Authenticated Key-Insulated Public Key Encryption and Timed-Release Cryptography

Jung Hee Cheon[1], Nick Hopper[2], Yongdae Kim[2], Ivan Osipkov[2]

[1] Seoul National University, Korea, jhcheon@math.snu.ac.kr
[2] University of Minnesota - Twin Cities, {hopper,kyd,osipkov}@cs.umn.edu

Abstract. In this paper we consider two security notions related to Identity-Based Encryption: Key-insulated public key encryption, introduced by Dodis, Katz, Xu and Yung; and Timed-Release Public Key cryptography, introduced independently by May and Rivest, Shamir and Wagner. We first formalize the notion of secure timed-release cryptography, and show that, despite several differences in its formulation, it is equivalent to strongly key-insulated public key encryption (with optimal threshold). Next, we introduce the concept of an authenticated timed-release cryptosystem, briefly consider generic constructions, and then give a construction based on a single primitive which is efficient and provably secure. Our construction also gives a strongly key-insulated authenticated encryption scheme.

Keywords: timed-release, authenticated encryption, key-insulated encryption

1 Introduction

Timed-Release cryptography. 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 [30], preventing early opening of votes in e-voting schemes, and delayed verification of a signed document, such as electronic lotteries [33] and check cashing. The problem of timed-release cryptography was first mentioned by May [23] and then discussed in detail by Rivest *et. al.* [30]. 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. The possible solutions fall into two categories:

– Time-lock puzzle approach. Alice encrypts her message and Bob needs to perform non-parallelizable computation without stopping for the required time to decrypt it.

– Agent-based approach. 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.

The first approach puts immense computational overhead on the message receiver, which makes it impractical for real-life scenarios. In addition, knowing the computational complexity of decryption, while giving us a lower bound on the 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 [10, 4, 33, 20, 19].

The agent-based approach, on the other hand, relieves Bob from performing non-stop 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 paper we concentrate on the agent-based approach. Several agent-based constructions were suggested by Rivest *et. al.* [30]. For example, the agent could encrypt messages on request with a secret key which will be published on a designated date by the agent. It also could precompute pairs of public/private keys, publish all public keys and release the private keys on the required days. A different scheme was proposed in [13], in which non-malleable encryption was used and receiver would engage in a conditional oblivious transfer protocol with the agent to decrypt the message. In [12], the authors proposed to use Boneh and Franklin's IBE scheme [9] for timed-release encryption: for that, one can replace the identity in an IBE scheme with the time of decryption. Similar proposals appear in [22, 6]. While some of these proposals contain informal proofs of security, none of them give a formal treatment of the security properties.

Since all known efficient constructions rely on the Boneh-Franklin IBE construction, a natural question to consider is if the existence of IBE is necessary for an efficient timed-release cryptosystem. In this paper, we formalize the security requirements of timed-release encryption and show that indeed this is the case: the existence of secure timed-release encryption is equivalent to the existence of strongly key-insulated encryption with optimal threshold; which in turn is known to imply the existence of IBE [5].

Strongly key-insulated encryption with Optimal Threshold. Strongly key-insulated encryption addresses the problem of computer intrusion by breaking up the lifetime of a public key into periods, and splitting the decryption key between the user (say, a mobile device) and a trusted "helper" (say, a desktop server) so that:

- 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.
- An adversary who is given access to usk_i for several time periods i cannot break the encryption for time periods j in which he has not obtained usk_j.
- An adversary given only the hsk cannot break the encryption scheme.

This notion, SKIE-OT, was introduced by Bellare and Palacio [5], who showed that without the 3rd requirement, key-insulated encryption is equivalent to Identity-Based Encryption. While key-insulated signature schemes and encryption schemes have appeared in the literature, to our knowledge no one has considered the notion of a strongly key-insulated authenticated encryption scheme.

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 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 paper, 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 [2, 14] 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 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 [12, 22, 6].

Our Contribution This paper proposes a new primitive that provides timed-release public key authenticated encryption (in short, TR-PKAE). The contribution of this paper is four fold:

- We give the first formal analysis of the security requirements for timed-release public key encryption (TR-PKE) and show that this notion is equivalent to SKIE-OT.
- We introduce the notion of TR-PKAE, as satisfying four notions: IND-CCA2, security against adaptive chosen ciphertext attacks under compromise of the timed-release agent and sender's private key; TUF-CTXT, or third-party unforgeability of ciphertexts; IND-RTR-CCA2, or receiver undecryptability before release time; and RUF-TR-CTXT, or receiver unforgeability before release time.
- We introduce a protocol that provides authenticated timed-release (or strongly key-insulated) public key encryption using a single primitive. The proposed protocol is essentially as efficient as Boneh and Franklin's chosen-ciphertext secure IBE scheme [9] in terms of computational and spatial complexity and is provably secure in the random oracle model. The proposed protocol requires minimal infrastructure (a single trusted agent) that can be shared among many applications and can be naturally converted to a threshold version, which provides robustness as well as stronger security by allowing outputs of multiple agents to be used.

In addition, we show how to efficiently adapt the resulting protocol to a Hierarchical Identity-Based Encryption scheme [8] to reduce the storage requirement of the supporting infrastructure.

Overview of our construction Consider a public agent (similar to NTP server [25]), called TiPuS (Timed-release Public Server), which at discrete time-intervals publishes new *self-authenticating* information $I_T = f(P_T, s)$ for current time T, where f and P_T are public, and s is secret. Alice can encrypt a message for Bob at time T using P_T, her private key and Bob's public key. *Only when I_T is published on day T*, will Bob be able to decrypt the message using I_T, his private key and Alice's public key.

We implement the above setting using an admissible bilinear map e (see Section 4.1), which along with the choice of groups and generator P is chosen independently of TiPuS. Each TiPuS chooses a secret $s \in \mathbb{Z}_q$ and publishes $P_{pub} = sP$. At time T, the TiPuS publishes $I_T = sP_T = sH(T)$ [1] (i.e. the private key for identity T in FullIdent), where H is a cryptographic hash function.

Let $(SK_A, PK_A) = (a, aP)$ and $(SK_B, PK_B) = (b, bP)$ be Alice's and Bob's authenticated private/public key pairs respectively. To *encrypt* message m for Bob, 1) Alice computes bilinear map $d = e(sP + bP, (r + a)P_T)$ for random r and applies hash function

[1] The value of I_T is *self-authenticating*: namely, each user can compute $e(P_{pub}, P_T)$ and verify that it is equal to $e(P, I_T)$, since by bilinearity $e(sP, H(T)) = e(P, sH(T)) = e(P, H(T))^s$.

H_2 to obtain $K = H_2(d)$, 2) she then encrypts message m as $E_K(m)$, where E_K is a symmetric encryption using key K. Bob also receives $r \cdot bP$. To *decrypt* the ciphertext, 1) Bob extracts rP from $r \cdot bP$, and having sP_T computes d as $e(rP + aP, sP_T + bP_T)$ [2], 2) applying hash function H_2, Bob computes K and uses it to decrypt $E_K(m)$.[3] The full detailed protocol and all required definitions/discussions are presented in later sections.

Note the following practical aspects already exhibited by the sketched scheme: 1) (*User Secret vs TiPuS Secret*) the secret value of TiPuS, system parameters and users' private keys are completely independent. It will be shown later that compromise of TiPuS does not jeopardize confidentiality and unforgeability of user ciphertexts; 2) (*Sharing*) the published value sP_T can be shared among multiple applications; 3) (*Scalability*) the protocol can take full advantage of a) several independent TiPuS's, [4] b) threshold generation of sP_T [27]. The increase in computational complexity is minimal when such schemes are applied to the protocol.

2 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 informal treatments given in previous work [23, 30, 12, 22, 6]; in particular they do not address the authentication requirements, which we add in section 3.

2.1 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.
- $\mathsf{TRSetup}$, 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).
- $\mathsf{TG}(\pi, \delta, T)$ computes the token corresponding to time T, tkn_T using (δ, π). This is the functionality performed by TiPuS which publishes tkn_T at time T.
- $\mathsf{Encrypt}(\pi, pk, m, T)$ computes the timed-release ciphertext c denoting the encryption with public key pk of message m with public parameters π and time encoding T.
- $\mathsf{Decrypt}(\pi, sk, \widehat{c}, tkn_T)$ outputs the plaintext corresponding to $\widehat{c}$ if decryption is successful or the special symbol `fail` otherwise.

[2] Note that according to properties of bilinear map, $e(rP + aP, sP_T + bP_T) = e((r+a)P, (s+b)P_T) = e((s+b)P, (r+a)P_T) = d$.

[3] We note that this scheme is somewhat similar to Bellare and Palacio's construction of an SKIE-OT scheme. Our results on the relationship between TR-PKE and SKIE-OT suggest this is not surprising; indeed, one can see our protocol as augmenting that scheme to add authentication.

[4] If s_iP is P_{pub} of the i-th token generator, then combined P_{pub} is $\sum s_iP$ and combined sP_T is $\sum s_iP_T$.

For consistency, we require that $\mathsf{Decrypt}(\pi, sk, \mathsf{Encrypt}(\pi, pk, m, T), \mathsf{TG}(\pi, \delta, T)) = m$, for all valid (pk, sk), (π, δ), T, and m,

2.2 Security

The introduction of TG and the master secret leads to the following question: how much should the security of the cryptosystem depend on these timed-release additions? Ideally, the cryptosystem should maintain receiver confidentiality properties even if the master secret is compromised. One of our goals (and a property of other recent proposals for timed-release encryption) is separation of the timed-release infrastructure from encryption security as much as possible. That is, the timed-release infrastructure should only affect the timed-release properties of the cryptosystem and not the PKE properties.

It is standard to require that the PKE cryptosystem be secure against adaptive chosen-ciphertext (IND-CCA2) adversaries [29, 3, 2]. This confidentiality must be provided even if all tokens tkn_T are given to the adversary, i.e. it should be time-independent. A stronger requirement is to demand IND-CCA2 security even if the master secret is out in the open. This requirement separates the timed-release infrastructure from the cryptosystem in the following sense: even if all master secrets are compromised, the receiver will still be guaranteed IND-CCA2 security against any third party. We model this attack by a slightly modified IND-CCA2 game, shown in Figure 1. Here, in addition to choosing two "challenge plaintexts" that the adversary will need to distinguish between, he also 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.

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

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

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

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

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

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

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

The timed-release functionality, i.e. stopping decryption until the designated time, 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. Therefore, any TR-PKE cryptosystem must provide some confidentiality guarantees 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 $\mathsf{IND\text{-}CCA}$ game by giving the adversary access to the receiver's secret key, arbitrary tokens $tkn_{T'}$, where $T' \neq T$, the "challenge time" chosen by the adversary, and a decryption oracle $\mathsf{Decrypt}^*(\pi, \delta, sk, \cdot, \cdot)$ which computes $\mathsf{Decrypt}(\pi, sk, \cdot, \mathsf{TG}(\pi, \delta, \cdot)$. The adversary may thus compute the decryption of any ciphertext for any time, *except* the challenge ciphertext in the challenge time. We say a timed-release public-key cryptosystem Γ is secure if every polynomial time adversary A has negligible advantages $\mathsf{Adv}_{A,\Gamma}^{\mathsf{IND-CCA2}}(k)$ and $\mathsf{Adv}_{A,\Gamma}^{\mathsf{IND-RTR-CCA2}}(k)$.

2.3 Strongly Key-Insulated Public Encryption and Timed-Release

The notion of key-insulated public key encryption has been discussed in [15, 16, 5]. As mentioned previously, Bellare and Palacio [5] have shown that the existence of (chosen-ciphertext) secure (not strongly) key-insulated encryption is a necessary and sufficient condition for the existence of secure IBE. Briefly, a strongly key-insulated encryption (SKIE) scheme is a quintuple of 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, j, pk, hsk_i, usk_j for $j \leq i$; 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.

