Timed-Release and Key-Insulated Public Key Encryption

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

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

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

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

1 Introduction

Timed-Release cryptography. The goal of timed-release cryptography is to "send a message into the future." One way to do this is to encrypt a message such that the receiver cannot decrypt the ciphertext until a specific time in the future. Such a primitive would have many practical applications, a few examples include preventing a dishonest auctioneer from prior opening of bids in a sealed-bid auction [26], preventing early opening of votes in e-voting schemes, and delayed verification of a signed document, such as electronic lotteries [28] and check cashing. The problem of timed-release cryptography was first mentioned by May [21] and then discussed in detail by Rivest *et. al.* [26]. Let us assume that Alice wants to send a message to Bob such that Bob will not be able to open it until a certain time. The possible solutions fall into two categories:
– Time-lock puzzle approach. Alice encrypts her message and Bob needs to perform non-parallelizable computation without stopping for the required time to decrypt it.

* The third and the fourth authors were supported, in part, by NSF Career Grant CNS-0448423 and by the Intelligent Storage Consortium at the Digital Technology Center (DTC), University of Minnesota. The first author was supported by Korea Telecom. $\ast$: Contact author

– Agent-based approach. Alice encrypts a message such that Bob needs some secret value, published by a trusted agent on the required date, in order to decrypt the message.

The first approach puts immense computational overhead on the message receiver, which makes it impractical for real-life scenarios. In addition, knowing the computational complexity of decryption, while giving us a lower bound on the time Bob may need to decrypt the message, does not guarantee that the plaintext will be available at a certain date. Still, this approach is widely used for specific applications [9, 4, 28, 19, 18]. The agent-based approach, on the other hand, relieves Bob from performing non-stop computation, sets the date of decryption precisely and does not require Alice to have information on Bob's capabilities. This comes at a price, though: the agents have to be trusted and they have to be available at the designated time.

In this paper we concentrate on the agent-based approach. Several agent-based constructions were suggested by Rivest *et. al.* [26]. For example, the agent could encrypt messages on request with a secret key which will be published on a designated date by the agent. It also could precompute pairs of public/private keys, publish all public keys and release the private keys on the required days. A different scheme was proposed in [13], in which non-malleable encryption was used and receiver would engage in a conditional oblivious transfer protocol with the agent to decrypt the message. In [11], the authors proposed to use Boneh and Franklin's IBE scheme [8] for timed-release encryption: for that, one can replace the identity in an IBE scheme with the time of decryption. Similar proposals appear in [20, 7]. While some of these proposals contain informal proofs of security, none of them consider and/or give a formal treatment of the security properties of timed-release public key encryption (or TR-PKE).

Since all known efficient constructions rely on the Boneh-Franklin IBE construction, a natural question to ask is if the existence of IBE is necessary for an efficient timed-release public key encryption. In this paper, we formalize the security requirements of TR-PKE and show that indeed this is the case: the existence of secure TR-PKE is equivalent to the existence of strongly key-insulated encryption with optimal threshold and random access key updates; existence of which in turn is known to be equivalent to the existence of IBE [5, 14].

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

- (*Sequential Key Updates*) At the beginning of each time period, the helper securely transmits a "helper secret key" hsk_i to the user, which he combines with his previous key, usk_{i-1}, to obtain a secret key usk_i that will decrypt messages encrypted during time period i.
- (*Random Access Key Updates*) Given any usk_i and hsk_j, the user can compute usk_j. This is useful for error recovery and it also allows the user to decrypt old messages.

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

Combining results of Bellare/Palacio [5] and Dodis/Katz [14] [3], it follows that *existence of SKIE-OTRU is equivalent to IBE.*

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

One avenue for developing a TR-PKAE scheme would be composing an unauthenticated TR-PKE scheme with either a signature scheme or a (non-timed-release) PKAE scheme. Although such constructions are possible, we note that the details of this composition are not trivial; examples from [2, 14] illustrate that naive constructions can fail to provide the expected security properties. Additionally, we note that such schemes are likely to suffer a performance penalty relative to a scheme based on a single primitive. Thus we also introduce a provably secure construction of a TR-PKAE scheme that is essentially as efficient as previous constructions of *non-authenticated* TR-PKE schemes [11, 20, 7].

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

– We give the first formal analysis of the security requirements for timed-release public key encryption (TR-PKE) and show that this notion is equivalent to SKIE-OTRU.
– We introduce the notion of TR-PKAE, as satisfying four notions: IND-KC-CCA2, security against adaptive chosen ciphertext attacks under compromise of the timed-release agent and sender's private key; TUF-CTXT, or third-party unforgeability of ciphertexts; IND-RTR-KC-CCA2, or receiver undecryptability before release time under compromise of sender's private key; and RUF-TR-CTXT, or receiver unforgeability before release time.
– We introduce a protocol that provides authenticated timed-release (or strongly key-insulated) public key encryption using a single primitive. The proposed protocol is essentially as efficient as Boneh and Franklin's chosen-ciphertext secure IBE scheme [8] (FullIdent, which will be referred to as BF-IBE in the rest of the paper) and is provably secure in the random oracle model. The proposed protocol requires minimal infrastructure (a single trusted agent) that can be shared among many applications and can be naturally converted

[3] Bellare/Palacio showed that KIE-OTRU is equivalent to IBE, while Dodis/Katz showed equivalence of SKIE-OTRU and KIE-OTRU

to a threshold version, which provides robustness as well as stronger security by allowing outputs of multiple agents to be used.

Overview of our construction Consider a public agent (similar to NTP server [23]), called TiPuS (Timed-release Public Server), which at discrete time-intervals publishes new *self-authenticating* information $I_T = f(P_T, s)$ for current time T, where f and P_T are public, and s is secret. Alice can encrypt a message for Bob at time T using P_T, her private key and Bob's public key. *Only when* I_T *is published on day* T, will Bob be able to decrypt the message using I_T, his private key and Alice's public key.

