

A Versatile Secure Protocol for Anonymous Timed-Release Encryption

D. Hristu-Varsakelis, K. Chalkias, and G. Stephanides

University of Macedonia

Department of Applied Informatics

Computational Systems and Software Engineering Laboratory

Thessaloniki, Greece

dcv@uom.gr, chalkias@java.uom.gr, steph@uom.gr

Abstract: We propose a new server-based protocol for timed-release encryption (TRE), sometimes referred to as “sending information into the future”. As with other recently-proposed schemes, ours is based on the use of bilinear pairings on a Gap Diffie-Hellman group, and allows a sender to specify with precision the release time of the encrypted data. Our protocol possesses various required properties related to security, user anonymity and server passivity. It also provides almost-costless scalability in settings with multiple time-servers, and improves significantly upon existing TRE schemes, in terms of communication cost. The basic version of our protocol is extended to include new desirable features, such as message pre-opening, and “hiding” of important details associated with a ciphertext, thus making our approach well-suited to a number of emerging e-applications that require future decryption of confidential data.

Keywords: cryptographic protocols, timed-release encryption, bilinear pairings, multiple time-servers, pre-open, hiding time information.

I. Introduction

The aim of *timed-release encryption* (TRE) is to support applications where encrypted confidential data must not be decrypted by anyone, including the designated recipient(s), until a predetermined future time. TRE was originally introduced in [22], and further studied in [28]. The TRE problem arises in many emerging applications with significant impact to daily personal and business activity, such as electronic voting, which requires delayed opening of votes [27]; sealed-bid auctions, where bids must stay sealed until the bidding period is over [28]; and Internet-based contests where participants must not access the challenge problem before the contest starts [4]. Other examples include contract signing, where two or more mutually suspicious parties need to exchange signatures on a contract [11], future release of documents (e.g., memoirs, wills, press releases) [28], delayed

release of escrowed keys [3], and online gambling and lotteries [8, 10].

Existing approaches to TRE are based upon the use of either a *time-lock puzzle* (TLP) or a so-called *trusted-agent* (TA). The former method was initially proposed for protecting communications against passive adversaries [23].

Later, [28] extended this technique to provide TRE. Briefly, the TLP approach consists of transforming a secret message in such a way that any serial or parallel machine must spend a significant amount of time solving a computational problem (puzzle) in order to reconstitute the message. Such techniques [16, 21, 28], do not require a trusted third-party, but they put immense computational overhead on the receiver, who must perform non-stop, non-parallelizable computation to retrieve the message. Except from “tying up” the receiver’s CPU, TLP techniques cannot guarantee a precise release time because they depend on the computational power of the receiver’s machine, and on the time at which the decryption process is started; thus they are impractical for many real-life scenarios.

To overcome the problems associated with TLPs, TA approaches make use of a so-called *time-server* who provides a common and absolute time reference to users. In these server-based schemes, receivers must retrieve a piece of information (trapdoor) from the time-server(s), akin to a secret key which is necessary (in conjunction with the user’s secret key) for the decryption process. Using TAs, one can set the decryption date with high precision. The tradeoff is that some interaction between the server and the users is required. In the early attempts at TRE [22, 28], the server was actively involved in the encryption or the decryption process, and user-anonymity was thus compromised. The situation improved with [13], which proposed a scheme based on a conditional oblivious transfer protocol [20], in which anonymity was achieved for the sender only. The non-pairing based scheme of [25] was the first to achieve server passivity, meaning that the TA’s only role is to publish

*universal*¹ time-specific trapdoors (e.g., on a web page). That approach was based on the quadratic residue assumption (QRA) and had a high communication cost compared to all of the elliptic curve (EC)-based protocols examined here.

Since the early work on TA-based TRE, researchers have focused on minimizing server-user interaction, to ensure scalability and user-anonymity. New and innovative TRE techniques appeared especially after the introduction of *Identity-Based Encryption* (IBE) [6]. The latter used elliptic curve cryptography (ECC) and the efficient implementation of bilinear pairings on ECs, leading to a series of important developments. Here, we review some of the best-known and most efficient protocols.

Some of the newer pairing-based schemes allow a user to recover past time-specific trapdoors from the currently published one. Specifically, [5] and [26] use the tree-like structure of [7] backwards, to construct previous trapdoors, while [10] uses an inverted hash chain, similarly to the S/Key system [17]. These approaches have a high communication cost, while the root of the tree-like structure (or hash chain) corresponds to the last available trapdoor to be published, implying an upper bound on the “lifetime” of the system.

Of particular interest in TA-based TRE is the problem of eliminating (or at least reducing) the possibility of collusion between the receiver and an unscrupulous time-server who might allow early access to a message. An often-used solution is to arrange matters so that the receiver is forced to obtain trapdoors from more than one servers to decrypt a message (more servers means that a larger number of entities must be corrupted in order for cheating to take place). In that setting, Blake’s and Chan’s [4] scheme formed the point of departure for a number of recent works, including ours, being the first to provide efficient scalable, server-passive, user-anonymous TRE with support for multiple time-servers. The work in [19] described a similar, but computationally less efficient scheme, that could also support message pre-opening². Improvements to [19] were later proposed in [12]. Also, [8, 9] designed user-anonymous TRE protocols that could make use of pre-computations (i.e., some of the calculations required to run the protocols can be performed offline, prior to specifying a message or a receiver), and thus be faster than previous approaches. The main disadvantage of those works was the high additional cost when using multiple time-servers. See also [27] for an attempt at authenticated TRE.

