Timed-Release Public Key Based Authenticated Encryption

Abstract. In this paper, we formally define a notion of timed-release public key based authenticated encryption (TR-PKAE). In addition to standard time-independent security properties (such as IND-CCA security for confidentiality and ciphertext/plaintext unforgeability), TR-PKAE introduces requirements such as timed-release receiver confidentiality (IND-RTR-CCA), which precludes the receiver from decrypting ciphertext before designated time, and ciphertext/plaintext unforgeability by the receiver itself for a future designated time among others. We propose a first provably secure TR-PKAE construction based on bilinear maps and prove the above security requirements in the random oracle model. Even though our protocol does not use digital signatures, receiver can still prove to a third party the message origin. The proposed protocol is compact, practical and efficient.

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 in such a way that the receiver cannot decrypt it until some 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 [23]. 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 [25] and check cashing. Other applications include release of important documents and press releases among many others.

The problem of timed-release cryptography was first mentioned by May [21] and then discussed in detail by Rivest *et. al.* [23]. Let us assume that the message sender is called Alice and the receiver is Bob, and Alice wants to send a message to Bob in such a way that Bob will not be able to open it until certain time. The possible solutions that do not require Alice's participation after initial communication are divided into two parts:

Time-lock puzzle approach. Alice would encrypt her message in a way that Bob would have to perform non-parallelizable computation non-stop for the required wait time in order to decrypt it.

Agent-based approach. Alice could use trusted agents to encrypt the message in such a way 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 to the message receiver, which make it impractical for real-life scenarios. In addition, knowing computational complexity does not always lead to correct estimation of time for Bob to decrypt the message. Still, this approach is widely used for a variety of applications [6, 5, 25, 18, 17]. Using agent-based approach relieves Bob from performing non-stop computation, fixes 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 end of the waiting period.

In this paper we concentrate on the agent-based approach only. Several agent-base constructions were suggested by Rivest *et. al.* [23]. For example, the agent could encrypt messages on request with a secret key which will be published on a designated date by the agent. Or it could precompute pairs of public/private keys, publish all public ones and release the private keys on the required days. A different scheme was proposed by Di Crescenzo *et. al.* [14], 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 [13], 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 [20], which again replaces identity in the encryption primitive with future time. Similar IBE-based approach was presented in [8]. Still security of these IBE-based approaches has not been proven rigorously by the authors.

Our Contribution We formally define a notion of timed-release public key based authenticated encryption (TR-PKAE) using the agent-based approach. We propose a provably secure TR-PKAE construction based on bilinear maps and provide security proofs in the random oracle model. Namely, we prove that 1) the proposed scheme stays IND-CCA even when sender's secret key is compromised, 2) the scheme is secure against third-party ciphertext/plaintext forgery (TUF-CTXT), 3) the scheme exhibits timed-release receiver confidentiality variation of IND-CCA (IND-RTR-CCA), which precludes the receiver from decrypting ciphertext before designated time, and 4) the scheme is secure against timed-release variation of receiver ciphertext/plaintext forgery. In addition, even though our protocol does not use digital signatures, receiver can still prove to a third party the message origin while maintaining necessary security properties. The proposed protocol is compact, practical and efficient.

Organization: The rest of the paper is organized as follows. In Section 2 we formulate the abstract TR-PKAE cryptosystem and security definitions. In Section 3 we review the background on bilinear maps and in Section 4 we describe proposed implementation of TR-PKAE and security results. In Section 5, we augment the basic cryptosystem with additional functionality that allows ciphertext receiver to prove message origin to a third party in the proposed encryption scheme. We formally state required security properties and corresponding security results. Section 6 discusses practical considerations, possible alternative constructions and performance. Finally, in Section 7 we conclude the paper. The detailed security proofs are provided in the Appendix B.

2 Basic Definitions

2.1 Basic Cryptosystem

The goal of *Timed-Release Public Key Based Authenticated Encryption* (TR-PKAE) is to provide public key based authenticated encryption that takes sender's secret key, receiver's public key and designated time so that the resulting ciphertext can be decrypted only by receiver and only starting with designated time using receiver's secret key, sender's public key and some secret that will be disclosed only on designated time. TR-PKAE is specified by the following randomized algorithms:

Setup : On input of security parameter k, it produces in a randomized manner a pair $\langle \delta, \pi \rangle$ where δ is a master secret and π the corresponding common public parameters. The public parameters include hash functions, message and ciphertext spaces among other parameters. The master secret key will be kept secret by a central entity while all other parameters are public.

$\mathsf{KeyGenerator}_\pi$: 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 $\langle \delta, \pi \rangle$. In practice, this functionality would be performed by an agent who (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 $\widehat{m} = m$.

2.2 Security

Confidentiality

IND-CCA: The ciphertext in the above cryptosystem takes private key of sender, public key of receiver and time-encoding T. A natural security requirement is to demand that the resulting ciphertexts will be secure against adaptive chosen-ciphertext attacks (IND-CCA) [26, 4]. Since we assume that $tkn[T]$ corresponding to ciphertext will be disclosed eventually, IND-CCA should be time-independent. Therefore, we demand that ciphertext will be secure even when $tkn[T]$ corresponding to the message is available. Since the master secret is used only for the purpose of generating $tkn[T]$, we make it available to the adversary.

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

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

Pre-Challenge : Adversary issues the following queries

Random Oracle Queries : Adversary may query any random oracle.

Decryption Queries : Adversary submits ciphertext and either arbitrary sender secret key sk or pk_a, and time encoding T. Challenger responds with decryption of ciphertext using sk (sender) or pk_a (depending on what was submitted), sk_b (receiver) and time encoding T.

Selection : Adversary chooses two distinct equal-size plaintexts m_0, m_1, time T 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), sk_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 $\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$, where k is the system security parameter and probability is taken over random bits used by the challenger and adversary.