We implement the above setting using an admissible bilinear map e (see Section 4.1), which along with the choice of groups and generator P is chosen independently of TiPuS. Each TiPuS chooses a secret $s \in \mathbb{Z}_q$ and publishes $P_{pub} = sP$. At time T, the TiPuS publishes $I_T = sP_T = sH(T)$ [4] (*i.e.* the private key for identity T in BF-IBE [8], where H is a cryptographic hash function.

Let $(sk_a, pk_a) = (a, aP)$ and $(sk_b, pk_b) = (b, bP)$ be Alice's and Bob's authenticated private/public key pairs respectively. To *encrypt* message m for Bob, 1) Alice computes bilinear map $d = e(sP + r_1 \cdot bP, (r_2 + a)P_T)$ for random r_1, r_2, and applies hash function H_2 to obtain $K = H_2(d)$, 2) she then encrypts message m as $E_K(m)$, where E_K is a symmetric encryption using key K. Bob also receives r_1P_T and r_2P. To *decrypt* the ciphertext, 1) Bob, having sP_T, computes d as $e(r_2P + aP, sP_T + b \cdot r_1P_T)$ [5], 2) applying hash function H_2, Bob computes K and uses it to decrypt $E_K(m)$.[6] The full detailed protocol and all required definitions/discussions are presented in later sections.

Note the following practical aspects exhibited by the scheme: 1) (*User Secret vs TiPuS Secret*) the secret value of TiPuS, system parameters and users' private keys are completely independent. It will be shown later that compromise of TiPuS does not jeopardize confidentiality and unforgeability of user ciphertexts; 2) (*Sharing*) the published value sP_T can be shared among multiple applications; 3) (*Scalability*) the protocol can take full advantage of a) several independent TiPuS's, [7] b) threshold generation of sP_T [24]. The increase in computational complexity is minimal when such schemes are applied to the protocol.

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

In this section we formalize the functionality and security requirements for a timed-release public key encryption system. These requirements are meant to

[4] The authenticity of I_T can be verified by checking equality $e(P_{pub}, P_T)$, since by bilinearity $e(sP, H(T)) = e(P, sH(T)) = e(P, H(T))^s$.

[5] Note that according to properties of bilinear map, $e(r_2P + aP, sP_T + b \cdot r_1P_T) = e((r_2 + a)P, (s + b \cdot r_1)P_T) = e((s + r_1 \cdot b)P, (r_2 + a)P_T) = d$.

[6] Without authentication, this scheme is similar to Bellare and Palacio's construction of an SKIE-OTRU scheme, in which $d = e(sP + bP, r_2P_T)$. However note that it cannot be used for timed-release: the receiver can publish as public key $bP = \tau P - sP$ for any chosen τ allowing him to decrypt any ciphertext before designated time.

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

capture the required security requirements not addressed in previous work [21, 26, 11, 20, 7]; in particular they do not address the authentication requirements, which we add in section 3.

2.1 Functional requirements

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

- Setup, which given input 1^k (the security parameter), produces public parameters π_g, which include hash functions, message and ciphertext spaces among others.
- $\mathsf{TRSetup}$, which on input π_g, produces a pair (δ, π_{tr}) where δ is a *master secret* and π_{tr} the corresponding timed-release public parameters. This setup is carried out by TiPuS which keeps the master secret key confidential, while all other parameters are public. *We denote the combined public parameters of π_g and π_{tr} by π.*
- KeyGen, given public parameters π_g, outputs a pair of secret key and public key (sk, pk).
- $\mathsf{TG}(\pi, \delta, T)$ computes the token tkn_T corresponding to time T using (δ, π). This functionality is performed by TiPuS which publishes tkn_T at time T.
- $\mathsf{Encrypt}(\pi, pk, m, T)$ computes the timed-release ciphertext c denoting the encryption with public key pk of message m with public parameters π and time encoding T.
- $\mathsf{Decrypt}(\pi, sk, \widehat{c}, tkn_T)$ outputs the plaintext corresponding to $\widehat{c}$ if decryption is successful or the special symbol `fail` otherwise.

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

2.2 Security

It is standard to require that the PKE cryptosystem be secure against adaptive chosen-ciphertext (IND-CCA2) adversaries [25, 3, 2]. Ideally, in TR-PKE, one should separate the timed-release security from security of PKE. Namely, TR-PKE should maintain receiver confidentiality properties even if the timed-release master secret is compromised. To that effect, we require that IND-CCA2 security against a third party is provided even when master secret is given to the adversary. We model this attack by a slightly modified $\mathsf{IND\text{-}CCA2}$ game, shown in Figure 1. Here, in addition to adaptively choosing two “challenge plaintexts” that the adversary will need to distinguish between, he also adaptively chooses a “challenge time” for which his challenge ciphertext will be decrypted; he wins when he can tell whether his challenge ciphertext is an encryption of his first or second plaintext for the challenge time, given access to a decryption oracle and the master secret key of the TiPuS.

The timed-release functionality is provided by the token-generating infrastructure (i.e. TiPuS). Not knowing the corresponding token is what keeps the

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

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

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

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

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

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

receiver from decrypting ciphertext until a designated time. To effect secure timed-release, any TR-PKE cryptosystem must provide confidentiality against the receiver itself until the corresponding token is made available. We model this property by the IND-RTR-CCA2 game, shown in Figure 1; in this game, we modify the basic IND-CCA2 game by allowing the adversary to adaptively choose receiver public key pk^* and time T^* for the challenge. Instead of access to the timed-release secret, the adversary is given access to arbitrary tokens tkn_T, where $T \neq T^*$, and a decryption oracle $\mathsf{Decrypt}^*(\pi, \delta, \cdot, \cdot, \cdot)$ which computes $\mathsf{Decrypt}(\pi, \cdot, \cdot, \mathsf{TG}(\pi, \delta, \cdot)$. The adversary may thus compute the decryption of any ciphertext for any time, *except* the challenge ciphertext in the challenge time T^* with chosen public key pk^*. We say a timed-release public-key cryptosystem Γ is secure if every polynomial time adversary A has negligible advantages $\mathsf{Adv}^{\mathsf{IND-CCA2}}_{A,\Gamma}(k)$ and $\mathsf{Adv}^{\mathsf{IND-RTR-CCA2}}_{A,\Gamma}(k)$.