A SKIE scheme is considered CCA-secure with optimal threshold if two conditions hold: (1) given access to pk, a decryption oracle, and as many pairs (hsk_i, usk_i) of his choosing, an adversary cannot break the encryption scheme for a stage j for which he has not been given hsk_j; and (2) given pk, hsk, and a decryption oracle, an adversary cannot break the encryption scheme for any stage. (For formal statements of these requirements, see [15, 16, 5]. The idea of separation of the timed-release master and user secrets in a timed-release cryptosystem 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 resemble the conditions $\mathsf{IND\text{-}CCA2}$ and $\mathsf{IND\text{-}RTR\text{-}CCA2}$ developed here.

However, there is a key difference between the SKIE and TR-PKE notions. In the SKIE 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; for instance, in a TR-PKE scheme, the user and master keys are generated independently, whereas in a SKIE scheme, they are generated jointly. Another difference is in the 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. Finally, the "user compromise" security notions for each scheme differ slightly, in that a SKIE scheme's notion of user compromise is limited to chosen time periods, whereas a TR-PKE's notion of user compromise gives away the user's secret.

The following theorem shows that despite these differences, these notions are essentially equivalent.

Theorem 1. *There exists a (chosen-ciphertext) secure timed-release public key cryptosystem if and only if there exists a secure strongly key-insulated public-key encryption scheme with optimal threshold that allows random-access key updates.*

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

It is easy to see that this scheme must be a secure SKIE-OT scheme. Suppose an 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 $\mathsf{IND\text{-}CCA2}$ attack on $\mathit{\Gamma}$, since these are exactly the resources given to an adversary in the $\mathsf{IND\text{-}CCA2}$ game. Likewise, an 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 $\mathsf{IND\text{-}RTR\text{-}CCA2}$ attack on $\mathit{\Gamma}$: when the SKIE adversary makes a corruption request for stage i, the corresponding $\mathsf{RTR\text{-}CCA2}$ adversary queries its TG oracle for tkn_i and can forward (sk, tkn_i) to the SKIE adversary since the $\mathsf{RTR\text{-}CCA2}$ adversary gets sk as an input; all other queries made by the SKIE adversary can be passed directly to the corresponding oracles of the $\mathsf{RTR\text{-}CCA2}$ adversary.

Now suppose we have a secure SKIE-OT scheme with random access updates, $\mathit{\Sigma}$. If $\mathit{\Sigma}$ 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 transform $\mathit{\Sigma}$ 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 $\mathit{\Sigma}$ there also exists an ordinary chosen-ciphertext secure PKC with labels $\mathit{\Pi} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$; and we can use standard techniques such as those proposed by Shoup [32] to add labels to $\mathit{\Sigma}$. The idea behind our construction is that **TRSetup** will sample $(spk, hsk, usk_0) \leftarrow \mathit{\Sigma}.\mathsf{KG}(1^k)$ and set $\pi = spk$ and $\delta = (hsk, usk_0)$; **KeyGen** will sample $(pk, sk) \leftarrow \mathit{\Pi}.\mathsf{PKGen}(1^k)$ and output (pk, sk). $\mathsf{TG}(\pi, \delta, \mathsf{i})$ will first compute $hsk_i = \mathsf{HKU}(spk, hsk, i)$ and then use usk_0 and hsk_i to compute $tkn_i = usk_i = \mathsf{UKU}(i, 0, spk, usk_0, hsk_i)$. Encryption and Decryption will use the multiple-encryption technique of Dodis and Katz [14].[5] Applying the results of [14], an $\mathsf{IND\text{-}CCA2}$ attack on this scheme reduces to a chosen-ciphertext attack on $\mathit{\Pi}$, while an $\mathsf{IND\text{-}RTR\text{-}CCA2}$ attack on this scheme reduces to an SKIE chosen-ciphertext attack on $\mathit{\Sigma}$.

[5] 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 a one-time signature scheme, (3) let $c_1 = \mathit{\Sigma}.\mathsf{Enc}^{VK}(spk, s_1, T)$, $c_2 = \mathit{\Pi}.\mathsf{PKEnc}^{VK}(pk, s_2)$, and (4) output $(VK, c_1, c_2, \mathsf{Sig}(VK, (T, c_1, c_2)))$. Decryption follows the scheme of [14], except that c_1 is decrypted using $tkn_T = usk_T$.

3 Authenticated Timed-release Public Encryption

The notion of authenticated encryption has been explored in depth in [2, 1]. In this section we adapt these definitions to give formal security and functionality requirements for a timed-release public-key authenticated encryption (TR-PKAE) scheme.

3.1 Basic Cryptosystem

The syntactic definition of a TR-PKAE is essentially the same as that of a TR-PKE 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:

- $\mathsf{Encrypt}(\pi, sk_A, pk_B, m, T)$ returns an authenticated timed-release ciphertext c denoting the encryption from sender A to receiver B of message m for time T.
- $\mathsf{Decrypt}(\pi, pk_A, sk_B, \widehat{c}, tkn_T)$ 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, $\mathsf{Decrypt}(\pi, pk_A, sk_B, \mathsf{Encrypt}(\pi, sk_A, pk_B, m, T), \mathsf{TG}(\pi, \delta, T)) = m$.

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. That is, the scheme should still provide *confidentiality* if an adversary knows the sender's secret key and either the master secret key or the user'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. If a cryptosystem has a goal of providing some kind of authentication, one should analyze what types of forgeries are possible or impossible. We concentrate on forgery of ciphertexts (plaintext forgery is defined analogously). We consider two types of ciphertext forgery: *third-party forgery*, by an adversary that does not know the sender's and receiver's private keys (TUF-CTXT); and *forgery by the ciphertext receiver* (RUF-CTXT) [2]. 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 decryptable (perhaps containing junk) ciphertexts between two users. If a TR-PKAE is not secure against RUF-CTXT, then the scheme provides no non-repudiation [6] and furthermore, if the receiver's private key is compromised, the attacker

[6] 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.

can impersonate any sender to this receiver. We introduce the following games to model unforgeability.

Algorithm 3.1: $\mathsf{Exp}^{\mathsf{IND-KC-CCA2}}_{A,\Gamma}(k)$

```
$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_A, sk_A) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(pk_B, sk_B) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(m_0, m_1, T^*)$
    $\leftarrow A^{\mathsf{Decrypt}(\pi, pk_A, sk_B, \cdot, \cdot)}(\pi, \delta, pk_A, sk_A, pk_B)$
$b \leftarrow_R \{0, 1\}$
$c^* \leftarrow \mathsf{Encrypt}(\pi, sk_A, pk_B, m_b, T^*)$
$b' \leftarrow A^{\mathsf{Decrypt}(\pi, pk_A, sk_B, \cdot, \cdot)}(\pi, \delta, pk_A, sk_A, pk_B, c^*)$
if ($A$ queried $\mathsf{Decrypt}(\pi, pk_A, sk_B, c^*, tkn_{T^*})$)
  then return (false)
  else return ($b' = b$)

$\mathsf{Adv}^{\mathsf{IND-CCA2}}_{A,\Gamma}(k) = \Pr[\mathsf{Exp}^{\mathsf{IND-CCA2}}_{A,\Gamma}(k) = \mathsf{true}] - \frac{1}{2}$
```

Algorithm 3.2: $\mathsf{Exp}^{\mathsf{IND-RTR-KC-CCA2}}_{A,\Gamma}(k)$

```
$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_A, sk_A) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(pk_B, sk_B) \leftarrow \mathsf{KeyGen}(\pi_g)$
$\boldsymbol{\kappa} \leftarrow (\pi, pk_A, sk_A, pk_B, sk_B)$
$(m_0, m_1, T^*)$
    $\leftarrow A^{\mathsf{TG}(\pi, \delta, \cdot), \mathsf{Decrypt}^*(\pi, \delta, pk_A, sk_B, \cdot, \cdot)}(\boldsymbol{\kappa})$
$b \leftarrow_R \{0, 1\}$
$c^* \leftarrow \mathsf{Encrypt}(\pi, sk_A, pk_B, m_b, T^*)$
$b' \leftarrow A^{\mathsf{TG}(\pi, \delta, \cdot), \mathsf{Decrypt}^*(\pi, \delta, pk_A, sk_B, \cdot, \cdot)}(\boldsymbol{\kappa}, c^*)$
if ($A$ queried $\mathsf{Decrypt}^*(\pi, pk_A, sk_A, c^*, T^*)$
 or $\mathsf{TG}(\pi, \delta, T^*)$)
  then return (false)
  else return ($b' = b$)

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

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

Timed-Release RUF-CTXT (RUF-TR-CTXT). We introduce a slightly weaker timed-release notion of RUF-CTXT, which requires that the receiver should not be able to forge ciphertext to himself for a future date. This notion 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 3rd party that a ciphertext was generated by the alleged sender, provided 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 [2] to allow adaptive adversarial behaviour: the receiver is not given access to the token for a single, adaptively-chosen *challenge* time period; in addition, the adversary adaptively chooses and fixes its public key only at the time of the first encryption query. We say that a TR-PKAE encryption is secure against timed-release RUF-CTXT, denoted by RUF-TR-CTXT, if every polynomial-time adversary A has negligible advantage (denoted by $\mathsf{Adv}^{\mathsf{RUF-TR-CTXT}}_{A,\Gamma}(k)$) against the challenger in the RUF-TR-CTXT game.

TUF-CTXT In addition to timed-release receiver unforgeability, we also require a time-independent third-party unforgeability (TUF-CTXT) condition. A good question would be to ask why would one require TUF-CTXT security if the best we can provide with respect to the receiver is RUF-TR-CTXT; after all RUF-TR-CTXT would automatically imply TUF-TR-CTXT. The main reason to require TUF-CTXT security is to ensure that some kind of unforgeability is guaranteed even if the master secret is compromised, that is, we would like to separate timed-release functionality from PKAE. Thus, in the

```
**Algorithm 3.3:** $\mathsf{Exp}^{\mathsf{TUF-CTXT}}_{\mathcal{A},\Gamma}(k)$

$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_A, sk_A) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(pk_B, sk_B) \leftarrow \mathsf{KeyGen}(\pi_g)$
$(c, t) \leftarrow \mathcal{A}^{\mathsf{Encrypt}^*(\pi, sk_A, pk_r, \cdot, \cdot)}(\pi, \delta, pk_A, pk_B)$
if ($\mathsf{Decrypt}^*(\pi, \delta, pk_A, sk_B, c, T_a) = \mathtt{fail}$
 or $\mathsf{Encrypt}^*(\pi, sk_A, pk_B, \cdot, T)$ returned $c$)
  then return (false)
  else return (true)
```

```
**Algorithm 3.4:** $\mathsf{Exp}^{\mathsf{RUF-TR-CTXT}}_{\mathcal{A},\Gamma}(k)$

