Scalable, Server-Passive, User-Anonymous Timed Release Cryptography

Aldar C-F. Chan
University of Toronto, Canada
aldar@comm.utoronto.ca

Ian F. Blake
University of Toronto, Canada
ifblake@comm.utoronto.ca

Abstract

We consider the problem of sending messages into the future, commonly known as timed release cryptography. Existing schemes for this task either solve the relative time problem with uncontrollable, coarse-grained release time (time-lock puzzle approach) or do not provide anonymity to senders and/or receivers and are not scalable (server-based approach). Using a bilinear pairing on any Gap Diffie-Hellman group, we solve this problem by giving scalable, server-passive and user-anonymous timed release public-key encryption schemes allowing precise absolute release time specifications. Unlike the existing server-based schemes, the trusted time server in our scheme is completely passive — no interaction between it and the sender or receiver is needed; it is even not aware of the existence of a user, thus assuring the privacy of a message and the anonymity of both its sender and receiver. Besides, our scheme also has a number of desirable properties including a single form of update for all users, self-authenticated time-bound key updates, and key insulation, making it a scalable and appealing solution. It could also be easily generalized to a more general policy lock mechanism.

1. Introduction

The idea of "sending information into the future", that is, encrypting a message so that it cannot be read (or decrypted) by anyone, including the designated recipients of the message, until a pre-determined, specified "release time" instant (chosen by the sender) is called timed release encryption. This problem was first discussed by May [15] in 1993 and further elaborated by Rivest et. al. [19]. Besides, the coordinated bit reveal of Beaver and So's [1] distributed unpredictable bit generator could also be considered as a time-lock mechanism of the generated bit.

When considering the notion of specified time, we need to distinguish between relative time (the amount of time between events, say one hour from the last information exchange) and absolute time (the exact time output of a universal common reference, say 2:00AM, January 18, 2003 GMT from the Denver Atomic Clock used in Global Positioning System (GPS)). In the context of this paper, we mainly consider the latter that is more interesting since the relative time specification could be implemented with the absolute time specification but not the reverse.

Since time is a critical aspect of many applications in distributed computing and networks, timed release encryption has several interesting real-world applications. In fact, there are many applications which depend on a common assumption of an absolute time reference in the future, that is, opening a document or proving the authenticity of a statement before a specified time is not allowed. An example is a sealed bid in which a bidder wants to seal his bid for a government tender so that it cannot be opened before the bidding period is closed so as to avoid anyone (say one of the government agents handling the bids) disclosing the information of his bid to his competitors who can gain advantage through this information. Another example is the Internet programming contest where teams located all over the world can only be granted access to the challenge problems at a certain time. As can be seen from these two examples, the essence of the timed release encryption problem is: a message has to be sent earlier prior to its desired release time and we need to ensure that it cannot be read before that moment. In the Internet programming contest example, we need to ensure that every participating team can access the problems when the contest starts; to avoid fairness issues arising from uncontrollable[1] network congestion or delivery delay, we want to ensure that every team has received the problem set well before the contest starts but cannot open it. This is the problem timed release encryption addresses.

Since its introduction, timed release encryption has been found useful in a number of scenarios, including electronic auctions, key escrow, chess moves, release of documents (like memoirs) over time, payment schedules, press releases and etc.. Bellare and Goldwasser [2] proposed the use of timed release encryption in key escrow; they suggested that

[1] Compared to the whole message, a timely delivery of the timing reference/update (within a reasonably small delay jitter bound) could be more easily achievable.

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

the delayed release of escrow keys may be a suitable deterrent in some contexts to the possible abuse of escrow.

Since the problem was posed in 1993, there have been a number of proposals for timed release encryption schemes, based on two approaches — time-lock puzzles [2, 19, 14, 6, 12] and trusted servers [15, 19, 4, 17]. However, none of these are fully satisfactory. Although no trusted server is needed, schemes based on the time-lock puzzle approach could only solve the relative time problem with a coarse-grained approximate release time, dependent on the speed of the recipients' machines and when the decryption is started. That is, they could only guarantee that a receiver cannot retrieve a certain message from its ciphertext for at least a certain minimum amount of time since he starts decrypting it; neither could they guarantee that the message can be retrieved immediately after the sender's desired release time has passed (if the recipient does not start decrypting the message immediately upon receiving it or he uses a machine slower than what the sender expected). Besides, the time-lock puzzle approach could take up considerable computational resources for decryption. Hence, in general these schemes (based on the time-lock puzzles) would be impractical. On the other hand, the existing schemes using trusted servers require interaction between the server and the sender or the receiver of a message, or even both. These schemes sacrifice the anonymity of users for solving the absolute time problem with a precise release time implementation. This interaction leaks out to the server the identities of the senders and/or the receivers, and sometimes the server would even know the release time and the content of all messages [15]. In addition, the fact that the server is actively involved in the encryption or decryption of every message limits the scalability of these schemes.

