(1+q_{tok})}$, where $e = 2.71828....$ This value of θ and P_θ will be used in the proof.

During Setup, B chooses $P_{pub} = sP$ to be $b'P$ and generates a random sender's private key $sk_a = a \in \mathsf{Z}_q^*$. The adversary A is given the public parameters and the sender's secret key sk_a.

When A makes a query to H_1 for P_T, B chooses random $c_T \in \mathsf{Z}_q^*$, flips the biased coin and returns $c_T P$ if the coin outcome is 0. Otherwise, it returns $c_T \cdot c'P$. When A makes a token query for $tkn[T] = sP_T$, B queries H_1, obtains corresponding c_T and 1) returns $sH_1(T) = c_T(b'P)$ if $H_1(T) = c_T P$, 2) fails otherwise.

Consider decryption query prior to the challenge: A submits ciphertext $\langle T, b, Q_1, Q_2, c_1, c_2\rangle$, where b is the receiver's secret key, pk_a is the sender, and T, Q_i, c_1 and c_2 carry the same meaning as in the previous proofs. In this case, as before, B obtains r_1 and r_2 from the databases, which allows it to decrypt correctly.

During the challenge, A chooses two equal-sized plaintexts m_0,m_1; public key pk_{b^*}; and time T^*. *If $P_{T^*} = c_{T^*}P$ the simulator quits*. It is assumed that A did not query (nor will query in the future) for $tkn[T^*]$. For challenge ciphertext, B chooses arbitrary $\beta \in \{0, 1\}$ and σ, and sets $r_2 = r_2^* = a'$ while the value of $r_1 = r_1^*$ is computed in a normal way. Then it chooses the value of the bilinear map at random and composes the resulting ciphertext $c^* = \langle T^*, pk_{b^*}, Q_1^* = r_1^* P_{T^*}, Q_2^* = a'P, c_1^*, c_2^*\rangle$.

After the challenge, A has a choice to continue queries or to reply to the challenge. However, A is not allowed to query for decryption of c^* using T^* and sk_{b^*} corresponding to pk_{b^*}. Suppose A submits ciphertext $\langle T, b, Q_1, Q_2, c_1, c_2\rangle$ after the challenge. If $P_T = c_T P$, then B can compute the corresponding bilinear map and answer the query correctly. If $P_T = c_T \cdot c'P$, then we consider two cases separately:

—If $Q_2 = Q_2^*$, then we return false and enter the tuple (T, b) in the database D_{aux}. Note that for ciphertext to be valid we must have at least that $Q_1 = r_1^* P_T$ and the bilinear map was computed correctly. Thus the simulator's response is incorrect only when the adversary made a query to H_2 with the correct value of the bilinear map constructed with the following parameters: $r_1 = r_1^*$, $r_2 = r_2^*$, time T and the receiver's secret key b.

—If $Q_2 \neq Q_2^*$, then we should be able to obtain the corresponding value of r_2 and the input pair (σ, m) from the queries to H_3. Likewise, we can obtain the value of r_1. Going through routine checks and noting that we can now compute the correct value of the bilinear map, we can answer correctly the decryption query.

We will run the above simulation twice. The second simulation is run using the same random tape for the adversary and changing the random tape of the simulator immediately after the adversary made the selection during the challenge, but ensuring that the only possible difference in the challenge step is in the value of r_1^*. After each simulation the following post-simulation steps are taken:

—After the first simulation, 1) we pick at random a query Y_1 that an adversary made to H_2 and a random pair (T, b) from database D_{aux}, computing $B_1 = (Y_1/[e(Q_2^* + aP, r_1^* \cdot b \cdot P_T) \cdot e(b'P, aP_T)])^{c_T/c_{T^*}}$; note that if Y_1 is the correct value of the bilinear map corresponding to r_1^*, r_2^*, T and b, then B_1 is the solution to BDHP; 2) we also pick at random a query F_1 that an adversary made to H_2 (a possible value of the bilinear map used in the challenge) and output the pair $(F_1, r_{1,1}^*)$, where $r_{1,1}^*$ is the value r_1^* used in the challenge.

—After the second simulation, as in the first simulation, 1) we compute the value of a possible solution to BDHP using adversarial queries to H_2 and database D_{aux}, but we mark this value now as B_2; 2) we pick at random a query F_2 that an adversary made to H_2 (a possible value of the bilinear map used in the challenge) and output the pair $(F_2, r_{1,2}^*)$, where $r_{1,2}^*$ is the value r_1^* used in the challenge.

After both simulations have been run, we flip a coin. If the coin output is "heads", we pick at random either B_1 or B_2 as the solution to BDHP. If the coin output is "tails", we compute $Z = (F_1/F_2)^{(r_{1,1}^* - r_{1,2}^*)^{-1}}$, then compute $Y = F_1/[e(sP, aP_{T^*}) \cdot Z^{r_{1,1}^*}]$ which is taken to power $c_{T^*}^{-1}$. The final result is output as the solution to BDHP. Note that if F_i are indeed correct values of the challenge bilinear maps used in the simulations, then $Z = e(bP, (r_2^* + a)P_{T^*})$, $Y = e(sP, r_2^* P_{T^*})$ and the final result is indeed the correct value of the solution to BDHP.

Note that the simulations above fail to be indistinguishable from the real game when either the adversary makes a query to H_2 with the correct value of the challenge bilinear map, or when after the challenge the simulator incorrectly answers the decryption query (in the part where D_{aux} is updated). On the other hand, without making a query to H_2 with the correct value of the challenge bilinear map, the adversary cannot succeed more than with negligible probability.

Denote by P the probability that in a single run of the simulation either the adversary makes a query to H_2 with the correct value of the bilinear map corresponding to some entry in D_{aux} or corresponding to the challenge, and the simulation does not fail due to the biased coin. Then $P \geq 2\epsilon \cdot P_\theta$. Using well-known probabilistic lemma [Pointcheval and Stern 1996] (aka the forking lemma), the probability that this event happens in both simulations is at least $(P/2)^3$. Given that in one of the simulations adversary makes a query to H_2 with the correct value of the bilinear map corresponding to some entry in D_{aux}, probability that we output the solution to BDHP is at least $P_1 = \frac{1}{4 \cdot q_d \cdot q_2}$. Given that in one of the simulations the adversary makes a query to H_2 with the correct value of the bilinear map corresponding to the challenge in both simulations, the probability that we output the solution to BDHP is at least $P_2 = \frac{1}{4 \cdot q_2^2}$. It follows that the probability that the combined simulation above outputs correct solution to BDHP is at least $(P/2)^3 \cdot \min(P_1, P_2) \geq \frac{1}{4q_2 \cdot \max(q_2, q_d)} [\epsilon \cdot P_\theta]^3$. □

PROOF. [RUF-TR-CTXT] The preliminaries in this proof are the same as in IND-KC-RTR-CCA2. More precisely, we are once again given BDH parameters $\langle X, a'X, b'X, c'X \rangle$, we set $P = X$, the oracles $H_2, ..., H_5$ return random answers, and we use the same biased coin with the same value of θ which maximizes probability P_θ that simulation does not quit during forgery and token queries. Our goal is to compute $e(X, X)^{a'b'c'}$.

During Setup, B chooses $P_{pub} = sP$ to be $b'P$ and sets $pk_a = a'P$. The adversary A obtains public parameters and pk_a. In addition, B maintains database D_s updated during encryption queries.