$\pi_g \leftarrow \mathsf{Setup}(1^k)$
$(\delta, \pi_{tr}) \leftarrow \mathsf{TRSetup}(1^k)$
$(pk_A, sk_A) \leftarrow \mathsf{KeyGen}(\pi_g)$
$pk_r \leftarrow \mathcal{A}^{\mathsf{TG}(\pi, \delta, \cdot)}(\pi, pk_A)$
$C \leftarrow \mathcal{A}^{\mathsf{TG}(\pi, \delta, \cdot), \mathsf{Encrypt}^*(\pi, \mathsf{sk_A}, \mathsf{pk_r}, \cdot, \cdot)}(\pi, pk_A, pk_r)$
$(c, T_a, pk_B, sk_B) \leftarrow C$
if ($\mathsf{Decrypt}^*(\pi, \delta, pk_A, sk_B, c, T_a) = \mathtt{fail}$
 or $\mathsf{Encrypt}^*$ returned $c$
 or $(pk_B, sk_B) \notin [\mathsf{KeyGen}(1^k)]$
 or $\mathcal{A}$ queried $\mathsf{TG}(\mathsf{T_a})$)
  then return (false)
  else return (true)
```

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

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

TUF-CTXT game defined in Figure 3, the master key is given to the adversary. We say that a TR-PKAE scheme Γ is secure against third-person chosen-plaintext ciphertext forgery (TUF-CTXT) if every polynomial time adversary $\mathcal{A}$ has negligible advantage, $\mathsf{Adv}^{\mathsf{TUF-CTXT}}_{\mathcal{A},\Gamma}(k)$, in k.

4 The Proposed TR-PKAE

Following the proof of Theorem 1, one approach to achieve TR-PKAE would be to combine a key-insulated encryption scheme with a PKAE scheme in a modular fashion using techniques such as given in [14]. However, modern authenticated encryption requires one primitive that achieves the desired security properties [11]: such solutions generally allow for a more efficient scheme, tighter security bounds and more stringent security. Below we construct an example of such a scheme that satisfies all of the above security requirements and is nearly as efficient as Boneh and Franklin's FullIdent scheme [9]. We start with a review of Bilinear Diffie-Hellman Problem.

4.1 Bilinear Diffie-Hellman Problem

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two abelian groups of prime order q. We will use additive notation for $\mathbb{G}_1$ (aP denotes the P added a times for element $P \in \mathbb{G}_1$) and multiplicative notation for $\mathbb{G}_2$ (g^a denotes the g multiplied a times for element g of $\mathbb{G}_2$). Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be an admissible bilinear map [9]. The properties of the groups and constructions of e are explained in detail in [9]. We assume that the *Decisional Diffie-Hellman Problem* (DDHP) is hard in $\mathbb{G}_2$. Note that as a trivial consequence of DDHP assumption, the *Discrete Logarithm Problem* (DLP) is also hard in $\mathbb{G}_2$. As a consequence of the above assumptions, it follows that 1) DLP is hard in $\mathbb{G}_1$ [24], 2) $\forall Q \in \mathbb{G}_1^*$, inverting the isomorphism that takes $P \in \mathbb{G}_1$ and computes $e(P, Q)$ is hard [9], 3) the DDHP is easy in $\mathbb{G}_1$ [21].

Let $\mathcal{G}$ be a *BDH Parameter Generator* [9], i.e. a randomized algorithm that takes positive integer input k, runs in polynomial time in k and outputs prime q, descriptions of $\mathbb{G}_1, \mathbb{G}_2$ of order q, description of admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ along with polynomial deterministic algorithms for group operations and e and generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$. We say that algorithm $\mathcal{A}$ has advantage $\epsilon(k)$ in solving the *computational* Bilinear Diffie-Hellman Problem (BDHP) for $\mathcal{G}$ if there exists k_0 such that:

$$\mathsf{Adv}^{\mathsf{cbdh}}_{\mathcal{A},\mathcal{G}}(k) = \Pr[\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle \leftarrow \mathcal{G}(1^k), P \leftarrow \mathbb{G}_1^*, a, b, c \leftarrow \mathbb{Z}_q^* : \\ \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, e, P, aP, bP, cP) = e(P,P)^{abc}] \geq \epsilon(k), \forall k > k_0 \quad (1)$$

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

We say that algorithm $\mathcal{A}$ has advantage $\epsilon(k)$ in solving the *decisional* Bilinear Diffie-Hellman Problem (BDHP) [7] for $\mathcal{G}$ if there exists k_0 such that:

$$\mathsf{Adv}^{\mathsf{dbdh}}_{\mathcal{A},\mathcal{G}}(k) = \Pr[\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle \leftarrow \mathcal{G}(1^k), P \leftarrow \mathbb{G}_1^*, a, b, c \leftarrow \mathbb{Z}_q^*, T \leftarrow \mathbb{G}_2^* : \\ |Pr[\mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, e, P, aP, bP, cP, e(P,P)^{abc}) = 0] - Pr[\mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, e, P, aP, bP, cP, T) = 0]| \\ \geq \epsilon(k), \forall k > k_0 \quad (2)$$

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

4.2 Description of the Scheme

Let $\mathcal{G}$ be a *BDH Parameter Generator*. Figure 4 gives a complete description of our construction[7]. The symmetric encryption scheme used is a straightforward adaptation of the Fujisaki-Okamoto scheme [18]. We briefly demonstrate the consistency of the scheme before moving on to security considerations. Given ciphertext $c = \langle Q, \sigma \oplus K, m \oplus H_4(\sigma)\rangle$ computed using sk_A, pk_B and T, we note that in the corresponding $\mathsf{Decrypt}$ computations we have 1) $R = rP$, 2) $\widehat{K} = K$ since $e(R+pk_a, sP_T+sk_bP_T) = e(rP+sk_aP, sP_T+sk_bP_T) = e([r+sk_a]P, [s+sk_b]P_T) = e([s+sk_b]P, [r+sk_a]P_T) = e(P_{pub}+pk_b, [r+sk_a]P_T)$, 3) as in Fujisaki-Okamoto, it follows that $\widehat{\sigma} = \sigma$, $\widehat{m} = m$ and 4) $R = rP = H_3(\widehat{\sigma}, \widehat{m})P$. Thus the original plaintext is retrieved.

4.3 Security of the Scheme

The following security results apply to TR-PKAE. Due to space considerations, the detailed proofs of these results are omitted from this extended abstract. First, we note the confidentiality properties of the proposed scheme. The proofs of these properties are similar to the security proofs of Bellare and Palacio's SKIE-OT scheme [5] and will appear in the full version [26].

[7] As in [9], we can weaken surjectivity assumption on hash function H_1. The security proofs and results will hold true with minor modifications. We skip the details and refer reader to [9].

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 and an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Arbitrary generator $P \in \mathbb{G}_1$ is chosen.

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) $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and 4) $H_4 : \{0,1\}^n \to \{0,1\}^n$. 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 $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n \times \{0,1\}^n$. The general system parameters are $\pi_g = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, H_i, i = 1...4\rangle$

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 $\pi = \pi_g || \pi_{tr} = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_{pub}, H_i, i = 1...4\rangle$

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

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) choose $\sigma \in \{0,1\}^n$ uniformly at random, compute $r = H_3(\sigma, m)$ and set $Q = r \cdot pk_b$; 2) compute $\mathcal{L} = e(P_{pub} + pk_b, (r + sk_a)P_T)$ and symmetric key $K = H_2(\mathcal{L})$ and 3) the ciphertext c is set to be $c = \langle Q, \sigma \oplus K, m \oplus H_4(\sigma)\rangle$

Decrypt: Given ciphertext $c = \langle Q, c_1, c_2\rangle$ encrypted using sk_A, pk_N and time T, one decrypts it as follows: (1) obtain $tkn_T = sP_T$; (2) compute $R = sk_B^{-1}Q$ and $\widehat{K} = H_2(e(R + pk_A, sP_T + sk_B P_T))$; 3) retrieve $\widehat{\sigma} = c_1 \oplus \widehat{K}$ and compute $\widehat{m} = c_2 \oplus H_4(\widehat{\sigma})$ and 4) one verify that $R = H_3(\widehat{\sigma}, \widehat{m})P$; if so, output $\widehat{m}$, otherwise output `fail`.

Fig. 4. The proposed TR-PKAE scheme, $\mathcal{TR} - \mathcal{PKAE}$

Theorem 2 (IND-KC-CCA2). *Let $\mathcal{A}$ be a* IND-KC-CCA2 *adversary that makes q_d decryption queries and q_2 queries to H_2. Assume that $\mathsf{Adv}^{\mathsf{IND-KC-CCA2}}_{\mathcal{A},\mathcal{TR}-\mathcal{PKAE}}(k) \geq \epsilon$. Then there exists an algorithm $\mathcal{B}$ that solves computational BDHP with advantage $\mathsf{Adv}^{\mathsf{cbdh}}_{\mathcal{B},\mathcal{G}}(k) \geq \frac{2\epsilon}{q_d + q_2}$ and running time $O(time(\mathcal{A}))$.*

Theorem 3 (IND-RTR-CCA2). *Let $\mathcal{A}$ be a* IND-RTR-CCA2 *adversary that makes q_d decryption queries, q_2 queries to H_2 and q_{tok} queries to* TG. *Assume that $\mathsf{Adv}^{\mathsf{IND-RTR-CCA2}}_{\mathcal{A},\mathcal{TR}-\mathcal{PKAE}}(k) \geq \epsilon$. Then there exists an algorithm $\mathcal{B}$ that solves computational BDHP with advantage $\mathsf{Adv}^{\mathsf{cbdh}}_{\mathcal{B},\mathcal{G}}(k) \geq \frac{2\epsilon}{e \cdot (1 + q_{tok})(q_d + q_2)}$ and running time $O(time(\mathcal{A}))$.*

The proposed protocol also satisfies the authentication properties specified in the previous section, i.e., TUF-CTXT and RUF-TR-CTXT. We briefly sketch the proof of Theorem 4; the proof of Theorem 5 is more involved and we include it in the full version of the paper.

Theorem 4 (TUF-CTXT). *Let $\mathcal{A}$ be a* TUF-CTXT *adversary that makes q_e encryption queries and q_2 queries to H_2, and let $\mathsf{Adv}^{\mathsf{TUF-CTXT}}_{\mathcal{A},\mathcal{TR}-\mathcal{PKAE}}(k) \geq \epsilon$. Then there exists an algorithm $\mathcal{B}$ with computational BDHP advantage $\mathsf{Adv}^{\mathsf{cbdh}}_{\mathcal{B},\mathcal{G}}(k) \geq \left[\frac{\epsilon}{(q_e \cdot q_2 + 1)}\right]^2$ and running time $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

Proof. (Sketch). Suppose we are given a triple (P, xP, yP). Then given $\mathcal{A}$ we can use relatively straightforward random oracle simulation techniques to compute $e(P,P)^{xy^2}$ with probability $\rho = \epsilon/(q_e q_2 + 1)$ as follows. We run $\mathcal{A}$, picking uniform s and setting setting $P_{pub} = sP$, $pk_A = xP$, $pk_B = yP$ and responding to oracle queries $H_1(T)$ with $c_T yP$ for a uniformly chosen c_T. Then on a successful forgery (T, ryP, c_1, c_2) we know that (except with negligible probability) $\mathcal{A}$ must have queried $e(P_{pub} + pk_B, (r + sk_A)H_1(T))$ to H_2, and from this value and c_T we can compute $e((s+y)P, (r+x)c_T yP) = e(P,P)^{ysr+y^2r+ysx+y^2x}$; since we know s and ryP, we can also compute $e(P,P)^{ysr}$,$e(yP, yrP) = e(P,P)^{y^2r}$, and $e(yP, xP)^s = e(P,P)^{ysx}$; dividing these off gives $e(P,P)^{xy^2}$. Given a BDH problem (P, aP, bP, cP) we can compute $e(P,P)$ with probability ρ^2 by using this procedure twice: once with $x_1P = r_1aP, y_1P = \frac{1}{2}(bP + cP)$, yielding $e(P,P)^{a(b/2+c/2)^2}$; and once with $x_2P = r_2aP, y_2P = \frac{1}{2}(bP - cP)$, yielding $e(P,P)^{a(b/2-c/2)^2}$. If both results are correct, dividing them gives $e(P,P)^{abc}$.

Theorem 5 (RUF-TR-CTXT). *Let $\mathcal{A}$ 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 $\mathsf{Adv}^{\mathsf{RUF-TR-CTXT}}_{\mathcal{A},\mathcal{TR-PKAE}}(k) \geq \epsilon$. Then there exists an algorithm $\mathcal{B}$ with decisional* [8] *BDHP advantage $\mathsf{Adv}^{\mathsf{dbdh}}_{\mathcal{B},\mathcal{G}}(k) \geq [\frac{\epsilon}{e\cdot(1+q_{tok})}]^4 \cdot \frac{1}{[16\cdot(q_e\cdot q_2+4)\cdot q_e \cdot q_2]}$ and running time $O(time(\mathcal{A})) + O([q_e \cdot q_2]^2)$.*

