A Scheme for Timed-Release Public Key Based Authenticated Encryption

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

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

Abstract. We propose the first provably-secure scheme that provides public key based authenticated encryption with timed-release property (TR-PKAE). Any application that requires delayed opening of information can use our construction. Detailed security model for TR-PKAE is derived from sealed-bid auction example. The proposed protocol has minimal overhead in storage, computation and communication, while providing strong security as well as diverse functionalities. The construction requires minimal infrastructure overhead, which can be shared among multiple applications.

Keywords: timed-release, public key authenticated encryption, timelock puzzle

1 Introduction

The goal of timed-release cryptography is to "send a message into the future". One way to do this is to encrypt a message such that the receiver cannot decrypt the ciphertext until specific time in the future. A solution to this problem has immediate applications in the real world. For example, in sealed-bid auctions, one can prevent prior opening of bids by a dishonest auction house [30]. E-voting is an another example that requires delayed opening of votes. It can be also used for delayed verification of a signed document, such as lottery [32] and check cashing. Other applications include release of important documents and press releases among many others. In order to further motivate the need for our construction, we discuss in detail sealed-bid auction example in Section 1.1.

The problem of timed-release cryptography was first mentioned by May [25] and then discussed in detail by Rivest *et. al.* [30]. Let us assume that the message sender is called Alice and the receiver is Bob, and Alice wants to send a message to Bob such that Bob will not be able to open it until certain time. The possible solutions are divided into two ways [1]:

Time-lock puzzle approach. Alice would encrypt her message such that Bob would have to perform non-parallelizable computation without stopping for the required wait time in order to decrypt it.

Agent-based approach. Alice could use trusted agents and encrypt the message such that Bob will need some secret value, published by the agents on the required date in order to decrypt the message.

Time-lock puzzle approach puts immense computational overhead on the message receiver, which makes it impractical for real-life scenarios. In addition, knowing computational complexity does not always lead to correct estimation of time that Bob needs to decrypt the message. Still, this approach is widely used for specific applications [6, 5, 32, 19, 18]. Using agent-based approach 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.

[1] We do not consider schemes that require Alice's presence at the time of message opening, since this will limit applications.

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 by Di Crescenzo *et. al.* [15], 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 [14], the authors proposed to use bilinear map based IBE scheme [9] for timed-release encryption. In particular, one can replace public key in IBE scheme by future time. An agent would publish a secret key corresponding to current day and consequently the ciphertext will not be decryptable until the specified future time. Another example using IBE was also proposed in [24], which again replaces identity in the encryption primitive with future time. Similar IBE-based approach was presented in [7]. Still security of these IBE-based approaches has never been proven. Furthermore, the constructions were either more expensive than our approach or did not provide sufficient functionalities.

1.1 Motivating Application: Sealed-Bid Auction

To motivate our construction, we first investigate the relation between timed-release cryptography and sealed-bid auction. In a sealed-bid auction, bidders submit their bids in closed form to the auction board. Once the bidding is closed, the bids are opened and the winning bid is chosen according to some publicly known metric [16]. Examples include government construction bidding auctions, artwork and real estate sales among others. Because of special nature of such auctions, many security requirements need to be carefully analyzed and appropriate measures should be specified [28]. One of the problems that may occur in such auctions is cheating by the auction board [21]. A bid may be opened before the closing time and communicated to another bidder who can adjust his own bid appropriately. Thus, enforcing delayed opening of the bid plays a central role in such applications. The following are the main requirements in sealed-bid auctions:

1. The bids should not be opened by anyone until the bidding is closed.
2. The bids should be decryptable only by the auction board and only after the bidding close.
3. The auction board should be able to verify the identity of the bidder when the bids are opened.
4. The auction board should not be able to disavow bid submission.
5. Bidder should not be able to repudiate his bid.
6. Other requirements include correct calculation of winning bid, verification by bidders among many more, but they are beyond the scope of this paper and we presently dispense with these considerations. See [16, 23, 21, 12, 28] for more details.

In the rest of the paper we concentrate solely on timed-release authenticated public key based encryption that addresses the first three requirements. [2]

[2] Non-repudiation of the bidder (fifth requirement) can be provided by digital signature mechanisms. An alternative way is to construct an encryption scheme that allows receiver to (efficiently) prove to a third party that the bid was indeed submitted by the alleged bidder. The bid submission can be repudiated by the auction board (fourth requirement) as follows: it can say that 1) the bid was never submitted, 2) the ciphertext is not decryptable, 3) the bid has a different value than what the bidder had submitted. The first attack can be resolved by requiring that the auction board signs the bids and publicly posts the signatures in the time-frame between bid-close and bid-opening. Note that dropping bids without knowing their contents may not be in the interests of the board. To deal with the second attack, the bidder can provide a proof that the ciphertext is decryptable. One way to resolve the third attack is for the bidder to reveal the bid value and prove that the ciphertext contains this bid. We suggest such mechanism in Appendix A due to page constraints.

1.2 PKAE and TR-PKAE

The first three requirements in Section 1.1 can be provided by combining timed-release cryptography with public key based authenticated encryption (PKAE) [2, 1]. The goal of PKAE is to provide privacy and authentication at the same time. In PKAE, the sender uses his private key and receiver's public key to encrypt a message such that 1) the resulting encryption stays confidential with respect to a third party (adaptive IND-CCA, or IND-CCA2), and 2) it provides ciphertext/plaintext unforgeability with respect to the third party (TUF-CTXT/PTXT). In addition, it is desirable to ensure that 3) sender's ciphertexts stay confidential even if his private key is compromised (IND-KC-CCA2) [3] and 4) the receiver cannot forge valid ciphertexts from the sender to itself (RUF-CTXT/PTXT).[4] If RUF-CTXT/PTXT does not hold and the receiver's private key is compromised, the adversary will be able to impersonate the sender. A generic PKAE scheme that satisfies all the above is given in [2] and combines public key encryption and digital signature. A more efficient PKAE scheme DHETM [2] is based on symmetric encryption using a shared Diffie-Hellman key (at the expense of losing unforgeability by the receiver).

One timed-release primitive can be derived from Identity-Based Encryption such as FullIdent by Boneh and Franklin [9]. The ID is replaced by time and the private key generator periodically outputs the private key associated with current time instead of the ID. The receiver will not be able to decrypt until the private key associated with the time used in encryption is published. One can extend FullIdent to provide also confidentiality and authenticity of messages: the plaintext can be first encrypted by PKAE and the result can be encrypted by this timed-release primitive, or vice versa [24]. So far, concrete schemes for these combinations as well as security or efficiency analysis are not known to our knowledge. This paper proposes an efficient scheme for TR-PKAE along with rigorous security analysis.

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

- The proposed protocol is as efficient as FullIdent in terms of computational and spatial complexity.
- The proposed protocol requires minimal infrastructure (i.e. agent) that can be shared among many applications. The proposed protocol can be naturally converted to threshold version, which provides robustness as well as stronger security by allowing outputs of multiple agents to be used.
- The proposed protocol provides provable security under the random oracle model. Namely, it provides
  - **IND-CCA2** even when sender's private key and infrastructure are compromised.
  - **TUF-CTXT/PTXT:** unforgeability of the ciphertext/plaintext by the third party (even when the infrastructure is compromised).
  - **IND-RTR-CCA2:** timed-release version of IND-CCA2 by the receiver. In other words, before the designated time, IND-CCA2 is provided with respect to the receiver.
  - **RUF-TR-CTXT/PTXT:** the receiver cannot forge ciphertext from a sender to the receiver for the future time.
- In addition, even though our protocol does not use digital signatures, receiver can still prove to a third party the message origin. [5]

Organization: The rest of the paper is organized as follows. An overview of our approach is given in Section 2. In Section 3 we formulate the TR-PKAE cryptosystem and security definitions. In Section 4 we describe proposed protocol and state security results. Section 5 discusses efficiency of the protocol. Appendix A shows how a ciphertext receiver can prove message origin to a third party, state security properties and corresponding security results. Finally, in Section 6 we conclude the paper.

[3] It stands for IND-CCA2 even after Key Compromise.
[4] IND-CCA2, TUF/RUF-CTXT/PTXT are introduced as desirable properties in [2].
[5] Once again, this part was moved to Appendix A due to page constraints

2 Timed-Release Infrastructure: Overview

The central concept of our approach is a public agent, call it TiPuS (Timed-release Public Server). Every unit of time T, say every day, it publishes new *self-authenticating* information I_T which combines publicly computable (i.e., it can be precomputed for any future or past day by anyone) information P_T and its unique private information S. In other words, TiPuS acts similarly to NTP server [27]. The value P_T can be published on its web page or broadcasted when broadcast channel is available. Alice can encrypt a message using P_T, her private key and Bob's (receiver'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 admissible bilinear map defined in Section 4.1. Public parameters such as groups, bilinear map and a generator $P \in \mathbb{G}_1$ are setup independently of TiPuS. Each TiPuS chooses a secret $s \in \mathbb{Z}_q$ and publishes authenticated $P_{pub} = sP$. On day T, it publishes $I_T = sH(T)$ (which corresponds to the private key for identity T in Full-Ident), where H is a cryptographic hash function. Returning to the previous notation, we have $s = S, H(T) = P_T$. The value of P_T is *self-authenticating*: namely, each user can compute $e(sP, H(T))$ and verify if it is equal to $e(P, sH(T))$, since by bilinearity $e(sP, H(T)) = e(P, sH(T)) = e(P, H(T))^s$.