It is thus fair to say that there does not exist a satisfactory solution to the problem of constructing a scalable, non-interactive and user-anonymous timed release encryption scheme, in which any encrypted messages could only be opened by the designated recipients at or after a precisely specified absolute release time (that can possibly be in the infinite future), as indicated by a common time reference server, and neither the sender nor the recipient of a message needs to interact with that server (which is completely passive) while encrypting or decrypting messages.[2] Based on a bilinear pairing over any Gap Diffie-Hellman group, we give a scalable solution. The main contribution of this paper is the model of a completely passive time reference server in timed release cryptography and the demonstration of its feasibility through designing scalable, server-passive and user-anonymous schemes which allow a precise absolute release time specification. Although we could use the hybrid cryptography paradigm to combine any public key encryption with an identity based encryption to achieve this goal,[3] the resulting constructions are considerably less efficient than our schemes in terms of computation and/or ciphertext size. Our schemes could have 50% reduction in most cases.

In our schemes, the time server merely provides a common absolute time reference, in the form of a sequence of signed messages called time-bound key updates (released/published when the referenced time instants come), and does not interact with either the sender or the receiver. Unlike the offline version of the Rivest's scheme [19], it does not need to pre-establish or remember any information of key updates for time instants in the future[4]. Neither does it need to remember any keys or information about the senders or receivers. In fact, the server would not even be aware of the existence of a sender or receiver unless queried by one of them (which is not necessary in most scenarios). The anonymity of the sender and receiver of a message and the privacy of the message and its release time are thus guaranteed. Our schemes are provably secure under the intractability assumption of the bilinear Diffie-Hellman (BDH) problem over any Gap Diffie-Hellman group in the random oracle model [3]. Besides, our first encryption scheme has a number of nice additional properties, rendering it an efficient and scalable solution to the timed release encryption problem requiring precise release time and complete recipient privacy. These include:

- The time-bound key update needed for the decryption at a particular release time is identical for all receivers; regardless of the number of receivers, the time server just need to publish/broadcast a single update enough for all receivers to recover their messages, thus making the scheme scalable.
- The key update published by the time server is a short signature inherently authenticating itself. Thus, no additional overhead of a server signature is needed.
- Certifying the authenticity of a receiver's public key can be done by a separate CA (Certificate Authority) not related to the time server. Although the public key of a receiver is tied to the time server public key, a

[2] Although the server in the offline version of the Rivest's scheme [19] is passive and is not involved in any encryption or decryption, it needs to (periodically) publish well in advance a long list of public keys for epochs in the future. This scheme is not scalable as a sender has to wait for the server to publish a public key (corresponding to his chosen release time) not in the existing list. In contrast, a sender in our scheme could choose any release time in the (possibly infinite) future at his own will without relying on any information from the server and the server only needs to publish information (for receivers) whose corresponding time has passed.

[3] We could use a public key encryption scheme to encrypt a sub-key K_1 and use an identity based encryption scheme to encrypt another sub-key K_2. These two sub-keys are then combined to feed into a symmetric key encryption scheme for encrypting the actual messages.

[4] In fact, the server does not need to remember the information of any key updates since it can generate a key update for any particular instant directly using its private key.

change of time servers does not need re-certification of the new public key.

- With slight modifications, our scheme inherently satisfies the key insulation property.
- Our scheme could be used as a more general policy-lock encryption scheme.

We discuss the previous work on timed release encryption in the next section. Then we present our model and the preliminaries needed in Section 3 and 4 respectively. In Section 5, we give our constructions and discuss their properties. Finally, we conclude with a discussion of future work on the resilience against missing updates.

2. Related Works

There are two main approaches adopted in the previous work, one based on the so called "time-lock puzzles" and the other on a trusted server.

2.1. Time-Lock Puzzle Approach

Time-locked puzzles were first suggested by Merkle [16] and extended by Bellare et. al. [2] and Rivest et. al. [19]. The idea is that a secret is transformed in such a way that any machine (serial or parallel), running continuously, takes at least a certain amount of time to solve the underlying computational problems (puzzles) in order to recover the secret. This minimum amount of time is the relative release time with respect to the start of solving the puzzle and could be different for different machines.