Timed-Release Receiver IND-CCA: 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-CCA game to restrict adversary access to $tkn[T]$ for designated time, which means that master secret is no longer available to the adversary. 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 either at least one encryption key or the ciphertext are different from the challenge. This models an adversary who may have some inside knowledge on some of the messages encrypted for that time. We also allow the adversary to choose receiver secret/public pair adaptively.

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

Setup : The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$, sender public/secret key pair (pk_a, sk_a) and designated time T_a. The public key pk_a 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, receiver secret key sk_b and time T. Challenger responds with decryption of ciphertext using sk_a (sender), sk_b (receiver) and $tkn[T]$.

Selection : Adversary chooses two distinct equal-size plaintexts m_0,m_1, receiver secret key sk_{b^*} 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), sk_{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 $\mathbf{Adv}^{\text{IND-RTR-CCA}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$.

The difference between IND-KC-CCA and IND-RTR-CCA is in reversal of adversary roles. In IND-TR-CCA, the goal is to ensure security against the receiver itself prior to designated time even if choice of receiver secret key is given to the adversary.

IND-CCA With Key Compromise: We also state the following game that defines IND-CCA with sender key compromise. The adversary is allowed to choose any sender public/private key. If TR-PKAE stays secure in this game, it follows that even if adversary knows sender's secret key, the encrypted message still stays IND-CCA secure. TR-PKAE encryption scheme is said to be semantically secure against an adaptive chosen ciphertext attack with key compromise (IND-KC-CCA) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-KC-CCA}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-KC-CCA}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$) against the challenger in the following IND-KC-CCA game:

Setup : The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$ and receiver public/secret key pair (pk_b, sk_b). The public key pk_b and both δ (master secret) and π (public parameters) are given to the adversary.

Pre-Challenge : Adversary issues the following queries

- Random Oracle Queries : Adversary may query any random oracle.
- Decryption Queries : Adversary submits ciphertext, arbitrary secret key sk_a and time encoding T. Challenger responds with decryption of ciphertext using sk_a (sender), sk_b (receiver) and time encoding T.

Selection : Adversary chooses two distinct equal-size plaintexts m_0, m_1, time T and sender secret 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), sk_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 $\mathbf{Adv}^{\text{IND-KC-CCA}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\widehat{\beta} = \beta] - 1/2$, where k is the system security parameter and probability is taken over random bits used by the challenger and adversary.

Ciphertext (Plaintext) Forgery [1]

We consider two types of ciphertext forgery: 1) forgery by adversary that does not know the sender's and receiver's secret keys (TUF-CTXT) and 2) forgery by ciphertext receiver itself (RUF-CTXT). 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 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. In the timed-release variation of TUF-CTXT, we require that a third-person cannot forge ciphertext for designated time T unless $tkn[T]$ is known (TUF-TR-CTXT). In the timed-release variation of RUF-CTXT we demand the same with respect to the receiver itself (RUF-TR-CTXT). In practice, this means that ciphertext forgery is hard for a future designated time.

Timed-Release TUF-CTXT (PTXT) We say that TR-PKAE encryption is secure against timed-release TUF-CTXT (denoted by TUF-TR-CTXT) if no polynomial adversary (denoted by $\mathcal{A}_{\text{TUF-TR-CTXT}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{TUF-TR-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the following TUF-TR-CTXT game:

Setup : The challenger runs setup with security parameter k and generates $\langle \delta, \pi \rangle$, public/private key pairs (pk_s, sk_s) and (pk_r, sk_r) of sender and receiver correspondingly, and time T_a. The public keys, T_a and π are given to the adversary.

$\mathsf{Pre\text{-}forgery}$:

- $\mathsf{Random\ Oracle\ Queries}$: Adversary may query any random oracle
- $\mathsf{Queries\ for}$ $tkn[T]$: Adversary submits $T \neq T_a$ and receives $tkn[T]$
- $\mathsf{Encryption\ Queries}$: Adversary submits plaintext m, time T and obtains encryption using sk_s (sender), pk_r (receiver) and T.

$\mathsf{Forgery}$: Adversary submits ciphertext c.

$\mathsf{Outcome}$: Adversary wins the game if c successfully decrypts using pk_s (sender), sk_r (receiver), T_a, and c was not obtained during encryption queries.

We define $\mathbf{Adv}^{\text{TUF-TR-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_s, sk_r, T_a] = true]$. 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 TUF-TR-PTXT. We skip the details.

Timed-Release RUF-CTXT (PTXT) 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:

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 $\langle \pi, pk_s, T_a \rangle$.

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, receiver's secret key sk_b, time T and obtains encryption using sk_s (sender), sk_b (receiver) and T.

Forgery : Adversary submits ciphertext c and receiver's secret key sk_{b^*}.

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

We define $\mathbf{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_s, sk_{b^*}, T_a] = true]$. 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-PTXT. We skip the details.

TUF-CTXT (PTXT) In addition, below we state a time-independent TUF-CTXT game. Since TUF-CTXT is time-independent, 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_s, sk_s) and (pk_r, sk_r) 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_s (sender), pk_r (receiver) and T.

Forgery : Adversary submits ciphertext c and T.

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

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

3 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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ 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 [12].

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 $\mathbb{G}_2$ [9]
3. Decisional Diffie-Hellman Problem (DDHP) is easy in $\mathbb{G}_1$ [9].
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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ along with polynomial deterministic algorithms for group operations and e and generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$.

We say that algorithm $\mathcal{A}$ has advantage $\epsilon(k)$ in solving 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 Proposed Scheme for TR-PKAE

4.1 Description of the Scheme

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

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

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

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

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

4 : 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 system parameters are $\pi = \langle q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_{pub}, H_i, i = 1...4\rangle$ and the master key δ is $s \in \mathbb{Z}_q^*$.