Every user A has private/public key pair $(SK_A, PK_A) = (a, aP)$, and aP is certified by a CA. One can authenticate if A indeed possesses A. (for example, using short signatures [10]). Using the information provided by TiPuS, the mentioned private/public keys and bilinear map, one can construct an efficient TR-PKAE. A high level description is as follows:

Setting: Alice is the sender and Bob is the receiver with private/public pairs (a, aP) and (b, bP) respectively.

Encryption: Alice chooses random r, computes bilinear map $d = e(sP+bP, (r+a)P_T)$, applies hash function H_2 to obtain $K = H_2(d)$, and then encrypts message m as $E_K(m)$, where E_K is a symmetric encryption using key K. Bob also receives $r \cdot bP$.

Decryption: Bob can extract rP from $r \cdot bP$, and having sP_T can compute d as $e(rP + aP, sP_T + bP_T)$ [6]. Applying hash function H_2, Bob computes K and uses it to decrypt $E_K(m)$.

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:

User Secret vs TiPuS Secret: The secret value of TiPuS is not related to private keys of users. It will be shown later that compromise of TiPuS does not jeopardize user secrets (more precisely, the protocol provides confidentiality and unforgeability with respect to TiPuS).

Usage: The published value sP_T can be shared among multiple applications.

Scalability: The protocol can take full advantage of 1) several independent TiPuS's (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 $\sum s_iP_T$), 2) threshold generation of sP_T (using Pederson's distributed threshold scheme [29], a brief setup is needed between token generators after which they can function independently). The increase in computational complexity is minimal when such schemes are applied to the protocol.

3 Basic Definitions

3.1 Basic Cryptosystem

The goal of proposed *Timed-Release Public Key Based Authenticated Encryption* (TR-PKAE) is to provide public key based authenticated encryption that takes sender's private key, receiver's public key and

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

designated time so that the resulting ciphertext can be decrypted only by receiver and only starting with designated time using receiver's private key, sender's public key and some secret that will be disclosed only on designated time. We specify TR-PKAE by the following randomized algorithms:

$\mathsf{General\ Setup}$: On input of security parameter k, it produces in a randomized manner public parameters π_g, which include hash functions, message and ciphertext spaces among others.

$\mathsf{Timed\text{-}Release\ Setup}$: On input of π_g, it produces in a randomized manner a pair $\langle \delta, \pi_{tr} \rangle$ 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.

$\mathsf{KeyGenerator}_{\pi_g}$: On input of valid private key sk computes corresponding public key pk.

$\mathsf{TokenGenerator}_{\pi,\delta}$: On input of valid time encoding T computes corresponding private token $tkn[T]$ using $\langle \delta, \pi \rangle$. This is the functionality performed by TiPuS which (at certain time-intervals) publishes $tkn[T]$ at time T.

$\mathsf{Encrypt}_{\pi}$: On input $\langle sk_A, pk_B, m, T \rangle$ returns authenticated timed-release ciphertext c denoting encryption from sender A to receiver B of message m and time encoding T.

$\mathsf{Decrypt}_{\pi}$: On input $\langle pk_A, sk_B, \widehat{c}, tkn[T] \rangle$ outputs plaintext $\widehat{m}$ and "true" if decryption is successful and "false" otherwise.

For consistency, we require that, $\forall pk_A, pk_B$, and setup values, if $c = \mathsf{Encrypt}_{\pi}[sk_A, pk_B, m, T]$ and $\widehat{m} = \mathsf{Decrypt}_{\pi}[pk_A, sk_B, c, tkn[T]]$ then $\widehat{m} = m$.

3.2 Security

The introduction of $\mathsf{TokenGenerator}$ and the $\mathsf{master\ secret}$ leads to the following question: how much should the security of the cryptosystem depend on these timed-release additions? Should the cryptosystem maintain typical PKAE [2] security properties even if the master secret is compromised? One of our goals is to separate the timed-release infrastructure from PKAE security as much as possible. That is, the timed-release infrastructure should only affect the timed-release properties of the cryptosystem and not the PKAE properties. With this in mind, we discuss required security properties below.

3.2.1 Confidentiality

Suppose Alice is the sender, Bob is the receiver and Alice composes a ciphertext for Bob using designated time T. It is standard to require that the PKAE cryptosystem be secure against adaptive IND-CCA (IND-CCA2) adversaries [33, 4, 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 sender and receiver will still be guaranteed IND-CCA2 security against any third party. To strengthen security even more, one can require that the scheme stays IND-CCA2 even if the private key of the sender is compromised, ensuring that the adversary will not be able to obtain information on any ciphertexts generated by the sender, even though it will be able to decrypt ciphertexts received by the sender.

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-PKAE cryptosystem must provide some confidentiality guarantees against the receiver itself until the corresponding token is made available.

Below, we sketch two particular games which will be used in security proofs: 1) IND-CCA2 game in which the adversary is given the private key of the sender and the master secret and 2) IND-RTR-CCA2 game in which the ciphertext receiver is launching an IND-CCA2 attack for a given designated time.

IND-CCA2 with Key Compromise (IND-KC-CCA2): Below we sketch a simple variation of adaptive IND-CCA (IND-CCA2) game in which the adversary is given the private key of the sender (i.e. we would like the scheme to provide IND-CCA2 confidentiality even in the case of sender's key compromise) and the master secret (to separate confidentiality against third party from the timed-release infrastructure).

We say that function $g : \mathbb{R} \rightarrow \mathbb{R}$ is negligible if $g(k)$ is smaller than $1/f(k)$ for any polynomial f (and $k > n_f$). TR-PKAE encryption scheme is said to be secure against an adaptive chosen ciphertext attack with key compromise (IND-KC-CCA2) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-KC-CCA2}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-KC-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$) against the challenger in the following IND-KC-CCA2 game:

Setup: The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$, receiver public/private key pair (pk_b, sk_b) and sender public/private pair set $\{(pk_a, sk_a)\}$. The adversary is given $\langle \pi, \delta, \{sk_a\}, pk_b \rangle$.

Pre-Challenge: Adversary issues the following queries

Random Oracle Queries: Adversary may query any random oracle, which will model hash functions.

Decryption Queries: Adversary submits ciphertext, time encoding T and pk_a. Challenger responds with decryption of ciphertext using pk_a (sender), sk_b (receiver) and time encoding T.

Selection: Adversary chooses two distinct equal-size plaintexts m_0, m_1, time T, sender key sk_a and submits it to the challenger.

Challenge: Challenger flips $\beta \in \{0, 1\}$ and returns encryption of m_β to adversary using sk_a (sender), pk_b (receiver) and time T.

Queries Repeated: Adversary repeats queries but does not ask to decrypt the challenge ciphertext using challenge time and keys.

Guess: Adversary answers the challenge with $\widehat{\beta}$ and wins if $\widehat{\beta} = \beta$

We define $\mathbf{Adv}^{\text{IND-KC-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$, where k is the system security parameter and probability is taken over random bits used by the challenger and adversary.

Timed-Release Receiver IND-CCA2 (IND-RTR-CCA2): A prerequisite of a secure TR-PKAE scheme is message confidentiality against the receiver itself prior to the time when the secret $tkn[T]$ that corresponds to the designated time is made available. We modify the IND-CCA2 game to restrict adversary access to $tkn[T]$ for designated time, which means that master secret is no longer available to the adversary. Note that this type of security is inherently dependent on the timed-release infrastructure. The adversary plays the ciphertext receiver in the game. In the decryption queries, we allow adversary to decrypt messages destined even for this designated time as long as the ciphertext is different from the challenge.

We say that TR-PKAE encryption scheme is timed-release secure against a receiver adaptive chosen ciphertext attack (IND-RTR-CCA2) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-RTR-CCA2}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-RTR-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$) against the challenger in the following IND-RTR-CCA2 game:

Setup: The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$, sender public/private key pair (pk_a, sk_a), receiver public/private pair set $\{(pk_b, sk_b)\}$ and designated time T_a. The public key pk_a, set $\{sk_b\}$ and T_a are given to the adversary.

Pre-Challenge :

Random Oracle Queries: Adversary may query any random oracle.

Queries for $tkn[T]$: Adversary submits T where $T \neq T_a$ and receives $tkn[T]$.

Decryption Queries: Adversary submits ciphertext and time T. Challenger responds with decryption of ciphertext using pk_a (sender), sk_b (receiver) and $tkn[T]$.

Selection: Adversary chooses two distinct equal-size plaintexts m_0,m_1 and submits them to the challenger.

Challenge: Challenger flips $\beta \in \{0, 1\}$ and returns encryption of m_β to adversary using sk_a (the sender), pk_b (the receiver) and time T_a.

Queries Repeated: Adversary repeats queries but does not ask to decrypt the challenge ciphertext using the same parameters used in the challenge.

Guess: Adversary answers the challenge with $\widehat{\beta}$ and wins if $\widehat{\beta} = \beta$

We define $\mathbf{Adv}^{\text{IND-RTR-CCA2}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$.

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

3.2.2 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 the ciphertext forgery (plaintext forgery is defined analogously). We consider two types of ciphertext forgery: 1) forgery by adversary that does not know the sender's and receiver's private keys (TUF-CTXT) and 2) forgery by ciphertext receiver itself (RUF-CTXT) [2]. If the TR-PKAE is not secure against TUF-CTXT then the scheme cannot claim authentication properties since a third-person may be able to forge decryptable (perhaps containing junk) ciphertext between two users. If TR-PKAE is not secure against RUF-CTXT, then 1) the receiver itself can generate the ciphertext allegedly coming from another user to itself, which means that 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, and 2) consequently, if receiver private key is compromised, the attacker can impersonate any sender to this receiver. We introduce the following games which will be used in security proofs.