In [2], Bellare and Goldwasser presented a time-lock puzzle based on the heuristic assumption that exhaustive search on the key space is the fastest method to recover the key of DES. However, Rivest et. al [19] pointed out that this only works on average since for a particular key, exhaustive search may find the key well before the assigned time. Rivest et. al. then proposed another time-lock puzzle based on the hardness of factoring which does not have this problem. In their puzzle, if factoring is difficult, repeated squaring mod n (where $n = pq$) is the fastest method to recover the secret. Using a function similar to this time-lock puzzle, a number of extensions for timed applications were introduced [6, 14, 12]. Mao [14] added a zero-knowledge proof to it and constructed a timed release RSA signature. Boneh and Naor [6] introduced the notion of (verifiable) timed commitments. Based on timed commitments, Garay et. al. [12] constructed timed release signatures.

Although elegant in the complexity theoretic sense, the time-lock puzzle approach is impractical, consumes a large amount of computational resources and lacks flexibility. Since time-lock puzzles try to make "CPU time" and "real time" agree as closely as possible, it can only solve the relative time problem (with reference to the start of solving the puzzle) with an approximately controllable time (different machines work at different speeds) and the puzzle does not automatically become solvable at a given time (if solving is not started immediately upon receipt). If absolute and precise timing of information release is essential, like the sealed bid scenario mentioned above, the approach based on a trusted time server is inevitable.

2.2 Trusted Server Approach

In order to support precise release time, an absolute or common time reference is necessary to synchronize the senders and receivers. Hence, the need of a trusted time server to provide this common reference is inevitable. Although inevitable, the time server should have as little involvement in the users' communication as possible, ideally with no interaction, to ensure scalability and anonymity. The time server should merely provide a common time reference for users by periodically releasing unforgeable time-embedded information that is necessary for decrypting timed release ciphertexts. However, none of the existing schemes could efficiently satisfy this requirement.

May [15] suggested that a third party can be used as a trusted escrow agent to store messages from senders and release them to the receivers at a specified release time. This does not scale well as the agent has to store all escrowed messages until their release time. Moreover, no anonymity is guaranteed because the server knows the message, its release time, and the identities of the sender and receiver.

Rivest et. al [19] used a combination of symmetric key and asymmetric key encryption to solve the problem. In their scheme, a server has a sequence of keys used for a symmetric key encryption like DES and releases them periodically. As the sequence of keys could be generated from a one way function, the server does not have to remember anything except the seed. The major disadvantage of this scheme is the sender has to interact with the server and give it his message, hence, his identity as well as his message and its release time are known to the server (anonymity guaranteed for receivers only). This interaction also limits its scalability. To eliminate this interaction, the authors suggested that the symmetric key encryption could be replaced by a public key encryption; however, the problem is the server has to publish in advance a huge number of public keys (corresponding to time instants in the future), whose private keys will be released in the future, or the senders cannot freely choose the desired release time of their messages. Hence, it is not scalable. Besides, a considerable amount of computation is also needed for encryption and decryption.

The distributed unpredictable bit generator in [1] has an implicit coordinated reveal property, namely, the consent

from a certain minimum fraction of the involved parties is needed before the generated bit is revealed to all. However, when used for timed release encryption, interaction between the sender and time server seems to be necessary.

While interaction between the sender and server is needed in [19], Di Crescenzo et. al. [9] proposed a scheme in which interaction is needed between the receiver and the server only. In their scheme, the receiver has to run a conditional oblivious transfer with the server, in which they engage in a multiple-round, interactive, and private conversation to evaluate the public predicate whether the release time is less than the server's current time. If it is true, the receiver gets the message, otherwise, he gets nothing. In addition, the server does not learn any information about the identity of the sender, the message and its release time; in particular the server does not learn whether the release time is less than, equal to, or greater than the current time. This protocol has a logarithmic complexity in the time parameter. However, the necessity of interaction between the server and the receiver makes the protocol not scalable and subject to denial of service attacks[5], and have no guarantee on the receiver's anonymity.