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

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

Encrypt : Given secret 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 $Q = rpk_b$, 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 $R = H_3(\widehat{\sigma}, \widehat{m})P$

[1] 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].

The symmetric encryption scheme above is due to Fujisaki and Okamoto [16]. 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.

4.2 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-CCA with sender key compromise.

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

Note 1. We note that adversary $\mathcal{A}_{\text{IND-KC-CCA}}$ is attacking only one public key. We note that IND-KC-CCA security in one-user setting implies security in multi-user case. For details, we refer reader to [3]

Also, the proposed protocol is TUF-CTXT secure which is stronger than TUF-TR-CTXT.

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 $\boldsymbol{Adv}^{\text{TUF-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) \geq \epsilon$. 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 $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

Note 2. We note that security against TUF-CTXT also implies security against third-person plaintext forgery (TUF-PTXT). More precisely, given the same bounds on queries and run-times, $\mathbf{Adv}^{\text{TUF-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) \geq \mathbf{Adv}^{\text{TUF-PTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$. The reader is referred to [7] for details.

Theorem 3 (IND-RTR-CCA). *Let $\mathcal{A}$ be IND-RTR-CCA adversary, let q_d be the number decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}^{\text{IND-RTR-CCA}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$. 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 $\boldsymbol{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) \geq \epsilon$. 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 $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

Note 3. Note 2 applies in this case as well, i.e. $\mathbf{Adv}^{\text{RUF-TR-CTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k) \geq \mathbf{Adv}^{\text{RUF-TR-PTXT}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$.

5 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.

5.1 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 (plaintext) was indeed generated by the alleged sender. 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 [10]).

We define the following additional algorithms:

$\mathsf{TokenTester}_{\mathsf{TokenGenerator}}$: Given designated time T, it outputs either "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 secret key (and possibly different cryptosystem). $Tokens_{pub}$ denotes a set of times for which $TokenTester$ outputs "published".

We will present a scheme for the proposed TR-PKAE scheme 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.

5.2 Security

Before we present the specific algorithm, we note that the prover may expose some information with respect to submitted ciphertext/plaintext and secret key sk_B. In this case, we need to answer the following questions:

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

Confidentiality

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

IND-KC$_V$-CCA We modify IND-KC-CCA game to include verifications. We say that TR-PKAE encryption is secure against adaptive chosen-ciphertext with verifications (and key compromise) attack (IND-KC$_V$-CCA) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-KC}_V\text{-CCA}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-KC}_V\text{-CCA}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the IND-KC$_V$-CCA game. The IND-KC$_V$-CCA game is identical to the IND-KC-CCA except that Pre-Challenge phase includes Verifications queries:

Verifications : Adversary submits secret key sk_a, valid ciphertext c 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. The ciphertext c denotes an encryption with sk_a (sender), sk_r (receiver) and time encoding T. (As an alternative, adversary can also submit plaintext m and the challenger would generate the ciphertext).

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

IND-RTR$_V$-CCA We also modify IND-RTR-CCA to add verifications. We say that TR-PKAE encryption is secure against timed-release receiver adaptive chosen-ciphertext with verifications attack (IND-RTR$_V$-CCA) if no polynomial adversary (denoted by $\mathcal{A}_{\text{IND-RTR}_V\text{-CCA}}$) has a non-negligible advantage (denoted by $\mathbf{Adv}^{\text{IND-RTR}_V\text{-CCA}}_{\mathcal{TR}-\mathcal{PKAE},\mathcal{A}}(k)$) against the challenger in the IND-RTR$_V$-CCA game. The IND-RTR$_V$-CCA game is identical to the IND-RTR-CCA except that Pre-Challenge phase includes Verifications queries:

Verifications : Adversary submits secret key sk, valid ciphertext c and time T. Adversary obtains information exposed during verification. The ciphertext c denotes an encryption with sk (sender), pk_a (receiver) and time encoding T. (As an alternative, adversary can also submit plaintext m and the challenger would generate the ciphertext).

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

Ciphertext (Plaintext) Forgery

In the scheme to be proposed, some information will be exposed to the verifier, that is, the proof of origin to a third party will not be zero-knowledge. As a consequence, the protocol will lose TUF-CTXT property, although only for the designated date of the ciphertext used in the proof. Still, the TUF-TR-CTXT (PTXT) will be retained. More than that, given verification for designated time T' it will be hard for the verifier to forge a ciphertext if either designated time $T \neq T'$ or the intended 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 following game is a modification of TUF-CTXT game in which the challenger generates public/secret key pairs of sender and receiver, time T_c, and provides adversary with the public keys, T_c and master secret. The adversary is allowed to obtain verification information on ciphertexts using the above sender and receiver only for designated time $T \neq T_c$. 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 and public/private key pairs (pk_a, sk_a), (pk_b, sk_b). The adversary receives $\langle\pi, \delta, T_c, pk_a, pk_b\rangle$.

Pre-Forgery :

Random Oracle Queries : Adversary may query any random oracle

Verifications :

Case 1 : Attacker submits a sender secret key sk, time T and ciphertext encrypted with sk, pk_b (receiver) and T. Attacker obtains the information exposed during the verification.

Case 2 : The same as the previous case except that now pk_a is the receiver.

Case 3 : Adversary submits ciphertext with pk_a (sender), pk_b (receiver) and designated time $T \neq T_c$. Adversary obtains the information exposed during the verification.

Case 4 : The same as the previous case except that sender/receiver roles are interchanged.

Encryption Queries :

Case 1 : Adversary submits plaintext m, time T and obtains encryption using pk_a (sender), pk_b (receiver) and time T.