This paper is an extended version of [18]. Its contribution is to propose a server-based TRE protocol which is inspired by [4] and makes significant improvements in communication cost, as well as scalability, over other published TRE

approaches. With respect to computational cost, our protocol compares favorably to those mentioned previously, and has the second-lowest total cost in the case of sending a message to an unknown receiver (in a single time-server setting). The decryption cost of our protocol is as good as the fastest schemes in its category. Its communication cost is kept low by effectively combining the best features offered in existing TRE protocols, namely simple public key format and small ciphertext space. Moreover, the proposed scheme can be easily extended to provide message pre-open capabilities, and to hide (from anyone except the designated recipient) information regarding i) the release time which is associated with a ciphertext, and ii) the identity of the time-servers whose trapdoors are required for decryption. Finally, under some conditions, the additional computational cost required in settings with multiple time-servers is negligible. To our knowledge, this is the first attempt to efficiently support these properties in a single protocol.

The remainder of this paper is structured as follows. In Section 2 we describe a new TA-based TRE protocol, whose security is discussed in Section 3. Section 4 details a series of improvements to the basic protocol, culminating in a version endowed with additional privacy and security properties. In Section 5 we compare our protocol with [4] and with other modern TRE approaches, in terms of computational efficiency, communication cost and scalability.

Proposed Protocol

We begin by fixing notation and by reviewing some related definitions and computational assumptions used in this work. We will require an abelian, additive finite group, \mathbb{G}_1 , of prime order q , and an abelian multiplicative group, \mathbb{G}_2 , of the same order. For example, \mathbb{G}_1 may be the group of points on an elliptic curve (EC). We will let P denote the generator of \mathbb{G}_1 and $t \in \{0, 1\}^\tau$, $\tau \in \mathbb{N}$ denote time. For instance, t could indicate the τ -bit string representation of a specific time instant (e.g., 09:30:00 AM August 29, 2007 GMT). Also, H_1, H_2 will be two secure hash functions, with $H_1 : \{0, 1\}^\tau \mapsto \mathbb{G}_1^*$, $H_2 : \mathbb{G}_2^* \mapsto \{0, 1\}^n$. Finally, $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ will be a bilinear pairing.

Definition 1 Bilinear Pairings Let \mathbb{G}_1 be an additive cyclic group of prime order q generated by P , and \mathbb{G}_2 be a multiplicative cyclic group of the same order. A map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ is called a bilinear pairing if it satisfies the following properties:

- *Bilinearity:* $\hat{e}(aP, bQ) = \hat{e}(bP, aQ) = \hat{e}(abP, Q) = \hat{e}(P, abQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}_1$, $a, b \in \mathbb{Z}_q^*$.
- *Non-degeneracy:* there exist $P, Q \in \mathbb{G}_1$ such that $\hat{e}(P, Q) \neq 1$.
- *Efficiency:* there exists an efficient algorithm to compute the bilinear map.

We note that all pairing algorithms currently employed in cryptography are based on ECs, and thus make use of Millers algorithm [24]. Currently, admissible types of pairings include the Weil and Tate pairings and their variants [1, 15].

¹Here, *universal* means that a trapdoor is the same for all users who decrypt messages released at the time instant for which the trapdoor was created.

²Pre-opening refers to the ability of a sender to “speed-up” the decryption process by sending a trapdoor key before the designated time at which a message was scheduled to be opened.

The security of our protocol is based on the assumed difficulty of the following problems:

Definition 2 Discrete Log Problem (DLP) in \mathbb{G}_1

Given $P, Q \in \mathbb{G}_1$, it is difficult to find an integer $a \in \mathbb{Z}_q^*$ such that $Q = aP$.

Definition 3 Discrete Log Problem (DLP) in \mathbb{G}_2

Given $g_1, g_2 \in \mathbb{G}_2$, it is difficult to find an integer $a \in \mathbb{Z}_q^*$ such that $g_2 = g_1^a$.

Definition 4 Computational Diffie-Hellman Problem (CDHP) in \mathbb{G}_1

Given $P, aP, bP \in \mathbb{G}_1$, for some unknowns $a, b \in \mathbb{Z}_q^*$, it is difficult to find $abP \in \mathbb{G}_1$.

Definition 5 Bilinear Diffie-Hellman Problem (BDHP)

Given $P, aP, bP, cP \in \mathbb{G}_1$, for some unknowns $a, b, c \in \mathbb{Z}_q^*$, it is difficult to find $\hat{e}(P, P)^{abc}$.

Proposed anonymous TRE The proposed agent-based TRE scheme, termed *NewTRE*, assumes two types of entities: a time-server which issues time-specific trapdoors with some pre-determined frequency (e.g., every minute), and users which may act as either senders or receivers. To send a message, m , that will be decrypted at (or after) a pre-defined time t , the following protocol is to be executed:

NewTRE.Setup(run by the time-server): Given a security parameter k , the setup algorithm:

1. Outputs a k -bit prime number q , two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q , an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ and an arbitrary generator $P \in \mathbb{G}_1$.
 2. Chooses the following cryptographic hash functions: $H_1 : \{0, 1\}^\tau \mapsto \mathbb{G}_1^*$, $H_2 : \mathbb{G}_2^* \mapsto \{0, 1\}^n$. These functions will be treated as random oracles when it comes to security considerations.
 3. Generates the time-server's private key $s \in \mathbb{Z}_q^*$ and the corresponding public key $S = sP \in \mathbb{G}_1^*$.
 4. Chooses the message space to be $m = \{0, 1\}^n$ and the ciphertext space to be $C = \mathbb{G}_1 \times \{0, 1\}^{n+\tau}$.
- The public parameters are $params := \{k, q, \mathbb{G}_1, \mathbb{G}_2, P, S, \hat{e}, H_1, H_2, n, \tau, m, C\}$.