Timed-Release RUF-CTXT/PTXT (RUF-TR-CTXT/PTXT): We introduce a slightly weaker notion of RUF-CTXT, which requires that the receiver should not be able to forge ciphertext to itself for a future date. Given such unforgeability: 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 ciphertext was generated by the alleged sender, provided he can produce a proof of ciphertext existence prior to the decryption date. The game below is a slight modification of RUF-CTXT in which the receiver is not given access to one particular token. We say that TR-PKAE encryption is secure against timed-release RUF-CTXT, denoted by RUF-TR-CTXT, if no polynomial adversary (denoted by $\mathcal{A}_{\text{RUF-TR-CTXT}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the following RUF-TR-CTXT game:

Challenger Setup: The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$, public/private key pair (pk_s, sk_s) of sender and time T_a. The adversary receives $\langle \pi, pk_s, T_a \rangle$.

Adversary Setup: Adversary runs setup with security parameter k, generates public key pk_r. Note that the corresponding private key is not known to the challenger. In fact, the adversary may not know it itself.

Pre-Forgery :

Random Oracle Queries: Adversary may query any random oracle

Queries for $tkn[T]$: Adversary submits $T \neq T_a$ and receives $tkn[T]$

Encryption Queries: Adversary submits plaintext m, time T and obtains encryption using sk_s (sender), pk_r (receiver) and T.

Forgery: Adversary submits ciphertext c and private key sk.

Outcome: Adversary wins the game if c successfully decrypts using pk_s (sender), sk (receiver) and T_a, and c was not obtained during encryption queries using the same parameters.

We define $\mathbf{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_s, sk, T_a] = true]$. By requiring that in the above game the decrypted plaintext m in the outcome was not submitted during encryption queries, we obtain corresponding notion of RUF-TR-PTXT. We skip the details.

TUF-CTXT (PTXT) In addition, below we state a time-independent TUF-CTXT game. 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. Perhaps we should only require TUF-TR-CTXT, which will automatically be provided given RUF-TR-CTXT security. The main reason for TUF-CTXT security is to ensure that some kind of unforgeability is guaranteed even if the master secret is compromised, i.e. we would like to separate timed-release functionality from PKAE. Thus, in TUF-CTXT the master key is given to the adversary. We say that TR-PKAE encryption is secure against third-person chosen-plaintext ciphertext forgery (TUF-CTXT) if no polynomial adversary (denoted by $\mathcal{A}_{\text{TUF-CTXT}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{TUF-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$) against the challenger in the following TUF-CTXT game:

Setup: The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$ and public/private key pairs (pk_a, sk_a) and (pk_b, sk_b) of sender and receiver correspondingly. The public keys and both δ and π are given to the adversary.

Pre-forgery :

Random Oracle Queries: Adversary may query any random oracle

Encryption Queries: Adversary submits plaintext m, time T and obtains encryption using sk_a (sender), pk_b (receiver) and T.

Forgery: Adversary submits ciphertext c and T.

Outcome: Adversary wins the game if c successfully decrypts using pk_a (sender) and sk_b (receiver), and c was not obtained during encryption queries.

We define $\mathbf{Adv}^{\text{TUF-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_a, sk_b, T] = true]$. As in the previous cases, we obtain corresponding TUF-PTXT game.

4 The Proposed TR-PKAE

First, we review the bilinear maps, the assumptions that we make and BDHP definition. Then we specify the proposed protocol and discuss security.

4.1 Bilinear Maps

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 $\mathbb{G}_2$).

A map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is called an admissible bilinear map if it satisfies the following conditions:

1. Bilinearity For any $P, Q \in \mathbb{G}_1$ and $a, b \in Z_q$, $e(aP, bQ) = e(P, Q)^{ab}$.
2. Non-degeneracy $e(P, Q) \neq 1$ for at least one pair of $P, Q \in \mathbb{G}_1$.
3. Efficiency There exists an efficient algorithm to compute the bilinear map.

The Weil and Tate pairings can be used to construct an admissible bilinear pairing. For groups, one can take $\mathbb{G}_1$ to be a subgroup of an elliptic curve and $\mathbb{G}_2$ a subgroup of the multiplicative group of a finite field. See the details of pairings and the conditions on curves in [13].

We make several comments about $\mathbb{G}_1, \mathbb{G}_2$ and $e(\cdot, \cdot)$.

1. Discrete Logarithm Problem (DLP) is assumed to be hard in $\mathbb{G}_2$
2. It follows that DLP is also hard in $\mathbb{G}_1$ [26]
3. Decisional Diffie-Hellman Problem (DDHP) is easy in $\mathbb{G}_1$ [22].
4. Decisional Diffie-Hellman Problem (DDHP) is hard in $\mathbb{G}_2$.
5. Hardness of DDHP in $\mathbb{G}_2$ implies that, $\forall Q \in \mathbb{G}_1^*$, inverting the isomorphism that takes $P \in \mathbb{G}_1$ and computes $e(P, Q)$ is hard [9]

Let $\mathcal{G}$ be *BDH Parameter Generator* [9], i.e. $\mathcal{G}$ is 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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ 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 BDHP for $\mathcal{G}$ if there exists k_0 such that:

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

We say that $\mathcal{G}$ satisfies Bilinear Diffie-Hellman Assumption (BDH assumption) if for any randomized polynomial algorithm $\mathcal{A}$ and any polynomial $f \in \mathbb{Z}[x]$ we have $Adv_{\mathcal{G},\mathcal{A}}(k) < 1/f(k)$ for sufficiently large k

4.2 Description of the Scheme

Let $\mathcal{G}$ be *BDH Parameter Generator* that satisfies BDH assumption.

General 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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Arbitrary generator $P \in \mathbb{G}_1$ is chosen.

2: The following cryptographic hash functions are chosen: 1)[7] $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$, 2) $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n, 3) $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{Z}_q^*$ and 4) $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$. 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 $\pi_g = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, H_i, i = 1...4\rangle$

Timed-Release Setup :

1: Random $s \in \mathbb{Z}_q^*$ is chosen and one sets $P_{pub} = sP$.

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

KeyGenerator: Given private key $sk = a \in \mathbb{Z}_q^*$, the corresponding public key pk is $aP \in \mathbb{G}_1^*$.

[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].

TokenGenerator: On input of time encoding $T \in \{0,1\}^n$ outputs sP_T where $P_T = H_1(T)$

Encrypt: Given private key sk_a of sender, public key pk_b of receiver, plaintext $m \in \mathcal{M}$ and designated time encoding T, encryption is done as follows: 1) random $\sigma \in \{0,1\}^n$ is chosen, one computes $r = H_3(\sigma, m)$ and sets $Q = r \cdot pk_b$, 2) symmetric key is computed as $K = H_2[e(P_{pub} + pk_b, (r + sk_a)P_T)]$ and 4) 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 and pk_b and time T, one decrypts it as follows: 1) $tkn[T] = sP_T$ is obtained, 2) one computes $R = sk_b^{-1}Q$ and $\widehat{K} = H_2[e(R+pk_a, sP_T+sk_bP_T)]$, 3) one retrieves $\widehat{\sigma} = c_1 \oplus \widehat{K}$ and then $\widehat{m} = c_2 \oplus H_4(\widehat{\sigma})$ and 4) one verifies $R = H_3(\widehat{\sigma}, \widehat{m})P$

The symmetric encryption scheme above is due to Fujisaki and Okamoto [17]. Next we show that the proposed encryption scheme is consistent. 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 Decrypt computations the following hold:

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. It follows that $\widehat{\sigma} = \sigma$ since $c_1 \oplus \widehat{K} = (\sigma \oplus K) \oplus K = \sigma$
4. $\widehat{m} = m$ since $c_2 \oplus H_4(\widehat{\sigma}) = (m \oplus H_4(\sigma)) \oplus H_4(\sigma) = m$
5. It follows that $R = rP = H_3(\widehat{\sigma}, \widehat{m})P$

Thus the original plaintext is retrieved.

4.3 Security of the Scheme

The following security results apply to the proposed TR-PKAE. The proofs are given in Appendix B. First, we note that proposed scheme satisfies a stronger version of IND-CCA2 with sender key compromise.

Theorem 1 (IND-KC-CCA2). *Let $\mathcal{A}$ be IND-KC-CCA2 adversary, q_d be the number of decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}^{\text{IND-KC-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq \frac{2\epsilon}{q_d+q_2}$ and running time $O(time(\mathcal{A}))$.*

Also, the proposed protocol is TUF-CTXT secure.

Theorem 2 (TUF-CTXT). *Let $\mathcal{A}$ be TUF-CTXT adversary, let q_e be the number of encryption queries and q_2 be the number of queries to random oracle H_2. Assume that $\boldsymbol{Adv}^{\text{TUF-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq [\frac{\epsilon}{q_e \cdot q_2+1}]^2$ and running time $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

The corresponding result for TUF-PTXT with the same inequality is proved similarly, only minor details need to be modified in the proof.

Theorem 3 (IND-RTR-CCA2). *Let $\mathcal{A}$ be IND-RTR-CCA2 adversary, let q_d be the number decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}^{\text{IND-RTR-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq \frac{2\epsilon}{q_d+q_2}$ and running time $O(time(\mathcal{A}))$.*

Theorem 4 (RUF-TR-CTXT). *Let $\mathcal{A}$ be RUF-TR-CTXT adversary, let q_e be the number of encryption queries and q_2 be the number of queries to random oracle H_2. Assume that $\boldsymbol{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq \frac{\epsilon^2}{(q_e \cdot q_2)^2+2}$ and running time $O(time(\mathcal{A})) + O([q_e \cdot q_2]^2)$.*

Again RUF-TR-PTXT with the same inequality holds as well. Only minor modifications in the proof are required.