2.3 Strongly Key-Insulated Public Encryption and Timed-Release

The notion of key-insulated public key encryption has been discussed in [15, 16, 5]. As mentioned previously, combining Bellare/Palacio [5] and Dodis/Katz [14] one obtains that the existence of secure SKIE-OTRU is a necessary and sufficient condition for the existence of secure IBE. Briefly, a SKIE-OTRU consists of following algorithms: KG, which generates a triple (pk, usk_0, hsk) of public key, initial user secret key, and master helper key; HKU which computes a *stage i helper secret key* hsk_i given (pk, hsk, i); UKU, which computes the *stage i user secret key* usk_i given i, pk, hsk_i, usk_{i-1}; RUKU, which computes the *stage i user secret key* usk_i given $i, j, pk, hsk_i, usk_j, \forall i \geq 1, j \geq 0$; Enc, which produces a ciphertext corresponding to m to be decrypted in stage i, given (pk, m, i); and Dec, which, given (i, pk, usk_i, c) attempts to decrypt a ciphertext for stage

i. Intuitively, hsk is given to a "helper", who will securely transmit, at the beginning of each stage i, the secret hsk_i to the user. The user can then compute usk_i, delete any old usk's in his possession, and use usk_i to decrypt messages sent to him during stage i. Existence of RUKU facilitates error recovery and allows for decryption of old ciphertexts.

A SKIE (and SKIE-OTRU) scheme is considered CCA-secure with optimal threshold if two conditions hold: (1) given access to pk, a decryption oracle, and pairs (hsk_i, usk_i) of his choosing, an adversary cannot break the encryption scheme for a stage j for which he has not been given hsk_j; and (2) given pk, hsk, and a decryption oracle, an adversary cannot break the encryption scheme for any stage [15, 16, 5]. The idea of separation of the timed-release master and user secrets in a TR-PKE very closely parallels the notions of helper and user secrets in a key-insulated cryptosystem; and both involve a "time period" parameter for encryption and decryption. Furthermore, the two security conditions for a SKIE scheme, in which either user keys or helper keys are assumed to be compromised, closely resemble the conditions $\mathsf{IND\text{-}CCA2}$ and $\mathsf{IND\text{-}RTR\text{-}CCA2}$ developed here.

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

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

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

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

Now suppose we have a secure SKIE-OTRU scheme Σ. If Σ has the additional property that KG can be implemented as two independent keying algorithms that generate (pk_h, hsk) and (pk_u, usk), then it is straightforward to transform Σ into a TR-PKE scheme. Since we would not expect this property to hold in general, we work around this problem as follows. We know that by the existence of Σ there also exists an ordinary chosen-ciphertext secure PKC $\Pi = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$. The idea behind our construction is that $\mathsf{TRSetup}$ will sample $(spk, hsk, usk_0) \leftarrow \Sigma.\mathsf{KG}(1^k)$ and set $\pi = spk$ and $\delta = (hsk, usk_0)$; KeyGen will sample $(pk, sk) \leftarrow \Pi.\mathsf{PKGen}(1^k)$ and output (pk, sk). $\mathsf{TG}(\pi, \delta, \mathsf{i})$ will first compute $hsk_i = \mathsf{HKU}(spk, hsk, i)$ and then use usk_0 and hsk_i to compute $tkn_i = usk_i = \mathsf{RUKU}(i, 0, spk, usk_0, hsk_i)$. Encryption and Decryption will use the multiple-encryption technique of Dodis and Katz [14].[8] Applying the results of [14], an $\mathsf{IND\text{-}CCA2}$ attack on this scheme reduces to a chosen-ciphertext attack on Π, while an $\mathsf{IND\text{-}RTR\text{-}CCA2}$ attack (even when receiver chooses its public key adaptively) on this scheme reduces to an SKIE chosen-ciphertext attack on Σ.

3 Authenticated TR-PKE (TR-PKAE)

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

3.1 Basic Cryptosystem

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

– $\mathsf{Encrypt}(\pi, sk_A, pk_B, m, T)$ returns an authenticated timed-release ciphertext c denoting the encryption from sender A to receiver B of m for time T.

[8] Specifically, to encrypt message m for time T, we: (1) pick $s_1 \leftarrow U_{|m|}$, and set $s_2 = m \oplus s_1$, (2) pick signing and verification keys (SK, VK) for a one-time signature scheme, (3) let $c_1 = \Sigma.\mathsf{Enc}^{VK}(spk, s_1, T)$, $c_2 = \Pi.\mathsf{PKEnc}^{VK}(pk, s_2)$, and (4) output $(VK, c_1, c_2, \mathsf{Sig}(VK, (T, c_1, c_2)))$. Decryption follows the scheme of [14], except that c_1 is decrypted using $tkn_T = usk_T$.

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

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

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

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

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

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

– $\mathsf{Decrypt}(\pi, pk_A, sk_B, \widehat{c}, tkn_T)$ outputs plaintext $\widehat{m}$ if both decryption and authentication are successful and the special symbol `fail` otherwise.