NewTRE.ReleaseT (run by the time-server): Given a time instant $t \in \{0, 1\}^\tau$ and the server's private key $s \in \mathbb{Z}_q^*$, it returns the time-specific trapdoor $sk_T = sT \in \mathbb{G}_1^*$, where $T = H_1(t) \in \mathbb{G}_1^*$. We note that the trapdoor is in fact a time-server's short signature (as this proposed in [5]) on that time, t , and is inherently self-authenticating. Thus, there is no need for an additional server signature: a user can simply check whether $\hat{e}(S, T) \stackrel{?}{=} \hat{e}(P, sk_T)$.

NewTRE.KeyGen (run by the receivers): Given $params$, it chooses a private key $b \in \mathbb{Z}_q^*$ and produces receiver's public key $B = bP \in \mathbb{G}_1^*$.

NewTRE.Enc (run by the senders): To encrypt $m \in \{0, 1\}^n$ using the time information $t \in \{0, 1\}^\tau$, the receiver's public key B and the server's public key S , the sender executes the

following:

1. Choose $r \in \mathbb{Z}_q^*$.
 2. Compute $T = H_1(t) \in \mathbb{G}_1^*$.
 3. Compute $Q = rT \in \mathbb{G}_1^*$.
 4. Compute $K = \hat{e}(S, Q) = \hat{e}(sP, rT) = \hat{e}(P, T)^{rs} \in \mathbb{G}_2^*$.
 5. Compute $c_1 = rB = rbP \in \mathbb{G}_1^*$ and $c_2 = m \oplus H_2(K) \in \{0, 1\}^n$, where \oplus denotes the XOR function.
- The ciphertext is $C := (c_1, c_2, t)$.

NewTRE.Dec (run by the receivers): Given $C := (c_1, c_2, t)$, the trapdoor sk_T and his private key b , the recipient computes:

1. $R = b^{-1}c_1 = b^{-1}brP = rP$. R can also be pre-computed (before the release time), and
2. the session key $K = \hat{e}(R, sk_T) = \hat{e}(rP, sT) = \hat{e}(P, T)^{rs} \in \mathbb{G}_2^*$.
3. He retrieves the message as $m = H_2(K) \oplus c_2$.

NewTRE is illustrated in Table 1.

Security and Adversarial Models

It will be helpful to distinguish between two kinds of adversaries. One is an *outside* attacker with knowledge of time-specific trapdoors for any time (e.g., a "curious" time-server), trying to decrypt a ciphertext he has intercepted. This attacker is to be challenged on a random user's public key for which he is equipped with a decryption oracle. A second adversary is an *inside* attacker that models a receiver who tries to decrypt a ciphertext before its designated release time. In that case, the adversary has knowledge of the receiver's private key, but does not know the time-server's private key or the trapdoor that will be published at the appointed time. We assume that an inside attacker can choose the public key on which he is challenged in a "find-then-guess" game, similar to that detailed in [8, 9], and can also access a release-time oracle returning trapdoors for any time period, except the one corresponding to the challenge ciphertext. In a chosen-ciphertext scenario, the adversary also has access to an oracle decrypting ciphertexts other than the challenge. In the *NewTRE* model, this adversary is called chosen-time period and ciphertext attacker (CTCA).

Definition 6 ([8]) Let \mathcal{A} be an outside adversary. A TRE scheme is said to be secure against chosen-ciphertext attacks (IND-CCA secure) if no polynomially bounded adversary \mathcal{A} has a non-negligible advantage in the following game:

1. A challenger, \mathcal{CH} , takes a security parameter 1^k and runs *TRE.Setup*(1^k) and *TRE.KeyGen* to obtain a list of public parameters, $params$, and a key pair (u_{pr}, u_{pub}) . The public key u_{pub} , $params$, and the server's private key, s_{pr} , are given to \mathcal{A} , while the private key, u_{pr} , is kept secret.
2. \mathcal{A} has access to a decryption oracle, *TRE.Decrypt*(.), which given a ciphertext (C, T) and the time-specific trapdoor s_T (always computable by anyone who knows s_{pr}), returns the decryption of C using the private key u_{pr} . At some point, \mathcal{A} outputs two equal-length messages m_0 ,

Table 1: Proposed improvement of Blake's and Chan's scheme (NewTRE)

(Sender)	(Receiver)	(Server)
	$(b, B = bP)$	$(s, S = sP)$
		$T = H_1(t)$ $sk_T = sT$
$r \xleftarrow{R} \mathbb{Z}_q^*$ $T = H_1(t)$ $Q = rT$ $K = \hat{e}(S, Q)$ $c_1 = rB$ $c_2 = m \oplus H_2(K)$	(c_1, c_2, t)	
		$\xleftarrow{sk_T}$ [at release time]
	$T = H_1(t)$ $K = \hat{e}(c_1, sk_T)^{b^{-1}}$ $m = c_2 \oplus H_2(K)$	

m_1 and a challenge time-period T^* . He gets a ciphertext $(C^*, T^*) = TRE.Encrypt(m_b, u_{pub}, params, T^*)$, for $b \xleftarrow{R} \{0, 1\}$, computed under the public key u_{pub} .