In all the schemes discussed so far, the information sent by the server is not inherently authenticated; the server needs to sign them. When proposing the use of bilinear maps to construct identity based encryption (IBE), Boneh and Franklin [4] named time released encryption as one of its applications. Then Mont et. al. [17] implemented their idea. In this scheme, a sender uses a receiver's identity augmented with a release time as the latter's public key (for that specified time instant) to encrypt messages. The server will give the receiver his private keys when the corresponding release time instants come. Although no certification of the receiver's public key is needed, this scheme is not scalable if the time granularity is small since the server needs to generate and individually transmit to each receiver his secret key at the start of each time epoch. Besides, the server could decrypt all the messages.

When investigating the application of the additive property of a bilinear pairing for a multiple trusted authority implementation in identity-based encryption, Chen et. al. [8] suggested that one of the authorities could be used for binding time information so as to give a timed press release application. This could be considered as an identity-based timed release encryption scheme. As independent work, we have come up with the same construction; it was later determined that it uses the idea in [8]. The main gain is the very desired property of using a single key update for all receivers. As in any identity based schemes, identity based timed release encryption could only provide moderate security (even though multiple servers could be used to lower the risk) since the server would be able to decrypt any messages for any users; key escrow is inherent. Constructing a non-identity based timed release encryption scheme seems to remain unknown and we will give one in this paper in which only a receiver would be able to know the decryption keys of the messages sent to him and nobody else, providing the highest possible privacy.

In [13], the notion of an oblivious envelop is proposed, which is in essence an encryption lock requiring a server's signature to unlock. In this sense, our scheme can be considered as an encryption, having an extra lock layer, which requires the receiver's private key, in addition to the time server's signature, to unlock.

[5] An adversary could keep querying the server with a ciphertext whose release time is in the very far future, hence overloading the server. Since the protocol is designed not to allow the server know the exact release time, the server cannot detect this kind of denial of service attacks.

3 Our Model

As can be seen, the existing techniques using a trusted time server either require interaction between the server and the sender [19] or receiver [9], or let the server know and deliver all the secret keys [4] or messages [15]. In some cases, there may just be a unidirectional channel between the server and a user, thus making these schemes (requiring interaction) out of function or impractical. As a result, we only consider techniques without requiring any online interaction. As usual, if the sender of a message is available at the release time, the solution to the timed release encryption problem is trivial. Therefore we assume that the sender is not available at the release time and has to send out the message before that.

In our model, we consider a passive server, that is, the server does not interact with either the sender or the receiver. Neither does it need to remember any keys or information about the senders or receivers. In fact, the server would not even be aware of the existence of a sender or receiver unless queried for time information (which is not necessary in most scenarios). Our model is analogous to the GPS scenario in which the satellites or the control center, where the Denver Atomic Clock is placed, are not aware of how many GPS receivers exist on earth while providing a precise time to them, and groups of users, based on this timing information from the GPS, can coordinate or synchronize their tasks (without the involvement of the satellites or the control center) while dispersed over the earth. What the server needs to do in our schemes is just merely to periodically output and publish/broadcast a certified piece of information, the time-bound key update I_t, which indicates the current time t (down to whatever granularity is needed), and to keep a list of old key updates (whose release time has passed) at a publicly accessible place. The fact that the server does not need to remember any information about the senders or receivers in our model implies that

the time-bound key update I_t is identical for all users for a particular t. That is, our model achieves the most desirable property that the server just needs to broadcast a single I_t for all receivers for a particular t.

When a sender wants to send a message with a certain release time, he just uses the public keys of the receiver and the server to encrypt the message and the release time in such a way that both the receiver's private key and the corresponding I_t are necessary for decryption. Upon receiving a timed release encrypted message, the receiver is usually very curious of its content and would wait (in alert) the release of the corresponding time-bound key update from the server. Once the update I_t (identical for all users) is published, everyone using the server's public key can verify the authenticity of I_t and if needed plug it into computation to decrypt a message (sent to him) with release time at t. Since the server's private key is unknown, nobody can create a I_t for an arbitrary time even after observing other values of $I_{t'}$ unless $t = t'$. In case a receiver has missed a particular key update, he could still look up from the list of old key updates to get the right update to decrypt a message whose release time has passed.

The only trust we assume on the server is that it outputs a consistent absolute timing and does not give out any time-bound key updates I_t before its release time. The first assumption means that if the server now outputs "02:01:01 AM JAN 18, 2003 GMT", an hour later (according to an accurate timing device, say a cesium clock) it should output a time within a reasonable error bound of "03:01:01 AM JAN 18, 2003 GMT". The second assumption means that the server should not give out any I_t at an instant t' where $t' < t$. From our viewpoint, these two are reasonable and easily achievable assumptions.