5 Efficiency

To compare the proposed scheme to Boneh and Franklin's FullIdent [9], note that, first, encryption in both schemes requires the same number of significant operations: *1 bilinear pairing, 1 MapToPoint, 2 exponentiations in* $\mathbb{G}_1$. The decryption in FullIdent requires *1 bilinear pairing and 1 exponentiation in* $\mathbb{G}_1$ while the proposed TR-PKAE adds 2 additional exponentiations in $\mathbb{G}_1$. Second, the proposed scheme shares the same spatial complexity with FullIdent. Therefore, the hybrid protocols that combine SKIE-OT with additional cryptographic primitives are bound to be at least as expensive as our scheme.

We implemented the proposed primitives using Miracl library v.4.8.2 [31] with Tate pairing for the bilinear map. The group $\mathbb{G}_1$ was chosen to be a subgroup of order q in a supersingular elliptic curve E over $\mathbb{F}_p$, where p is a 512 bit and q is a 160 bit primes. Group $\mathbb{G}_2$ was a subgroup of a finite field of order 1024 bits. We used a P3-977 MHz desktop with 512 MB of memory. The performance measurements are summarized in Table 1. As expected, the proposed TR-PKAE is slightly more expensive than FullIdent in decryption, but when FullIdent is extended to provide comparable functionality to TR-PKAE we expect the resulting scheme to be at least as expensive as the proposed protocol.

6 Adaptation of HIBE to TR-PKAE

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

[8] Computational BDH assumption suffices when RUF-TR-CTXT is adapted from [2]. The resulting bound is of the order ϵ^2 order

Table 1. Cost of basic operations

Function	modulus (bits)	exponent (bits)	performance (msec)
RSA(Sig/Dec)	1024	1024	4.65
RSA(Ver/Enc)	1024	16 ($e = 2^{16} + 1$)	0.36
Scalar Mul in EC over $\mathbb{F}_p$	160	160	3.44
MapToPoint	512	-	2.42
Pairing	512	160	31.71
TR-PKAE Enc	512	160	41
TR-PKAE Dec	512	160	42
FullIdent Enc	512	160	41
FullIdent Dec	512	160	35

time well in the past. As a result, the required storage for tokens grows linearly over time. A Hierarchical IBE (HIBE) scheme recently proposed by Boneh *et al.* [8] allows to reduce the required token storage to $O(\log^{3/2} T)$, where T is an upper bound on the number of time periods when tokens are published [9]. This adaptation works as follows. Using the notation of [8] (in particular, multiplicative notation for both groups), the sender now computes $\{e(g^b \cdot g_1, g_2)^{r+a}, g^{rb}, g^a, L\}$, where g replaces P, (b, g^b) and (a, g^a) are now the private/public key pairs of the receiver and sender, and L is the remaining part of the ciphertext as in [8] using public key $(I_1, ..., I_k)$ (which denotes the time) and $s = r + a$. Note that the g^s in the ciphertext of [8] is replaced by g^a and g^{rb}. To decrypt the ciphertext using private key b and private key $(a_0, b_1, ...)$ corresponding to $(I_1, ..., I_k)$, the receiver replaces a_0 by $a_0 \cdot g_2^b$ and then applies the decryption mechanism used in [8] with $g^s = g^r \cdot g^a$ and message $M = 1$ to obtain the required bilinear map. Note that the number of pairings required does not increase when we add authentication. We also note that now we can use the property of [8] to reduce the amount of storage required to store tokens. In particular, at each time-period the agent only needs to store $O(\log^{3/2} T)$ group elements, from which all previous tokens can be derived. The security in [8] relies on the Bilinear Diffie Hellman Exponent assumption, which is stronger than computational BDH assumption; aside from this, the security analysis of the adapted scheme is similar to the original schemes and is left to the full paper due to space constraints.

References

1. M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *CT-RSA*, 2001.
2. J. H. An. Authenticated Encryption in the Public-Key Setting: Security Notions and Analyses. `http://eprint.iacr.org/2001/079/`, 2001.
3. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO*, 1998.
4. M. Bellare and S. Goldwasser. Encapsulated key escrow. Technical report, MIT/LCS/TR-688, 1996.
5. M. Bellare and A. Palacio. Protecting against Key Exposure: Strongly Key-Insulated Encryption with Optimal Threshold. `http://eprint.iacr.org/2002/064/`, 2002.

[9] The possibility of using HIBE for this purpose was initially suggested in online `sci.crypt` newsgroup [17] and further discussed in [8].

6. I. F. Blake and A. C.-F. Chan. Scalable, Server-Passive, User-Anonymous Timed Release Public Key Encryption from Bilinear Pairing. In *ICDCS*, 2005.
7. D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*, 2004.
8. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *EUROCRYPT*, 2005.
9. D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. In *CRYPTO*, 2003.
10. D. Boneh and M. Naor. Timed commitments. In *CRYPTO*, 2000.
11. X. Boyen. Multipurpose Identity-Based Signcryption: A Swiss Army Knife for Identity-Based Cryptography. In *CRYPTO*, 2003.
12. L. Chen, K. Harrison, D. Soldera, and N. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *InfraSec*, 2002.
13. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional Oblivious Transfer and Timed-Release Encryption. In *EUROCRYPT*, 1999.
14. Y. Dodis and J. Katz. Chosen-Ciphertext Security of Multiple Encryption. In *Theory of Cryptography Conference*, 2005.
15. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-Insulated Public Key Cryptosystems. In *EUROCRYPT*, 2002.
16. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. In *PKC*, 2003.
17. P. from newsgroup sci.crypt. Key Evolving Encryption. Available from `http://www.groupsrv.com/science/viewtopic.php?t=61168&start=0`, 2004.
18. E. Fujisaki and T. Okamoto. Secure Integration of Assymetric and Symmatric Encryption Schemes. In *CRYPTO*, 1999.
19. J. Garay and C. Pomerance. Timed Fair Exchange of Arbitrary Signatures. In *Financial Cryptography*, 2003.
20. J. A. Garay and C. Pomerance. Timed Fair Exchange of Standard Signatures. In *Financial Cryptography*, 2002.
21. A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. *Journal of Cryptology*, 16, 2003.
22. K. H. Marco Casassa Mont and M. Sadler. The HP Time Vault Service: Exploiting IBE for Timed Release of Confidential Information . In *WWW*, 2003.
23. T. May. Timed-Release Crypto. `http://www.cyphernet.org/cyphernomicon/chapter14/14.5.html-`.
24. A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *IEEE Transactions on Information Theory IT-39, 5*, 1993.
25. D. Mills. Network Time Protocol (Version 3) Specification, Implementation. Technical Report 1305, RFC, 1992.
26. I. Osipkov, Y. Kim, and J. H. Cheon. A Scheme for Timed-Release Public Key Based Authenticated Encryption. Available from `http://eprint.iacr.org/2004/231`, 2004.
27. T. P. Pederson. A Threshold Cryptosystem Without a Trusted Party. In *EUROCRYPT*, 1991.
28. D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *EUROCRYPT*, 1996.
29. C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *CRYPTO*, 1991.
30. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and Time-released Crypto. Technical report, MIT/LCS/TR-684, 1996.
31. Shamus Software Ltd. MIRACL: Multiprecision Integer and Rational Arithmetic C/C++ Library. `http://indigo.ie/~mscott/`.

32. V. Shoup. ISO 18033-2: An Emerging Standard for Public-Key Encryption. Available from `http://shoup.net/iso/`, 2004.
33. P. F. Syverson. Weakly Secret Bit Commitment: Applications to Lotteries and Fair Exchange. In *Computer Security Foundations Workshop*, 1998.

A Selected Security Proofs

Proof of Theorem 2 [**IND-KC-CCA2**] The Theorem result follows from Corollary 9. Let $\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle$ (output by $\mathcal{G}(1^k)$) and a random instance of BDH parameters $\langle X, a'X, b'X, c'X\rangle$ be given, where X is a generator of $\mathbb{G}_1$. Consider an adversary $\mathcal{A}$ against IND-KC-CCA2. We design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real IND-KC-CCA2 game for the adversary in order to compute solution to BDHP $e(X, X)^{a'b'c'}$

Setup :

Choice of Generator: $\mathcal{B}$ chooses generator P to be $P = a'X$.

Choice of s: $\mathcal{B}$ chooses master secret s and makes it public.

Choice of pk_b: $\mathcal{B}$ chooses receiver public key pk_b to be X. The adversary $\mathcal{A}$ receives pk_b.

Choice of Set $\{(sk_a, pk_a)\}$: $\mathcal{B}$ chooses $a_i \in \mathbb{Z}_q^*$ at random and forms the set $\{(a_i, a_iP)\}$. The adversary $\mathcal{A}$ receives $\{a_i\}$.

Databases: Databases corresponding to $H_i, i = 1, ..., 4$ are maintained indexed by queries with replies being the values. In addition, $\mathcal{B}$ maintains database $\mathcal{L}$ of possible values of $e(X, X)^{a'b'c'}$ updated in the *Decryption Queries After Challenge* phase.

Oracle queries :

P_T (or H_1) Queries: $\mathcal{B}$ returns c_TZ for random $c_T \in \mathbb{Z}_q^*$, where $Z = c'X$, and stores the query T in the database coupled with c_T. Repeated queries retrieve answers from the database.

H_2,H_3,H_4 Queries: $\mathcal{B}$ returns a random value and stores it in its database coupled with the query. Whenever a query is made, this query is stored in a database along with the answer given. Repeated queries retrieve answers from the database.

Decryption Queries Before Challenge: $\mathcal{A}$ submits ciphertext $\langle T, Q, aP = a_iP, c_1, c_2\rangle$ where c_1 denotes $\sigma \oplus K$ and c_2 denotes $m \oplus H_4(\sigma)$, Q represents $r \cdot pk_b$, a (note that simulator can extract $a = a_i$ from $aP = a_iP$) is the sender private key and T is the designated time.

$\mathcal{B}$ goes through the database of H_3 searching for appropriate r (by multiplying each r by pk_b and comparing with Q; alternatively, the multiplication can be done whenever a query to H_3 is made). If it is not found, false is returned. If it is found, then corresponding σ and m are retrieved. Then database of H_4 is searched for query with σ. If this σ was not queried in H_4 then false is returned. Otherwise, $\mathcal{B}$ computes $c_2 \oplus H_4(\sigma)$ and compares it with m. If they are not equal, false is returned. Next, database of H_1 is queried: if it never returned $H_1(T)$ false is returned. Next $\mathcal{B}$ computes $K = c_1 \oplus \sigma$ and queries the database of H_2 to see if this K was ever returned. If it was not, false is returned. If it was, it obtains corresponding query given to H_2 and compares it with the true value of the bilinear map which can be computed as $e(rP, sH_1(T)) \cdot e(aP, sH_1(T)) \cdot e(pk_b, aH_1(T)) \cdot e(Q, H_1(T))$ (note that simulator knows r). If they are equal, true is returned. Otherwise, false is returned.

Selection: $\mathcal{A}$ chooses two equal-sized plaintexts m_0,m_1, sender private key $a^* \in \{a_i\}$ and $T = T^*$.