3. \mathcal{A} issues a new sequence of queries but is prohibited from asking for the decryption of the challenge for the time period T^* . He eventually outputs a bit b' , and wins if $b' = b$. His advantage is $Adv_{TRE, \mathcal{A}}^{IND-CTCA}(\mathcal{A}) := |Pr[b' = b] - 1/2|$.

Definition 7 ([8]) Let \mathcal{A} be an inside adversary. A TRE scheme is said to be secure against chosen-time period and ciphertext attacks (IND-CTCA secure) if no polynomially bounded adversary, \mathcal{A} , has a non-negligible advantage in the following game:

1. The challenger, CH, takes the security parameter 1^k and runs $NewTRE.Setup(1^k)$ to return the resulting public parameters $params$ to \mathcal{A} . The server's public key, s_{pub} , is given to \mathcal{A} , while the corresponding private key, s_{pr} , is kept secret.

2. \mathcal{A} has access to a release-time oracle $TRE.ReleaseT(.)$ returning trapdoors s_T for any time T . \mathcal{A} is also given access to a decryption oracle, $TRE.Dec(.)$, which given a ciphertext C and a receiver's public key, u_{pub} , provided by \mathcal{A} , computes the decryption of C using s_T , but without knowing the corresponding user's private key u_{pr} . At some moment, \mathcal{A} outputs messages m_0, m_1 , an arbitrary public key u_{pub}^* , and a time instant T^* that has not been submitted to the $TRE.ReleaseT$ oracle. He receives the challenge $(C^*, T^*) = TRE.Enc(m_b, u_{pub}^*, params, T^*)$, for a hidden bit $b \xleftarrow{R} \{0, 1\}$.

3. \mathcal{A} issues a new sequence of release-time queries for any time instant except T^* , and decryption queries for any ciphertext but the challenge (C^*, T^*) , for the public key u_{pub}^* . He eventually outputs a bit b' and wins if $b' = b$. His

advantage is $Adv_{TRE, \mathcal{A}}^{IND-CTCA}(\mathcal{A}) := |Pr[b' = b] - 1/2|$.

By applying the FO transformation [14] to NewTRE, we obtain a modified version of our protocol which can be shown to be secure against IND-CTCA and IND-CCA attackers. The FO transform consists of altering the protocol so that the sender encrypts their message not with a random integer r but rather with the hash value of a string which contains the plaintext, a random integer, the decryption time, and the receiver's public key. See [8, 9] for examples of its use.

Theorem 1 Assume that a polynomial-time IND-CTCA attacker has a non-negligible advantage $\epsilon(k)$ against NewTRE. Then the BDH problem can be solved (in polynomial time) with non-negligible probability.

Theorem 2 Assume that a polynomial-time IND-CCA attacker has a non-negligible advantage $\epsilon(k)$ against NewTRE. Then the CDH problem can be solved (in polynomial time) with non-negligible probability.

The proofs of the last two theorems are very similar to those in [8], [9]. A sketch is provided below. Similar statements can be made for the optimized version of NewTRE, OptTRE, discussed in Section I.

Sketch of Security Proofs The proof proceeds by initially showing that NewTRE is secure against plaintext attackers (based on a modification of the proof in [4]).

1. The server's and receiver's private keys are safe because it is difficult to find s or b given P and sP or bP , respectively (it is an instance of the DL problem).

2. The server's key is also safe from an attacker who tries to compute s from the sT that the server produces for various times, t_i . Rewriting sT_i as $w_i sP$ for some unknown w_i , the attacker is faced with the problem of computing s from $P, sP, w_1 sP, w_2 sP, \dots$, which is at least as difficult as the DL problem.

3. It is difficult for the receiver to decrypt a ciphertext without the release key, sT . To do so, he must compute $\hat{e}(P, T)^{rs}$ from P, b, rbP, sP . By rewriting $T = wP$ for some un-

known w , we see that computing $\hat{e}(P, T)^{rs} = \hat{e}(P, P)^{wrs}$ from b, wP, sP, rbP is at least as hard as the BDH problem. If the receiver tries to find sT from $sT_i, T_i \neq T$, then by rewriting $T = w_i T_i$ the receiver has the problem of finding $sw_i T_i$ from $T_i, w_i T_i, sT_i$, which is equivalent to the CDH problem over a Gap Diffie-Hellman group. Thus, the receiver cannot decrypt a message before its release time unless he colludes with the time server.

4. A malicious server who wants to decrypt a message must compute $K = \hat{e}(P, T)^{rs}$ from s, P, bP, rbP and T . As s is known to the server, the problem can be transformed into finding $K^{s^{-1}} = \hat{e}(P, T)^r$. The last problem could be solved if one of r, rP or rT were known. However, the only available quantity that “embeds” r is rbP . Using rbP , the malicious server cannot extract the value of r . This is because if we assume that $Q = bP$ (b is unknown), then the problem of finding r from rQ is an instance of the DLP in \mathbb{G}_1 . Also, the problem of finding rP from bP and rbP is equivalent to the CDHP in \mathbb{G}_1 . Moreover, the problem of finding rT from rQ , where $Q = bP$ (r, b unknown) is at least as difficult as breaking the short signature scheme of [5]. Finally, the malicious server could try to compute the pairing $\hat{e}(rbP, T) = \hat{e}(P, T)^{rb}$ but he then needs to solve the DLP in \mathbb{G}_2 in order to produce K .