When A queries H_1 for P_T, B chooses random $c_T \in \mathsf{Z}_q^*$, flips the biased coin and returns $c_T P$ if the coin outcome is 0. Otherwise, it returns $c_T \cdot c'P$. When A queries for token sP_T, B queries H_1, obtains corresponding c_T and returns $sH_1(T) = c_T(b'P)$ if $H_1(T) = c_T P$, 2), or the simulation fails and stops otherwise.

During encryption query, A submits T, m and $b\,P$. The simulator is expected to output the encryption of m using a' (sender) and $b\,P$ (receiver). If $P_T = c_T P$, B computes the ciphertext in a normal way. It chooses arbitrary σ, queries H_3 for r_1, H_4 for r_2, and then queries H_5 with input σ. Then it computes the bilinear map as $e(sP + r_1 \cdot b\,P, r_2 P_T + aP_T)$ by noting that $aP_T = a' \cdot c_T P = c_T \cdot a'P$. The corresponding query is made to H_2 and B returns resulting ciphertext. If $P_T = c_T \cdot c'P$, B chooses σ and computes r_2 in the normal way. Then it picks random r_1' and sets $r_1 P_T = r_1' P$, updating appropriately the database of H_2. The value of bilinear map is chosen at random and the resulting ciphertext is given to the adversary. The simulator enters the parameters $(T, r_2, r_1', b\,P)$ in database D_{aux}.

Finally, either simulation halts or the adversary outputs forged ciphertext $c^* = \langle Q_1^*, Q_2^*, T^*, c_1^*, c_2^* \rangle$ and the receiver secret key b^*, which will be used for verification. *If* $P_{T^*} = c_{T^*} P$, the simulation fails. The simulator flips a fair coin. Next we describe what one does depending on the outcome of the coin: if the output is “tails” and the description below does not have data to proceed, then the coin is set to “heads” and vice versa.

—If the coin outcome is “heads”, B picks a random entry $(T, r_2, r'_1, b P)$ from database D_{aux}, and picks at random a query Y that the adversary made to H_2. Then it computes $Z = e(r_1 \cdot b P, (r_2 + a)P_T) = e(b P, (r_2 + a) \cdot r_1 P_T) = e(r'_1 \cdot b P, r_2 P + aP)$, computes $Y/[e(sP, r_2 P_T) \cdot Z]$, and takes the result to power c_T^{-1}. The final result is output as the solution to BDHP. Note that if Y was the correct value of the bilinear map corresponding to $(T, r_2, r'_1, b P)$, then we do obtain the solution to BDHP.

—If the coin outcome is “tails”, B picks at random a query Y that the adversary made to H_2. If r_2^* for Q_2^* was not found in database of H_4, the simulation fails. If the actual value of r_1^* corresponding to Q_1^* was found in the database of H_4, then B computes $Z = e(sP, r_2^* P_{T^*}) \cdot e(r_2^* P + aP, r_1^* \cdot b \cdot P_T)$. Then it computes $(Y/Z)^{c_{T^*}^{-1}}$ and outputs the result as the solution to BDHP. If no actual value of r_1^* was found and there exists an entry $(T, r_2, r'_1, b P)$ in D_{aux} such that $r_1^* = r_1$ (where $r_1 = r'_1/c_T$), then as in the “heads” case we can compute $Z = e(r_1^* \cdot b^* P, (r_2^* + a)P_{T^*}) = e(b^* P, (r_2^* + a) \cdot r_1^* P_{T^*}) = e(r'_1 \cdot c_{T^*}/c_T \cdot b^* P, r_2^* P + aP)$, then compute $Y/[e(sP, r_2^* P_{T^*}) \cdot Z]$ and take the result to power $c_{T^*}^{-1}$ producing our solution to BDHP. Note that if Y was the correct value of the bilinear map in the forgery, then we do obtain the solution to BDHP.

Note that the simulation fails to be indistinguishable from the real game when the adversary makes a query to H_2 with a real value of one of the bilinear maps used in the encryption queries. However, in this case, the probability that we output correct solution to BDHP is at least $\frac{1}{2 \cdot q_2 \cdot q_e}$. If no such query was made and the forgery is correct then the probability that we solve BDHP is at least $\frac{1}{2 \cdot q_2}$.

Taking into account the probability of failure due to the biased coin and the advantage ϵ of the adversary in the real game, we obtain that the probability of outputting correct solution to BDHP is at least $\frac{\epsilon \cdot P_\theta}{2 \cdot q_2 \cdot q_e}$. □

Appendix B. Proofs of Theorems 5.3.2 and 5.3.4

For completeness, below we show the proofs of IND-KC-CCA2 and TUF-CTXT for the proposed single-primitive TR-PKAE$_{\text{bm}}$. As in IND-KC-TR-CCA2 and RUF-TR-CTXT, we are given $\langle q, \mathsf{G}_1, \mathsf{G}_2, e\rangle$ (output by $\mathsf{G}(1^k)$) and a random instance of BDH parameters $\langle X, a'X, b'X, c'X\rangle$, where X is a generator of G_1. Our goal is to compute $e(X, X)^{a'b'c'}$ using the adversarial advantage in the real games. Once again, we set $P = X$ and random oracles $H_2, \ldots, H_5$ return random answers, while the way queries to H_1 are handled will be specified later.

PROOF. [IND-KC-CCA2] During Setup, B chooses random master secret s and makes it public, receiver public key $pk_b = b'P$, and random $sk_a = a \in_R \mathsf{Z}_q^*$. The adversary A, in addition to public parameters, receives s, sk_a, and pk_b. In addition, B maintains database L of possible values of $e(P, P)^{a'b'c'}$ updated during decryption queries made after the challenge.

When A queries H_1 for P_T, B samples $c_T \in_R \mathbb{Z}_k$ and returns $P_T = c_T \cdot P$, storing the query T in the database coupled with c_T.

Consider the decryption query made by A before the challenge: A submits ciphertext $\langle T, Q_1, Q_2, c_1, c_2\rangle$, where c_1 denotes $\sigma \oplus K$, and c_2 denotes $m \oplus H_4(\sigma)$, Q_1 represents $r_1 P_T$, Q_2 represents $r_2 P$, $sk_a = a$ is the sender private key, and T is the designated time. In this case, B goes through the oracle databases, checks that relevant queries have been made, and that the ciphertext is consistent. In particular, it can retrieve the message m and verify that the bilinear map is the same as $e(sP, (r_2 + a)P_T) \cdot e([r_2 + a] \cdot b\,P, Q_1)$ using retrieved r_1, r_2. Should any step fail, false is returned. Otherwise, the message m is returned.

During selection, A chooses two equal-sized plaintexts m_0,m_1, and $T = T^*$. To generate challenge ciphertext,

—B chooses arbitrary $\beta \in \{0, 1\}$, and assigns $Q_1^* = r_1^* P_{T^*} = a' P$, $Q_2^* = r_2^* P = c' P$. Then B chooses σ^*, two random strings c_1^* and c_2^*, and composes and returns ciphertext $c^* = \langle T^*, Q_1^*, Q_2^*, c_1^*, c_2^*\rangle$.

—Next, B updates the databases. For database of H_3: B, instead of r_1^*, puts $Q_1^* = r_1^* P_{T^*}$ as a value (marked that it's already multiplied by P_T) and (σ^*, m_β) as the query. Similar steps are taken with respect to Q_2^* and the database of H_4. Next, B puts $m_\beta \oplus c_2^*$ as a value and σ^* as the query into database of H_5. If $H_1(T^*)$ was never queried, then the query is made. Finally, the database of H_2 is instructed never to return the corresponding value of $K = K^* = \sigma^* \oplus c_1^*$.