Case 2 : Adversary submits plaintext m, time T and obtains encryption using pk_b (sender), pk_a (receiver) and time T.

Forgery : Adversary submits ciphertext c.

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

We define $\mathbf{Adv}^{\text{TUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) = \Pr[\mathsf{Decrypt}[c, pk_b, sk_a, T_c] = true]$. 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 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 that the *Pre-Forgery* phase includes in addition the verifications:

Verifications : Attacker submits a sender secret key sk, time T 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.

5.3 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 public/secret pair $\langle pk_p, sk_p \rangle$, verifier V, ciphertext $c = \langle Q, c_1, c_2 \rangle$, 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 [2]. 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 $\langle X, sk_p X \rangle$ for some $X \in \mathbb{G}_1$. Note that $KB_2 \cdot e(pk_a, sk_p P_T) = e(rP + pk_a, tkn[T] + sk_p P_T)$. Thus the prover has to show that $KB = KB_2 \cdot e(pk_a, sk_p P_T)$, or equivalently that $KB_{part} = KB/KB_2 = e(pk_a, sk_p P_T)$. Verifier can compute $e(pk_a, sk_p X)$. Zero-knowledge proof proves knowledge of $k^* = \log_{P_T} X$ and that $e(pk_a, sk_p X)^{1/k^*} = KB_{part}$. Noting that $1/k^* = \log_X P_T$, we obtain that $e(pk_a, sk_p X)^{1/k^*} = e(pk_a, sk_p \log_X(P_T)X) = e(pk_a, sk_p P_T)$. 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. Consistency requirement is clearly satisfied as well. The verification protocol exposes the following information to the verifier: $sk_p(kP_T)$, kP_T and $e(pk_a, sk_p P_T)$. Next we state security results for TR-PKAE with verification protocol.

5.4 Security

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

Theorem 5 (IND-KC$_V$-CCA). *Let $\mathcal{A}$ be IND-KC$_V$-CCA adversary, q_d be the number of decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}_{\mathcal{TR-PKAE},\mathcal{A}}^{\text{IND-KC}_V\text{-CCA}}(k) \geq \epsilon$. 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$_V$-CCA). *Let $\mathcal{A}$ be IND-RTR$_V$-CCA adversary, let q_d be the number decryption queries and q_2 the number of queries made to the H_2 oracle. Assume that $\boldsymbol{Adv}_{\mathcal{TR-PKAE},\mathcal{A}}^{\text{IND-RTR}_V\text{-CCA}}(k)$. 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 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 $\boldsymbol{Adv}_{\mathcal{TR-PKAE},\mathcal{A}}^{\text{TUF-TR}_V\text{-CTXT}}(k) \geq \epsilon$. 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 $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

Note 4. Analogously to Note 2, TUF-TR$_V$-CTXT security implies TUF-TR$_V$-PTXT security. More precisely, $\mathbf{Adv}^{\text{TUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \mathbf{Adv}^{\text{TUF-TR}_V\text{-PTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$

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 $\boldsymbol{Adv}^{RUF\text{-}TR_V\text{-}CTXT}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \epsilon$. 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 $O(time(\mathcal{A})) + O(q_e \cdot q_2)$.*

Note 5. As before, RUF-TR$_V$-CTXT security implies RUF-TR$_V$-PTXT security. More precisely, $\mathbf{Adv}^{\text{RUF-TR}_V\text{-CTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k) \geq \mathbf{Adv}^{\text{TUF-TR}_V\text{-PTXT}}_{\mathcal{TR-PKAE},\mathcal{A}}(k)$

6 Discussion

6.1 Practical Considerations

This protocol can be used for many practical applications that require delayed opening of messages. The TokenGenerator can be played by an agent, which behaves like an NTP server [22] and at regular time-intervals publishes sP_t for current time t. We note that this value is *self-authenticated*: each user can compute $e(sP, P_t)$ and verify if it is equal to $e(P, sP_t)$ (because of bilinearity). Therefore, it can be replicated to any server. Output of the TokenGenerator can be shared by multiple instances of applications, since public key encryption is independent of the token.

Time-stamping authority and cryptosystem used are independent from TR-PKAE. In fact, it can be replaced by any mechanism which proves existence of the required ciphertext prior to designated opening time. For example, in sealed-bid auction the auction house could sign all encrypted bids and publish them prior to the closing of bidding.

In addition, the proposed scheme can be easily transformed into more restricted protocols such as non-authenticated designated timed-release encryption (by replacing sender's secret key with 0), authenticated public timed-release encryption (by replacing receiver's secret/public key with 0 and setting $Q = rP$). In the extreme case of public non-authenticated timed-release encryption, the protocol reduces to IBE scheme (where ID is replaced by time). The transformed protocol retains appropriate security properties. For example, unauthenticated designated timed-release encryption is still IND-CCA and IND-RTR-CCA, but other properties are naturally dropped. For space constraints, and because the resulting proofs are simple modifications of the proofs provided, we dispense with them in this paper.

6.2 Alternative Constructions

The functionalities of our scheme can be also achieved by adapting the IBE based on Weil/Tate pairing [9]. To this end, one can simply replace the ID in IBE by the target time [13, 20]. Then one can use this encryption scheme as a "time-capsule" containing perhaps additional cryptographic operations as the payload [20]. On the other hand, one can instead use the IBE encryption as the payload encrypted with a public key. Even though these constructions are more straightforward, the precise choice of how exactly IBE should be combined with other cryptographic operations in a provably secure way is not clear.

However, we note that the proposed scheme is almost as efficient as the bare IBE (FullIdent) [9] in terms of computational and spatial complexity. First, encryption operation in bare IBE 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 bare IBE. Therefore, the hybrid protocols that combine IBE with additional cryptographic primitives are bound to be at least as expensive as our scheme.