Now, using the technique from [8], [9], one can show that the plaintext security of NewTRE implies the IND-CTCA, IND-CCA security of the protocol that results by applying the FO transformation [14], as modified in [8] to account for the use of time as a parameter in the protocol.

Enhancements to NewTRE We go on to describe a series of enhancements that reinforce the security and privacy afforded by the basic NewTRE protocol.

Encrypting using multiple time-servers NewTRE can be easily modified to support multiple time servers, as in [4]. Suppose that there are N time-servers, each using a secret key s_i and a generator $P_i \in \mathbb{G}_1$, where $i = 1, \dots, N$. Then, their corresponding server public keys are $P_i, S_i = s_i P_i$, and the trapdoors are of the form $s_i T$ (where $T = H_1(t)$).

In the special case where all servers use the same generator $P_i = P$ (this could well be the case if, for example, the time-servers follow NIST recommendations for chosen ECs and generators), the additional computational burden for NewTRE is negligible. To encrypt a message m using $i = 1, 2, \dots, N$ multiple servers, a sender follows the steps of the basic single-server protocol, but computes $K = \hat{e}(\sum S_i, Q) = \hat{e}(P, T)^{r \sum s_i}$ using the common generator, P . On the receiver side, once the trapdoors ($s_i T$) are published, the receiver can also compute $K = (R, \sum s_i T) = \hat{e}(P, T)^{r \sum s_i}$.

Message Pre-Opening In a typical TRE scheme, decryption is possible only at (or after) the release time. It is reasonable, however, to expect that in some cases the sender may change her mind and may want to allow the receiver to read the confidential information immediately. One way to accomplish this would be to re-encrypt the message with a conventional

encryption algorithm (non-TRE). Such a solution has a number of drawbacks. First, it places additional computational overhead on the sender, because the whole message must be re-encrypted. Second, from the receiver’s point of view, there is no indication that the initial timed-release encrypted message and the re-encrypted one correspond to the same message. These problems highlight the need for an “embedded” mechanism that enables early decryption in a protocol. As recently as two years ago, Hwang, Yum, and Lee [19] proposed such an idea, which was later developed improved further in [12]. Briefly, the idea is to encrypt a message so that it can only be decrypted at a designated future time, or when the sender releases a kind of secondary trapdoor called a pre-open key, whichever occurs first.

Release-Time Confidentiality

Another improvement to be made vis-a-vis TRE is to provide a method for hiding the release time. This may be a requirement in a number of e-business applications, where information about the exact date an event (e.g., contract signing, or announcement) is to take place must be kept secret from competitors. In the majority of TRE protocols release-times are transmitted in the clear, and nothing can prevent an eavesdropper from observing release-times of ciphertexts. Such information could be used, for example, to find out the identity of the recipient (by waiting to see who will download a trapdoor when the release time comes). Such a threat would also hamper key privacy [2].

In [8], the authors advanced the notion of “Confidentiality of Release Time” (CRT) and proposed a mechanism through which the the release time is hidden (i.e., it is part of the ciphertext) and must be unmasked using the receiver’s private key, so that the correct trapdoor can be obtained subsequently. Existing literature lacks a “complete” secure TRE scheme that can provide pre-opening and confidentiality of release time simultaneously. Motivated by [12, 19] and [8], we have extended NewTRE to include those two features. In order to reduce the additional computational cost, one can combine the methods of [8] and [12] so that the computations performed for release-time confidentiality are re-used to calculate the pre-open key.

Confidentiality of Time-Servers Used In a multiple time-server setting, we would also like the sender to be able to hide information regarding the time-servers which she selects, for enhanced privacy and anonymity. Confidentiality of Time-Servers (CTS) can prevent malicious users from obtaining information about TRE transactions by eavesdropping the list of time-servers attached to a ciphertext and “matching” them to the trapdoors which will be downloaded later. To achieve CTS, the list of selected time-servers, $sList$, must be encrypted with a secret value which the receiver can compute immediately, without waiting for any trapdoor. As we will see, both $sList$ and the decryption time can be masked using the same secret value thus providing CTS and CRT simultaneously.

Optimized TRE scheme In the following, we describe the full

version of our protocol, named OptTRE, which arises as a modified version of NewTRE. It includes the CTS and CRT capabilities discussed above, as well as support for multiple time-servers, and makes use of the FO transformation to provide security against chosen-time chosen-ciphertext attacks. By embedding the properties discussed in Section 2.4 and by both allowing multi-server support and the modified FO transform, the optimized versatile protocol, named OptTRE, consists of the following 8-tuple of algorithms:

OptTRE.Setup(run by the time-servers): It is similar to that of NewTRE, with modifications to support multiple time-servers and to apply the FO transform.

1. Outputs a k -bit prime number q , two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q , an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ and an arbitrary generator $P \in \mathbb{G}_1$.

2. Chooses the following cryptographic hash functions: $H_1 : \{0, 1\}^\tau \mapsto \mathbb{G}_1^*$, $H_2 : \mathbb{G}_2^* \mapsto \{0, 1\}^{n+k_0+k}$, $H_3 : \mathbb{G}_1^* \mapsto \{0, 1\}^{\tau+l}$, $H_4 : \{0, 1\}^{n+k_0+\tau+l} \times \mathbb{G}_1^* \mapsto \{0, 1\}^k$, $H_5 : \{0, 1\}^{k_0} \mapsto \mathbb{Z}_q^*$.