After the challenge ciphertext, A has a choice to continue queries or to reply to the challenge. However, A is not allowed to query for decryption of c^* using T^* chosen for the challenge. Now, when A submits ciphertext $\langle T, Q_1, Q_2, c_1, c_2\rangle$ for decryption, slightly different steps are carried out by B. First, B searches for r_1, r_2 corresponding to Q_1, Q_2 in databases of H_3, H_4. If either one is not found even in the form that we put during challenge, B simply returns false. If $Q_1 = Q_1^*$, $Q_2 = Q_2^*$, $T = T^*$ we also return false.[17] Otherwise,

—First suppose $Q_2 = Q_2^*$. If $Q_1 = Q_1^*$ and $T \neq T^*$, then if we cannot find the exact value of r_1 in the databases we return false. Otherwise, we note that $r_1 P_T = c_T r_1 P = a' P$ which allows us to compute the solution to BDHP. If $Q_1 \neq Q_1^*$, then we again return false if we cannot find the exact value of r_1. Otherwise, we can compute correctly the bilinear map used in the encryption as $e(sP + r_1 \cdot b\,P, aP_T + c_T \cdot Q_2)$ and thus answer the decryption query correctly.

—Now suppose that $Q_2 \neq Q_2^*$. In that case, if exact value of r_2 could not be retrieved, then false is returned. Otherwise, we can compute the bilinear map as $e((r_2 + a)P, sP_T)e((r_2 + a) \cdot b\,P, Q_1)$ and thus correctly answer the decryption query.

In the end, the simulation may either halt or A returns β. If during the above simulation we managed to compute the solution to BDHP, then that is

[17]Note that if $c_1 \neq c_1^*$, and adversary computed bilinear map correctly, then the correct query was made to H_2.

the value output by B. Otherwise, B outputs at random one of the queries that the adversary has made to H_2.

Note that if the simulation answers a decryption query incorrectly, then (up to probability of guessing) the adversary will have made a query to oracle H_2 with correct value of the challenge bilinear map, which will allow us to compute solution to BDHP. Simple computations show that the resulting advantage is at least $2\epsilon/q_2$. $\square$

PROOF. [TUF-CTXT] During Setup, B generates random master secret $s \in \mathsf{Z}_q^*$, sets public key of the receiver to be $pk_b = b'P$, and the public key of the sender to be $pk_a = a'P$. The adversary A receives public parameters, master secret s, pk_a and pk_b. In addition, B maintains database D_s updated during encryption queries.

When A makes a query to H_1 for P_T, B chooses random $c_T \in \mathsf{Z}_q^*$ and returns $c_T(c'P)$. Query T along with c_T are stored and replies for repeated queries use the database.

When A submits T and m for an encryption query, B chooses σ, r_1, and r_2 the same way as in the protocol. However, the value of the bilinear map is chosen at random. Other than that, the ciphertext is formed in a normal way and the databases are updated accordingly. Also B keeps the local database D_s in which it enters the information about the inputs of the bilinear map, i.e., T and which r_1,r_2 were chosen.

Finally, either A returns forged ciphertext $\langle T^*, Q_1^*, Q_2^*, c_1^*, c_2^* \rangle$ or simulation halts. Then B flips a coin. If D_s is empty and/or the adversary did not make any queries to H_2, then we reset the coin outcome to "tails". If no forgery was submitted then the coin outcome is changed to "heads". If coin outcome is "heads", B picks a random entry from database D_s, and obtains corresponding T, r_1, r_2. Then it picks the random adversarial query Y to H_2 and computes $Y/[e(r_2P + aP, sP_T) \cdot e(r_1 \cdot bP, r_2P_T)]]^{c_T^{-1} r_1^{-1}}$, which is output as the solution to BDHP. If the coin outcome is "tails", B first obtains corresponding values of r_1 and r_2 since they had to be queried from H_3 and H_4. Then it extracts $K = \sigma \oplus c_1^*$ and looks up the query Y that was made to H_2 and returned K. Then as in the previous case, a candidate solution to BDHP is computed.

The above simulation fails to be indistinguishable from a real game if the adversary correctly computed the bilinear map corresponding to one of the encryption queries and then made the correct query to the H_2 oracle. However, in this case, with probability $\frac{1}{2 \cdot q_e \cdot q_2}$, the simulation will output the correct solution to BDHP. Otherwise, the simulation is indistinguishable from a real game and if forgery is successful the simulation outputs the correct solution to BDHP. It follows that the BDHP solution is output with a probability of at least $\frac{\epsilon}{2 \cdot q_e \cdot q_2}$. $\square$

Appendix C. Proof of Theorems 6.2.2 and 6.2.1

PROOF. [IND-KEM-CCA2] Let aP, bP, cP, K be DBDHP challenge, i.e., with equal probability K is random or $\mathsf{K} = e(P, P)^{abc}$.

Set $m = 2p$ (where p is the maximum number of adversarial queries). The simulator randomly chooses $x_0, ..., x_n$ in Z_q, $y_0', y_1, ..., y_n$ in Z_m, k in Z_{n+1}, and sets $y_0 = q - km + y_0'$. Then set $U' = x_0P + y_0L_1$ and $V_i = x_iP + y_iL_1, i = 1, ..., n$. The simulator can compute P_T as $x(T)P + y(T)L_1$, where $x(T) = x_0 + \sum_{i=1}^{n} T_ix_i$ and $y(T) = y_0 + \sum_{i=1}^{n} T_iy_i$. Note that distribution of U', V is indistinguishable from random in the adversarial view.

Simulator B sets $L_1 = aP$, $L_2 = -h(cP + bP) \cdot aP + d \cdot P$ for random d, random $s \in \mathsf{Z}_q$. Also $P_2 = \alpha \cdot aP$ for random α, and $P_3 = \beta P$ for random β. The adversary A obtains public parameters, $Q = sP_2$ and receiver's public key $\{bP, bP_3 = \beta \cdot bP\}$.

Suppose tuple $(T, C = \{C_1, rP, rP_T\})$ is queried to the decapsulation oracle with consistent C. First consider the case when $h(cP + bP) = h(rP_T + bP)$:

—$cP \neq rP_T$, then we found collision in the hash function; we stop and return a random bit.

—$cP = rP_T$ and $y(T) = 0 \mod q$. In this case, we abort and return a random bit.

—$cP = rP_T$ and $y(T) \neq 0 \mod q$. Since $y(T) \neq 0 \mod q$ and $rP_T = x(T) \cdot rP + r \cdot y(T) \cdot aP$, we can compute $r \cdot aP$. Then the encapsulated key is $e(Q, rP) \cdot e(bP, \alpha \cdot raP)$, which we return to the adversary.

If none of the above cases happen during the decapsulation query, we compute response by noting that $r[h(C_2+bP)L_1+L_2] = r[([h(C_2+bP)-h(cP+bP))aP+dP]$ which allows us to compute $r \cdot aP$. Then the encapsulated key is $e(Q, rP) \cdot e(bP, \alpha \cdot raP)$, which we can compute.

The challenge ciphertext for challenge time T^* is computed as follows. If $y(T^*) \neq 0$, then we stop and return a random bit. Otherwise, we choose $cP = r^*P_{T^*} = x(T^*) \cdot r^*P$. The challenge ciphertext is $\{d/x(T^*) \cdot cP, 1/x(T^*) \cdot cP, cP\}$ with the corresponding encapsulated key $K^* = e(Q, 1/x(T^*) \cdot cP) \cdot \mathsf{K}^{\alpha/x(T^*)}$.