5 Efficiency

We note that the proposed scheme is almost as efficient as the FullIdent [9] in terms of computational and spatial complexity. First, encryption operation in FullIdent and the proposed scheme for TR-PKAE both require the same number of significant operations: *1 bilinear pairing, 1 MapToPoint, 2 exponentiations in* $\mathbb{G}_1$. The decryption in IBE 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 (suggested in Section 1.2) that combine IBE 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]. 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. The library uses Tate pairing for the bilinear map. 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.

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
Expo in $\mathbb{F}_p$	1024	160	3.93
Scalar Mul in EC over $\mathbb{F}_p$	160	160	3.44
BLS sign	512	160	7.33
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

6 Concluding Remarks and Future Work

In this paper, we presented a new cryptographic scheme for timed-release public-key based authenticated encryption (TR-PKAE), and proved that it is IND-CCA2 (even when sender key is compromised) and TUF-CTXT secure. Our security model introduces additional timed-release security notions such as timed-release IND-CCA2 (against the receiver) and timed-release RUF-CTXT: the receiver cannot distinguish ciphertexts until the designated time and cannot forge ciphertexts to itself for the future designated time.

In the proposed schemes, the past tokens have to be stored in a repository in case a user attempts to decrypt message with designated time well in the past. As a result, the required storage grows linearly. The authors recently became aware of a new development proposed by Boneh *et al.* [8] in HIBE (hierarchical identity based encryption) [20] that allows to reduce required storage to $O(\log^{3/2} T)$, where T is the upper bound on the number of time periods when tokens are published. In proposed TR-PKAE, the bilinear map used to generate symmetric key is $e(sP + bP, (r + a)P_T)$. We can adapt proposed TR-PKAE to HIBE as follows. Using notation in [8] (in particular multiplicative notation for both groups) for Hybrid scheme, the sender computes now $\{e(g^b \cdot g^\alpha, g_2)^{r+a}, g^{rb}, g^a, L\}$, where g replaces P, (b, g^b) and (a, g^a) are now the private/public key pairs of receiver and sender, L is the remaining part of 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 bilinear map computations does not increase when we add authentication property. We also note that now we can now use the property of [8] to reduce the amount of storage required to store tokens. In particular, at each time-period the agent need only $O(\log^{3/2} T)$ amount of storage from which all previous tokens can be derived. The HIBE proposed by Boneh *et al.* relies on BDHE assumption while TR-PKAE relies on BDHP. In the future we plan to analyze thoroughly the security of proposed TR-PKAE when adapted to HIBE.

References

1. M. Abdalla, M. Bellare, and P. Rogaway. The oracle diffie-hellman assumptions and an analysis of dhies. In *LNCS, volume 2020, pages 143–158. Springer-Verlag.*, 2001.
2. J. H. An. Authenticated encryption in the public-key setting: Security notions and analyses. `http://eprint.iacr.org/2001/079/`, 2001.
3. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Proceedings CRYPTO 2000, Springer LNCS 1880, pp 255 - 270*, 1999.
4. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Proceedings CRYPTO 1998, Springer LNCS 1462, pp 26 - 45*, 1998.
5. M. Bellare and S. Goldwasser. Encapsulated key escrow. In *In MIT/LCS/TR-688*, 1996.
6. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Proc. of Asiacrypt '00, Lecture Notes in Computer Science, Vol. 1976*, 2000.
7. I. F. Blake and A. C.-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. `http://eprint.iacr.org/2004/211/`, 2004.
8. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *to appear in Eurocrypt 2005*, 2005.
9. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Proc. of Crypto '01*, 2003.
10. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proc. of Asiacrypt '01*, 2001.
11. X. Boyen. Multipurpose identity-based signcryption: A swiss army knife for identity-based cryptography. In *Proceedings CRYPTO 2003, Springer LNCS 2729, pp 382 - 398*, 2003.
12. F. Brandt. Fully private auctions in a constant number of rounds. In *In Proceedings of the 7th Annual Conference on Financial Cryptography (FC)*, 2003.
13. J. Cha and J. Cheon. An id-based signature from gap-diffie-hellman groups. In *In Public Key Cryptography - PKC 2003*, 2003.
14. L. Chen, K. Harrison, D. Soldera, and N. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *Proceedings InfraSec 2002, Springer LNCS 2437, pp 260-275*, 2002.
15. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Proc. of Eurocrypt '99*, 1999.
16. M. K. Franklin and M. K. Reiter. The design and implementation of a secure auction service. In *Proceedings of 1995 IEEE Symposium on Security and Privacy, pp. 2-14, Oakland, California*, 1995.
17. E. Fujisaki and T. Okamoto. Secure integration of assymetric and symmatric encryption schemes. In *Proceedings CRYPTO 1999, Springer LNCS 1666, pp 537 - 554*, 1999.
18. J. Garay and C. Pomerance. Timed fair exchange of arbitrary signatures. In *In Financial Crypto*, 2003.
19. J. A. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *In Financial Cryptography '02*, 2002.
20. C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *Proceedings Asiacrypt 2002, Springer LNCS 2501, pp 548 - 566*, 2002.
21. J. T. Harkavy and H. Kikuchi. On cheating in sealed-bid auctions. In *EC'03*, 2003.

22. A. Joux and K. Nguyen. Separating decision diffie-hellman from diffie-hellman in cryptographic groups. Available from `http://eprint.iacr.org/2001/003/`, 2001.
23. J. T. M. Harkavy and H. Kikuchi. Electronic auctions with private bids. In *3 rd USENIX Workshop on Electronic Commerce, Boston, Mass., pp. 61–73*, 1998.
24. K. H. Marco Casassa Mont and M. Sadler. The hp time vault service: Exploiting ibe for timed release of confidential information. In *WWW2003*, 2003.
25. T. May. Timed-release crypto. `http://www.cyphernet.org/cyphernomicon/chapter14/14.5.html-`.
26. 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), 1639–1646*, 1993.
27. D. Mills. Network time protocol (version 3) specification, implementation. Technical Report 1305, Mar. 1992.
28. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of ACM Conference on Electronic Commerce, pp. 129–139*, 1999.
29. T. P. Pederson. A threshold cryptosystem without a trusted party. In *In Advances in Cryptology-Eurocrypt 91*, 1991.
30. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and time-released crypto. In *MIT laboratory for Computer Science,MIT/LCS/TR-684*, 1996.
31. Shamus Software Ltd. Miracl: Multiprecision integer and rational arithmetic c/c++ library. `http://indigo.ie/~mscott/`.
32. P. F. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *1998 IEEE Computer Security Foundations Workshop (CSFW11)*, 1998.
33. C. Tackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Proceedings CRYPTO 1991, Springer LNCS 576, pp 433 - 444*, 1992.

A Proof of Ciphertext/Plaintext Origin to a Third Party For The Proposed Scheme

In this section, we restrict ourselves to the specific implementation of TR-PKAE proposed in the previous sections.

A.1 Basic Definitions

Let pk_b be the public key of receiver, sk_a the public key of sender and T the designated time. We note that given security against RUF-TR-CTXT (PTXT) and TUF-TR-CTXT (PTXT), the receiver cannot forge a ciphertext with specified parameters unless $tkn[T]$ is disclosed. If receiver obtains a time-stamp on the ciphertext from a trusted signing authority at time at which $tkn[T]$ has not been disclosed (where T is the designated time), and eventually proves that the ciphertext can be constructed using pk_a (sender), sk_b (receiver) and $tkn[T]$ to a third party, then this would prove that the ciphertext was indeed generated by the alleged sender. [8].

Now suppose the sender received a time-stamped signature from the receiver on the ciphertext prior to the decryption time. Suppose now that the receiver decides to 1) deny that pk_a had sent the ciphertext, or 2) claim that the ciphertext is not decryptable, or 3) claim that the plaintext contains a value different from what the sender alleges. In this case, if the sender manages to prove to a third party that the ciphertext was formed correctly this type of attack could be prevented. Both of these situations are directly applicable to sealed-bid auction as discussed in Section 1.1. Below we concentrate only on proof by the receiver, as we proceed, but we will comment on the case when the sender has to prove to a third party.

When we look at the scheme, it is rather easy to see that the receiver may be able to (efficiently) prove ciphertext origin to a third party only if he is willing to: 1) disclose the plaintext, 2) disclose the symmetric encryption key K. In addition, he will have to prove that the hash pre-image of K has a certain form which may leak some information to the verifier. This is somewhat expected since the scheme uses symmetric encryption. Still, it turns out that the scheme will retain forward/backward security with respect to time.

[8] We stress that "non-repudiation" provided by this kind of proof is inherently different from non-repudiation provided by digital signatures (and also by signcryption schemes such as [11])

More precisely, the verifier will only be able to forge ciphertexts for this particular sender/receiver pair and only for the designated time used in the verified ciphertext.

We define the following additional algorithms:

$\mathsf{TokenTester}_{\mathsf{TokenGenerator}}$: Given designated time T, it outputs either (corresponding token is) "published" or "unpublished". Note that this algorithm depends on the internal state of $\mathsf{TokenGenerator}$.