3. Generates for each time-server the private key $s_i \in \xleftarrow{R} \mathbb{Z}_q^*$ and the corresponding public key $S_i = s_i P \in \mathbb{G}_1^*$, with $i = 1, \dots, N$, where N is the number of time servers.

4. Chooses the message space to be $m = \{0, 1\}^n$, the ciphertext space to be $C = \mathbb{G}_1 \times \{0, 1\}^{n+k_0+k} \times \{0, 1\}^{\tau+l}$, and $sList \in \{0, 1\}^l$ to be the l -bit string representation of the list of servers to be used.

The public parameters are $params := \{k, k_0, q, \mathbb{G}_1, \mathbb{G}_2, P, S_i, \hat{e}, H_1, H_2, H_3, H_4, n, \tau, l, m, C\}$.

OptTRE.ReleaseT (run by the time-servers): It is similar to NewTRE; the only difference is that this algorithm is now run by each server who is to publish his own trapdoor, $sk_{i_T} = s_i T$.

OptTRE.KeyGen (run by the receivers): As in NewTRE.

Opt.Enc (run by the senders): To encrypt $m \in \{0, 1\}^n$ using the time information $t \in \{0, 1\}^\tau$, the receiver's public key B , the list of selected time-servers, $sList$, and the servers' public keys, S_i , the sender executes the following:

1. Choose $\sigma \xleftarrow{R} \{0, 1\}^{k_0}$.
2. Compute $h = H_4(m || \sigma || t || sList || B)$.
3. Compute $r = H_5(\sigma)$.
4. Compute $T = H_1(t)$.
5. Compute $Q = rT$.
6. Compute the encryption key $K = \hat{e}(\sum S_i, Q)$.
7. Compute $c_1 = rB$.
8. Encrypt the message $c_2 = (m || \sigma || h) \oplus H_2(K)$.
9. Compute the secret value needed to hide time information and server list $R = rP$.
10. Encrypt time information and time-server list $c_3 = (t || sList) \oplus H_3(R)$.

The ciphertext is $C := \langle c_1, c_2, c_3 \rangle$.

OptTRE.ImDec (run by the receivers immediately af-

ter receiving the ciphertext): Given $C := \langle c_1, c_2, c_3 \rangle$, the appropriate trapdoors sk_{i_T} and his private key b , the recipient:

1. Computes the secret value $R = b^{-1}c_1$.
2. Unmasks time information and list of time-servers $(t || sList) = H_3(R) \oplus c_3$.
3. Computes $T = H_1(t)$.

OptTRE.PreSend (run by the sender, to allow early decryption): Given Q, R the sender

1. Computes the pre-open trapdoor $L = R + Q$. and transmits it to the receiver.

OptTRE.PreDec (run by the receiver, if and only if the sender has run the OptTRE.PreSend algorithm): Given a ciphertext C , the pre-open trapdoor L , and the value R (computed at OptTRE.ImDec), the recipient:

1. Verifies the authenticity of the pre-open trapdoor $\hat{e}(R, P + T) \stackrel{?}{=} \hat{e}(P, L)$.
2. Computes $Q = L - R$.
3. Computes the decryption key $K = \hat{e}(\sum S_i, Q)$.
4. Retrieves the message $m || \sigma || h = c_2 \oplus H_2(K)$.
5. Accepts the message if $H_4(m || \sigma || t || sList || B) \stackrel{?}{=} h$.

OptTRE.Dec (run by the receiver at the designated time): Given the ciphertext C , the trapdoors sk_{i_T} , and the value R (computed at OptTRE.ImDec), the recipient:

1. Verifies the authenticity of the trapdoors $\hat{e}(\sum S_i, T) \stackrel{?}{=} \hat{e}(P, \sum sk_{i_T})$.
2. Computes the decryption key $K = \hat{e}(R, \sum sk_{i_T})$.
3. Retrieves the message $m || \sigma || h = c_2 \oplus H_2(K)$.
4. Accepts the message if $H_4(m || \sigma || t || sList || B) \stackrel{?}{=} h$.

OptTRE is illustrated in a more convenient form in Table 2. Comparisons We go on to compare NewTRE with five of the best-known, best-performing approaches to non-authenticated, non-interactive server-based anonymous TRE. These are the BC-TRE [4], HYL-TRE [19], DT-TRE [12], CLQ-TRE³ [8], and CHS-TRE [9] protocols.

Computational Efficiency For the purposes of comparing the computational efficiency of NewTRE with that of the five protocols mentioned previously, we will ignore operations whose cost is negligible compared to that of a scalar multiplication in \mathbb{G}_1 . These include generating random numbers, integer multiplication, plain hashes and point additions in \mathbb{G}_1 . Also, because some of the protocols enable pre-computations under certain circumstances, we distinguish between three cases of anonymous TRE: i) typical message transmission to *unknown* receivers, ii) transmission to *known* receivers (in which case there is no need to verify their public keys), and iii) TRE with multiple time-servers.

Table 3 lists the cost (in *msec*) of the basic operations required to run the protocol(s), taken from [27]. The results were obtained using the *MIRACL* open-source library [29]

³Unlike this work, [8] use multiplicative notation for both groups \mathbb{G}_1 and \mathbb{G}_2 .