If the game did not abort before the end, at the end of the game we fix A's random tape and all values that it sees from the simulator during execution. In particular, set $ID^* = \{T^1, ..., T^{p_0}, T^*\}$, the set of times queried during decapsulations and challenge, is fixed. Note that $Y = (y_0', y_1, ..., y_n, k)$ can still be varied randomly without changing the adversarial view. Define event ForcedAbort to be the case when either 1) $y(T^i) = 0 \mod q$ for some i, or 2) $y(T^*) \neq 0 \mod q$. Set $\eta = \Pr_Y[\neg\text{ForcedAbort}] \geq \lambda = \frac{1}{4(n+1)p}$. If ForcedAbort did occur during the game, we abort and return a random bit. If no ForcedAbort has occurred, B determines a good estimate η' of η (the estimate is a function of $T^1, ..., T^{p_0}, T^*$). If $\eta' > \lambda$, then with probability $1 - \lambda/\eta'$ it stops and outputs random bit – this event is called ArtAbort, or artificial abort. Otherwise, we return to the DBDHP challenger the output of A.

The main thing to notice is that if the K is in fact the solution to computational BDHP, then the challenge ciphertext is correct, including the returned encapsulated key. If instead K is random then so is the returned encapsulated key. Thus, if the adversary can tell the random challenge encapsulated key from a real one, we will be able to solve decisional BDHP.

Next, we provide the necessary analysis. Consider a game X_1 where we know a, b, c and we answer all decapsulation queries correctly without any aborts. The adversarial advantage in this game is the same as in the real IND-KEM-CCA2 game. Next, let us call X_2 the game which is the same as X_1, except that the simulator aborts when a hash collision happens, in which case it returns a random bit. Let us set γ to be 0 if K is random, and $\gamma = 1$ if $\mathsf{K} = e(P, P)^{abc}$. Let us call γ' the output of adversary A. Denote by β' the output of the simulator for the DBDHP challenge. Using Difference Lemma, we obtain $|Pr_{\mathsf{X}_2}[\beta' = \gamma] - Pr_{\mathsf{X}_1}[\beta' = \gamma]| \leq Pr_{\mathsf{X}_2}[\text{HashCollision}]$.

Denote by X_3 the game which is the same as X_2 except that the simulator at the very end computes if ForcedAbort or ArtAbort happen, in which case it returns a random bit. Denote $\kappa_1 = Pr_{\mathsf{X}_3}[\neg\text{ForcedAbort} \wedge \neg\text{ArtAbort}|\gamma = \gamma']$ and $\kappa_2 = Pr_{\mathsf{X}_3}[\neg\text{ForcedAbort} \wedge \neg\text{ArtAbort}|\gamma \neq \gamma']$. Then, following absolutely the same analysis as in Kiltz and Galindo [2006], we have $Pr_{\mathsf{X}_3}[\beta' = \gamma] - 1/2 = 1/2 \cdot (Pr_{\mathsf{X}_2}[\beta' = \gamma] \cdot (\kappa_1 + \kappa_2) - \kappa_2)$. As in Kiltz and Galindo [2006], we have $|\kappa_i - \lambda| \leq \lambda\rho/4$, where $\lambda = \frac{1}{4(n+1)p}$. After additional manipulations, we obtain $|Pr_{\mathsf{X}_3}[\beta' = \gamma] - 1/2 - \lambda \cdot (Pr_{\mathsf{X}_2}[\beta' = \gamma] - 1/2)| \leq \lambda\rho/2$.

The value ρ will be determined later and is a parameter in how many samples the simulator takes during ArtAbort computations. We assume that $\mathsf{O}(\rho^{-2} \ln \rho \ln \lambda/\lambda)$ samples are taken which leads to the inequality $|\kappa_i - \lambda| \leq \lambda\rho/4$ used above. The run-time complexity of the simulator, then, is $\mathsf{O}(Time(\mathsf{A}) + \rho^{-2} \ln \rho \ln \lambda/\lambda + p)$.

Denote by X_4 the final game described above, which is the same as X_3 except that we immediately abort, returning a random bit, if $y(T^*) \neq 0 \mod q$ during the challenge or if we abort during the decapsulation query. In case $y(T^*) \neq 0 \mod q$, it does not matter if we abort immediately or at the very end—in both cases, the simulator will return the random bit. During decapsulation, abort happens when $cP = rP_T$ and $y(T) = 0 \mod q$:

—If this abort happens before the challenge, then the adversary does not have information about cP and equality $cP = rP_T$ can happen only due to random chance. Thus, the probability of such an abort is less than or equal to $2p/q$.

—If this abort happens after the challenge, two cases are possible. If $T = T^*$, then the queried ciphertext is the same as the challenge ciphertext and thus the query is invalid. If $T \neq T^*$, then the game X_3 would also return a random bit at the end during ForcedAbort check.

From the above comments, we conclude that $|Pr_{\mathsf{X}_3}[\beta' = \gamma] - Pr_{\mathsf{X}_4}[\beta' = \gamma]| \leq 2p/q$. Thus, combining all inequalities and equations, we have $|Pr_{\mathsf{X}_1}[\beta' = \gamma] - 1/2| \leq |Pr_{\mathsf{X}_2}[\beta' = \gamma] - 1/2| + Pr_{\mathsf{X}_2}[\text{HashCollision}] \leq \rho/2 + |Pr_{\mathsf{X}_3}[\beta' = \gamma] - 1/2|/\lambda + Pr_{\mathsf{X}_2}[\text{HashCollision}] \leq \rho/2 + (|Pr_{\mathsf{X}_4}[\beta' = \gamma] - 1/2| + 2p/q)/\lambda + Pr_{\mathsf{X}_2}[\text{HashCollision}]$. The run-time of X_2 and X_1 is $\mathsf{O}(Time(\mathsf{A}))$, and the run-time of X_4 is $\mathsf{O}(Time(\mathsf{A}) + \rho^{-2} \ln \rho \ln \lambda/\lambda + p)$.

Let A_{bdh} be the resulting algorithm that, using A, outputs the answer to DBDHP in the game X_4. And let A_{hash} denote the resulting algorithm that, using A, outputs the hash collision input y such that $cP \neq y$ and $h(cP) = h(y)$ in the game X_2. Since X_1 is indistinguishable from a real IND-KEM-CCA2 game, we obtain $\mathrm{Adv}^{\text{IND-KEM-CCA2}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}} \leq \rho/2 + (\mathrm{Adv}^{\text{dbdh}}_{\mathsf{A}_{\text{bdh}},\mathsf{G}} + 2p/q)/\lambda + \mathrm{Adv}^{\text{tcr}}_{\mathsf{A}_{\text{hash}},\text{h}}$, where the run-time of A_{hash} is $\mathsf{O}(Time(\mathsf{A}))$, and the run-time of A_{bdh} is $\mathsf{O}(Time(\mathsf{A}) + \rho^{-2} \ln \rho \ln \lambda/\lambda + p)$. The final conclusion of the Theorem follows by setting $\rho = \mathrm{Adv}^{\text{IND-KEM-CCA2}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}}$. □

PROOF. [IND-KEM-RTR-CCA(2)]

First, we describe construction of the simulation in full details. Let aP, bP, cP, K be DBDHP challenge, i.e., with equal probability K is random or $\mathsf{K} = e(P, P)^{abc}$.

Set $m = 2p$ (where p is the maximum number of adversarial queries). The simulator randomly chooses $x_0, ..., x_n$ in Z_q, $y_0', y_1, ..., y_n$ in Z_m, k in Z_{n+1} and sets $y_0 = q - km + y_0'$. Then set $U' = x_0P + y_0L_1$ and $V_i = x_iP + y_iL_1, i = 1, ..., n$. The simulator can compute P_T as $x(T)P + y(T)L_1$, where $x(T) = x_0 + \sum_{i=1}^{n} T_i x_i$ and $y(T) = y_0 + \sum_{i=1}^{n} T_i y_i$. Note that the distribution of U', V still looks random.