The consistency requirement is modified to require that, for all valid (pk_A, sk_A), (pk_B, sk_B), (π, δ), T, and m, $\mathsf{Decrypt}(\pi,\ pk_A,\ sk_B,\ \mathsf{Encrypt}(\pi,\ sk_A,\ pk_B,\ m,\ T),\ \mathsf{TG}(\pi,\ \delta,\ T)) = m$.

3.2 Security

Confidentiality. The confidentiality requirements of a TR-PKAE are essentially the same as the confidentiality requirements of a TR-PKE; *except* that we make the conservative assumption that the third party (in the case of IND-CCA2) or the receiver (in the case of IND-RTR-CCA2) has compromised the sender's secret key. This results in two new notions, IND-KC-CCA2 and IND-RTR-KC-CCA2, which we define formally in Figure 2. As before, we say that a TR-PKAE scheme provides confidentiality if every polynomial time adversary has negligible advantage, as defined in Figure 2.

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

Ciphertext (Plaintext) Forgery. For authentication properties of TR-PKAE, we concentrate on ciphertext forgery (plaintext forgery is defined analogously). We consider two types of ciphertext forgery: *third-party forgery* (TUF-CTXT), by an adversary that does not know the sender's and receiver's private keys but knows the master secret; and *forgery by the ciphertext receiver* (RUF-CTXT) [2].

If the TR-PKAE is not secure against TUF-CTXT then the scheme cannot claim authentication properties since a third party may be able to forge new (perhaps decrypting to junk) ciphertexts between two users. If a TR-PKAE is not secure against RUF-CTXT, then the scheme does not provide non-repudiation [9] and furthermore, if the receiver's private key is compromised, the attacker can impersonate any sender to this receiver. We introduce the following games to model unforgeability (see Figure 3).

Timed-Release RUF-CTXT (RUF-TR-CTXT). We introduce a slightly weaker timed-release notion of RUF-CTXT, which requires that the receiver should not be able to forge ciphertext to himself for a future date. This notion has two important implications: (1) the receiver should discard any ciphertexts received past decryption dates if his private key may be compromised; and (2) the receiver may be able to prove to a third party that a ciphertext was generated by the alleged sender if he can produce a proof of ciphertext existence prior to the decryption date. The game in Figure 3 is an enhancement of the RUF-CTXT condition proposed by An [2] to allow adaptive adversarial behavior: the receiver is not given access to the token for a single, adaptively-chosen *challenge* time period; in addition, the adversary can choose any receiver public key in the encryption queries. We say that a TR-PKAE encryption is secure against RUF-TR-CTXT, if every polynomial-time adversary A has negligible advantage, $\mathsf{Adv}^{\mathsf{RUF-TR-CTXT}}_{A,\Gamma}(k)$, against the challenger in the RUF-TR-CTXT game.

TUF-CTXT. In addition to timed-release receiver unforgeability, we also require a time-independent third-party unforgeability (TUF-CTXT) condition, which allows to separate timed-release functionality from PKAE. Thus, in the TUF-CTXT game defined in Figure 3, the master key is given to the adversary. We say that a TR-PKAE scheme Γ is secure against TUF-CTXT if every polynomial time adversary $\mathcal{A}$ has negligible advantage, $\mathsf{Adv}^{\mathsf{TUF-CTXT}}_{\mathcal{A},\Gamma}(k)$, in k.

4 The Proposed TR-PKAE

Following the proof of Theorem 1, one approach to achieve TR-PKAE would be to combine a key-insulated encryption scheme with a PKAE scheme in a modular fashion using techniques such as given in [14]. However, it is desirable for modern authenticated encryption to have one primitive that achieves the desired security properties [10]: such solutions generally allow for a more efficient scheme, tighter security bounds and more stringent security. Below we construct an example of such a scheme that satisfies all of the above security requirements and is nearly as efficient as BF-IBE scheme [8]. We start with a review of Bilinear Diffie-Hellman Problem.

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

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

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

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

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

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

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

4.1 Bilinear Diffie-Hellman Problem

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two abelian groups of prime order q. We will use additive notation for group operation in $\mathbb{G}_1$ (where aP denotes P added a times for $P \in \mathbb{G}_1, a \in \mathbb{Z}_q$) and multiplicative notation for $\mathbb{G}_2$ (g^a denotes the g multiplied a times for element g of $\mathbb{G}_2$). Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be an admissible bilinear map [8]. The properties of the groups and constructions of e are explained in detail in [8]. We assume that the *Decisional Diffie-Hellman Problem* (DDHP) is hard in $\mathbb{G}_2$. Note that as a trivial consequence of DDHP assumption, the *Discrete Logarithm Problem* (DLP) is also hard in $\mathbb{G}_2$. As a consequence of the above assumptions, it follows that DLP is hard in $\mathbb{G}_1$ [22].

Let $\mathcal{G}$ be a *Bilinear Diffie-Hellman* (BDH) *Parameter Generator* [8], *i.e.* a randomized algorithm that takes positive integer input k, runs in polynomial time in k and outputs prime q, descriptions of $\mathbb{G}_1$, $\mathbb{G}_2$ of order q, description of admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ along with polynomial deterministic algorithms for group operations and e and generators $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$. We say that algorithm $\mathcal{A}$ has advantage $\epsilon(k)$ in solving the *computational* BDH Problem (BDHP) for $\mathcal{G}$ if there exists k_0 such that:

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

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

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

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

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

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

TRSetup :