Based on the measurements oaf cryptographic primitives given in Appendix A, the encryption and decryption are expected to take about 41 and 42 msec, respectively. We used Miracl library v.4.8.2 [24] in P3-977 MHz with 512 Mbytes memory. In MapToPoint and Pairing, we considered a subgroup of order q in a supersingular elliptic curve E over $\mathbb{F}_p$, where p is a 512 bit prime and q is a 160 bit prime. Note that the pairing value belongs to a finite field of order 1024 bits.

7 Concluding Remarks and Future Work

We proposed a formal security model for timed-release public-key based authenticated encryption cryptosystem. Our security model introduces additional timed-release security notions such as timed-release IND-CCA and timed-release RUF-CTXT. We also presented a new cryptographic scheme for TR-PKAE and proved that it is IND-CCA- (even when sender key is compromised) and TUF-CTXT-secure. In addition, we proved timed-release equivalent of IND-CCA security against receiver itself and timed-release variation of RUF-CTXT which does not allow the receiver itself to forge ciphertexts with future designated time. The scheme proposed allows for a proof of message origin to a third party while at the same time maintaining its IND-KC-CCA property and a slightly weaker variation of TUF-CTXT in which third party cannot forge ciphertexts for future times. This result is quite surprising, since general authenticated encryption scheme constructed from symmetric key encryption does not allow message origin proof to a third party without exposing the secret key of the prover. The proposed scheme is at least as efficient and compact as other possible IBE-based constructions.

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. Recently, there was a discussion in newsgroup sci.crypt [15] regarding whether HIBE (hierarchical id-based encryption) [19] can remove the infrastructure required to store the agent's published tokens. Strictly speaking, FSE [11] (forward secure encryption) would be sufficient to provide this functionality, since in this case one can compute a previous token from a later one. However, existing FSE schemes (including HIBE) are too expensive for practical applications and the question whether there exists a practical scheme that does not require a repository still remains unanswered.

References

1. J. H. An. Authenticated encryption in the public-key setting: Security notions and analyses. `http://eprint.iacr.org/2001/079/`, 2001.
2. 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.
3. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Proceedings CRYPTO 2000, Springer LNCS 1807, pp 259 - 274*, 2000.
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. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Asiacrypt 2000, Springer LNCS 1976, pp 531 - 545*, 2000.
8. 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.

9. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Proc. of Crypto '01*, 2003.
10. 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.
11. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *Proceedings EUROCRYPT 2003, Springer LNCS 2656, pp 255 - 271*, 2003.
12. J. Cha and J. Cheon. An id-based signature from gap-diffie-hellman groups. In *In Public Key Cryptography - PKC 2003*, 2003.
13. 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.
14. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Proc. of Eurocrypt '99*, 1999.
15. P. from newsgroup sci.crypt. Key evolving encryption. Available from `http://www.groupsrv.com/science/viewtopic.php?t=61168&start=0`, 2004.
16. E. Fujisaki and T. Okamoto. Secure integration of assymetric and symmatric encryption schemes. In *Proceedings CRYPTO 1999, Springer LNCS 1666, pp 537 - 554*, 1999.
17. J. Garay and C. Pomerance. Timed fair exchange of arbitrary signatures. In *In Financial Crypto*, 2003.
18. J. A. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *In Financial Cryptography '02*, 2002.
19. C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *Proceedings Asiacrypt 2002, Springer LNCS 2501, pp 548 - 566*, 2002.
20. K. H. Marco Casassa Mont and M. Sadler. The hp time vault service: Exploiting ibe for timed release of confidential information. In *WWW2003*, 2003.
21. T. May. Timed-release crypto. `http://www.cyphernet.org/cyphernomicon/chapter14/14.5.html-`.
22. D. Mills. Network time protocol (version 3) specification, implementation. Technical Report 1305, Mar. 1992.
23. 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.
24. Shamus Software Ltd. Miracl: Multiprecision integer and rational arithmetic c/c++ library. `http://indigo.ie/~mscott/`.
25. P. F. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *1998 IEEE Computer Security Foundations Workshop (CSFW11)*, 1998.
26. 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 Measurements of Cryptographic Operations

The experimental results for the cost of several cryptographic primitives are presented in Table 1. We used Miracl library v.4.8.2 [24] in P3-977 MHz with 512 Mbytes memory. In MapToPoint and Pairing, we considered a subgroup of order q in a supersingular elliptic curve E over $\mathbb{F}_p$, where p is a 512 bit prime and q is a 160 bit prime. Note that the pairing value belongs to a finite field of order 1024 bits.

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	-	-	2.42
Pairing	512	-	31.71

B Security Proofs

Proof of Theorem 1 [**IND-KC-CCA**] The Theorem result follows from Corollary 13. 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-CCA. We design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real IND-KC-CCA 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.