Simulator B sets $L_1 = aP$, $L_2 = -h(cP) \cdot aP + d \cdot P$ for random d, $Q = sP_2 = bL_1 = abP$. Also $P_2 = \alpha bP$ for random α, $P_3 = bP$ and $P_{pub} = sP = \alpha^{-1} \cdot aP$. The adversary A obtains public parameters only.

When queried for token for time T, if $y(T) = 0 \mod q$ we immediately abort and return a random bit. If $y(T) \neq 0 \mod q$, then, as in Kiltz and Galindo [2006], we pick random r', implicitly define $r = -b/y(T) + r'$, and compute the token as $\{-x(T)/y(T) \cdot bP + r'x(T)P + r'y(T)L_1, -1/y(T) \cdot bP + r'P\}$.

During a decapsulation query, we are given a tuple $(T, C = \{C_1, rP, rP_T\})$ with consistent C, where the adversary also submits the decryption key $\widehat{b}P_2$ and public key $\{0 \neq \widehat{b}P, \widehat{b}P_3\}$:

—If $h(cP) = h(rP_T + \widehat{b}P)$ and $y(T) = 0 \mod q$, then we abort and return a random bit. If $cP \neq rP_T + \widehat{b}P$, then we found a collision in the hash function. Otherwise, we mark this event as CoAbort.

—If $y(T) \neq 0 \mod q$, then decryption can be carried out using token for time T and the public/private keys submitted by the adversary.

—If $h(cP) \neq h(rP_T + \widehat{b}P)$, we return $K = e(C_1 - d \cdot rP, bP)^{(h(rP_T + \widehat{b}P) - h(cP))^{-1}} \cdot e(rP, \widehat{b}P_2)$.

The challenge ciphertext is computed once again similarly to Kiltz and Galindo [2006], but a few technical modifications are made due to presence of the receiver public key submitted by the adversary. Given challenge time T^* and public key $\{\widehat{b}^*P, \widehat{b}^*P_3 = \widehat{b}^* \cdot bP\}$, the challenge ciphertext is computed as follows. If $y(T^*) \neq 0 \mod q$ we immediately abort returning a random bit. Otherwise, we choose $cP = r^*P_{T^*} + \widehat{b}^*P$, and create challenge ciphertext $\{d \cdot \frac{cP - \widehat{b}^*P}{x(T^*)}, \frac{cP - \widehat{b}^*P}{x(T^*)}, cP - \widehat{b}^*P\}$ with the corresponding session key $K^* = \mathsf{K}^{1/x(T^*)} \cdot e(\widehat{b}^*P, P_2)^{(c-\widehat{b}^*)/x(T^*)} \cdot e(-aP, \widehat{b}^* \cdot bP)^{1/x(T^*)}$. Note that $e(\widehat{b}^*P, P_2)^{(c-\widehat{b}^*)/x(T^*)} = e(P, \alpha \cdot \widehat{b}^* bP)^{(c-\widehat{b}^*)/x(T^*)} = e((c - \widehat{b}^*)P, \alpha \cdot \widehat{b}^* bP)^{1/x(T^*)}$.

If the game did not abort before the end, at the end of the game we fix A's random tape and all values that it sees from the simulator during execution. In particular, set $ID^* = \{T^1, ..., T^{p_0}, T^*\}$, the set of times queried during decapsulations and challenge, is fixed. Note that $Y = (y'_0, y_1, ..., y_n, k)$ can still be varied randomly without changing adversarial view. Define event ForcedAbort to be the case when either 1) $y(T^i) = 0 \mod q$ for some i, or 2) $y(T^*) \neq 0 \mod q$. Set $\eta = \Pr_Y[\neg\text{ForcedAbort}] \geq \lambda = \frac{1}{4(n+1)p}$. If no ForcedAbort has occurred, B determines a good estimate η' of η (the estimate is a function of $T^1, ..., T^{p_0}, T^*$). If $\eta' > \lambda$, then with probability $1 - \lambda/\eta'$, it stops and outputs a random bit, this event is called ArtAbort, or artificial abort. Otherwise, we return to the DBDHP challenger the output of A.

The main thing to notice is that if the K is in fact the solution to computational BDHP, then the challenge ciphertext is correct including the returned encapsulated key. If instead K is random then so is the returned encapsulated key. Thus, if the adversary can tell the random challenge encapsulated key from a real one, we will be able to solve decisional BDHP.

In the analysis, we follow an approach similar to the one given in IND-KEM-CCA2. At first, we will consider adaptive and non-adaptive games together and then separate the analyses when needed. Consider a game X_1 where we know a, b, c and we answer all decapsulation and token queries correctly without any aborts. The adversarial advantage in this game is the same as in the real IND-KEM-RTR-CCA game. Next, let us call X_2 the game which is the same as X_1 except that the simulator aborts when hash collision happens, in which case it returns a random bit. Let us set γ to be 0 if K is random, and $\gamma = 1$ if $\mathsf{K} = e(P, P)^{abc}$. Let us call γ' the output of adversary A. Denote by β' the output of the simulator for the DBDHP challenge. Using Difference Lemma, we obtain $|Pr_{\mathsf{X}_2}[\beta' = \gamma] - Pr_{\mathsf{X}_1}[\beta' = \gamma]| \leq Pr_{\mathsf{X}_2}[\text{HashCollision}]$.

Denote by X_3 the game which is the same as X_2 except that the simulator at the very end computes if ForcedAbort or ArtAbort happen, in which case it returns a random bit. Denote $\kappa_1 = Pr_{\mathsf{X}_3}[\neg\text{ForcedAbort} \wedge \neg\text{ArtAbort}|\gamma = \gamma']$ and $\kappa_2 = Pr_{\mathsf{X}_3}[\neg\text{ForcedAbort} \wedge \neg\text{ArtAbort}|\gamma \neq \gamma']$. Then, following absolutely the same analysis as in Kiltz and Galindo [2006], we have $Pr_{\mathsf{X}_3}[\beta' = \gamma] - 1/2 = 1/2 \cdot (Pr_{\mathsf{X}_2}[\beta' = \gamma] \cdot (\kappa_1 + \kappa_2) - \kappa_2)$. As in Kiltz and Galindo [2006], we have $|\kappa_i - \lambda| \leq \lambda\rho/4$, where $\lambda = \frac{1}{4(n+1)p}$. After additional manipulations, we obtain $|Pr_{\mathsf{X}_3}[\beta' = \gamma] - 1/2 - \lambda \cdot (Pr_{\mathsf{X}_2}[\beta' = \gamma] - 1/2)| \leq \lambda\rho/2$.

The value ρ will be determined later and is a parameter in how many samples the simulator takes during ArtAbort computations. We assume that $\mathsf{O}(\rho^{-2} \ln \rho \ln \lambda/\lambda)$ samples are taken, which leads to the inequality $|\kappa_i - \lambda| \leq \lambda\rho/4$ used above. The run-time complexity of the simulator, then, is $\mathsf{O}(Time(\mathsf{A}) + \rho^{-2} \ln \rho \ln \lambda/\lambda + p)$.

Denote by X_4 the game, which is the same as X_3 except that the simulator immediately aborts returning a random bit when 1) a query T is made to the token oracle with $y(T) = 0 \mod q$, or 2) if $y(T^*) \neq 0 \mod q$. If $y(T^*) \neq 0 \mod q$ happens, then both X_4 and X_3 return a random bit (although X_3 does it at the end, which in this case does not matter). The same comment goes for the case $y(T) = 0 \mod q$ during token queries. Thus, $Pr_{\mathsf{X}_3}[\beta' = \gamma] = Pr_{\mathsf{X}_4}[\beta' = \gamma]$.