Challenge: $\mathcal{B}$ chooses arbitrary $\beta \in \{0, 1\}$, arbitrary $t^* \in \mathbb{Z}_q^*$ and assigns $Q^* = t^*(b'X)$. Then $\mathcal{B}$ chooses σ^*, two random strings c_1^* and c_2^*, and composes and returns ciphertext $c^* = \langle T^*, Q^*, a^*, c_1^*, c_2^* \rangle$. The databases are updated as follows:

- H_3: $\mathcal{B}$ puts $r \cdot pk_b = Q^*$ as a value (marked appropriately in the database) and (σ^*, m_β) as the query. If such (σ^*, m_β) was queried previously, a new choice of σ^* is made. In addition, Q^* is checked against existing replies in the database (by multiplying each reply by pk_b and comparing it with Q^*) and if it already exists, a new choice for t^* is made.
- H_4: $\mathcal{B}$ puts $m_\beta \oplus c_2^*$ as a value and σ^* as the query into database of H_4. If σ^* was already queried, a new choice of σ^* is made (in addition, corresponding (σ^*, m_β) should not have been queried from H_3). If $m_\beta \oplus c_2^*$ was returned previously as a reply to some query, a new choice of c_2^* is made.
- H_1: If $H_1(T^*)$ was never queried then the query is made.
- H_2: The database of H_2 is instructed never to return the corresponding value of $K = K^* = \sigma^* \oplus c_1^*$ (if it returned this value previously, a new choice of c_1^* is made)

Queries Cont'd: $\mathcal{A}$ has a choice to continue queries or to reply to the challenge. $\mathcal{A}$ is not allowed to query for decryption of c^* using a^* and T^* chosen for the challenge. For decryption queries, $\mathcal{B}$ behaves according to *Decryption Queries After Challenge* phase.

Decryption Queries After Challenge: $\mathcal{A}$ submits ciphertext $\langle T, Q, aP = a_iP, c_1, c_2 \rangle$. $\mathcal{B}$ searches for r corresponding to Q in database of H_3. Three cases are possible:

- Q is found without r: Then $Q = Q^*$ and $\mathcal{B}$ returns false independent of the rest of the ciphertext. In addition the following local actions are carried out. If $c_2 = c_2^*$ and $c_1 \neq c_1^*$, $\mathcal{B}$ retrieves appropriate $\sigma = \sigma^*$ and computes $K = c_1 \oplus \sigma^* \neq K^*$. If H_2 did return this value of K for query Y, then $\mathcal{B}$ computes $[Y/[e(sP+pk_b, aH_1(T)) \cdot e(Q, H_1(T))]]^{(sc_T t^*)^{-1}}$ and writes the result in the list $\mathcal{L}$ as a possible value of $e(X, X)^{a'b'c'}$.
- r is found: If $Q = Q^*$, then $\mathcal{B}$ quits and computes $e(rP, sH_1(T)) = e(b'X/sk_b, Z)^{sc_T t^*}$. Thus $\mathcal{B}$ can calculate $e(b'X/\log_P X, Z) = e(\log_X(P) \cdot b'X, Z) = e(X, X)^{a'b'c'}$. Otherwise, the same procedure as in the *Before Challenge* case is followed.
- None of the above: false is returned

Outcome: β is returned or simulation halts.

1. If r corresponding to challenge Q^* was found in the *After Challenge* phase, then the procedure specified there produces $e(X, X)^{a'b'c'}$. This value is the solution to BDHP and is output by $\mathcal{B}$.
2. Otherwise, $\mathcal{B}$ goes through all q_2 adversary queries to H_2 and the list $\mathcal{L}$ that was produced in the *After Challenge* phase and picks a random value Y. If Y comes from queries to H_2, $\mathcal{B}$ computes $[Y/[e(sP+pk_b, a^*H_1(T^*)) \cdot e(Q^*, H_1(T^*))]]^{(sc_{T^*} t^*)^{-1}}$ and outputs the result as the solution to BDHP. If the choice came from the *After Challenge* list, this choice in its original form is output as a solution to BDHP.

Definition 6. We say that simulation above becomes inconsistent when: 1) $\mathcal{A}$ makes a query to H_2 with a true value of challenge bilinear map $e(sP + pk_b, (r + a^*)H_1(T^*)$ where $r \cdot pk_b = t^* b' X$ or 2) in the *After Challenge* phase $\mathcal{B}$ returns false where true is due, were the calculation done the same way as in *Before Challenge* phase.

Lemma 7. *If the simulation above becomes inconsistent, then $\mathcal{B}$ outputs correct answer to BDHP with probability $\frac{1}{q_d+q_2}$*

Proof: Suppose simulation becomes inconsistent due to queries to H_2 and let Y be the query which is the true value of the challenge bilinear map. Then $Y/[e(sP + pk_b, a^* H_1(T^*)) \cdot e(Q^*, H_1(T^*))] = e(sP, rH_1(T^*))$ where $r \cdot pk_b = t^*(b'P)$ and $e(sP, rH_1(T^*))$ $= e(b'X/sk_b, Z)^{sc_{T^*}t^*} = e(b'X/\log_P X, Z)^{sc_{T^*}t^*} = e(\log_X(P) \cdot b'X, Z)^{sc_{T^*}t^*} = e(X, X)^{(a'b'c')(sc_{T^*}t^*)}$. Thus if this Y is chosen in the *Outcome* phase, the corresponding computation by $\mathcal{B}$ will output the true solution to BDHP.

If simulation becomes inconsistent due to incorrect reply in the *After Challenge* phase, then $\mathcal{A}$ must have submitted ciphertext $\langle T, Q, a, c_1, c_2 \rangle$ where $Q = Q^*$. To return true to this query we must have:

1. $c_2 = c_2^*$ (since σ and m are the same in both cases)
2. and $c_1 \neq c_1^*$. If $c_1 = c_1^*$, then $K^* = K$ which is true only when $a = a^*$ and $T = T^*$ (up to some negligible probability) provided that no query to H_2 with a true value of the challenge bilinear map was made. In this case, submitted ciphertext is the same as the challenge ciphertext and $\mathcal{B}$ should return false.

If true should have been returned, then $\mathcal{A}$ must have made a query Y to H_2 and received $K = c_1 \oplus \sigma$, where Y is the correct value of the bilinear map $e(sP + aP, (r + sk_b)P_T)$. In this case, Y can be re-written as $e(sP + pk_b, aH_1(T)) \cdot e(Q, H_1(T))e(rP, sH_1(T))$ where $e(sP, rH_1(T)) = e(b'X/sk_b, Z)^{sc_T t^*} = e(X, X)^{(a'b'c')(sc_T t^*)}$ as before. It follows that the corresponding computation carried out in the *After Challenge* phase will in fact yield the true solution to BDHP and thus the list $\mathcal{L}$ will contain $e(X, X)^{a'b'c'}$. It follows that if the simulation becomes inconsistent then one of the output choices of $\mathcal{B}$ will be the solution to BDHP and since the size of the output list is at most $q_d + q_2$, the conclusion follows. □

To show that advantage obtained is at least $\frac{2\epsilon}{q_2+q_d}$, we construct a new simulation with challenger denoted by $\mathcal{C}$. The new game will be denoted as $\texttt{Game}_{\mathcal{C}}$ while the game with challenger $\mathcal{B}$ specified above will be denoted by $\texttt{Game}_{\mathcal{B}}$.

In $\texttt{Game}_{\mathcal{C}}$, challenger $\mathcal{C}$ runs $\mathcal{G}(1^k)$ to generate $(q, \mathbb{G}_1, \mathbb{G}_2, e)$ and then chooses at random X, a', b' and c'. Up to the challenge, $\mathcal{C}$ behaves the same way as $\mathcal{B}$ including answering the random oracle queries. In addition, $\mathcal{C}$ calculates correctly the bilinear map in the challenge and assigns the hash value to this pairing the same way as $\mathcal{B}$ unless this input was already queried by adversary from H_2, in which case $\mathcal{C}$ uses the value of K returned by H_2. In $\texttt{Game}_{\mathcal{C}}$, this value of K is put in the database of H_2 with input being the correct calculation of the pairing. In both games, Q and c_2 of the ciphertext are chosen in the same way with the only possible difference being in c_1. $\mathcal{C}$ replies to decryption queries in *Decryption Queries After Challenge* the same way as in *Decryption Queries Before Challenge* using its knowledge of a', b' and c'.

Lemma 8. *If $\mathcal{A}$ wins with advantage ϵ in the real game then he also wins with advantage of at least ϵ in the $\texttt{Game}_{\mathcal{C}}$ (up to negligible probability of guessing).*

Proof: We note that in the *Decryption Queries Before/After Challenge* $\mathcal{C}$ provides incorrect answer only if adversary guessed one of the values. In the *Challenge* phase, behavior of $\mathcal{C}$ differs from a real game only in the fact that some choices may be replaced with new random choices to ensure that adversary did not query those choices before. Probability that these choices have to be replaced with new ones is similar to probability of guessing in the previous case. Other than these remarks, $\texttt{Game}_{\mathcal{C}}$ is indistinguishable from a real game since all values are chosen at random starting with random initial seeds. □

Corollary 9. *If $\mathcal{A}$ attains advantage of at least ϵ in the real game, then the probability that $\texttt{Game}_{\mathcal{B}}$ outputs solution to BDHP is at least $\frac{2\epsilon}{q_d+q_2}$.*

Proof: Some additional notation is needed first:

- Denote by $r_{\mathcal{B}}$ the random tape of $\mathcal{B}$, $r_{\mathcal{A}}$ the random tape of $\mathcal{A}$ and $r_{\mathcal{C}}$ the random tape of $\mathcal{C}$ used by $\mathcal{C}$ after generation of BDHP parameters.
- Denote by $Par_{\mathcal{C}(r_{\mathcal{C}})}$ the set $(q, \mathbb{G}_1, \mathbb{G}_2, e, X, a'X, b'X, c'X)$ of BDHP parameters generated by $\mathcal{C}$ with random tape $r_{\mathcal{C}}$.
- Denote by $Inc(Par, r_{\mathcal{A}}, r_{\mathcal{B}})$ the event that the run of $\texttt{Game}_{\mathcal{B}}$ with BDHP parameters Par, random tapes $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$, is inconsistent.
- Denote by $Succ(r_{\mathcal{A}}, r_{\mathcal{C}})$ the event that the adversary $\mathcal{A}$ wins in $\texttt{Game}_{\mathcal{C}}$ with random tapes $r_{\mathcal{A}}$ and $r_{\mathcal{C}}$.

From Lemma 8 it follows that $\mathcal{A}$ achieves advantage ϵ in $\texttt{Game}_{\mathcal{C}}$ and, therefore, $Pr_{r_{\mathcal{A}},r_{\mathcal{B}}}[Succ(r_{\mathcal{A}}, r_{\mathcal{C}})] = 1/2+\epsilon$. We have $Pr_{r_{\mathcal{A}},r_{\mathcal{C}}}[Succ(r_{\mathcal{A}}, r_{\mathcal{C}}) \mid \neg Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})] = 1/2$ since no correct query of the challenge bilinear map was made to H_2 by $\mathcal{A}$ and, therefore, $\mathcal{A}$ cannot distinguish ciphertexts other than by guessing. Note that B is running with random tape $r_{\mathcal{C}}$ and $\mathcal{C}$'s BDHP parameters, therefore, $\texttt{Game}_{\mathcal{B}}$ is identical to $\texttt{Game}_{\mathcal{C}}$ until $\texttt{Game}_{\mathcal{B}}$ becomes inconsistent.

We have

$$\begin{aligned}
&Pr[Succ(r_{\mathcal{A}}, r_{\mathcal{C}})] = \\
&\qquad Pr[Succ(r_{\mathcal{A}}, r_{\mathcal{C}}) \mid \neg Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})] \cdot Pr[\neg Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})] \\
&\qquad + \\
&\qquad Pr[Succ(r_{\mathcal{A}}, r_{\mathcal{C}}) \mid Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})] \cdot Pr[Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})] \\
&= 1/2 + \epsilon
\end{aligned}$$

where all probabilities are taken over random tapes $r_{\mathcal{A}}$ and $r_{\mathcal{C}}$

Denote $p_f = Pr[Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})]$ and $k = Pr_{r_{\mathcal{A}},r_{\mathcal{C}}}[Succ(r_{\mathcal{A}}, r_{\mathcal{C}}) \mid Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})]$. Then the above equation becomes $1/2 \cdot (1 - p_f) + p_f \cdot k = 1/2 + \epsilon$. It follows that $p_f \cdot (k - 1/2) = \epsilon$ and, therefore, $p_f \geq 2\epsilon$.