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

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

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

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

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

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

Fig. 4. The proposed TR-PKAE scheme

4.2 Description of the Scheme

Let $\mathcal{G}$ be a *BDH Parameter Generator*. Figure 4 gives a complete description of our construction[10]. The symmetric encryption scheme used is a straightforward adaptation of the Fujisaki-Okamoto scheme [17]. We briefly demonstrate the consistency of the scheme before moving on to security considerations. Given ciphertext $c = \langle Q_1, Q_2, \sigma \oplus K, m \oplus H_4(\sigma)\rangle$ computed using sk_A, pk_B and T, we note that in the corresponding Decrypt computations we have 1) $\widehat{K} = K$ since $e(Q_2 + pk_a, sP_T + sk_b \cdot Q_1) = e(r_2 P + sk_a P, sP_T + sk_b \cdot r_1 P_T) = e([r_2 + sk_a]P, [s + r_1 \cdot sk_b]P_T) = e([s + r_1 \cdot sk_b]P, [r_2 + sk_a]P_T) = e(P_{pub} + r_1 \cdot pk_b, [r_2 + sk_a]P_T)$, 3) as in Fujisaki-Okamoto, it follows that $\widehat{\sigma} = \sigma$, $\widehat{m} = m$ and 4) $Q_1 = H_3(\widehat{\sigma}, \widehat{m})P$ and $Q_2 = H_4(\widehat{\sigma}, \widehat{m})P$. Thus the original plaintext is retrieved.

[10] As in [8], 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 [8].

4.3 Security of the Scheme

The following security results apply to TR-PKAE. The hash functions are modeled as random oracles [6]. Due to space considerations, the detailed proofs of these results are omitted from this extended abstract and are available online [12]. First, we note the confidentiality properties of the proposed scheme.

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

Theorem 3 (IND-RTR-KC-CCA2). *Let $\mathcal{A}$ be a* IND-RTR-KC-CCA2 *adversary that makes q_d decryption queries, q_2 queries to H_2 and q_{tok} queries to* TG. *Assume that $\mathsf{Adv}^{\mathsf{IND-RTR-KC-CCA2}}_{\mathcal{A},TR\text{-}PKAE}(k) \geq \epsilon$. Then there exists an algorithm $\mathcal{B}$ that solves computational BDHP with advantage $\mathsf{Adv}^{\mathsf{cbdh}}_{\mathcal{B},\mathcal{G}}(k) \geq \frac{1}{4q_2 \cdot \max(q_2,q_d)} \left[\frac{\epsilon}{e\cdot(1+q_{tok})}\right]^3$ and running time $O(time(\mathcal{A}))$, where $e = 2.71828...$.*

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

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

Theorem 5 (RUF-TR-CTXT). *Let $\mathcal{A}$ be a* RUF-TR-CTXT *adversary that makes q_e encryption queries, q_2 queries to H_2, and q_{tok} queries to* TG, *and let $\mathsf{Adv}^{\mathsf{RUF-TR-CTXT}}_{\mathcal{A},TR\text{-}PKAE}(k) \geq \epsilon$. Then there exists an algorithm $\mathcal{B}$ with computational BDHP advantage $\mathsf{Adv}^{\mathsf{cbdh}}_{\mathcal{B},\mathcal{G}}(k) \geq \frac{\epsilon}{2\cdot q_2 \cdot q_e \cdot e \cdot (1+q_{tok})}$ and running time $O(time(\mathcal{A}))$, where $e = 2.71828...$.*

5 Efficiency of TR-PKAE

To compare the proposed scheme to BF-IBE [8], note that, in terms of significant operations – *bilinear pairings, MapToPoint, exponentiations* – TR-PKAE adds 3 additional exponentiations in $\mathbb{G}_1$ for encryption and 2 for decryption. More precisely, encryption in TR-PKAE involves 1 bilinear map, 4 exponentiations in $\mathbb{G}_1$ and 1 MapToPoint (to compute P_T). The decryption involves 1 bilinear map and 3 exponentiations in $\mathbb{G}_1$ (assuming P_T is pre-computed). Second, the proposed scheme adds additional point in $\mathbb{G}_1$ to the ciphertext. Taking into account functionality of TR-PKAE and the fact that naive combinations yielding hybrid protocols generally fail to provide required security, we expect hybrid constructions of TR-PKAE to be at least as expensive as our scheme.

We implemented the proposed primitives using Miracl library v.4.8.3 [27] with Tate pairing for the bilinear map. The group $\mathbb{G}_1$ was chosen to be a subgroup of order q in a supersingular elliptic curve E over $\mathbb{F}_p$, where p is a 512 bit and q is a 160 bit primes. Group $\mathbb{G}_2$ was a subgroup of a finite field of order 1024 bits. We used a P4-3.2 GHz "Northwood" (800MHz FSB) with 2GB of 400 MHz RAM desktop. The performance measurements are summarized in Table 1 and are all averaged over 10000 runs, except that the RSA results were obtained by running OpenSSL v.0.9.8 *speed* command. As expected, the proposed TR-PKAE is somewhat more expensive than BF-IBE in encryption/decryption, but when BF-IBE is extended to provide comparable functionality to TR-PKAE we expect the resulting scheme to be at least as expensive as the proposed protocol.

Table 1. Cost of basic operations

Function	modulus (bits)	exponent (bits)	performance (msec)
RSA(Sig/Dec)	1024	1024	2.96
RSA(Ver/Enc)	1024	16 ($e = 2^{16} + 1$)	0.14
Scalar Mul in EC over $\mathbb{F}_p$	160	160	2.23
MapToPoint	512	-	1.52
Pairing	512	160	18.15
TR-PKAE Enc	512	160	29
TR-PKAE Dec	512	160	25
BF-IBE Enc	512	160	24
BF-IBE Dec	512	160	21

6 Acknowledgements

The authors thank Moti Yung for the excellent suggestion to bridge the link between timed-release and key-insulated encryption and many other invaluable comments, and the anonymous reviewers for helpful feedback.