Denote by X_5 the final game, which is the same as X_4 except that we immediately abort and return a random bit when CoAbort happens. Using Difference Lemma, $Pr_{\mathsf{X}_4}[\beta' = \gamma] - Pr_{\mathsf{X}_5}[\beta' = \gamma]| \leq Pr_{\mathsf{X}_5}[\text{CoAbort}]$.

Combining all equations and inequalities, we have $|Pr_{\mathsf{X}_1}[\beta' = \gamma] - 1/2| \leq |Pr_{\mathsf{X}_2}[\beta' = \gamma] - 1/2| + Pr_{\mathsf{X}_2}[\text{HashCollision}] \leq \rho/2 + |Pr_{\mathsf{X}_4}[\beta' = \gamma] - 1/2|/\lambda + Pr_{\mathsf{X}_2}[\text{HashCollision}] \leq \rho/2 + (|Pr_{\mathsf{X}_5}[\beta' = \gamma] - 1/2| + Pr_{\mathsf{X}_5}[\text{CoAbort}])/\lambda + Pr_{\mathsf{X}_2}[\text{HashCollision}]$. The run-time of X_2 and X_1 is $\mathsf{O}(Time(\mathsf{A}))$, and the run-time of X_5 is $\mathsf{O}(Time(\mathsf{A}) + \rho^{-2} \ln \rho \ln \lambda/\lambda + p)$.

Let A_{bdh} be the resulting algorithm that, using A, outputs the answer to DBDHP in the game X_5. And let A_{hash} denote the resulting algorithm that, using A, outputs the hash collision input y such that $cP \neq y$ and $h(cP) = h(y)$ in the game X_2. Since X_1 is indistinguishable from a real IND-KEM-RTR-CCA game, we obtain $\text{Adv}^{\text{IND-KEM-RTR-CCA2}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}} \leq \rho/2 + (\text{Adv}^{\text{dbdh}}_{\mathsf{A}_{\text{bdh}},\mathsf{G}} + \text{Pr}_{\mathsf{X}_5}[\text{CoAbort}])/\lambda + \text{Adv}^{\text{tcr}}_{\mathsf{A}_{\text{hash}},\text{h}}$, where the run-time of A_{hash} is $\mathsf{O}(Time(\mathsf{A}))$, and the run-time of A_{bdh} is $\mathsf{O}(Time(\mathsf{A}) + \rho^{-2} \ln \rho \ln \lambda/\lambda + p)$.

In non-adaptive IND-KEM-RTR-CCA, the probability of CoAbort is less than $2p/q$, since the adversary obtains no information about cP until the challenge. Thus, we have $\text{Adv}^{\text{IND-KEM-RTR-CCA}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}} \leq \rho/2 + (\text{Adv}^{\text{dbdh}}_{\mathsf{A}_{\text{bdh}},\mathsf{G}} + 2p/q)/\lambda + \text{Adv}^{\text{tcr}}_{\mathsf{A}_{\text{hash}}}, \text{h}$. By setting $\epsilon = \text{Adv}^{\text{IND-KEM-RTR-CCA}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}}$ and $\rho = \epsilon$, we readily obtain the Theorem statement.

In adaptive IND-KEM-RTR-CCA2, the CoAbort may happen with higher probability after the challenge since the adversary knows cP. The probability before the challenge is the same as above. Thus, we concentrate on cases when CoAbort happens after the challenge. If in this event we have $T \neq T^*$, then we can assume that a query to the token query has been made with $y(T) = 0 \bmod q$, in which case we would abort in game X_4. Thus, the new event, which we call AcAbort, happens after the challenge and when $T = T^*$, $cP = rP_{T^*} + \widehat{b}\,P$ and the queried ciphertext is different from the challenge. In this case, the adversary finds $rP, rP_{T^*}, \widehat{b}\,P, \widehat{b}\,P_2, \widehat{b}\,P_3$ such that $r^*P_{T^*} + \widehat{b}^*P = rP_{T^*} + \widehat{b}\,P$ and $\widehat{b} \neq \widehat{b}^*$. We get the following inequality $\text{Adv}^{\text{IND-KEM-RTR-CCA2}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}} \leq \rho/2 + (\text{Adv}^{\text{dbdh}}_{\mathsf{A}_{\text{bdh}},\mathsf{G}} + 2p/q + Pr_{\mathsf{X}_5}[\text{AcAbort}])/\lambda + \text{Adv}^{\text{tcr}}_{\mathsf{A}_{\text{hash}},\text{h}}$.

Now, we design a game, which we will call Y, to estimate the probability of AcAbort. Let $\zeta P, \nu P, R, \mathsf{T}$ be the DTDH challenger where we have to decide if $\mathsf{T} = \zeta\nu R$. In game Y, we set $U' = \alpha \cdot \zeta P$ and $V = \beta \cdot \zeta P$ for some random α, β. We also set $P_3 = R$. We set the rest of parameters in this game in absolutely the same way as in a normal game. In particular, we now can compute any token and answer any decapsulation query. And we compute the challenge ciphertext in a normal way. Consider the case $r^*P_{T^*} + \widehat{b}^*P = rP_{T^*} + \widehat{b}\,P$ mentioned above and rewrite this equation as $r^*P_{T^*} + (\widehat{b}^* - \widehat{b})P = rP_{T^*}$, or equivalently $\kappa \cdot r^*\zeta P + (\widehat{b}^* - \widehat{b})P = \kappa \cdot r\zeta P$ for some known κ. Dividing both sides by ζ, we obtain that we can compute $\frac{(\widehat{b}^* - \widehat{b})}{\zeta}P$. Next we compute $A_1 = e(\frac{(\widehat{b}^* - \widehat{b})}{\zeta}P, \mathsf{T})$ and $A_2 = e((\widehat{b}^* - \widehat{b})R, \nu P)$, since $R = P_3$ and $\widehat{b}^*P_3$ and $\widehat{b}\,P_3$ are part of the public keys. If and only if $A_1 = A_2$, then $\mathsf{T} = \zeta\nu R$: indeed, if $\mathsf{T} = \zeta\nu R$, then $A_1 = e((\widehat{b}^* - \widehat{b})P, \nu R) = e((\widehat{b}^* - \widehat{b})R, \nu P) = A_2$. Therefore we can solve decisional TDHP with exactly the probability of the event AcAbort. We note that Y is indistinguishable from a real IND-KEM-RTR-CCA2 game, and the runs of X_5 where no abort (other than possibly AcAbort) happens are indistinguishable from Y in adversarial view. Thus, we have

Pr_{X_5}[AcAbort] $\leq$ Pr_{Y}[AcAbort], where Y runs in time $\mathsf{O}(Time(\mathsf{A}))$. Denote by $\mathsf{A}_{\mathrm{tdh}}$ the resulting adversary against decisional TDHP. Finally, we obtain $\mathrm{Adv}^{\text{IND-KEM-RTR-CCA2}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}} \leq \rho/2 + (\mathrm{Adv}^{\mathrm{dbdh}}_{\mathsf{A}_{\mathrm{bdh}}}, \mathsf{G} + 2p/q + \mathrm{Adv}^{\mathrm{dtdh}}_{\mathsf{A}_{\mathrm{tdh}},\mathsf{G}})/\lambda + \mathrm{Adv}^{\mathrm{tcr}}_{\mathsf{A}_{\mathrm{hash}},\mathrm{h}}$. The final conclusion of the Theorem follows by setting $\rho = \mathrm{Adv}^{\text{IND-KEM-RTR-CCA2}}_{\mathsf{A},\text{TR-PKE}_{\text{std}}}$. □

Appendix D. Key-Insulated Equivalence Proofs

There are two parts to proving equivalence of key-insulated encryption and timed-release: 1) construction of SKIE-OTRU from TR-PKE; and 2) construction of TR-PKE from SKIE-OTRU. As the first part is fairly trivial (including the resulting security proofs and reductions), in this section we concentrate only on the second part. The construction of TR-PKE from SKIE-OTRU is given in the main text and here we concentrate in more detail on the security proofs and reductions.