Table 4: Computational cost comparison sending to *unknown* receivers (in msec) (single time-server)

Protocol	Encryption	Decryption	Total
BC-TRE	$3Pa + 2Sm + 1Mtp = 104.43$	$1Pa + 1Sm = 35.15$	139.58
HYL-TRE	$1Pa + 1PSm + 2Sm + 1Mtp = 45.31$	$2Pa + 1Sm = 66.86$	112.17
DT-TRE	$1Pa + 3Sm + 1Mtp = 44.45$	$1Pa + 1Sm = 35.15$	79.6
CLQ-TRE	$2Pa + 1PSm + 1Ex = 71.65$	$1Pa + 1PSm + 1Ex = 39.94$	111.59
CHS-TRE	$1PSm + 2Sm + 1Ex = 15.11$	$1Pa + 1Sm = 35.15$	50.26
NewTRE	$1Pa + 2Sm + 1Mtp = 41.01$	$1Pa + 1Sm = 35.15$	76.16

Table 5: Computational cost comparison sending to *known* receivers (in msec)

Protocols	Encryption	Decryption	Total
BC-TRE	$1Pa + 2Sm + 1Mtp = 41.01$	$1Pa + 1Sm = 35.15$	76.16
HYL-TRE	$1Pa + 1PSm + 2Sm + 1Mtp = 45.31$	$2Pa + 1Sm = 66.86$	112.17
DT-TRE	$1Pa + 3Sm + 1Mtp = 44.45$	$1Pa + 1Sm = 35.15$	79.6
CLQ-TRE	$1PSm + 1Ex = 8.23$	$1Pa + 1PSm + 1Ex = 39.94$	48.17
CHS-TRE	$1PSm + 2Sm + 1Ex = 15.11$	$1Pa + 1Sm = 35.15$	50.26
NewTRE	$1Pa + 2Sm + 1Mtp = 41.01$	$1Pa + 1Sm = 35.15$	76.16

TRE, CHS-TRE and NewTRE all have the lowest complexity in the decryption phase. BC-TRE and CLQ-TRE are the only protocols whose cost depends on knowledge of the receiver’s public key. This is because they use a slightly different public key format, with users’ public keys consisting of two points in \mathbb{G}_1 instead of just one, as in conventional cryptographic schemes (Diffie-Hellman keys). The implication for BC-TRE and CLQ-TRE is that on the first use of any public key (for transmitting to an *unknown* receiver) the sender must verify the validity of this two-point public key, to ensure that it is properly formed and that the recipient will be able to decrypt the message. Such verification is not needed in the other four schemes.⁴

In a multiple time-server setting, NewTRE is much faster than BC-TRE (the only protocol of those surveyed that explicitly handles multiple time-servers) when all time-servers use the same generator: with N servers, the additional computation required under NewTRE is only $N - 1$ scalar additions in \mathbb{G}_1 . There is no additional communication cost, besides the unavoidable retrieval of the servers’ trapdoors. With BC-TRE on the other hand, a receiver’s public key includes $N - 1$ additional EC points due to that protocol’s more complex public key format. An examination of that protocol reveals that a sender must perform an additional $2(N - 1)$ pairings to verify the authenticity of each of an unknown receiver’s public keys, and an additional $N - 1$ scalar multiplications (these correspond to EC points which are appended to the ciphertext). The receiver must perform $N - 1$ additional pairings to compute the decryption key. If the time-servers have chosen different generators, a NewTRE receiver would need to send public keys of the form aP_i , for each different

P_i , $i = 1, \dots, N$. In that case the additional computational and communication cost of NewTRE are the same as those of BC-TRE.

It is important to note that in BC-TRE, it is the receiver who in effect chooses the time-servers, because the time-server’s public key is used to create the receiver’s public key. In our approach, a sender can use any time-server of his choice, and any number of them, without restrictions, which may increase the level of trust in transactions undertaken via NewTRE and OptTRE.

Communication Cost The protocols’ communication complexity depends on the bit-length of the transmitted public keys and the ciphertext space. As mentioned previously, in BC-TRE and CLQ-TRE the users’ public keys consist of two EC points, while the rest use a single Diffie-Hellman public key. Consequently, if the recipient is an *unknown* entity, the cost to download the recipient’s public key from a public database for BC-TRE and CLQ-TRE is twice that of the HYL-TRE, DT-TRE, CHS-TRE and NewTRE schemes.

As for the ciphertext space, all of the TRE schemes under comparison require an EC point and a ℓ -bit string to be transmitted; the HYL-TRE scheme requires an additional EC point and pairing value, and both the DT-TRE and CHS-TRE require an additional EC point. The value of ℓ is the same for all schemes; it is either $\ell = n + \tau$ or $\ell = n + \tau + q$, depending on whether or not the FO transformation [14] is applied, where n is the length of the cleartext message, q is the security parameter (e.g., 160-bits) and τ is the bit-length of the string used to represent a time instant. We emphasize that in order to provide sufficient security, an EC point must be represented using at least 160 bits; a pairing value should be allotted approximately 1024 bits. To summarize, NewTRE has the lowest communication cost, while HYL-TRE has the highest one, in large part because of the pairing value being transmitted.

⁴When comparing the cost of implementations of the above schemes, we did not include the cost of a group membership test for the public keys. If that were taken into account, [4, 8] would require some additional checking because of the use of “two-point” public keys.