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 $c_T Z$ 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 $\langle T, Q, a, c_1, c_2 \rangle$ where c_1 denotes $\sigma \oplus K$ and c_2 denotes $m \oplus H_4(\sigma)$, Q represents rpk_b, a is the sender secret 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). 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 $e(rP, sH_1(T))e(aP, sH_1(T))e(pk_b, aH_1(T))e(Q, H_1(T))$ (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 secret key $a = a^*$ 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 $rpk_b = Q^*$ 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 $\langle T, Q, a, c_1, c_2 \rangle$. $\mathcal{B}$ 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 $[Y/[e(sP + pk_b, aH_1(T))e(Q, H_1(T))]]^{(sc_T t^*)^{-1}}$ 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 $[Y/[e(sP + pk_b, a^*H_1(T^*))e(Q^*, H_1(T^*))]]^{(sc_{T^*} t^*)^{-1}}$ 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 $rpk_b = t^*b'X$ 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 $Y/[e(sP + pk_b, a^*H_1(T^*))e(Q^*, H_1(T^*))] = e(sP, rH_1(T^*))$ where $rpk_b = t^*(b'P)$ 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 $e(sP + pk_b, aH_1(T))e(Q, H_1(T))e(rP, sH_1(T))$ 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 simulator SIM_{new}. Denote by SIM_{old} the original simulator. In SIM_{new} to be constructed, challenger knows a',b' and c'. Up to the challenge, SIM_{new} behaves the same way as SIM_{old} including the random oracle queries. In addition, SIM_{new} calculates correctly the bilinear map in the challenge and assigns the hash value to this pairing as in SIM_{old} unless this input was already queried by adversary from H_2, in which case SIM_{new} uses the value of K returned by H_2. In SIM_{new}, this value of K is put in the database of H_2 with input being the correct calculation of the pairing. In both simulations, Q and c_2 of the ciphertext are chosen in the same way with the only possible difference being in c_1. SIM_{new} 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 SIM_{new} simulation (up to probability of guessing).*

Proof: We note that in the *Decryption Queries Before/After Challenge* SIM_{new} provides incorrect answer only if adversary guessed one of the values. In the *Challenge* phase SIM_{new} 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, SIM_{new} is indistinguishable from a real game since all values in SIM_{new} are chosen at random starting with random initial seeds. □

Lemma 12. *If $\mathcal{A}$ attains advantage of at least ϵ in the real game, then the fraction of inconsistent runs of SIM_{old} is at least 2ϵ and therefore SIM_{old} will have a solution to BDHP in its database with probability of at least 2ϵ.*

Proof: From Lemma 11 it follows that $\mathcal{A}$ achieves advantage ϵ in SIM_{new}. We construct the following bi-simulation:

1. Both simulators start to run simultaneously
2. Replies by SIM_{old} are given to SIM_{new} which checks them for correctness and passes them to adversary.
3. When SIM_{old} becomes inconsistent (when $\mathcal{A}$ queries H_2 with the true value of the challenge bilinear map, or SIM_{new} intercepts incorrect reply of SIM_{old} to decryption query according to its own calculations), SIM_{old} goes through the current step locally (by recording in H_2 adversary query or finishing completely decryption phase) and stops.
4. SIM_{new} takes over from SIM_{old} at this point (e.g. the incorrect decryption query reply of SIM_{old} is replaced with correct one, where SIM_{new} uses the *Before Challenge* phase for all decryption queries) and runs alone.

The advantage that adversary wins in this game is ϵ since the bi-simulation is indistinguishable from SIM_{new}. This means that adversary wins in fraction $1/2+\epsilon$ of runs of this bi-simulation. Let p_{fail} be the fraction of the runs in which SIM_{old} stops pre-maturely. Then

– Simulator SIM_{old} has a value of BDHP solution in its database in p_{fail} of the runs

– In runs where SIM_{old} does not stop, adversary wins with probability 1/2, since no correct query of the challenge bilinear map was made to H_2 and, therefore, adversary cannot distinguish ciphertexts other than by guessing

Let k be the fraction of times adversary wins when SIM_{old} stops. Then $(1-p_{fail})1/2 + p_{fail} \cdot k = 1/2 + \epsilon$. It follows from this equation that $p_{fail}(k - 1/2) = \epsilon$ and, therefore, $p_{fail} \geq 2\epsilon$. Thus, the fraction of all runs of bi-simulation in which SIM_{old} stops is at least $p_{fail} \geq 2\epsilon$, which means that SIM_{old} will have a value of BDHP in its databases with probability of at least 2ϵ. □

Corollary 13. *Probability that a random run of SIM_{old} produces the solution to BDHP is at least $\frac{2\epsilon}{q_d+q_2}$*

Proof: From Lemma 12 it follows that probability that a random run of SIM_{old} contains a solution to BDHP in its list of possible answers is at least 2ϵ. From SIM_{old} we recall that size of this list is at most $q_d + q_2$ and SIM_{old} returns a random value from this list (with some pre-processing, possibly). Therefore, it follows that SIM_{old} solves BDHP with probability at least $\frac{2\epsilon}{q_d+q_2}$. □

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_s and pk_r : $\mathcal{B}$ chooses public key of receiver pk_r to be $b'P = b'X$ and public key of sender pk_s 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 $Q = rb'P \in \mathbb{G}_1$ for some $r \in \mathbb{Z}_q$, $rH_1(T) = r(c_T b'P) = c_T(rb'P) = c_T Q$, i.e. knowing $rb'P$ 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 $pk_s = a'P$ being the sender and $pk_r = b'P$ 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_r 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 $e(sP + pk_r, (r + sk_s)H_1(T))$, where $rpk_b = Q$

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 $Q = rpk_b$ (r is unknown to $\mathcal{B}$) from each entry and computes $[Y/[e(sP, rH_1(T))e(sk_s, sH_1(T))e(pk_r, rH_1(T))]]^{c_T^{-1}}$, 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_r) 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 $e(sP, rH_1(T^*))e(pk_s, sH_1(T^*))e(pk_r, rH_1(T^*))$ 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 14. 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 15. *If the simulation above becomes inconsistent, then $\mathcal{B}$ will have correct answer for $e(X,X)^{a'b'^2}$ 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 $rpk_r = Q$. Then $Y = e(sP + pk_r, (r + sk_s)H_1(T)) = e(pk_r, sk_sH_1(T))e(sP, rH_1(T))e(pk_s, sH_1(T))e(pk_r, rH_1(T))$. In the *Outcome* phase of the simulation, $\mathcal{B}$ computes $Y/[e(sP, rH_1(T))e(pk_s, sH_1(T))e(pk_r, rH_1(T)) = e(pk_r, sk_sH_1(T)) = e(pk_r, sk_s(c_T b'P)) = e(P,P)^{a'b'^2 c_T}$. Since $\mathcal{B}$ takes the result to power c_T^{-1}, the true value of $e(P,P)^{a'b'^2}$ is indeed in the list of possible values. It follows that at least one out of $q_e \cdot q_2 + 1$ possible values in the list is a true value of $e(P,P)^{a'b'^2}$. □

Next, one constructs SIM_{new} and bi-simulation analogously to the proof of Theorem 1 (details skipped). And Lemma 11 carries over here as well with obvious modifications. The following Lemmas are slightly different from those in the proof of Theorem 1.