Let us consider adversary A against IND-CCA2 of the resulting TR-PKE scheme. In this case, the adversary knows the timed-release secret and thus can compute the helper tokens of the related SKIE-OTRU. We fix the receiver's public/private keys and give the receiver's public key to A. The adversary is also given access to the decryption oracle which decrypts using the above receiver's secret key and helper tokens. The adversary chooses adaptively time T^*, two equal-sized plaintexts and asks for IND-CCA2 challenge with these parameters and the above receiver's public key. The adversary can continue with decryption queries, except that it cannot ask for decryption of the challenge ciphertext unless $T \neq T^*$. In the end, A guesses which plaintext was encrypted and wins if the guess is correct.

Next, we design an algorithm B which plays a SUF-CMA game against the one-time signature challenger, and at the same time plays an IND-CCA2 game against the underlying PKC. The PKC gives us public key pk which we will use as the receiver's public key in the TR-PKE, and the $\mathrm{DS}_{\mathrm{one}}$ challenger provides us with a one-time signature verification key VK^*. When A gives us a decryption query, we decrypt using the SKIE-OTRU secrets, obtaining s_1. Then we query the PKC decryption oracle with c_2 and label VK obtaining s_2, allowing us to extract m. When A gives us challenge (T^*, m_0^*, m_1^*), we choose random s_1^*, compute $s_{2,i} = m_i^* \oplus s_1^*$ and submit to PKC the pair $(s_{2,0}, s_{2,1})$ and VK^* for challenge encryption. The PKC challenger chooses β and returns $c_2^* = \Pi.\mathrm{PKEnc}^{VK^*}(pk, s_{2,\beta})$. We set $c_1^* = \Sigma.\mathrm{Enc}^{VK^*}(spk,^* s_1, T^*)$, query the one-time signature oracle for signature $\mathrm{Sig}(VK^*, (T^*, c_1^*, c_2^*))$, and return the complete TR-PKE ciphertext to the adversary.

After the challenge, if the adversary queries the decryption oracle with $VK = VK^*$, and either $\mathrm{Sig}(VK^*, (T, c_1, c_2))$ is different from the challenge and/or $(T, c_1, c_2) \neq (T^*, c_1^*, c_2^*)$, then we return this signature to the $\mathrm{DS}_{\mathrm{one}}$ challenger, thus winning the SUF-CMA game. When we break SUF-CMA, we abort and return a random bit to the PKC challenger. If the adversary queries the decryption oracle with $VK \neq VK^*$, then we can legitimately use the PKC decryption oracle (since we will not be querying the PKC oracle with the label used in the PKC challenge). In the end (unless we already broke SUF-CMA), we return the response of A to the PKC challenger.

The adversarial advantage against TR-PKE is the same as our advantage against IND-CCA2 with respect to PKC, since essentially it was the PKC challenger who chose which plaintext to encrypt. Noting that, in case of SUF-CMA break, we win against PKC with probability exactly 1/2, we obtain that $\mathrm{Adv}^{\text{IND-CCA2}}_{\mathsf{A},\text{TR-PKE}}(k) \leq \frac{1}{2}\mathrm{Adv}^{\text{SUF-CMA}}_{\mathsf{B},\mathrm{DS}_{\mathrm{one}}}(k) + \mathrm{Adv}^{\text{IND-CCA2}}_{\mathsf{B},\text{PKE}}(k)$, where B runs with complexity of A. The algorithm B_1 against SUF-CMA is a modification of B, where B simulates the PKC challenger; similarly, algorithm B_2 against IND-CCA2 of PKC is a modification of B, where B simulates the SUF-CMA challenger. Note that the game from the view of A, the IND-CCA2 game against TR-PKE is indistinguishable from a real one.

Now let us consider adversary A against IND-RTR-CCA2 of TR-PKE. In this case, the adversary makes token queries and decryption queries. In the decryption queries, A also submits the secret key of the receiver. The adversary chooses adaptively time T^*, receiver's public key, two distinct equal-sized plaintexts, and asks for the IND-CCA2 challenge with these parameters. The adversary is not allowed to query the token for time T^*, and cannot ask for decryption of the challenge ciphertext unless either $T \neq T^*$ or the submitted secret key does not correspond to the challenge public key. The A guesses which plaintext was encrypted and wins if the guess is correct.

As in the case of IND-CCA2 against TR-PKE, we design an algorithm B which plays a SUF-CMA game against the one-time signature challenger, and at the same time plays an IND-KIE-CCA2 game against the underlying KIE. The KIE gives us public information *spk* (so that we can encrypt, with labels, in KIE any message for any time period), provides us access to its decryption oracle and allows us to query for usk_i.[18] The $\mathrm{DS}_{\mathrm{one}}$ challenger provides us with a one-time signature verification key VK^*.

When A requests a token, we simply ask the KIE oracle for the decryption key for the specified time. During decryption queries, we decrypt using supplied private/public key pair, and then ask the KIE to decrypt the rest. When the adversary submits $(pk^*, m_0^*, m_1^*, T^*)$ for the challenge, we choose random s_2^* and set $s_{1,i} = m_i^* \oplus s_2^*$. We submit $(s_{1,0}, s_{1,1}, T^*)$ and VK^* to the KIE challenger, which returns the $c_1^* = \Sigma.\mathrm{Enc}^{VK^*}(spk, s_{1,\beta}, T^*)$ for some randomly chosen β. We compute $c_2^* = \Pi.\mathrm{PKEnc}^{VK^*}(pk^*, s_2)$ using the public key pk^* that A has given us, and query the one-time signature oracle for signature $\mathrm{Sig}(VK^*, (T^*, c_1^*, c_2^*))$. The resulting challenge ciphertext is returned to A, which is a correct encryption of m_β^*.

After the challenge, if the adversary queries the decryption oracle with $VK = VK^*$, and either $\mathrm{Sig}(VK^*, (T, c_1, c_2))$ is different from the challenge and/or $(T, c_1, c_2) \neq (T^*, c_1^*, c_2^*)$, then we return this signature to the $\mathrm{DS}_{\mathrm{one}}$ challenger, thus winning the SUF-CMA game. When we break SUF-CMA, we abort and return a random bit to the KIE challenger. If the adversary queries the decryption oracle with $VK \neq VK^*$, then we can legitimately use the KIE decryption oracle to decrypt the c_1 part (since we will not be querying

[18]The restriction is that we cannot query for usk_i for the challenge period, and we cannot ask for decryption of the IND-KIE-CCA2 challenge ciphertext with the same parameters that were used during the challenge.

the KIE oracle with the label used in the IND-KIE-CCA2 challenge); we decrypt the rest of the ciphertext using the supplied receiver's secret key. In the end (unless we already broke SUF-CMA), we return the response of A to the KIE challenger. The adversarial advantage against TR-PKE is the same as our advantage against IND-KIE-CCA2, since essentially it was the KIE challenger who chose which plaintext to encrypt. Noting that, in case of SUF-CMA break, we win against KIE with probability exactly 1/2, we obtain that $\mathrm{Adv}^{\mathrm{IND\text{-}RTR\text{-}CCA2}}_{\mathsf{A},\mathrm{TR\text{-}PKE}}(k) \leq \frac{1}{2}\mathrm{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{B},\mathrm{DS}_{\mathrm{one}}}(k) + \mathrm{Adv}^{\mathrm{IND\text{-}KIE\text{-}CCA2}}_{\mathsf{B},\mathrm{SKIE\text{-}OTRU}}(k)$, where B runs with complexity of A. The algorithm B_1 against SUF-CMA is a modification of B, where B simulates the KIE challenger; similarly, algorithm B_2 against IND-KIE-CCA2 of KIE is a modification of B, where B simulates the SUF-CMA challenger. As before, the game IND-RTR-CCA2 that A plays here is indistinguishable from a real IND-RTR-CCA2 game. This finishes the proof.