Putting pieces together, we can now state the problem.

Timed Release Encryption (TRE) Problem: How can a sender, without talking to the time server, encrypt a message with a release time (defined using the notion of time marked by the server) in the future using a receiver's public key (as well as the time server's public key) such that the receiver can decrypt this message with his private key only after the release time has passed as indicated by a signed piece of information on the current time (i.e. time-bound key update) published by the time server that would learn nothing about the identities of the two parties, the messages or its release time?

Besides, in a secure and private timed release public key encryption scheme, only the intended receiver holding the corresponding private key and at a time instant after the specified release time (enforced by a trusted time server) could recover a secret. To achieve the highest possible privacy, even the trusted authority or time server should not be able to decrypt a message sent to any users[6].

[6]Identity based encryption schemes do not satisfy this requirement because the server also possess a user's private key.

4 Preliminaries

Throughout this paper, we will use the following notations, definitions and computational assumptions.

We denote the set of all binary strings by $\{0,1\}^*$, the finite field of integers modular q (where q is a prime) by $\mathbb{Z}_q$, and $\mathbb{Z}_q \backslash \{0\}$ by $\mathbb{Z}_q^*$. Below are the computational assumptions we have used in this paper. Let $\mathbb{G}_1$ be a cyclic additive group whose order is a prime q and $\mathbb{G}_2$ be a cyclic multiplicative group with the same order q.

Definition 1 Discrete Log (DL) Problem in $\mathbb{G}_2$
Given $g_1, g_2 \in \mathbb{G}_2$, find an integer $a \in \mathbb{Z}_q^*$ such that $g_2 = g_1^a$ whenever such an integer exists.

Definition 2 Discrete Log (DL) Problem in $\mathbb{G}_1$
Given $P, Q \in \mathbb{G}_1$, find an integer $a \in \mathbb{Z}_q^*$ such that $Q = aP$ whenever such an integer exists.

Definition 3 Decisional Diffie-Hellman (DDH) Problem in $\mathbb{G}_1$
Given $P \in \mathbb{G}_1$, aP, bP and cP for some unknowns a, b, $c \in \mathbb{Z}_q^*$, decide whether $c \equiv ab \pmod q$.

Definition 4 Computational Diffie-Hellman (CDH) Problem in $\mathbb{G}_1$
Given $P \in \mathbb{G}_1$, aP and bP for some unknowns a, $b \in \mathbb{Z}_q^*$, find abP.

Definition 5 Bilinear Pairing
A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties:

1. **Bilinearity:** $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ *for all* $P, Q \in \mathbb{G}_1$, $a, b \in \mathbb{Z}_q^*$.
2. **Non-degeneracy:** $\hat{e}(P, Q) \neq 1$ *unless P or Q is an identity element of* $\mathbb{G}_1$.
3. **Computability:** *There is an efficient algorithm to compute* $\hat{e}(P, Q)$.

Definition 6 Bilinear Diffie-Hellman (BDH) Problem
Given $P \in \mathbb{G}_1$, aP, bP and cP for some unknowns a, b, $c \in \mathbb{Z}_q^*$, find $\hat{e}(P, P)^{abc}$.

With carefully chosen groups, the DL problem is usually assumed to be difficult for most instances (DL assumption). Both the decisional and computational Diffie-Hellman problems are also assumed difficult in such groups; however, if a bilinear pairing exists in the underlying additive group, the decisional Diffie-Hellman problem over it can be solved in polynomial time by testing whether $\hat{e}(aP, bP) = \hat{e}(P, cP)$. This leads to the Gap

Diffie-Hellman (GDH) Assumption that for a certain additive group $\mathbb{G}_1$, the decisional Diffie-Hellman problem on it can be solved in polynomial time but there is no polynomial time algorithm to solve the computational Diffie-Hellman problem (with the help of an oracle who can solve the decisional Diffie-Hellman problem) with non-negligible probability. The additive group $\mathbb{G}_1$ is called a Gap Diffie-Hellman group which can be found in supersingular elliptic curves over a finite field, with the bilinear pairing derived from a Weil or Tate pairing.[7] The bilinear Diffie-Hellman problem is usually assumed to be difficult and is the basis of the security of our schemes. It is obvious that solving the DL or GDH problem over $\mathbb{G}_1$ implies solving the BDH problem. Besides that, the DL problem over $\mathbb{G}_2$ should also be hard so that the DL problem over $\mathbb{G}_1$ is not easily solved. Hence, we need the assumption that BDH problem is hard. As long as the BDH assumption holds, our schemes are secure in the random oracle model [3].