$\mathsf{TimeStamp}_{\mathsf{TokenGenerator}}$: Given signing authority $\mathcal{SA}$, on input of any message c it generates $ts_{\mathcal{SA}}(c, T, Tokens_{pub})$, signature on $\langle c, T, Tokens_{pub}\rangle$ using $\mathcal{SA}$'s private key (and possibly different cryptosystem). $Tokens_{pub}$ denotes a set of times for which $TokenTester$ outputs "published".

We will present a protocol for the proposed TR-PKAE scheme that allows receiver to prove to a third-party the ciphertext/plaintext origin. Abstractly, the corresponding algorithm is defined as follows:

Prove_{π}: This is an abstract function which involves a prover P and verifier V.
Prover submits $\langle pk_A, pk_B, T, \mathcal{R}(sk_B, pk_A, T, tkn[T], c), ts_{\mathcal{SA}}(c, T', Tokens_{pub})\rangle$ to the verifier, where $c \in \{0,1\}^n$ is the corresponding ciphertext allegedly encrypted using sk_A (sender), pk_B (receiver) and time T. Then both parties engage in an interactive proof. Verifier outputs either "true" or "false", where "true" means that verifier confirms that ciphertext (and corresponding plaintext) were indeed generated by A.

For consistency, we require that Prove_{π} outputs "true" in the case of honest-prover and honest-verifier.

A.2 Protocol Description

As we have seen previously, our specific construction for TR-PKAE is based on symmetric key encryption. In general, an authenticated encryption based on symmetric key encryption does not allow for the receiver to prove the origin of the message to a third party. Nevertheless, this property would be desirable, even though perhaps counter-intuitive. In this section, we show how the proposed TR-PKAE scheme allows for proof of ciphertext/plaintext origin to a third party.

The Prove algorithm works as follows:

Setting: Prover P with private/public key pair $\langle sk_p, pk_p\rangle$, verifier V, ciphertext $c = \langle Q, c_1, c_2\rangle$ with receiver pk_p and sender pk_a, time T, time-stamp $ts_{\mathcal{SA}}[c, Tokens_{pub}]$. Assume that $tkn[T]$ has been made public, i.e. TokenTester(T) outputs "published".

Decryption: Prover decrypts c using pk_a (sender), sk_p (receiver) and $tkn[T]$. Corresponding σ, plaintext m are retrieved.

Step 1: Prover picks random $r \in \mathbb{Z}_q^*$ and submits $\langle T, m, \sigma, KB, J_1 = kP_T, J_2 = sk_p J_1, ts_{\mathcal{SA}}[c, Tokens_{pub}]\rangle$ to the verifier where $KB = e(sk_p^{-1}Q + pk_a, tkn[T] + sk_p P_T)$.

Step 2 :
1. Verifier computes ciphertext using submitted σ, m, KB and T.
2. It verifies the time-stamp $ts_{\mathcal{SA}}$ using the computed ciphertext and public key of $\mathcal{SA}$.
3. It checks that $T \notin Tokens_{pub}$

Step 3: Verifier checks equality $e(J_2, P) = e(J_1, pk_p)$ and then computes
1. $KB_2 = e(rP + pk_a, tkn[T])e(pk_p, rP_T)$ where $r = H_3(\sigma, m)$
2. $KB_{part} = KB/KB_2$
3. $KB^* = e(pk_a, J_2)$

Interactive Proof: Let $L^* = e(J_1, P_T)$ and $L = e(P_T, P_T)$. Prover proves to verifier that $KB_{part}^{k^*} = KB^*$ and $L^{k^*} = L^*$ for the same k^* (and knowledge of k^*) using zero-knowledge proof [3]. Alternatively expressed, P proves equality $\log_{KB_{part}} KB^* = \log_L L^*$ and knowledge of the logarithms.

From group properties, it follows that if $e(J_2, P) = e(J_1, pk_p)$, then $\langle J_1, J_2\rangle$ must have the form $\langle X, sk_pX\rangle$ for some $X \in \mathbb{G}_1$. Note that $KB_2 \cdot e(pk_a, sk_pP_T) = e(rP + pk_a, tkn[T] + sk_pP_T)$. Thus the prover has to show that $KB = KB_2 \cdot e(pk_a, sk_pP_T)$, or equivalently that $KB_{part} = KB/KB_2 = e(pk_a, sk_pP_T)$. Verifier can compute $e(pk_a, sk_pX)$. Zero-knowledge proof proves knowledge of $k^* = \log_{P_T} X$ and that $e(pk_a, sk_pX)^{1/k^*} = KB_{part}$. Noting that $1/k^* = \log_X P_T$, we obtain that $e(pk_a, sk_pX)^{1/k^*} = e(pk_a, sk_p \log_X(P_T)X) = e(pk_a, sk_pP_T)$. As a result, this proves that KB has the required form. Since TR-PKAE is secure against RUF-TR-CTXT and TUF-CTXT and provided the time-stamp verifies, it follows that the alleged sender had generated the ciphertext.

Leaked Information The verification protocol exposes the following information to the verifier: plaintext/ciphertext, $sk_p(kP_T)$, kP_T and $e(pk_a, sk_pP_T)$.

Note that due to the symmetric nature of the protocol (once $tkn[T]$ is known), the sender can prove to a third party that the ciphertext/plaintext is generated by either the sender or receiver if he saves the value r used in the encryption. The protocol is almost identical except that in **Step 3**, the sender submits $J_2 = sk_aJ_1$. This property is useful in sealed-bid auction.

A.3 Security Experiments

From now on we consider solely the case when the prover is the receiver. The case when the prover is the sender is almost the same with obvious modifications. We need to answer the following questions:

1. *Does the above protocol affect confidentiality of prover's other ciphertexts?*
2. *Can the verifier generate ciphertexts on behalf of the prover using the obtained information?*

A.3.1 Confidentiality

To answer the first question, we need to determine if the proposed TR-PKAE scheme will stay IND-KC-CCA2 and IND-RTR-CCA2 given the information obtained by verifier.

IND-KC$_V$-CCA2 We modify IND-KC-CCA2 game to include verifications. More precisely, the adversary is again in possession of a sender's private key and obtains information leaked by the receiver during verifications for ciphertexts sent by any sender, other than for ciphertext generated during the challenge. We say that the proposed scheme is secure against adaptive chosen-ciphertext with verifications (and key compromise) attack (IND-KC$_V$-CCA2) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-KC}_V\text{-CCA2}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-KC}_V\text{-CCA2}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the IND-KC$_V$-CCA2 game. The IND-KC$_V$-CCA2 game is identical to the IND-KC-CCA2 except for the following addition:

Verifications : Adversary submits valid ciphertext c encrypted with $pk \in \{pk_a\}$ (sender), pk_b (receiver) and time T. Adversary is not allowed to submit the ciphertext obtained during the challenge with the same keys and time. Adversary obtains information exposed during verification.

We define $\mathbf{Adv}^{\text{IND-KC}_V\text{-CCA2}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$.

IND-RTR$_V$-CCA2 We also modify IND-RTR-CCA2 to add verifications. Now, the receiver obtains information exposed during verifications carried out by the sender. We say that the proposed scheme is secure against timed-release receiver adaptive chosen-ciphertext with verifications attack (IND-RTR$_V$-CCA2) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-RTR}_V\text{-CCA2}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-RTR}_V\text{-CCA2}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the IND-RTR$_V$-CCA2 game. The IND-RTR$_V$-CCA2 game is identical to the IND-RTR-CCA2 except for the following addition:

Verifications : Adversary submits valid ciphertext c, pk_s (from the set of receiver public keys) and time T. The receiver in the ciphertext is pk_a and sender is pk_s. Adversary obtains information exposed during verification.

We define $\mathbf{Adv}^{\text{IND-RTR}_V\text{-CCA2}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$.

A.3.2 Ciphertext (Plaintext) Forgery

Note that in the proposed scheme, the verifier obtains $e(pk_a, sk_p P_T)$ which will allow the verifier to forge ciphertexts between this sender and receiver for time T. Thus, we lose TUF-CTXT/PTXT. Still, it turns out that given verification for designated time T' it will be hard for the verifier to forge a ciphertext if one the following holds: 1) designated time of the forgery $T \neq T'$, 2) either the sender or receiver of the forgery was not part of the verified ciphertext. This will be true even if the master key is known to the adversary. Besides TUF-CTXT, we also need to ask ourselves if RUF-TR-CTXT is retained, that is, if the verifier can forge ciphertext with the prover as the sender and verifier as the receiver for a designated time T without knowledge of corresponding $tkn[T]$.

TUF-TR$_V$-CTXT (PTXT) The corresponding game is a modification of TUF-CTXT game. Now, the challenger also generates time T_c. The adversary is allowed to obtain verification information on ciphertexts using the above sender and receiver only for designated time $T \neq T_c$. Otherwise, it can obtain any verification information. The goal is to forge a valid ciphertext with these public keys (representing sender and receiver) and time T_c. We say that TR-PKAE encryption is secure against timed-release third-party chosen-plaintext ciphertext forgery with verifications attack (TUF-TR$_V$-CTXT) if no polynomial adversary (denoted by $\mathcal{A}_{\text{TUF-TR}_V\text{-CTXT}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{TUF-TR}_V\text{-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the following TUF-TR$_V$-CTXT game:

Setup: The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$, time T_c, public/private key pairs (pk_a, sk_a), (pk_b, sk_b) and a random set of public/private pairs $\{(pk_v, sk_v)\}$. The adversary receives $\langle \pi, \delta, T_c, pk_a, pk_b, \{sk_v\} \rangle$. Denote $S_v = \{pk_v\} \bigcup pk_a \bigcup pk_b$ and $R_v = pk_a \bigcup pk_b$.