References

1. M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *CT-RSA*, 2001.
2. J. H. An. Authenticated Encryption in the Public-Key Setting: Security Notions and Analyses. `http://eprint.iacr.org/2001/079/`, 2001.
3. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO*, 1998.
4. M. Bellare and S. Goldwasser. Encapsulated Key Kscrow. Technical report, MIT/LCS/TR-688, 1996.
5. M. Bellare and A. Palacio. Protecting against Key Exposure: Strongly Key-Insulated Encryption with Optimal Threshold. `http://eprint.iacr.org/2002/064/`, 2002.
6. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS*, 1995.
7. I. F. Blake and A. C.-F. Chan. Scalable, Server-Passive, User-Anonymous Timed Release Public Key Encryption from Bilinear Pairing. In *ICDCS*, 2005.

8. D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. In *CRYPTO*, 2003.
9. D. Boneh and M. Naor. Timed Commitments. In *CRYPTO*, 2000.
10. X. Boyen. Multipurpose Identity Based Signcryption: A Swiss Army Knife for Identity Based Cryptography. In *CRYPTO*, 2003.
11. L. Chen, K. Harrison, D. Soldera, and N. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *InfraSec*, 2002.
12. J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Authenticated Key-Insulated Public Key Encryption and Timed-Release Cryptography. Available from `http://eprint.iacr.org/2004/231`, 2004.
13. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional Oblivious Transfer and Timed-Release Encryption. In *EUROCRYPT*, 1999.
14. Y. Dodis and J. Katz. Chosen-Ciphertext Security of Multiple Encryption. In *Theory of Cryptography Conference*, 2005.
15. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-Insulated Public Key Cryptosystems. In *EUROCRYPT*, 2002.
16. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong Key-Insulated Signature Schemes. In *PKC*, 2003.
17. E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO*, 1999.
18. J. Garay and C. Pomerance. Timed Fair Exchange of Arbitrary Signatures. In *Financial Cryptography*, 2003.
19. J. A. Garay and C. Pomerance. Timed Fair Exchange of Standard Signatures. In *Financial Cryptography*, 2002.
20. K. H. Marco Casassa Mont and M. Sadler. The HP Time Vault Service: Exploiting IBE for Timed Release of Confidential Information . In *WWW*, 2003.
21. T. May. Timed-Release Crypto. `http://www.cyphernet.org/cyphernomicon/chapter14/14.5.html-`.
22. A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *IEEE Transactions on Information Theory IT-39, 5*, 1993.
23. D. Mills. Network Time Protocol (Version 3) Specification, Implementation. Technical Report 1305, RFC, 1992.
24. T. P. Pederson. A Threshold Cryptosystem Without a Trusted Party. In *EUROCRYPT*, 1991.
25. C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *CRYPTO*, 1991.
26. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock Puzzles and Time-released Crypto. Technical report, MIT/LCS/TR-684, 1996.
27. Shamus Software Ltd. MIRACL: Multiprecision Integer and Rational Arithmetic C/C++ Library. `http://indigo.ie/~mscott/`.
28. P. F. Syverson. Weakly Secret Bit Commitment: Applications to Lotteries and Fair Exchange. In *Computer Security Foundations Workshop*, 1998.

A Selected Security Proofs

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 the game under consideration. Below, for each game, we design an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ by simulating a real game for the adversary in order to compute solution to BDHP $e(X,X)^{a'b'c'}$.

Proof of Theorem 2 [IND-KC-CCA2]

Setup :

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

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

Choice of pk_b: $\mathcal{B}$ chooses receiver public key $pk_b = b'P$, which is given to adversary $\mathcal{A}$.

Choice of sk_a: $\mathcal{B}$ chooses $a \in_R \mathbb{Z}_q^*$ at random which is given to adversary.

Databases: Databases corresponding to $H_i, i = 1, ..., 5$ 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 :

Queries for P_T (or H_1): $\mathcal{B}$ samples $c_T \in_R \mathbb{Z}_k$ and returns $P_T = c_T \cdot P$, storing the query T in the database coupled with c_T. Repeated queries retrieve answers from the database.

Queries to $H_i, i = 1, ..., 5$: $\mathcal{B}$ 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 $\langle T, Q_1, Q_2, c_1, c_2 \rangle$ where c_1 denotes $\sigma \oplus K$ and c_2 denotes $m \oplus H_4(\sigma)$, Q_1 represents $r_1 P_T$, Q_2 represents $r_2 P$, $sk_a = a$ is the sender private key and T is the designated time.

$\mathcal{B}$ goes through the databases of H_3 and H_4 searching for appropriate r_1 and r_1. If either is not found, false is returned. Otherwise, the corresponding σ and m are retrieved. Then database of H_5 is searched for query with σ. If this σ was not queried in H_5 then false is returned. Otherwise, $\mathcal{B}$ computes $c_2 \oplus H_5(\sigma)$ and compares it with m. If they are not equal, false is returned. Next, database of H_1 is queried: if it never returned P_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 $e(sP, (r_2 + a)P_T) \cdot e([r_2 + a] \cdot bP, Q_1)$. If they are equal, true is returned. Otherwise, false is returned.

Selection: $\mathcal{A}$ chooses two equal-sized plaintexts m_0,m_1, and $T = T^*$.

Challenge: $\mathcal{B}$ chooses arbitrary $\beta \in \{0,1\}$, and assigns $Q_1^* = a'P = a'X$, $Q_2^* = c'P = c'X$. Then $\mathcal{B}$ chooses σ^*, two random strings c_1^* and c_2^*, and composes and returns ciphertext $c^* = \langle T^*, Q_1^*, Q_2^*, c_1^*, c_2^* \rangle$. The databases are updated as follows:

H_3, H_4: $\mathcal{B}$ puts Q_1^* as a value (marked that it's already multiplied by P_T) and (σ^*, m_β) as the query into database of H_3. Similar steps are taken with respect to Q_2.

H_5: $\mathcal{B}$ puts $m_\beta \oplus c_2^*$ as a value and σ^* as the query into database of H_5.