ACKNOWLEDGMENT

The authors thank Moti Yung, Ran Canetti, Rosario Gennaro, Shai Halevi, and Hugo Krawczyk for many invaluable comments.

REFERENCES

Abdalla, M., Bellare, M., and Rogaway, P. 2001. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Cryptographer's Track at the RSA Conference*.

An, J. H. 2001. Authenticated encryption in the public-key setting: security notions and analyses. http://eprint.iacr.org/2001/079/.

Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. 1998. Relations among notions of security for public-key encryption schemes. In *Annual International Cryptology Conference (CRYPTO'98)*.

Bellare, M. and Goldwasser, S. 1996. Encapsulated key escrow. Tech. rep., Laboratory for Computer Science, MIT, TR-688.

Bellare, M. and Palacio, A. 2002. Protecting against key exposure: Strongly key-insulated encryption with optimal threshold. http://eprint.iacr.org/2002/064/.

Bellare, M. and Rogaway, P. 1995. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security (ACM CCS'95)*.

Blake, I. F. and Chan, A. C.-F. 2005. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. In *International Conference on Distributed Computing System (ICDCS'05)*.

Boneh, D. and Boyen, X. 2004. Efficient selective-ID secure identity based encryption without random oracles. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'04)*.

Boneh, D., Boyen, X., and Goh, E.-J. 2005. Hierarchical identity based encryption with constant size ciphertext. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'05)*.

Boneh, D., Canetti, R., Halevi, S., and Katz, J. 2006. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput*. To appear.

Boneh, D. and Franklin, M. 2003. Identity based encryption from the weil pairing. In *Annual International Cryptology Conference (CRYPTO'03)*.

Boneh, D. and Naor, M. 2000. Timed commitments. In *Annual International Cryptology Conference (CRYPTO'00)*.

Boyen, X. 2003. Multipurpose identity based signcryption: A swiss army knife for identity based cryptography. In *Annual International Cryptology Conference (CRYPTO'03)*.

BOYEN, X., MEI, Q., AND WATERS, B. 2005. Simple and eficient CCA2 security from IBE techniques. In *ACM Conference on Computer and Communications Security (ACM CCS'05)*.

BOYEN, X. AND WATERS, B. 2006. Anonymous hierarchical identity-based encryption (without random oracles). In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'06)*.

CATHALO, J., LIBERT, B., AND QUISQUATER, J.-J. 2005. Efficient and non-interactive timed-release encryption. In *International Conference on Information, Communications and Signal Processing (ICICS'05)*.

CHATTERJEE, S. AND SARKAR, P. 2005. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In *International Conference on Information Security and Cryptology (ICISC'05)*.

CHEN, L., HARRISON, K., SOLDERA, D., AND SMART, N. 2002. Applications of multiple trust authorities in pairing based cryptosystems. In *Proceedings of Infrastructure Security Conference*.

CHEON, J. H., HOPPER, N., KIM, Y., AND OSIPKOV, I. 2004. Authenticated key-insulated public key encryption and timed-release cryptography. http://eprint.iacr.org/2004/231.

CHEON, J. H., HOPPER, N., KIM, Y., AND OSIPKOV, I. 2006. Timed-release and key-insulated public key encryption. In *Financial Cryptography*.

CRAMER, R. AND SHOUP, V. 1998. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Annual International Cryptology Conference (CRYPTO'98)*.

CRAMER, R. AND SHOUP, V. 2003. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput. 33*.

CRESCENZO, G. D., OSTROVSKY, R., AND RAJAGOPALAN, S. 1999. Conditional oblivious transfer and timed-release encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'99)*.

DODIS, Y. AND KATZ, J. 2005. Chosen-ciphertext security of multiple encryption. In *Theory of Cryptography Conference*.

DODIS, Y., KATZ, J., XU, S., AND YUNG, M. 2002. Key-insulated public key cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'02)*.

DODIS, Y., KATZ, J., XU, S., AND YUNG, M. 2003. Strong key-insulated signature schemes. In *Conference on Theory and Practice of Public-Key Cryptography*.

FUJISAKI, E. AND OKAMOTO, T. 1999. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference (CRYPTO'99)*.

GARAY, J. AND POMERANCE, C. 2003. Timed fair exchange of arbitrary signatures. In *Financial Cryptography*.

GARAY, J. A. AND POMERANCE, C. 2002. Timed fair exchange of standard signatures. In *Financial Cryptography*.

GENTRY, C. AND SILVERBERG, A. 2002. Hierarchical ID-based cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'02)*.

HORWITZ, J. AND LYNN, B. 2002. Toward hierarchical identity-based encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'02)*.

KILTZ, E. 2006. Chosen-ciphertext secure identity-based encryption in the standard model with short ciphertexts. http://eprint.iacr.org/2006/122/.

KILTZ, E. AND GALINDO, D. 2006. Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. http://eprint.iacr.org/2006/034/.

LAGUILLAUMIE, F., PALLIER, P., AND VERGNAUD, D. 2005. Universally convertible directed signatures. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'05)*.

MARCO CASASSA MONT, K. H. AND SADLER, M. 2003. The HP time vault service: Exploiting IBE for timed release of confidential information. In *World Wide Web Consortium*.

MAY, T. 1993. Timed-release crypto. http://www.cyphernet.org/cyphernomicon/chapter14/14.5.html.

MENEZES, A., OKAMOTO, T., AND VANSTONE, S. 1993. Reducing elliptic curve logarithms to logarithms in a finite field. In *IEEE Trans. Inform. Theory IT-39, 5*.

NACCACHE, D. 2005. Secure and practical identity-based encryption. http://eprint.iacr.org/2005/369/.

PEDERSON, T. P. 1991. A threshold cryptosystem without a trusted party. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'91)*.

POINTCHEVAL, D. AND STERN, J. 1996. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'96)*.

RACKOFF, C. AND SIMON, D. R. 1991. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual International Cryptology Conference (CRYPTO'91)*.

RIVEST, R. L., SHAMIR, A., AND WAGNER, D. A. 1996. Time-lock puzzles and timed-release crypto. Tech. rep., Laboratory for Computer Science, MIT, TR-684.

SHAMUS SOFTWARE LTD. MIRACL: Multiprecision integer and rational arithmetic C/C++ library. http://indigo.ie/ mscott/.

SHOUP, V. 2000. Using hash functions as hedge against chosen ciphertext attack. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'00)*.

SHOUP, V. 2004. ISO 18033-2: An emerging standard for public-key encryption. http://shoup.net/iso/.

SYVERSON, P. F. 1998. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Computer Security Foundations Workshop*.

WATERS, B. 2005. Efficient identity-based encryption without random oracles. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'05)*.

Received September 2006; revised May 2007; accepted July 2007