We note that $Pr_{r_{\mathcal{A}},r_{\mathcal{C}}}[Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})] = Pr_{Par,r_{\mathcal{A}},r_{\mathcal{B}}}[Inc(Par, r_{\mathcal{A}}, r_{\mathcal{B}})]$ since C generates $Par_{\mathcal{C}(r_{\mathcal{C}})}$ independently from $r_{\mathcal{C}}$ using a separate random tape. It follows that probability that $\texttt{Game}_{\mathcal{B}}$ is inconsistent is at least 2ϵ. Applying Lemma 7 we obtain the result.□

Proof of Theorem 4 [**TUF-CTXT**] Let $\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle$ (output by $\mathcal{G}(1^k)$) and a random instance of BDH parameters $\langle X, a''X, b''X, c''X\rangle$ be given, where X is a generator of $\mathbb{G}_1$. Consider an adversary $\mathcal{A}$ against TUF-CTXT. First we design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real TUF-CTXT game for the adversary in order to compute solution to special case of BDHP with parameters $\langle X, a'X, b'X, b'X\rangle$.

Setup :

Choice of Generator: $\mathcal{B}$ chooses generator P to be X.

Choice of s: $\mathcal{B}$ chooses $s \in \mathbb{Z}_q^*$ and makes it public.

Choice of pk_a and pk_b: $\mathcal{B}$ chooses public key of receiver pk_b to be $b'P = b'X$ and public key of sender pk_a to be $a'P = a'X$. The public keys are given to $\mathcal{A}$.

Databases: Databases corresponding to $H_i, i = 1, ..., 4$ are maintained indexed by queries with replies being the values. In addition, $\mathcal{B}$ maintains database D_s updated in the *Encryption Queries* phase.

Oracle queries :

P_T (or H_1) Queries: $\mathcal{B}$ chooses random $c_T \in \mathbb{Z}_q^*$ and returns $c_T(b'P)$. Query T along with c_T are stored and replies for repeated queries use the database. Note that given $Q = r \cdot b'P \in \mathbb{G}_1$ for some $r \in \mathbb{Z}_q$, $rH_1(T) = r \cdot c_T b'P = c_T \cdot rb'P = c_T Q$, i.e. knowing $r \cdot b'P$ we can compute $rH_1(T)$ for arbitrary T without knowledge of r.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem 2.

Encryption queries: When $\mathcal{A}$ submits T and m, $\mathcal{B}$ chooses random $Q \in \mathbb{G}_1^*$, σ and two random strings c_1 and c_2 and returns ciphertext $c = \langle T, Q, c_1, c_2\rangle$. The ciphertext represents encryption of m with $pk_a = a'P$ being the sender and $pk_b = b'P$ the receiver. The databases are updated as follows:

H_3: $\mathcal{B}$ puts Q as a value (marked appropriately in the database) and (σ, m) as the query. If such (σ, m) was queried previously, a new choice of σ is made. In addition, Q is checked against existing replies in the database (by multiplying each reply by pk_b and comparing it with Q; in addition $\mathcal{B}$ ensures that this choice of Q was not submitted in one of the previous Encryption Queries) and if it already exists, a new choice for Q is made.

H_4, H_1, H_2: updated the same way as in the *Challenge* phase of the proof of Theorem 2

$\mathcal{B}$ keeps the local database D_s in which it enters the pair $\langle T, Q\rangle$. Denote by $TRUE[T, Q]$ the true value of $e(sP + pk_b, (r + sk_a)H_1(T))$, where $r \cdot pk_b = Q$

Forgery: $\mathcal{A}$ submits ciphertext $\langle T^*, Q^*, c_1^*, c_2^*\rangle$.

Outcome: $\mathcal{A}$ returns forged ciphertext or simulation halts.

1. $\mathcal{B}$ goes through database D_s, obtains a pair of T and $Q = r \cdot pk_b$ (r is unknown to $\mathcal{B}$) from each entry and computes $[Y/[e(sP, rH_1(T)) \cdot e(sk_a, sH_1(T)) \cdot e(pk_b, rH_1(T))]]^{c_T^{-1}}$, for every query Y of $\mathcal{A}$ to H_2. The results are written down as possible values of $e(P, P)^{a'b'^2}$.
2. If $\mathcal{A}$ submitted a forgery, $\mathcal{B}$ first verifies that Q^* is in the database of H_3, either in the form of r (this is checked by multiplying each r by pk_b) or Q. If the answer is yes, two cases are possible:

Corresponding r is absent: It follows that Q^* was entered by $\mathcal{B}$. $\mathcal{B}$ retrieves corresponding σ and m. If $c_2^* = m \oplus H_4(\sigma)$ and c_1^* is not equal to the corresponding part of a ciphertext generated in the encryption queries, $\mathcal{B}$ computes $K = c_1^* \oplus \sigma$. If K was returned by H_2, the corresponding query is divided by $e(sP, rH_1(T^*)) \cdot e(pk_a, sH_1(T^*)) \cdot e(pk_b, rH_1(T^*))$ and the result is taken to c_T^{-1}-th power (note that $rH_1(T^*)$ can be computed as $c_{T^*}Q^*$). The answer is written down as possible value of $e(P,P)^{a'b'^2}$

Corresponding r is found: $\mathcal{B}$ obtains m and σ and goes through the same steps as in the previous case (except that c_1^* is not compared) to obtain possible value of $e(P,P)^{a'b'^2}$

Note that if $\mathcal{A}$ wins then the query corresponding to K will be the correct calculation of the corresponding bilinear map and, therefore, the answer computed by $\mathcal{B}$ will in fact be equal to $e(P,P)^{a'b'^2}$ (up to probability of guessing).

Out of calculated possible values of $e(P,P)^{a'b'^2}$, $\mathcal{B}$ picks one at random and outputs it as the value of $e(P,P)^{a'b'^2}$. Note that the size of the list of possible values of $e(P,P)^{a'b'^2}$ is at most $q_e \cdot q_2 + 1$.

Definition 10. We say that simulation above becomes inconsistent when $\mathcal{A}$ makes a query to H_2 with a true value corresponding to one of the $TRUE[T,Q]$ in D_s.

Lemma 11. *If the simulation above becomes inconsistent, then $\mathcal{B}$ contains $e(X,X)^{a'b'^2}$ in its output list.*

Proof: Let Y be a query to H_2 which happens to be the correct computation of the bilinear map corresponding to some $TRUE[T,Q]$ in D_s. Denote $r \cdot pk_b = Q$. Then $Y = e(sP + pk_b, (r + sk_a)H_1(T)) = e(pk_b, sk_aH_1(T)) \cdot e(sP, rH_1(T)) \cdot e(pk_a, sH_1(T)) \cdot e(pk_b, rH_1(T))$. In the *Outcome* phase of the simulation, $\mathcal{B}$ computes $Y/[e(sP, rH_1(T)) \cdot e(pk_a, sH_1(T)) \cdot e(pk_b, rH_1(T))] = e(pk_b, sk_aH_1(T)) = e(pk_b, sk_a(c_T b'P)) = e(P,P)^{a'b'^2 c_T}$. Since $\mathcal{B}$ takes the result to power c_T^{-1}, the true value of $e(P,P)^{a'b'^2}$ is indeed in the list of possible values. □

Next, one constructs $\texttt{Game}_{\mathcal{C}}$ analogously to the proof of Theorem 2 (details skipped – the reader is asked to refer to analysis in Theorem 2 for notation). And Lemma 8 carries over here as well with obvious modifications. The following Lemma is slightly different from the corresponding one in the proof of Theorem 2.

Lemma 12. *If $\mathcal{A}$ attains advantage of at least ϵ in the real game, then the probability that $\texttt{Game}_{\mathcal{B}}$ outputs $e(P,P)^{a'b'^2}$ is at least $\frac{\epsilon}{q_e \cdot q_2 + 1}$.*

Proof: We use the same notation as in Corollary 9. In addition to notation used in Corollary 9, denote $k^* = Pr_{r_{\mathcal{A}}, r_{\mathcal{C}}}[Succ(r_{\mathcal{A}}, r_{\mathcal{C}}) \mid \neg Inc(Par_{\mathcal{C}(r_{\mathcal{C}})}, r_{\mathcal{A}}, r_{\mathcal{C}})]$. Then, as in Corollary 9, $k^* \cdot (1 - p_f) + k \cdot p_f = \epsilon$.

Note that when $\mathcal{A}$ is successful in $\texttt{Game}_{\mathcal{C}}$ and $\texttt{Game}_{\mathcal{B}}$ (using $r_{\mathcal{C}}$ as $\mathcal{B}$'s random tape, Par generated by $\mathcal{C}$ and the same random tape for $\mathcal{A}$) is consistent, the output list of $\mathcal{B}$ will contain $e(P,P)^{a'b'^2}$ (namely, the candidate for $e(P,P)^{a'b'^2}$ extracted by $\mathcal{B}$ from the forgery). From this remark and Lemma 11, it follows that the probability that $\mathcal{B}$

contains $e(P,P)^{a'b'^2}$ in its output list is at least $p_f+(1-p_f)\cdot k^* \geq k^*\cdot(1-p_f)+k\cdot p_f = \epsilon$. Since the output list contains $q_e \cdot q_2 + 1$ entries, the result follows. □

The $\texttt{Game}_{\mathcal{B}}$ is used to solve BDHP $\langle X, a''X, b''X, c''X\rangle$ as follows. We run $\texttt{Game}_{\mathcal{B}}$ with BDHP parameters $\langle X, a''X, Y_1, Y_1\rangle$ where $Y_1 = b_1X = (c''X + b''X)/2$, where $b_1 = (c''+b'')/2$, and obtain $E_1 = e(X,X)^{b_1^2a''}$ with advantage at least $\frac{\epsilon}{q_e\cdot q_2+1}$. Then we run $\texttt{Game}_{\mathcal{B}}$ with BDHP parameters $\langle X, a''X, Y_2, Y_2\rangle$ where $Y_2 = b_2X = (c''X - b''X)/2$, where $b_2 = (c''-b'')/2$, and obtain $E_2 = e(X,X)^{b_2^2a''}$ with advantage at least $\frac{\epsilon}{q_e\cdot q_2+1}$. Dividing E_1 by E_2, we obtain $e(X,X)^{a''b''c''}$ with advantage $[\frac{\epsilon}{q_e\cdot q_2+1}]^2$

Proof of Theorem 3 [**IND-RTR-CCA2**] The Theorem result follows from Corollary 13. Let $\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle$ and a random instance of BDH parameters $\langle X, a'X, b'X, c'X\rangle$ be given. Consider an adversary $\mathcal{A}$ against IND-RTR-CCA2. We design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real IND-RTR-CCA2 game for the adversary in order to compute solution to BDHP $e(X,X)^{a'b'c'}$.

We use a biased coin that with probability $\theta > 0$ returns 0 and otherwise returns 1. The optimal value of θ will be determined later.

Setup :

Choice of Generator: $\mathcal{B}$ chooses generator P to be X.

Choice of P_{pub}: $\mathcal{B}$ chooses $P_{pub} = sP$ to be $b'P$.

Choice of pk_a and set $\{(sk_b, pk_b)\}$: $\mathcal{B}$ chooses random $sk_a = a \in \mathbb{Z}_q^*$ and $sk_{b_i} = b_i \in \mathbb{Z}_q^*$. Adversary $\mathcal{A}$ receives $pk_a = aP$ and $\{sk_{b_i} = b_i\}$. Public key pk_a denotes the message sender that will be used in the simulation.

Databases: Databases corresponding to $H_i, i = 1, ..., 4$ are maintained indexed by queries with replies being the values. In addition, $\mathcal{B}$ maintains database $\mathcal{L}$ of possible values of $e(X,X)^{a'b'c'}$ updated in the *Decryption Queries After Challenge* phase.

Oracle queries :

P_T (or H_1) Queries: $\mathcal{B}$ chooses random $c_T \in \mathbb{Z}_q^*$, flips coin and returns c_TP if coin outcome is 0. Otherwise, it returns $c_T \cdot c'P$. Results are stored and repeated queries retrieve answers from the database.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem 2.

Queries for $tkn[T] = sP_T$: $\mathcal{B}$ queries H_1, obtains corresponding c_T and 1) if returns $sH_1(T) = c_T(b'P)$ if $H_1(T) = c_TP$, 2) fails otherwise.

Decryption Queries Before Challenge: $\mathcal{A}$ submits ciphertext $\langle b, T, Q, c_1, c_2\rangle$, where $b = sk_{b_i}$ is the choice of the receiver, pk_a is the sender and T, Q, c_1 and c_2 carry the same meaning as in the previous proofs.

$\mathcal{B}$ computes $rP = Q/b$ and goes through the database of H_3 searching for appropriate r (by multiplying each r by P and comparing with Q/b). If it is not found, false is returned. If it is found, then corresponding σ and m are retrieved. Then database of H_4 is searched for query with σ. If this σ was not queried in H_4 then false is returned. Otherwise, $\mathcal{B}$ computes $c_2 \oplus H_4(\sigma)$ and compares it with m. If they are not equal, false is returned. Next, database of H_1 is queried: if it never returned $H_1(T)$ false is returned. Next $\mathcal{B}$ computes $K = c_1 \oplus \sigma$ and queries the database of H_2 to see if this K