Pre-Forgery :

- Random Oracle Queries: Adversary may query any random oracle
- Verifications: Adversary chooses different $pk_s \in S_v$ (sender), $pk_r \in R_v$ (receiver) and time T. It submits ciphertext with these parameters and obtains information exposed during verification. The only restriction is for time $T = T_c$: in this case it is not allowed that both pk_s and pk_r come from $\{pk_a, pk_b\}$.
- Encryption Queries: Adversary submits plaintext m, time T and obtains encryption using pk_a (sender), pk_b (receiver) and time T.

Forgery: Adversary submits ciphertext c.

Outcome: Adversary wins the game if c successfully decrypts using pk_a (sender), pk_b (receiver) and T_c, and c was not obtained during encryption queries.

We define $\mathbf{Adv}^{\text{TUF-TR}_V\text{-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_a, sk_b, T_c] = true]$. By requiring that in the above game the decrypted plaintext m in the outcome was not submitted during encryption queries, we obtain corresponding notion of TUF-TR$_V$-PTXT. We skip the details.

RUF-TR$_V$-CTXT (PTXT) We modify the RUF-TR-CTXT(PTXT) in which the adversary (receiver) obtains information exposed during verifications carried out by the sender. We say that TR-PKAE encryption is secure against timed-release receiver chosen-plaintext ciphertext forgery with verifications attack (RUF-TR$_V$-CTXT) if no polynomial adversary (denoted by $\mathcal{A}_{\text{RUF-TR}_V\text{-CTXT}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{RUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$) against the challenger in the RUF-TR$_V$-CTXT game. The game is identical to that of RUF-TR-CTXT except for the following addition:

Verifications: Attacker submits a sender private key sk (or alternatively, a set of such keys could be generated by the challenger and given to the adversary), time $T \neq T_a$ and ciphertext encrypted with sk, pk_s (receiver) and T. Attacker obtains the information exposed during the verification.

We define $\mathbf{Adv}^{\text{RUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_s, sk_{b^*}, T_a] = true]$. By requiring that in the above game the decrypted plaintext m in the outcome was not submitted during encryption queries, we obtain corresponding notion of RUF-TR$_V$-PTXT. We skip the details.

A.4 Security Results

Below we state security properties of TR-PKAE against IND-KC$_V$-CCA2, IND-RTR$_V$-CCA2, TUF-TR$_V$-CTXT and RUF-TR$_V$-CTXT. The proofs are given in Appendix B.

Theorem 5 (IND-KC$_V$-CCA2). *Let $\mathcal{A}$ be IND-KC$_V$-CCA2 adversary, q_d be the number of decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}^{\text{IND-KC}_V\text{-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq [\frac{2\epsilon}{q_d+q_2}]^2$ and running time $O(time(\mathcal{A}))$.*

Theorem 6 (IND-RTR$_V$-CCA2). *Let $\mathcal{A}$ be IND-RTR$_V$-CCA2 adversary, let q_d be the number of decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}^{\text{IND-RTR}_V\text{-CCA2}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq \frac{2\epsilon}{q_d+q_2}$ and running time $O(time(\mathcal{A}))$.*

Theorem 7 (TUF-TR$_V$-CTXT). *Let $\mathcal{A}$ be TUF-TR$_V$-CTXT adversary, let q_e be the number of encryption queries and q_2 be the number of queries to random oracle H_2. Assume that $\boldsymbol{Adv}^{\text{TUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq \frac{\epsilon}{q_e \cdot q_2+1}$ and running time $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

We have TUF-TR$_V$-PTXT security with the same inequality. Only minor modifications are required in the proof.

Theorem 8 (RUF-TR$_V$-CTXT). *Let $\mathcal{A}$ be RUF-TR$_V$-CTXT adversary, let q_e be the number of encryption queries and q_2 be the number of queries to random oracle H_2. Assume that $\boldsymbol{Adv}^{\text{RUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. Then there exists an algorithm that solves BDHP with advantage $\boldsymbol{Adv}(k) \geq \frac{\epsilon^2}{(q_e \cdot q_2)^2+2}$ and running time $O(time(\mathcal{A})) + O([q_e \cdot q_2]^2)$.*

We have RUF-TR$_V$-PTXT security with the same inequality. Only minor modifications are required in the proof.

B Security Proofs

Proof of Theorem 1 [IND-KC-CCA2] The Theorem result follows from Corollary 12. 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 9. 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 10. *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 11. *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 12. *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 11 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

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

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 10 we obtain the result.□

Proof of Theorem 2 [**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 1.

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 1

$\mathcal{B}$ 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 13. 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 14. *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 $Y/[e(sP, rH_1(T)) \cdot e(pk_a, sH_1(T)) \cdot e(pk_b, rH_1(T))] = e(pk_b, sk_aH_1(T)) = e(pk_b, sk_a(c_Tb'P)) = e(P,P)^{a'b'^2c_T}$. 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 1 (details skipped – the reader is asked to refer to analysis in Theorem 1 for notation). And Lemma 11 carries over here as well with obvious modifications. The following Lemma is slightly different from the corresponding one in the proof of Theorem 1.

Lemma 15. *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 12. In addition to notation used in Corollary 12, 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 12, $k^* \cdot (1 - p_f) + k \cdot p_f = \epsilon$.

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 14, 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 $\mathtt{Game}_{\mathcal{B}}$ is used to solve BDHP $\langle X, a''X, b''X, c''X\rangle$ as follows. We run $\mathtt{Game}_{\mathcal{B}}$ 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 $E_1 = e(X, X)^{b_1^2 a''}$ with advantage at least $\frac{\epsilon}{q_e \cdot q_2+1}$. Then we run $\mathtt{Game}_{\mathcal{B}}$ 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 $E_2 = e(X, X)^{b_2^2 a''}$ 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 16. 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 $e(X, X)^{a'b'c'}$

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, set $\{(sk_b, pk_b)\}$ and T_a: $\mathcal{B}$ chooses random $sk_a = a \in \mathbb{Z}_q^*$, $sk_{b_i} = b_i \in \mathbb{Z}_q^*$ and T_a. Adversary $\mathcal{A}$ receives $pk_a = aP$, $\{sk_{b_i} = b_i\}$ and T_a. 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: If $T \neq T_a$, $\mathcal{B}$ returns c_TP, for random $c_T \in \mathbb{Z}_q^*$, and stores the query T in the database coupled with the c_T. Repeated queries retrieve answers from the database. If $T = T_a$, simulator returns $c'P$.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem 1.

Queries for $tkn[T] = sP_T$: When $\mathcal{A}$ submits $T \neq T_a$, $\mathcal{B}$ queries H_1, obtains corresponding c_T and returns $sH_1(T) = c_T(b'P)$.

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 and $pk_{b^*} \in \{pk_{b_i}\}$. (Note that simulator can determine b^*)

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 1

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.

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

$T = T_a$: $\mathcal{B}$ returns false and computes $[J/[e(sP, aH_1(T_a)) \cdot e(rbP, H_1(T_a)) \cdot e(bP, aH_1(T_a))]]^{t^{*-1}}$. 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 definition of inconsistency, construction of $\mathtt{Game}_\mathcal{C}$ and the Lemmas in the proof of Theorem 1 naturally carry over with minor modifications. We skip the details and just state the final Corollary:

Corollary 16. *Probability that a random run of the above simulation produces the solution to BDHP is at least* $\frac{2\epsilon}{q_d+q_2}$.

Proof of Theorem 4 [**RUF-TR-CTXT**] The Theorem result follows from Corollary 20. 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. 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 compute solution to the BDHP with parameters $\langle X, a'X, b'X, c'X\rangle$.

Since the simulator will not know the private key corresponding to the public key of the adversary which is generated during setup, the simulation will be run twice. The idea is to obtain two different simulation results with the same choice of adversarial public key that will allow us to cancel some terms that involve this public key. The second simulation uses the same BDHP and setup parameters. The only difference is in the choice of sP. Moreover, the simulations are completely identical from adversarial view. The final answer to BDHP is given after both simulations have been run. The random tape of the adversary is the same in both cases (i.e. we restart adversarial Turing machine with the same random tape), while the random tape of $\mathcal{B}$ should be different in both 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 $u \cdot b'P$ for random u. *Denote by u_1 the value of u used in the 1st simulation and by u_2 the value used in the 2nd simulation*

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

Adversary Setup: Adversary chooses public key denoted by bP.

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: If $T \neq T_a$, $\mathcal{B}$ returns $c_T P$ for random $c_T \in \mathbb{Z}_q^*$ and stores c_T indexed by T in the database. If $T = T_a$, $\mathcal{B}$ returns $c'P$.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem 1.

Queries for sP_T: $\mathcal{A}$ submits $T \neq T_a$. $\mathcal{B}$ queries H_1 with T and returns $sH_1(T) = c_T(u \cdot b'P)$. Queries for sP_T with $T = T_a$ are not allowed.

Encryption queries: $\mathcal{A}$ submits T, m. The simulator is expected to output the encryption of m using a' (sender) and bP (receiver). Two cases are considered:

$T \neq T_a$: $\mathcal{B}$ 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 $sH_1(T) = c_T(u \cdot b'P)$ and $bH_1(T) = c_T(bP)$. The corresponding query is made to H_2 and $\mathcal{B}$ returns resulting ciphertext $c = \langle rbP, b, T, c_1, c_2\rangle$.