Conclusions We have presented a new CCA and CTCA-secure, anonymous TRE protocol, inspired by that in [4], and compared it to other well-known TRE schemes from the recent literature. Ours has the second-lowest computational cost when sending data to unknown receivers, and is as good as [19], [9] and [4] in decryption cost for both known and unknown receivers. In terms of communication cost, our protocol has an advantage overall, because it combines the best features of [9, 12, 19] (simple public-key format) and [4, 8] (small ciphertext space), together in one scheme. Our approach offers excellent scalability in multiple time-server settings. In particular, if all time-servers employ the same group generator, the additional cost is negligible (one scalar addition per additional server). The extended version of our protocol also includes message pre-opening capabilities, and provides release-time confidentiality, and confidentiality of time-servers used.

References

- [1] P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology CRYPTO 2002, LNCS 2442*, pp. 354 - 368. Springer-Verlag, 2004.
- [2] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology - ASIACRYPT 2001, LNCS 2248*, p. 568-584. Springer, 2001.
- [3] M. Bellare and S. Goldwasser. Encapsulated key-escrow. In *Technical Report MIT/LCS/TR-688*, 1996.
- [4] I. F. Blake and A. C.-F. Chan. Scalable, server-passive, user-anonymous timed-release cryptography. In *25th IEEE Int'l. Conf. on Distributed Computing Systems*, pp. 504-513. IEEE Computer Society, 2005.
- [5] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology EUROCRYPT 2005, LNCS 3494*, pp. 440-456. Springer-Verlag, 2005.
- [6] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in Cryptology - CRYPTO '01, LNCS 2139*, pp. 213-229. Springer-Verlag, 2000.
- [7] R. Canetti, S. Halevi, and J. Katz. A forward secure public key encryption scheme. In *Advances in Cryptology - EUROCRYPT '03, LNCS 2656*, pp. 254-271. Springer-Verlag, 2003.
- [8] J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In *Intl. Conf. on Information and Communications Security, LNCS 3783*, pp. 291-303. Springer-Verlag, 2005.
- [9] K. Chalkias, D. Hristu-Varsakelis, and G. Stephanides. Improved anonymous timed-release encryption. In *12th European Symp, On Research In Computer Security, ESORICS, LNCS*, vol. 4734, pages 311-326. Springer, 2007.
- [10] K. Chalkias and G. Stephanides. Timed-release cryptography from bilinear pairings using hash chains. In *10th IFIP CMS*, pp. 130-140. Springer-Verlag, 2006.
- [11] I. Damgard. Practical and provably secure release of a secret and exchange of signatures. In *Advances in Cryptology - EUROCRYPT '93, LNCS 765*, pp. 200-217. Springer-Verlag, 1994.
- [12] A. W. Dent and Q. Tang. Revisiting the security model for timed-release public-key encryption with pre-open capability. In *Cryptology ePrint Archive: Report 2006/306*, 2006.
- [13] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Advances in Cryptology - EUROCRYPT '99, LNCS 1592*, pp. 74-89. Springer-Verlag, 1999.
- [14] E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In *PKC '99, LNCS 1560*, pp. 53-68. Springer-Verlag, 1999.
- [15] S. Galbraith, H. K., and S. D. Implementing the Tate pairing. In *Algorithmic Number Theory Symposium ANTS V, LNCS 2369*, pp. 324 - 337. Springer-Verlag, 2002.
- [16] J. Garay and M. Jakobsson. Timed-release of standard digital signatures. In *Financial Cryptography '02, LNCS 2357*, pp. 168-182. Springer-Verlag, 2002.
- [17] N. Haller. The s/key one-time password system. In <http://www.rfc-archive.org/getrfc.php?rfc=1760>, 2005.
- [18] D. Hristu-Varsakelis, K. Chalkias, and G. Stephanides. Low-cost anonymous timed-release encryption. In *Information Assurance and Security*, pp. 77-82. IEEE Computer Society Press, 2007.
- [19] Y. H. Hwang, D. H. Yum, and P. J. Lee. Timed-release encryption with pre-open capability and its application to certified e-mail system. In *Information Security Conf., LNCS 3650*, pp. 344-358. Springer-Verlag, 2005.
- [20] J. Killian. Basing cryptography on oblivious transfer. In *Proc. of STOC*, pp. 20-31, 1988.
- [21] W. Mao. Timed-release cryptography. In *Selected Areas in Cryptography 2001, LNCS 2259*, pp. 342-357. Springer-Verlag, 2001.
- [22] T. May. Timed-release crypto. In *Manuscript*, <http://www.hks.net/cpunks/cpunks-0/1460.html>, 1993.
- [23] R. C. Merkle. Secure communications over insecure channels. In *Communications of ACM, 21(4)*, pp. 294-299, 1978.
- [24] V. S. Miller. The Weil pairing, and its efficient calculation. In *Journal of Cryptology, volume 17*, pp. 235 - 261, 2004.
- [25] M. C. Mont, K. Harrison, and M. Sadler. The HP time vault service: Innovating the way confidential information is disclosed at the right time. In *Intl. World Wide Web Conf.*, pp. 160-169. ACM Press, 2003.
- [26] D. Nali, C. Adams, and A. Miri. Time-based release of confidential information in hierarchical settings. In *Information Security, LNCS 3650*, pp. 29-43. Springer-Verlag, 2005.
- [27] I. Osipkov, Y. Kim, and J.-H. Cheon. Timed-release public key based authenticated encryption. In <http://eprint.iacr.org/2004/231>, 2004.
- [28] R. Rivest, A. L. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. In *MIT Laboratory for Computer Science Technical Report 684*. Massachusetts Institute of Technology, 1996.
- [29] L. Shamus Software. Miracl - multiprecision integer and rational arithmetic C/C++ library. In <http://indigo.ie/mscott>, 2006.