was ever returned. If it was not, false is returned. If it was, it obtains corresponding query given to H_2 and verifies that it is equal to $e(sP + bP, (r + a)H_1(T))$. If they are equal, true is returned. Otherwise, false is returned.

Selection: $\mathcal{A}$ chooses two equal-sized plaintexts m_0,m_1, key $pk_{b^*} \in \{pk_{b_i}\}$ and time T_a. (Note that simulator can determine b^*). *If* $P_{T_a} = c_{T_a}P$ *the simulator quits.* It is assumed that $\mathcal{A}$ did not query (nor will in the future) for $tkn[T_a]$.

Challenge: $\mathcal{B}$ chooses arbitrary $\beta \in \{0, 1\}$, arbitrary $t^* \in \mathbb{Z}_q^*$ and assigns $Q^* = t^*b^*(a'X)$. Then σ^* is chosen, $\mathcal{B}$ chooses two random strings c_1^* and c_2^* and composes and returns ciphertext $c^* = \langle T_a, b^*, Q^*, c_1^*, c_2^* \rangle$ denoting encryption using aP (sender), b^*P (receiver), m_β and T_a. The databases are updated as follows:

H_3: $\mathcal{B}$ puts $rP = t^*a'P$ as a value (marked appropriately in the database) and (σ^*, m_β) as the query. If such (σ^*, m_β) was queried previously, a new choice of σ^* is made. In addition, $t^*a'P$ is checked against existing replies in the database (by multiplying each reply by P and comparing it with $t^*a'P$) and if it already exists, a new choice for t^* is made.

H_4, H_2: updated the same way as in the *Challenge* phase of the proof of Theorem 2

Queries Cont'd: $\mathcal{A}$ has a choice to continue queries or to reply to the challenge. $\mathcal{A}$ is not allowed to query for decryption of c^* using b^* as the receiver and T_a. For decryption queries, $\mathcal{B}$ behaves according to *Decryption Queries After Challenge* phase.

Decryption Queries After Challenge: $\mathcal{A}$ submits ciphertext $\langle T, b, Q, c_1, c_2 \rangle$. $\mathcal{B}$ searches for r corresponding to $Q/b = rP$ in database of H_3. If rP is not found, $\mathcal{B}$ returns false. Otherwise, two cases are possible:

rP is found without r: Then $b^*(rP) = Q^*$. If $c_2 = c_2^*$, then $\sigma = \sigma^*$ and $m = m_\beta$ are retrieved and $\mathcal{B}$ computes $K = c_1 \oplus \sigma$. Otherwise false is returned. If H_2 never returned K, false is returned. Otherwise, the corresponding query J is retrieved.

$P_T = c_T P$: $\mathcal{B}$ can compute the true value of the bilinear map, compare it to J and based on that return true or false.

$P_T = c_T \cdot c'P$: $\mathcal{B}$ returns false and computes $[J/[e(sP, aH_1(T)) \cdot e(rbP, H_1(T)) \cdot e(bP, aH_1(T))]]^{t^{*-1}}$. The answer is written down as possible value of $e(P, P)^{a'b'c'}$ in a list $\mathcal{L}$.

rP is found with r: If $rb^*P = Q^*$, then $\mathcal{B}$ quits, computes $e(rP, sH_1(T_a)) = e(t^*a'P, b'c'P)$ and, taking the result to power t^{*-1}, obtains $e(P, P)^{a'b'c'}$. Otherwise, the same procedure as in the *Before Challenge* case is followed.

Outcome: β is returned or simulation halts.

1. If r corresponding to challenge Q^* was found in the *After Challenge* phase, then the procedure specified there produces $e(X, X)^{a'b'c'}$. This value is the solution to BDHP and is output by $\mathcal{B}$.
2. Otherwise, $\mathcal{B}$ goes through all q_2 adversary queries to H_2 and the list $\mathcal{L}$ that was produced in the *After Challenge* phase and picks a random value. If the choice comes from queries to H_2, then result is divided by $e(sP + b^*P, aH_1(T_a)) \cdot e(Q^*, H_1(T))$ to obtain possible value of $e(rP, sH_1(T)) = e(a'P, b'c'P)^{t^*}$. $\mathcal{B}$ takes the t^{*-1} root and outputs the result as a solution to BDHP. If the choice came from the *After Challenge* list, this choice in its original form is output as a solution to BDHP.

The optimal value of θ maximizes the probability that simulation does not quit during token queries or during selection. This probability is at least $\theta^{q_{tok}} \cdot (1-\theta)$, where q_{tok} is the number of token queries made by $\mathcal{A}$, and is maximized when $\theta = 1 - 1/(q_{tok}+1)$ with value $\frac{1}{e\cdot(1+q_{tok})}$.

The definition of inconsistency, construction of $\texttt{Game}_{\mathcal{C}}$ and the Lemmas in the proof of Theorem 2 naturally carry over with minor modifications. We skip the details and just state the final Corollary:

Corollary 13. *Probability that a random run of the above simulation produces the solution to BDHP is at least* $\frac{2\epsilon}{e\cdot(1+q_{tok})(q_d+q_2)}$.

[**Auxiliary Simulation for RUF-TR-CTXT**] Let $\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle$ and a random instance of BDH parameters $\langle X, a'X, b'X, c'X\rangle$ be given. Consider an adversary $\mathcal{A}$ against RUF-TR-CTXT. Denote $Z = e(a'P, c'P)^b$, where bP is the adversarial public key that will be used in the encryption queries. The challenger is given candidate value of Z. A biased coin is used which with probability $\theta > 0$ outputs 0 and outputs 1 otherwise.

Setup :

Choice of Generator: $\mathcal{B}$ chooses generator P to be X.

s and P_{pub}: $\mathcal{B}$ chooses $P_{pub} = sP$ to be $b'P$.

Choice of pk_s and T_a: $\mathcal{B}$ chooses pk_s to be $a'P$. Adversary receives pk_s.

Databases: Databases corresponding to $H_i, i = 1, ..., 4$ are maintained indexed by queries with replies being the values. In addition, $\mathcal{B}$ maintains database D_s updated in the *Encryption Queries* phase.

Oracle queries :

P_T (or H_1) Queries: $\mathcal{B}$ picks random $c_T \in \mathbb{Z}_q^*$ and flips the coin. If the output is 0, it returns c_TP, otherwise it returns $c_T \cdot c'P$. Result is stored and repeated queries retrieve the answer from the database.

H_2,H_3,H_4 Queries: Same as in the proof of IND-KC-CCA2 Theorem.

Queries for sP_T: $\mathcal{B}$ queries H_1 with T and 1) returns $sH_1(T) = c_T \cdot b'P$ if $P_T = c_TP$, 2) fails otherwise.

Encryption queries: $\mathcal{A}$ submits T, m and public key bP. *The value bP stays fixed after the 1st initial query.* The simulator is expected to output the encryption of m using a' (sender) and bP (receiver). Two cases are considered:

$P_T = c_TP$: $\mathcal{B}$ computes the ciphertext in a normal way. It chooses arbitrary σ, queries H_3 for r and queries H_4 with input σ. Then it computes bilinear map as $e(rP + a'P, sH_1(T) + bH_1(T))$ by noting that $sH_1(T) = c_T \cdot b'P$ and $bH_1(T) = c_T(bP)$. The corresponding query is made to H_2 and $\mathcal{B}$ returns resulting ciphertext $c = \langle rbP, T, c_1, c_2\rangle$.

$P_T = c_T \cdot c'P$: $\mathcal{B}$ chooses random $t \in \mathbb{Z}_q^*$, σ and two random strings c_1, c_2, and returns ciphertext $c = \langle Q = t \cdot a'P, T, c_1, c_2\rangle$. The databases are updated as follows:

H_3: $\mathcal{B}$ puts Q as a value and (σ, m) as the query. If such (σ, m) was queried previously, a new choice of σ is made. In addition, Q is checked against existing replies in the database and if it already exists, a new choice for t is made.

H_4, H_1, H_2: updated the same way as in the *Challenge* phase of the proof of IND-KC-CCA2 Theorem

$\mathcal{B}$ keeps the local database D_s in which it enters the triple $\langle T, t, bP\rangle$. Denote by $TRUE[T, t, bP]$ the true value of $e(sP + bP, (r + a')H_1(T))$, where $rbP = Q$.

Forgery: $\mathcal{A}$ submits ciphertext $c^* = \langle Q^*, T_a, c_1^*, c_2^*\rangle$ and the receiver private key b^* that will be used for verification.

Outcome of Simulation: $\mathcal{A}$ returns forged ciphertext or simulation halts.

Inspection of Databases: $\mathcal{B}$ goes through database D_s, obtains $\langle T, t, bP\rangle$ from each entry, computes $Y/[e(c'P, t \cdot a'P) \cdot e(bP, rH_1(T)) \cdot Z^{c_T}]$ for every query Y to H_2, and then takes the result to power c_T^{-1}. The results are written down as possible values of $e(P, P)^{a'b'c'}$ in a database D_{aux}.

Forgery Examination: If $\mathcal{A}$ submitted a forgery, $\mathcal{B}$ computes $r^*P = {b^*}^{-1}Q^*$ and searches query for r^* in database of H_3. If this query is found, $\mathcal{B}$ retrieves corresponding σ^* and computes $K^* = c_1^* \oplus \sigma^*$. Then H_2 is queried for query corresponding to K^*. If it exists, $\mathcal{B}$ divides the query by $e(sP + b^*P, r^*H_1(T_a)) \cdot e(a'P, b^*H_1(T_a)))$, and writes down the answer as a possible value of $e(P, P)^{a'b'c'}$. Note that if $\mathcal{A}$ wins, Z is correct and the simulation stays consistent then the query corresponding to K^* will be the correct calculation of the corresponding bilinear map and, therefore, the answer computed by $\mathcal{B}$ will in fact be equal to $e(P, P)^{a'b'c'}$ (up to probability of guessing).

Candidate Solutions to BDHP: We output the set of all candidates for $e(P, P)^{a'b'c'}$ compiled in the previous phase.

Definition 14. We say that the simulation is inconsistent if $\mathcal{A}$ makes a query to H_2 with a true value corresponding to one of the $TRUE[T, r]$ in D_s.

The optimal value of θ maximizes the probability that simulation does not quit during token queries or during forgery. This probability is at least $\theta^{q_{tok}} \cdot (1 - \theta)$, where q_{tok} is the number of token queries made by $\mathcal{A}$, and is maximized when $\theta = 1 - 1/(q_{tok} + 1)$ with value $\frac{1}{e \cdot (1+q_{tok})}$.

Assume that Z is correctly computed and simulation does not fail during token queries. Then if $\mathcal{A}$ succeeds in forgery and the simulation stays consistent then the query corresponding to K^* will be the correct calculation of the corresponding bilinear map and, therefore, the answer computed by $\mathcal{B}$ will in fact be equal to $e(P, P)^{a'b'c'}$ (up to probability of guessing). If the simulation is inconsistent, then (irrespective of forgery outcome) the database D_{aux} will a correct value of $e(P, P)^{a'b'c'}$ in its list. We summarize these observations in the following Lemma.

Lemma 15. *Assume that $Z = e(a'P, c'P)^b$ is correct. Let $\epsilon > 0$ be adversarial advantage in the RUF-TR-CTXT and q_{tok} the number of token queries. Then the probability that the set output in the above simulation contains correct solution to BDHP is at least $\frac{\epsilon}{e \cdot (1+q_{tok})}$*

Proof of Theorem 5 [**RUF-TR-CTXT**] Let $\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle$ and a random instance of BDH parameters $\langle X, a'X, b'X, c'X\rangle$ be given along with decisional BDH challenge F. Consider an adversary $\mathcal{A}$ against RUF-TR-CTXT. We design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real RUF-TR-CTXT game for the adversary in order to decide if equality $F = e(P, P)^{a'b'c'}$ holds.