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 $K = K^* = \sigma^* \oplus c_1^*$.

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 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 $\langle T, Q_1, Q_2, c_1, c_2\rangle$. $\mathcal{B}$ searches for r_1, r_2 corresponding to Q_1, Q_2 in databases of H_3, H_4. If either one is not found (even in the form that we put during challenge), $\mathcal{B}$ simply returns false. If $Q_1 = Q_1^*, Q_2 = Q_2^*, T = T^*$ we also return false (note that if $c_1 \neq c_1^*$, and adversary computed bilinear map correctly, then the correct query was made to H_2). Otherwise consider two cases:

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

$Q_2 \neq Q_2^*$: If r_2 could not be retrieved, then false is returned. Otherwise, we can compute bilinear map as $e((r_2 + a)P, sP_T)e((r_2 + a) \cdot bP, Q_1)$ and thus answer correctly the decryption query.

Outcome: β is returned or simulation halts.

1. If during the above simulation we managed to compute the solution to BDHP, then that is value is output by $\mathcal{B}$.
2. Otherwise, $\mathcal{B}$ outputs at random one of the queries that the adversary queries has made to H_2.

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

Proof of Theorem 4 [TUF-CTXT]

Setup :

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

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

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

Databases: Databases corresponding to $H_i, i = 1, ..., 5$ 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 :

Queries for P_T (or H_1) : $\mathcal{B}$ chooses random $c_T \in \mathbb{Z}_q^*$ and returns $c_T(c'P)$. Query T along with c_T are stored and replies for repeated queries use the database.

Queries to $H_i, i = 2, ..., 5$: Same as in the proof of Theorem 2.

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

Forgery: $\mathcal{A}$ submits ciphertext $\langle T^*, Q_1^*, Q_2^*, c_1^*, c_2^* \rangle$.

Outcome: $\mathcal{A}$ returns forged ciphertext or simulation halts. Then $\mathcal{B}$ flips a coin. If D_s is empty and/or adversary did not make any queries to H_2, then we reset coin outcome to 'tails'. If no forgery was submitted then coin outcome is changed to 'heads'.

Coin is 'heads': $\mathcal{B}$ picks a random entry from database D_s, and obtains corresponding T, r_1, r_2. Then it picks random adversarial query Y to H_2 and computes $Y/[e(r_2P + aP, sP_T) \cdot e(r_1 \cdot bP, r_2P_T)]]^{c_T^{-1} r_1^{-1}}$, which is output as the solution to BDHP.

Coin is 'tails': $\mathcal{B}$ first obtains corresponding values of r_1 and r_2 since they had to be queried from H_3 and H_4. Then it extracts $K = \sigma \oplus c_1^*$ and looks-up the query Y that was made to H_2 and returned K. Then as in the previous case, a candidate solution to BDHP is computed.

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

Proof of Theorem 3 [**IND-RTR-KC-CCA2**] We use a biased coin that with probability $\theta > 0$ returns 0 and otherwise returns 1. The optimal value of θ will be determined later.

Setup :

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

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

Choice of pk_a: $\mathcal{B}$ chooses random sender's private key $sk_a = a \in \mathbb{Z}_q^*$ which is given to adversary $\mathcal{A}$.

Databases: Databases corresponding to $H_i, i = 1, ..., 5$ 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 :

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

Queries to $H_i, i = 2, ..., 5$: Same as in the proof of Theorem 2.

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

Decryption Queries Before Challenge: $\mathcal{A}$ submits ciphertext $\langle T, b, Q_1, Q_2, c_1, c_2 \rangle$, where b is the receiver's secret key, pk_a is the sender and T, Q_i, c_1 and c_2 carry the same meaning as in the previous proofs. As before, $\mathcal{B}$ obtains r_1 and r_2 from the databases, which allows it to decrypt correctly.

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

Challenge: $\mathcal{B}$ chooses arbitrary $\beta \in \{0, 1\}$ and σ, and sets $r_2 = r_2^* = a'$ while the value of $r_1 = r_1^*$ is computed in a normal way. Then it chooses the value of the bilinear map at random and composes the resulting ciphertext $c^* = \langle T^*, pk_{b^*}, Q_1^* = r_1^* P, Q_2^* = a'P, c_1^*, c_2^* \rangle$.

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 T^* and sk_{b^*} corresponding to pk_{b^*}. For decryption queries, $\mathcal{B}$ behaves according to *Decryption Queries After Challenge* phase.

Decryption Queries After Challenge: $\mathcal{A}$ submits ciphertext $\langle T, b, Q_1, Q_2, c_1, c_2 \rangle$. If $P_T = c_T P$, then $\mathcal{B}$ can compute the corresponding bilinear map and answer the query correctly. If $P_T = c_T \cdot c'P$, then we consider two cases separately: 1) $Q_2 = Q_2^*$, or 2) $Q_2 \neq Q_2^*$.

Case $Q_2 = Q_2^*$: Assume that $Q_1 \neq Q_1^*$ and $T \neq T^*$ (otherwise we return false). Note that in this case r_1 used in this ciphertext must be the same as r_1^* used in the challenge – if not (and we can check that) we return false. Let us write the bilinear map used in the ciphertext as $B = e(sP, r_2 P_T) \cdot D_1$. Knowing sk_a, sk_b, r_1, $r_2 P$ and sP, we can correctly compute $D_1 = e(sP, aP_T) \cdot e(r_2 P + aP, r_1 \cdot bP)$. However, we cannot necessarily compute $e(sP, r_2 P_T)$ (which would solve the BDHP). We enter the pair (D_1, T) in database D_{aux} and return false to the adversary.

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

Outcome of First Simulation: β is returned or simulation halts.