Lemma 16. *If $\mathcal{A}$ attains advantage of at least ϵ in the real game, then the probability that SIM_{old} will have the correct value of $e(P,P)^{a'b'^2}$ in its output list is at least ϵ.*

Proof: Let p_f be the fraction of runs of the bi-simulation in which SIM_{old} becomes inconsistent. From Lemma 15, in these runs, SIM_{old} will contain the true value of $e(P,P)^{a'b'^2}$ in its output list. Then we note that in fraction $1 - p_f$ of the runs the probability that SIM_{old} will have correct value of $e(P,P)^{a'b'^2}$ in its output list depends on success of $\mathcal{A}$, i.e. it is ϵ. If $\mathcal{A}$ is successful, the possible value of $e(P,P)^{a'b'^2}$ extracted by SIM_{old} from the forgery result will be the true value of $e(P,P)^{a'b'^2}$. It follows that probability that output list of SIM_{old} will contain correct value of $e(P,P)^{a'b'^2}$ is $p_f + (1 - p_f)\epsilon \geq \epsilon$.□

Corollary 17. *Probability that a random run of the above simulation produces the correct value of $e(P,P)^{a'b'^2}$ is at least $\frac{\epsilon}{q_e \cdot q_2 + 1}$*

Proof: Since the size of the output list is at most $q_e \cdot q_2 + 1$ and $\mathcal{B}$ makes a random choice from this list, the result follows from Lemma 16 $\square$

The above simulation is used to solve BDHP $\langle X, a''X, b''X, c''X\rangle$ as follows. Algorithm $\mathcal{B}$ runs the above simulation with 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 computes $E_1 = e(X,X)^{b_1^2 a''}$ with advantage at least $\frac{\epsilon}{q_e \cdot q_2 + 1}$. Then $\mathcal{B}$ runs the simulation with 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 computes $E_2 = e(X,X)^{b_2^2 a''}$ with advantage at least $\frac{\epsilon}{q_e \cdot q_2 + 1}$. Dividing E_1 by E_2, $\mathcal{B}$ obtains $e(X,X)^{a''b''c''}$ with advantage $[\frac{\epsilon}{q_e \cdot q_2 + 1}]^2$

Proof of Theorem 3 [**IND-RTR-CCA**] The Theorem result follows from Corollary 18. 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-CCA. We design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real IND-RTR-CCA 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 and T_a : $\mathcal{B}$ chooses random $sk_a = a \in \mathbb{Z}_q^*$ and T_a. Adversary $\mathcal{A}$ receives $pk_a = aP$ 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 $\langle T, b, Q, c_1, c_2\rangle$, where b is the private key of receiver 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 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^* (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 $[J/[e(sP, aH_1(T_a))e(rbP, H_1(T_a))e(bP, aH_1(T_a))]]^{t^{*-1}}$. 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 $e(sP + b^*P, aH_1(T_a))e(Q^*, H_1(T))$ 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 SIM_{new} and bi-simulation, and the Lemmas that follow the simulation of the proof of Theorem 1 naturally carry over with minor modifications. We skip the details and just state the final Corollary:

Corollary 18. *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 21. 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. First, 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 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.

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

Choice of pk_s and T_a : $\mathcal{B}$ chooses pk_s to be $a'P$ and a random T_a. Adversary receives pk_s and T_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 : 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 $sH_1(T) = c_T(b'P)$.

Encryption queries : $\mathcal{A}$ submits T, m and receiver secret key b. 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 $sH_1(T) = c_T(b'P)$. 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 ciphertext represents encryption of m with sender $a'P$ and receiver b. 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 $c^* = \langle Q^*, b^*, T_a, c_1^*, c_2^* \rangle$.

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

1. $\mathcal{B}$ goes through database D_s, obtains $\langle T_a, r, b \rangle$ from each entry and computes $Y/[e(sP, rH_1(T_a))e(bP, rH_1(T_a))e(a'P, bH_1(T_a))]$ for every query Y to H_2. The results are written down as possible values of $e(P, P)^{a'b'c'}$.
2. 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 $e(sP + b^*P, r^*H_1(T_a))e(a'P, b^*H_1(T_a)))$ and writes down the answer as a possible value of $e(P, P)^{a'b'c'}$. Note that if $\mathcal{A}$ wins 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).

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 of $e(P, P)^{a'b'c'}$ is at most $q_e \cdot q_2 + 1$.

Definition 19. 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 $TRUE[T_a, r, b]$ in D_s.