Note that the public key of the adversary used in the encryption queries needs to be revealed only during the encryption queries. Moreover, the simulator does not know the corresponding secret key. Also the *challenge* time is not known *apriori* but can be chosen by adversary adaptively. To deal with lack of apriori knowledge of challenge time, the simulator will use a biased coin that with probability $\theta > 0$ returns 0 and otherwise returns 1. The optimal value of θ will be determined later. To deal with lack of knowledge of secret key corresponding to the public key chosen by adversary during encryption queries, we will run the simulation two times. Finally, to deal with the fact that the adversarial public key used in the encryption queries can be chosen adaptively, in the end we will run the *Auxiliary RUF-TR-CTXT* simulation above feeding it the output of the first two simulations.

Setup :

Choice of Generator: $\mathcal{B}$ chooses generator P to be X.

s and P_{pub}: $\mathcal{B}$ chooses $P_{pub} = sP$ to be $b'P$.

Choice of pk_s: $\mathcal{B}$ chooses pk_s to be $a'P$. Adversary receives pk_s.

Databases: Databases corresponding to $H_i, i = 1, ..., 4$ are maintained indexed by queries with replies being the values. In addition, $\mathcal{B}$ maintains database D_s updated in the *Encryption Queries* phase.

Oracle queries :

P_T (or H_1) Queries: $\mathcal{B}$ chooses random $c_T \in \mathbb{Z}_q^*$, flips coin and returns $c_T P$ if coin outcome is 0. Otherwise, it returns $c_T \cdot c'P$. Results are stored and repeated queries retrieve answers from the database.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem on IND-KC-CCA2.

Queries for sP_T: $\mathcal{B}$ queries H_1, obtains corresponding c_T and 1) it returns $sH_1(T) = c_T(b'P)$ if $H_1(T) = c_T P$, 2) fails otherwise.

Encryption queries: $\mathcal{A}$ submits T, m and bP. *After 1st submission of* bP *it stays fixed for future encryption queries.* The simulator is expected to output the encryption of m using a' (sender) and bP (receiver). Two cases are considered:

$P_T = c_T P$: $\mathcal{B}$ computes the ciphertext in a normal way. It chooses arbitrary σ, queries H_3 for r and queries H_4 with input σ. Then it computes bilinear map as $e(rP + a'P, sH_1(T) + bH_1(T))$ by noting that $sH_1(T) = c_T \cdot b'P$ and $bH_1(T) = c_T(bP)$. The corresponding query is made to H_2 and $\mathcal{B}$ returns resulting ciphertext $c = \langle rbP, bP, T, c_1, c_2 \rangle$.

$P_T = c_T \cdot c'P$: $\mathcal{B}$ chooses $Q = rbP = t \cdot a'P$ for random t, σ and two random strings c_1, c_2, and returns ciphertext $c = \langle Q, bP, T, c_1, c_2 \rangle$. The databases are updated as follows:

H_3: $\mathcal{B}$ puts Q as a value rbP and (σ, m) as the query. If such (σ, m) was queried previously, a new choice of σ is made. In addition, Q is checked against existing replies in the database and if it already exists, a new choice for Q is made.

H_4, H_1 and H_2: updated the same way as in the *Challenge* phase of the proof of IND-KC-CCA2 Theorem

$\mathcal{B}$ keeps the local database D_s in which it enters the triple $\langle c_T, T, t, bP \rangle$. Denote by $TRUE[c_T, T, t, bP]$ the true value of $e(sP + bP, (r + a')H_1(T))$.

Forgery: $\mathcal{A}$ submits ciphertext $c^* = \langle Q^*, T_a, c_1^*, c_2^* \rangle$ and the receiver private key b^* that will be used for verification. *If $P_{T_a} = c_{T_a} P$, the simulation fails*

Outcome of Simulation: $\mathcal{A}$ returns forged ciphertext or simulation halts.

Inspection of Databases: $\mathcal{B}$ goes through database D_s, obtains $\langle c_T, T, Q = t \cdot a'P, bP \rangle$ from each entry and computes $Y/[F \cdot e(Q, H_1(T))]$ for every query Y to H_2. *The results are written down as possible values of $e(a'P, b'c'P)^{t/b} \cdot e(a'P, c'P)^b$ in a database D_{aux}.*

Forgery Examination: If $\mathcal{A}$ submitted a forgery, $\mathcal{B}$ computes $r^*P = {b^*}^{-1}Q^*$ and searches query for r^* in database of H_3.

r^* is found: $\mathcal{B}$ retrieves corresponding σ^* and computes $K^* = c_1^* \oplus \sigma^*$. Then H_2 is queried for query corresponding to K^*. If it exists, $\mathcal{B}$ divides the query by $e(sP + b^*P, r^*H_1(T_a)) \cdot e(a'P, b^*H_1(T_a)))$, and writes down the answer as a possible value of $e(P,P)^{a'b'c'}$. *We record this value as as L_1 (in the 2nd simulation, it would be marked S_2).*

r^* is not found: If $r^* \cdot b \neq t \cdot a'$, $\forall t$, we can safely disregard this case (note that we can check equality $e(r^*P, bP) = e(t \cdot a'P, P)$ to determine that). If equality holds for some $t = t^*$, we can retrieve the corresponding values (σ, m) and derive K from the ciphertext. The query to H_2 (that resulted in K) should be equal to $e(b'P + b^*P, (r^* + a') \cdot c_{T_a} \cdot c'P)$ which we divide by $e(r^*P + a'P, b^* \cdot c_{T_a} \cdot c'P) \cdot F^{c_{T_a}}$ to obtain possible value of $e(P,P)^{a'b'c' \cdot t/b \cdot c_{T_a}}$ and consequently $e(P,P)^{a'b'c' \cdot t/b}$. *This value is entered as S_1 (in the 2nd simulation, it would be marked S_2)*

Second Simulation: We run the adversary with the same random tape. We replace the portion of simulator's random tape used to generate values t in the encryption queries with a new one, keeping the rest of the tape the same. Note that in this case the choice of bP will be the same in both simulations, while the values of t used by simulator in the encryption queries are fresh. Denote by D_{aux_1}, D_{aux_2} the databases D_{aux} in the 1st and 2nd run correspondingly.

Combined Outcome: Three possibilities are considered:

L_1 and L_2: We combine values L_1 and L_2 (possible solutions to BDHP) and put them in set $\mathcal{S}$

D_{aux_1} and D_{aux_2}: We go through $x \in D_{aux_1}$ and $y \in D_{aux_2}$, and compute possible values of $\beta = e(a'P, c'P)^b$. Namely, provided that x and y are computed correctly (and F is correct) we have $x = F^{t_1/b} \cdot \beta$ and $y = F^{t_2/b} \cdot \beta$. Then $\beta = \frac{x^{t_2/(t_2 - t_1)}}{y^{t_1/(t_2 - t_1)}}$.

S_1 and S_2: We go through $x \in S_i$ and $y \in D_{aux_j}$, where $i, j = 1, 2$, computing possible values of $e(a'P, c'P)^b$.

Single run of Auxiliary RUF-TR-CTXT simulator: Finally, we run a single simulaion of original RUF-TR-CTXT feeding it BDH parameters, F and randomly chosen possible value of $e(a'P, c'P)^b$ obtaining a set of possible values of $e(P,P)^{a'b'c'}$. The random tape of adversary stays the same, while the random tape of simulator is the same except for the portion used to compute values t in the encryption queries which is replaced with a new one.

Decision on equality $F = e(P,P)^{a'b'c'}$: If one of the values in $\mathcal{S}$ or in the output of Auxiliary simulator is equal to F we output 1, otherwise we output 0.

Definition 16. We say that 1st (2nd) simulation above becomes inconsistent when $\mathcal{A}$ makes a query to H_2 with a true value corresponding to one of the $TRUE[c_T, T, t, bP]$ in D_s.

Suppose that $F = e(P,P)^{a'b'c'}$ holds and both simulations do not fail. If one simulation is consistent with correct forgery, and second simulation is inconsistent, then either one of the values in $\mathcal{S}$ will be equal to F or we will have correct value of $e(a'P, c'P)^b$. In the latter case, we will output 1 if Auxiliary simulation will have BDHP solution in its output set. If both simulations are inconsistent, then once again we will have a correct value of $e(a'P, c'P)^b$. If both simulations are consistent, the only case of concern is when both output forgeries in the "r^* is not found" category, in which case we can compute neither feeding value to Auxiliary simulation nor BDHP solution. However, we note that in case of consistent simulation, forgery cases from "r^* is not found" and "r^* is found" categories happen with the same probability since adversary cannot distinguish if $r^* \cdot b = t \cdot a'$ or not. The formal analysis follows below.

We use well-known probability lemma (see [28] for the statement of this Lemma) to ensure that all runs of simulations behave in expected way (since we use the same random tape of adversary and random tape of the simulator is partially changed). Denote by ϵ the adversarial advantage in RUF-TR-CTXT. Denote by P the probability that the RUF-TR-CTXT simulation either 1) stays consistent, does not fail and produces correct forgery, or 2) is inconsistent and does not fail, computed over all possibilities of adversarial and simulator random tapes. Then the probability that all three simulations fall in one of the above categories is at least $P^4/16$. [10]

The optimal value of θ maximizes the probability that simulation does not quit during token queries or during forgery. This probability is at least $\theta^{q_{tok}} \cdot (1-\theta)$, where q_{tok} is the number of token queries made by $\mathcal{A}$, and is maximized when $\theta = 1 - 1/(q_{tok}+1)$ with value $\frac{1}{e \cdot (1+q_{tok})}$. Let J denote probability that simulation is consistent. Then $P = \frac{1}{e \cdot (1+q_{tok})} \cdot [J \cdot \epsilon + (1-J)] \geq \frac{\epsilon}{e \cdot (1+q_{tok})}$. We make the following comments for the cases with respect to the first two simulations:

Case 1: *One simulation run is inconsistent and the other one either is inconsistent or consistent producing correct forgery.* In this case, one of possible $(q_e \cdot q_2 + 4) \cdot q_e \cdot q_2$ values that we can possibly feed to the auxiliary simulation is correct and, provided that we feed correct value and auxiliary simulation does not fail, we obtain correct answer to the decisional BDHP. Thus, in this case our advantage is at least $1/[(q_e \cdot q_2 + 4) \cdot q_e \cdot q_2]$.

Case 2: *Both runs are consistent.* Note that if correct forgery is produced then with equal probability it can be in the "r^* is not found" or "r^* is found" categories. Thus, our advantage in this case is at least $1/4$.

Case 3: *Either both runs are consistent with incorrect forgery or one is inconsistent and the other one is consistent with incorrect forgery.* In this case, our simulations do not provide results and we simply flip a coin returning 1 or 0 at random. Our advantage is 0.

[10] Let $\mathcal{X}$ be the space of possible values of portion of simulator tape that stays the same, $\mathcal{Y}$ the remaining space of simulator's random tape and $\mathcal{Z}$ the set of adversarial random tapes. Then there exists a subset $\mathcal{D} \subset \mathcal{X} \times \mathcal{Z}$ of size at least $P/2$ such that for any fixed $(x,y) \in \mathcal{D}$, the probability that simulation falls within 1) or 2) above is at least $P/2$ for random $y \in \mathcal{Y}$. It follows that if we pick $\mathcal{X}$ and $\mathcal{Z}$ at random, the probability that we obtain "good" values is at least $P/2$. Using these "good" values and picking $\mathcal{Y}$ at random, the probability that we get desired outcome is at least $P/2$. The result follows

Probability that either Case 1 or Case 2 happen is at least $P^4/16$ and the our advantage in this case is at least $1/[(q_e \cdot q_2 + 4) \cdot q_e \cdot q_2]$. Probability of Case 3 is at most $(1-P^4/16)$ and our advantage is 0. Recalling that $P \geq \frac{\epsilon}{e \cdot (1+q_{tok})}$, the Theorem statement follows immediately.