$T = T_a$: $\mathcal{B}$ chooses random $r \in \mathbb{Z}_q^*$, σ and two random strings c_1, c_2, and returns ciphertext $c = \langle rbP, b, T_a, c_1, c_2\rangle$. The databases are updated as follows:

H_3: $\mathcal{B}$ puts r 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, r is checked against existing replies in the database and if it already exists, a new choice for r is made.

H_4, H_1, H_2: updated the same way as in the *Challenge* phase of the proof of Theorem 1

$\mathcal{B}$ keeps the local database D_s in which it enters the triple $\langle T_a, r, b\rangle$. Denote by $TRUE[T_a, r, b]$ the true value of $e(sP + bP, (r + a')sH_1(T_a))$.

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 1st Simulation: $\mathcal{A}$ returns forged ciphertext or simulation halts.

Inspection of Databases: $\mathcal{B}$ goes through database D_s, obtains $\langle T_a, r, b\rangle$ from each entry and computes $Y/[e(sP, rH_1(T_a)) \cdot e(bP, rH_1(T_a))]$ for every query Y to H_2. The results are written down as possible values of $e(P, P)^{u_1 \cdot a'b'c'} \cdot e(bP, a'c'P)$ 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 $e(sP + b^*P, r^*H_1(T_a)) \cdot e(a'P, b^*H_1(T_a)))$, computes $1/u_1$-th root of the result and writes down the answer as a possible value of $e(P, P)^{a'b'c'}$.

Note that if $\mathcal{A}$ wins (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). To see this, note that $e(sP + b^*P, (r^* + a')P_{T_a}) = e(P, P)^{c'(s+b^*)(r^*+a')}$ and this value is equal to a bilinear map $e(P, P)^{c'(s+b)(r+a')}$ (used in the encryption queries for T_a) iff $(s + b^*)(r^* + a') = (s + b)(r + a') \mod q$. We can modify the above simulation by taking s to be known (i.e. $a'P$ is the DLP parameter here and we need to determine a'). Then if collision happens we can compute the value of a'. Under the assumption of hardness of DLP in $\mathbb{G}_1$, this happens only in a negligible number of cases. Thus we may safely ignore such collisions.

Outcome of 2nd Simulation: $\mathcal{A}$ returns forged ciphertext or simulation halts.

Inspection of Databases: Note that, with high probability, $u_1 \neq u_2$. $\mathcal{B}$ goes through database D_s, obtains $\langle T_a, r, b\rangle$ from each entry and computes $Y/[e(sP, rH_1(T_a)) \cdot e(bP, rH_1(T_a))]$ for every query Y to H_2 (the result is a possible value of $e(P,P)^{u_2 \cdot a'b'c'} \cdot e(bP, a'c'P)$). Then for each $f \in D_{aux}$, the result is divided by f and then taken to $1/(u_2 - u_1)$-th power. The final result is written down as a possible value of $e(P,P)^{a'b'c'}$.

Forgery Examination: Same as in the 1st simulation. The same comments apply here as well.

Combined Outcome: The final outcome is produced after the 2nd simulation. Out of calculated possible values of $e(P,P)^{a'b'c'}$ produced in both simulations (in the 1st simulation, only forgery examination may produce such possible value), $\mathcal{B}$ picks one at random and outputs it as the value of $e(P,P)^{a'b'c'}$. Note that the size of the list of possible values of $e(P,P)^{a'b'c'}$ is at most $(q_e \cdot q_2)^2 + 2$.

Definition 17. 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 $TRUE[T_a, r, b]$ in D_s.

Lemma 18. *If the 1st simulation becomes inconsistent, then $\mathcal{B}$ will have correct answer for $e(P,P)^{u_1 \cdot a'b'c'} \cdot e(bP, a'c'P)$ in a database D_{aux}.*

Proof: Let Y be a query to H_2 which happens to be the correct computation of the bilinear map corresponding to some $TRUE[T_a, r, b]$ in D_s. Then $Y = [e(sP, rH_1(T_a)) \cdot e(bP, rH_1(T_a))] \cdot e(a'P, bH_1(T_a)) \cdot e(sP, a'P_{T_a})$. In the *Outcome of 1st Simulation* phase, $\mathcal{B}$ computes $Y/[e(sP, rH_1(T_a)) \cdot e(bP, rH_1(T_a))] = e(a'P, bH_1(T_a)) \cdot$ $\cdot e(sP, a'P_{T_a}) = e(bP, a'c'P) \cdot e(P,P)^{u_1 \cdot a'b'c'}$. Therefore, the true value of $e(P,P)^{u_1 \cdot a'b'c'} \cdot e(bP, a'c'P)$ will be in D_{aux}. □

Lemma 19. *If both simulations are inconsistent, then $\mathcal{B}$ outputs correct answer for $e(P,P)^{a'b'c'}$ with probability $\frac{1}{(q_e \cdot q_2)^2 + 2}$.*

Proof: As in the previous lemma, correct value of $e(P,P)^{u_2 \cdot a'b'c'} \cdot e(bP, a'c'P)$ will be calculated in the 2nd simulation run. In addition, a correct value of $e(P,P)^{u_1 \cdot a'b'c'} \cdot e(bP, a'c'P)$ will be retrieved from D_{aux}. Now one can easily see that 2nd simulation in fact will calculate $e(P,P)^{a'b'c'}$ using these two values. There will be at most $(q_e \cdot q_2)^2 + 2$ calculations and one of them will involve the mentioned values. Thus, the probability that $e(P,P)^{a'b'c'}$ is output will be at least $\frac{1}{(q_e \cdot q_2)^2 + 2}$. □

In case both simulations are consistent, success of $\mathcal{B}$ depends on the forgery. Next we follow a similar line of reasoning to Theorem 2. More precisely, we construct $\texttt{Game}_{\mathcal{C}}$ and prove Lemma 11 for the above simulation (details omitted). Next, similarly to Lemma 15, we prove the following result:

Corollary 20. *If $\mathcal{A}$ attains advantage of at least ϵ in the real game, then the probability that simulation 2 outputs solution to BDHP is at least $\frac{\epsilon^2}{(q_e \cdot q_2)^2 + 2}$.*

Proof: We use the same notation as in Lemma 15. The proof is very similar to that of Lemma 15. Recall that k^* denotes probability of event $Succ(r_{\mathcal{A}}, r_{\mathcal{C}})$ for one simulation run given that this run is consistent and p_f denotes probability that a single run is inconsistent. When both simulation runs are inconsistent, one of the possible values of $e(P,P)^{a'b'c'}$ produced will in fact be the true value of $e(P,P)^{a'b'c'}$. When one run is consistent the probability that we have a true value of $e(P,P)^{a'b'c'}$ among output possibilities (i.e.

when 2nd simulation chooses its answer) depends on k^*. The probability that the the 2nd run above will have a true value of $e(P,P)^{a'b'c'}$ in its output list is at least $p_f^2+(1-p_f^2)\cdot k^*$ (the 1st term is the probability that both runs are inconsistent, and the 2nd term is a lower bound on probability that at least one run is consistent and $\mathcal{A}$ achieves forgery on this run). As in Lemma 15, we have equation $k^*(1-p_f)+k\cdot p_f=\epsilon$. In particular, $k^*(1-p_f)+p_f\geq\epsilon$. It is a routine check to verify that $p_f^2+(1-p_f^2)\cdot k^*\geq[k^*(1-p_f)+p_f]^2\geq\epsilon^2$. Since the BDHP answer is chosen from the list of at most $(q_e\cdot q_2)^2+2$ possible values and one of them is a true value of $e(P,P)^{a'b'c'}$ with probability at least ϵ^2, we obtain the claim. □

Proof of Theorem 7 [**TUF-TR$_V$-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 TUF-TR$_V$-CTXT. We design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real TUF-TR$_V$-CTXT 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 X.

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

Choice of pk_a, pk_b, T_c and set $\{(pk_v,sk_v)\}$: $\mathcal{B}$ chooses pk_a to be $a'X$ and pk_b to be $b'X$. Also, random T_c and set $\{(pk_v,sk_v)\}$ is chosen. The adversary $\mathcal{A}$ receives pk_a, pk_b, $\{sk_v\}$ and T_c.

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: If $T\neq T_c$, $\mathcal{B}$ chooses random $c_T\in\mathbb{Z}_q^*$ and returns c_TP. Reply c_T is indexed by T and stored in the database and replies for repeated queries use the database. If $T=T_c$, $\mathcal{B}$ returns $c'X$.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem 1.

Encryption Queries: Adversary submits plaintext m, time T. In the generated ciphertext, pk_a is the sender and pk_b is the receiver.

$T\neq T_c$: In this case, $\mathcal{B}$ can compute the bilinear map as $e(sP+pk_b, rH_1(T)+c_T\cdot pk_a)$. Therefore, it goes through normal encryption algorithm and makes all necessary queries to $H_i, i=1,...4$. The resulting ciphertext is given to the adversary.

$T=T_c$: In this case, $\mathcal{B}$ generates random $r\in\mathbb{Z}_q^*$, σ and two random strings c_1, c_2 and updates all necessary databases the same way as in simulation of the proof of Theorem 4 (the $T=T_a$ case). $\mathcal{B}$ returns ciphertext $c=\langle Q=r\cdot pk_b,T,c_1,c_2\rangle$.

$\mathcal{B}$ keeps the local database D_s for case $T=T_c$ in which it enters r. Let us denote by $TRUE[r]$ the true value of $e(sP+pk_b,(r+sk_a)H_1(T_c))$.