Lemma 20. *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 $TRUE[T_a, r, b]$ in D_s. Then $Y = [e(sP, rH_1(T_a))e(bP, rH_1(T_a))e(a'P, bH_1(T_a))]e(sP, a'P_{T_a})$. In the *Outcome* phase of the simulation, $\mathcal{B}$ computes $Y/[e(sP, rH_1(T_a))e(bP, rH_1(T_a))e(a'P, bH_1(T_a))] = e(sP, a'P_{T_a}) = e(b'P, a'c'P) = e(P, P)^{a'b'c'}$. Therefore, the true value of $e(P, P)^{a'b'c'}$ will be in the output list of $\mathcal{B}$. $\square$

Next, with minor modifications the steps carried out at this point in the proof of Theorem 2 apply here as well. The proofs are almost the same. We state the final Corollary only:

Corollary 21. *Probability that a random run of the above simulation produces the correct value of* $e(P, P)^{a'b'c'}$ *is at least* $\frac{\epsilon}{q_e \cdot q_2 + 1}$

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 and T_c : $\mathcal{B}$ chooses pk_a to be $a'X$ and pk_b to be $b'X$. Also, random T_c is chosen. The adversary $\mathcal{A}$ receives pk_a, pk_b 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 $c_T P$. 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 and makes a choice which one of pk_a and pk_b will be receiver and which one will be the sender. Due to symmetry and since master secret is public, we restrict ourseleves to the case where pk_a is the sender and pk_b is the receiver. The other case is carried out identically.

- $T \neq T_c$: In this case, $\mathcal{B}$ can compute the bilinear map as $e(sP + pk_b, rH_1(T) + c_T pk_a)$. 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 $c = \langle Q = rpk_b, pk_a, pk_b, T, c_1, c_2\rangle$.

$\mathcal{B}$ keeps the local database D_s for case $T = T_c$ in which it enters triple $\langle r, pk_a, pk_b\rangle$. Let us denote by $TRUE[r, pk_a, pk_b]$ the true value of $e(sP + pk_b, (r + sk_a)H_1(T_c))$. Similarly, when roles of pk_a and pk_b are interchanged above, $\mathcal{B}$ enters corresponding triple $\langle r, pk_a, pk_b\rangle$, and $TRUE[r, pk_b, pk_a]$ denoted the true value of $e(sP + pk_a, (r + sk_b)H_1(T_c))$.

Verification Queries :

- Case 1 : Adversary submits a sender secret key sk, valid ciphertext encrypted with sk, r, pk_b (receiver) and time T. The new information that the adversary can obtain is the pair $(kP_T, sk_b(kP_T))$. Note that we can omit the zero-knowledge proof here since one can easily remove it from the algorithm $\mathcal{A}$, for it provides no new state.

  $\mathcal{B}$ chooses random $k \in \mathbb{Z}_q^*$ and returns $\langle kP, k(b'X)\rangle$.
- Case 2 : Symmetrical to Case 1
- Case 3 : 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, $e(pk_b, sk_a P_{T_c}) = e(pk_a, c_T pk_b)$ and pair $\langle kP, k(b'X)\rangle$

Case 4 : Symmetrical to Case 3

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 $TRUE[r, pk_a, pk_b]$ and query Y to H_2 computes $Y/[e(sP, rH_1(T_c))e(pk_a, sH_1(T_c))e(pk_b, rH_1(T_c))]$. The answer is written down as a possible value of $e(P,P)^{a'b'c'}$. The case $TRUE[r, pk_a, pk_b]$ is symmetric.
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 $e(sP, r^*H_1(T_c))e(pk_a, sH_1(T_c))e(pk_b, r^*H_1(T_c))$. 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 22. 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 23. *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 $TRUE[r, pk_a, pk_b]$ in D_s. Then $Y = [e(sP, rH_1(T_c))e(pk_a, sH_1(T_c))e(pk_b, rH_1(T_c))]e(pk_a, sk_bH_1(T_c))$. In the *Outcome* phase, $\mathcal{B}$ computes $Y/[e(sP, rH_1(T_c))e(pk_a, sH_1(T_c))e(pk_b, rH_1(T_c))] = e(pk_a, sk_bH_1(T_c))$ $= e(a'X, b'(c'X)) = e(X,X)^{a'b'c'}$ 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$_V$-CCA**] The Theorem statement follows from Corollary 24. 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$_V$-CCA. First we design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real IND-KC$_V$-CCA 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.

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 secret key sk_a, 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 $kH_1(T) = k(c_T P)$ 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 $\langle T, Q, a, c_1, c_2\rangle$ where c_1 denotes encryption of σ and c_2 is the encryption of plaintext, Q represents rbP, a is the sender secret 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 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 $e(rP, sH_1(T))e(aP, sH_1(T))e(bP, aH_1(T))e(Q, H_1(T))$ (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 secret key $a = a^*$ 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 $\langle T, Q, a, c_1, c_2\rangle$. $\mathcal{B}$ 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 $[Y/[e(sP + bP, aH_1(T))e(Q, H_1(T))]]^{(sc_T)^{-1}}$ and writes the result as a possible value of $e(X, X)^{a'^2 b'}$ 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 $e(X, X)^{a'^2 b'}$. 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 $e(X, X)^{a'^2 b'}$. 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

$[Y/[e(sP+bP, a^*H_1(T^*))e(Q^*, H_1(T^*))]]^{(sc_{T^*})^{-1}}$ 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. Since all replies in the verifications phase are consistent, all proofs and statements stay the same and we obtain the following Corollary (which parallels Corollary 13):

Corollary 24. *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 that in the simulation the Verifications phase is added as follows:

Verifications : Adversary submits a sender secret key sk, valid ciphertext $c = \langle Q, c_1, c_2\rangle$ encrypted with sk, pk_s (receiver) and time T. $\mathcal{B}$ 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 $\langle m, \sigma, kP, k(a'X)\rangle$ (again note that we can omit the zero-knowledge proof here since one can easily remove it from the algorithm $\mathcal{A}$, for it provides no new state)

Proof of Theorem 6 [**IND-RTR$_V$-CCA**] The proof is identical to the proof of Theorem 3 except that in the simulation the Verifications phase is added as follows:

Verifications : Adversary submits a sender secret key sk, valid ciphertext $c = \langle Q, c_1, c_2\rangle$ encrypted with sk, $pk_a = aP$ (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 (which in this case is, moreover, valid).