1. We pick at random a query Y_1 that an adversary made to H_2 and a random pair (Y_2, T) from database D_{aux}. We then compute $B_1 = (Y_1/Y_2)^{c_T^{-1}}$. *Note that if Y_1 is the correct value of the bilinear map corresponding to entry (Y_2, T), then B_1 is the solution to BDHP*
2. We pick at random a query F_1 that an adversary made to H_2 (a possible value of the bilinear map used in the challenge). We output the pair $(F_1, r_{1,1}^*)$, where $r_{1,1}^*$ is the value of r_1 used in the challenge.

Second Simulation: Second simulation is run using the same random tape for the adversary and changing the random tape of the simulator immediately after the *Selection* step, but ensuring that the only possible difference in the *Challenge* step is in the value of r_1.

Outcome of Second Simulation: β is returned or simulation halts.

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

Combined Outcome of Both Simulations: We flip a coin and based on the outcome proceed in one of the following ways:

– We pick output at random either B_1 or B_2 as the solution to BDHP.

– We compute $Z = (F_1/F_2)^{(r^*_{1,1}-r^*_{1,2})^{-1}}$, then compute $Y = F_1/[e(sP, aP_{T^*})\cdot Z^{r^*_{1,1}}]$ which is taken to power $c^{-1}_{T^*}$. The final result is output as the solution to BDHP. *Note that if F_i are indeed correct values of the challenge bilinear maps used in the simulations, then $Z = e(bP, (r^*_2 + a)P_{T^*})$, $Y = e(sP, r^*_2 P_{T^*})$ and the final result is indeed correct value of solution to BDHP.*

The optimal value of θ maximizes the probability that simulation does not quit during token queries or during selection. This probability is at least $\theta^{q_{tok}} \cdot (1-\theta)$, where q_{tok} is the number of token queries made by $\mathcal{A}$, and is maximized when $\theta = 1 - 1/(q_{tok}+1)$ with value $\frac{1}{e\cdot(1+q_{tok})}$, where $e = 2.718281828....$

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

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

$(P/2)^3 \cdot \min(P_1, P_2) \geq \frac{1}{4q_2 \cdot \max(q_2, q_d)} \left[\frac{\epsilon}{e \cdot (1+q_{tok})}\right]^3$

Proof of Theorem 5 [RUF-TR-CTXT]

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

Setup :

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

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

Choice of pk_a: $\mathcal{B}$ sets $pk_a = a'P$ and gives the public key to the adversary.

Databases: Databases corresponding to $H_i, i = 1, ..., 5$ 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 :

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

Queries to $H_i, i = 2, ..., 5$: Same as in the proof of Theorem on IND-KC-CCA2.

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

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

$P_T = c_T P$: $\mathcal{B}$ computes the ciphertext in a normal way. It chooses arbitrary σ, queries H_3 for r_1, H_4 for r_2, and then queries H_5 with input σ. Then it computes bilinear map as $e(sP + r_1 \cdot bP, r_2 P_T + aP_T)$ by noting that $aP_T = a' \cdot c_T P = c_T \cdot a'P$. The corresponding query is made to H_2 and $\mathcal{B}$ returns resulting ciphertext.

$P_T = c_T \cdot c'P$: $\mathcal{B}$ chooses σ and computes r_2 in the normal way. Then it picks random r_1' and sets $r_1 P_T = r_1' P$, updating appropriately the database of H_2. The value of bilinear map is chosen at random and the resulting ciphertext is given to the adversary. The simulator enters the parameters (T, r_2, r_1', bP) in database D_{aux}.

Forgery: $\mathcal{A}$ submits ciphertext $c^* = \langle Q_1^*, Q_2^*, T^*, c_1^*, c_2^* \rangle$ and the receiver secret key b^*, which will be used for verification. *If $P_{T^*} = c_{T^*} P$, the simulation fails.*

Outcome of Simulation: $\mathcal{A}$ returns forged ciphertext or simulation halts. Simulator flips a fair coin. If one of the cases below does not have data to proceed, simulator picks the other case.

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

Coin outcome is 'tails': $\mathcal{B}$ picks at random a query Y that adversary made to H_2. If r_2 for Q_2^* was not found in database of H_4, the simulation fails. If the actual value of r_1 corresponding to Q_1^* was found in the database of H_4, then $\mathcal{B}$ computes $Z = e(sP, r_2P_{T^*}) \cdot e(r_2P + aP, r_1 \cdot b \cdot P_T)$. Then it computes $(Y/Z)^{c_{T^*}^{-1}}$ and outputs the result as the solution to BDHP. If no actual value of r_1 was found and there exists an entry (T, r_2, r_1', bP) in D_{aux} such that $T = T^*$ and $r_1'P = Q_1^*$, then as in the 'heads' case we can compute $Z = e(r_1 \cdot b^*P, (r_2 + a)P_{T^*}) = e(b^*P, (r_2 + a) \cdot r_1P_{T^*}) = e(r_1' \cdot b^*P, r_2P + aP)$, then compute $Y/[e(sP, r_2P_*T) \cdot Z]$ and take the result to power $c_{T^*}^{-1}$ producing our solution to BDHP. *Note that if Y was the correct value of the bilinear map in the forgery, then we do obtain the solution to BDHP.*

The optimal value of θ maximizes the probability that simulation does not quit during token queries or during selection. This probability is at least $\theta^{q_{tok}} \cdot (1 - \theta)$, where q_{tok} is the number of token queries made by $\mathcal{A}$, and is maximized when $\theta = 1 - 1/(q_{tok} + 1)$ with value $\frac{1}{e \cdot (1+q_{tok})}$, where $e = 2.718281828....$

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

Taking into account the probability of failure due to the biased coin and advantage ϵ of the adversary in the real game, we obtain that the probability of outputting correct solution to BDHP is at least $\frac{\epsilon}{2 \cdot q_2 \cdot q_e \cdot e \cdot (1+q_{tok})}$.