Verification Queries :

Case 1: Adversary submits valid ciphertext encrypted with $sk_s\in\{sk_v\}$ (sender), either $pk_r\{pk_a,pk_b\}$ as the receiver and time T. Challenger decrypts the ciphertext (by querying the databases and computing the bilinear map). $\mathcal{B}$ chooses random $k\in\mathbb{Z}_q^*$ and returns $\langle kP,k(pk_r),m\rangle$. (Note that we can omit the zero-knowledge proof here since one can easily remove it from the algorithm $\mathcal{A}$).

Case 2: Adversary submits $T\neq T_c$, pk_a (sender), pk_b (receiver) and a corresponding valid ciphertext $c=\langle Q,c_1,c_2\rangle$. $\mathcal{B}$ verifies validity of c as follows: 1) r and corresponding σ, m are obtained from H_3, 2) equality $c_2=m\oplus H_4(\sigma)$ is verified, 3) $K=c_1\oplus\sigma$ is extracted, corresponding query is obtained from H_2 and one verifies that it is equal to the true value (note that it can be calculated by $\mathcal{B}$ in this case). If either one of these steps fails, c is deemed to be invalid. Next, $\mathcal{B}$ chooses random $k\in\mathbb{Z}_q^*$ and returns rP, $e(pk_b,sk_a\cdot P_{T_c})=e(pk_a,c_T\cdot pk_b)$ and pair $\langle kP,k(b'X)\rangle$. The case when the roles of sender and receiver are interchanged is symmetric.

Forgery: $\mathcal{A}$ submits ciphertext $c^* = \langle Q^*, c_1^*, c_2^* \rangle$. Ciphertext represents encryption using pk_b as the sender and pk_a as the receiver with designated time T_c.

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

1. $\mathcal{B}$ goes through its database D_s, and for each entry of $TRUE[r]$ and query Y to H_2 computes $Y/[e(sP, rH_1(T_c)) \cdot e(pk_a, sH_1(T_c)) \cdot e(pk_b, rH_1(T_c))]$. The answer is written down as a possible value of $e(P,P)^{a'b'c'}$.
2. If $\mathcal{A}$ submitted a forgery, $\mathcal{B}$ searches H_3 for corresponding r^* (by multiplying every r in H_3 by pk_a and comparing it with Q^*). If r^* is found, then $\mathcal{B}$ obtains σ^* and m^* and computes $K^* = c_1^* \oplus \sigma^*$. If query corresponding to K^* is in database of H_2, then this query is divided by $e(sP, r^*H_1(T_c)) \cdot e(pk_a, sH_1(T_c)) \cdot e(pk_b, r^*H_1(T_c))$. The answer is written down as a possible value of $e(P,P)^{a'b'c'}$. Note that if the query corresponding to K^* is the true value of the bilinear map, this calculation produced the correct $e(P,P)^{a'b'c'}$

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

Definition 21. 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 $TRUE[r]$ in D_s.

Lemma 22. *If the simulation above becomes inconsistent, then $\mathcal{B}$ will have correct answer for $e(X,X)^{a'b'c'}$ 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 $TRUE[r]$ in D_s. Then $Y = [e(sP, rH_1(T_c)) \cdot e(pk_a, sH_1(T_c)) \cdot e(pk_b, rH_1(T_c))] \cdot e(pk_a, sk_b H_1(T_c))$. In the *Outcome* phase, $\mathcal{B}$ computes $Y/[e(sP, rH_1(T_c)) \cdot e(pk_a, sH_1(T_c)) \cdot e(pk_b, rH_1(T_c))] = e(pk_a, sk_b H_1(T_c))$ $= e(a'X, b'(c'X)) = e(X,X)^{a'b'c'}$ and the conclusion follows.□

Next, we follow analogous chain of discussion as in the corresponding part of the proof of Theorem 2. In fact, all results and proofs are almost identical with obvious modifications. We skip this part of the proof and conclude that the advantage attained in solving the BDHP problem is at least $\frac{\epsilon}{q_e \cdot q_2 + 1}$

Proof of Theorem 5 [**IND-KC$_V$-CCA2**] The Theorem statement follows from Corollary 23. 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-KC$_V$-CCA2. First we design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real IND-KC$_V$-CCA2 game for the adversary in order to compute solution to special case of BDHP with parameters $\langle X, a'X, a'X, b'X \rangle$.

Setup :

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

s and P_{pub}: $\mathcal{B}$ chooses s and makes it public.

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

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 $\{a_i\}$ (alternatively, the adversary can choose this set by itself).

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 $e(X,X)^{a'^2b'}$ updated in the *Decryption Queries After Challenge* phase.

Oracle Queries :

P_T (or H_1) Queries: $\mathcal{B}$ returns c_TP for random $c_T \in \mathbb{Z}_q^*$ and stores the query in the database coupled with the reply. Repeated queries retrieve answers from the database.

H_2,H_3,H_4 Queries: Same as in the proof of Theorem 1.

Verification Queries: $\mathcal{A}$ submits public key $pk_a \in \{a_iP\}$, time T and ciphertext $c = \langle Q, c_1, c_2\rangle$. $\mathcal{B}$ follows the routine specified in the *Decryption Queries After Challenge*. If "true" is output by this phase, then corresponding r, m, σ and the value of the bilinear map KB are present. $\mathcal{B}$ computes $kH_1(T) = k(c_TP)$ and $k(bH_1(T)) = kc_T(b'P)$ for random $k \in \mathbb{Z}_q^*$, and returns $\langle T, m, \sigma, KB, kH_1(T), k(bH_1(T))\rangle$. Note that in this case even zero-knowledge protocol will succeed.

Decryption Queries Before Challenge: $\mathcal{A}$ submits ciphertext $\langle T, Q, pk_a, c_1, c_2\rangle$ where c_1 denotes encryption of σ and c_2 is the encryption of plaintext, Q represents $r \cdot bP$, $pk_a = aP \in \{a_iP\}$ and T is the designated time.

$\mathcal{B}$ goes through the database of H_3 searching for appropriate r (by multiplying each r by bP and comparing with Q). 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 $e(rP, sH_1(T)) \cdot e(aP, sH_1(T)) \cdot e(bP, aH_1(T)) \cdot e(Q, H_1(T))$ (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 $a = a^* \in \{a_i\}$ 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 σ^* is chosen, $\mathcal{B}$ chooses 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 $rbP = Q^*$ 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 bP and comparing it with Q^*) 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 Theorem 1

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 $\langle T, Q, pk_a, c_1, c_2\rangle$. $\mathcal{B}$ 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 $[Y/[e(sP + bP, aH_1(T)) \cdot e(Q, H_1(T))]]^{(sc_T)^{-1}}$ and writes the result as a possible value of $e(X, X)^{a'^2b'}$ in the list $\mathcal{L}$.

r is found: If $Q = Q^*$, then $\mathcal{B}$ quits and computes $e(rP, sH_1(T)) = e(Q^*/b, P)^{sc_T}$ and by taking the root obtains $e(X, X)^{a'^2b'}$. 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 $e(X, X)^{a'^2b'}$. 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

$[Y/[e(sP+bP, a^*H_1(T^*)) \cdot e(Q^*, H_1(T^*))]]^{(sc_{T^*})^{-1}}$ 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.

We define inconsistency the same way as in Definition 9 and go through absolutely the same Lemmas as in the proof of Theorem 1, where in addition verifications phase is added. All proofs and statements stay the same and we obtain the following Corollary (which parallels Corollary 12):

Corollary 23. *Probability that a random run of the above simulation produces the solution to BDHP* $\langle X, a'X, a'X, b'X\rangle$ *is at least* $\frac{2\epsilon}{q_d+q_2}$

The above simulation is used to solve BDHP $\langle X, a''X, b''X, c''X\rangle$ in the same way as at the end of the proof of Theorem 2. Thus, the advantage in solving for $e(X,X)^{a''b''c''}$ is $[\frac{2\epsilon}{q_d+q_2}]^2$

Proof of Theorem 8 [**RUF-TR$_V$-CTXT**] The proof is identical to the proof of Theorem 4 except for the following addition:

Verifications: Adversary submits a sender public key pk, valid ciphertext $c = \langle Q, c_1, c_2\rangle$ encrypted with sk (private key corresponding to pk), pk_s (receiver) and time $T \neq T_a$. $\mathcal{B}$ verifies validity of c as follows: 1) r and corresponding σ, m are obtained from H_3, 2) equality $c_2 = m \oplus H_4(\sigma)$ is verified, 3) $K = c_1 \oplus \sigma$ is extracted, corresponding query is obtained from H_2 and one verifies that it is equal to the true value (note that it can be calculated by $\mathcal{B}$ in this case since $skP_T = c_T(skP)$). If either one of these steps fails, c is deemed to be invalid and verification fails. Next, $\mathcal{B}$ chooses random $k \in \mathbb{Z}_q^*$ and returns $\langle m, \sigma, kP, k(a'X)\rangle$ (again note that we can omit the zero-knowledge proof here since one can easily remove it from the algorithm $\mathcal{A}$)

Proof of Theorem 6 [**IND-RTR$_V$-CCA2**] The proof is identical to the proof of Theorem 3 except for the following addition:

Verifications: Adversary submits a sender public key $pk_s \in \{pk_b\}$, valid ciphertext $c = \langle Q, c_1, c_2\rangle$ encrypted with pk_s (sender), $pk_a = aP$ (receiver) and time T. We note that $\mathcal{B}$ knows a and, therefore, can verify validity (decrypt) of submitted ciphertext in the usual way. As a result, $\mathcal{B}$ can supply $\mathcal{A}$ with all required information.