5 Timed Release Public Key Encryption Constructions

In this section, we will describe a simple construction of timed release encryption based on bilinear maps. The security of this scheme is based on the hardness of the Bilinear Diffie-Hellman (BDH) Problem over a Gap Diffie-Hellman (GDH) group. These constructions are one-way encryption [4] semantically secure against chosen plaintext attacks in the random oracle model [3]. For the sake of clarity and simplicity, the Fujisaki-Okamoto Transform [11] has not been applied in the following discussion. Similar to the technique in [4], this transform can be applied to our schemes to obtain chosen-ciphertext secure schemes. Alternatively, the REACT conversion introduced by Okamoto and Pointcheval [18] could be used instead.

5.1 Timed Release Encryption (TRE)

Suppose $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive and multiplicative cyclic groups of order q (prime) respectively and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a computable, non-degenerate bilinear map. Given the following cryptographic hash functions:

$$H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1, \qquad H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$$

where n is the size of plaintext. The TRE scheme runs as follows.

Server Key Generation: The time server randomly picks a generator of $\mathbb{G}_1$, say G, and a private key $s \in \mathbb{Z}_q^*$, and computes the public key sG. G and sG are made public. The public key of the server is $PK_{\mathcal{S}} = (G, sG)$

[7] They motivate our choice of notating the image in multiplicative notation and the preimage in additive notation.

User Key Generation: Each user picks a secret key $a \in \mathbb{Z}_q^*$ and computes the public key $PK_{\mathcal{U}} = (aG, asG)$.

The secret key a could be generated by applying a good hash function to a human-memorable password chosen by the user. (*Note that the public key here is not directly derived from the user's identity; a CA type of certification is still needed.*)

Time Server Broadcast: At a time instant $T \in \{0,1\}^*$, the time server publishes a time-bound key update of the form $sH_1(T)$. Every user can verify its authenticity[8] by checking $\hat{e}(sG, H_1(T)) = \hat{e}(G, sH_1(T))$, where (G, sG) is the server's public key.

At a particular instant t as described by a string T, the server sends out to every user its signature on the string T, that is $sH_1(T)$, to certify it is now instant t.

Encryption: Given a message M, a receiver public key (aG, asG), a server public key (G, sG), and a release time $T \in \{0,1\}^*$,

1. Verify that $\hat{e}(aG, sG) = \hat{e}(G, asG)$; proceed with the encryption only if this is true.

   The verification is to ensure that the receiver's public key is of the form $a \times sG$ so that he really needs the server's timed release information (the time-bound key update) to decrypt the message.

2. Randomly pick $r \in \mathbb{Z}_q^*$ and compute rG and $rasG$.

3. Compute:
   $K = \hat{e}(rasG, H_1(T)) = \hat{e}(G, H_1(T))^{ras}$.

4. Then the ciphertext is: $C = \langle rG, M \oplus H_2(K) \rangle$.

Decryption: Given a ciphertext $C = \langle U, V \rangle$, a receiver's private key a, and the needed time-bound key update $\sigma_{\mathcal{S}}(T) = sH_1(T)$ from the server,

1. Compute:$K' = \hat{e}(U, \sigma_{\mathcal{S}}(T))^a$.

2. Recover M by computing $V \oplus H_2(K')$.

If C is a correct ciphertext with release time T, then $U = rG$ and $V = M \oplus H_2(K)$ with $K = \hat{e}(G, H_1(T))^{ras}$. The decryption works because:

$$K' = \hat{e}(U, \sigma_{\mathcal{S}}(T))^a = \hat{e}(rG, sH_1(T))^a$$
$$= \hat{e}(G, H_1(T))^{ras} = K$$

and $V \oplus H_2(K') = M \oplus H_2(K) \oplus H_2(K) = M$.

[8] Note that the authenticity of a time-bound key update is implicitly provided by the content itself, and the time server does not need to generate an additional signature for this purpose.

A Sketch of Security Proof for TRE

1. Given G, sG, it is difficult to find s (DL problem). Hence, the server's private key s should be safe.

2. Given (G, sG) and (aG, asG), it is difficult to find a. The argument is as follows: Suppose we have a polynomial time algorithm $\mathcal{A}_1(G, sG, aG, asG) = a$ which solves the above problem, we can use it to solve the DL problem in the following way: Given G and aG, we randomly pick a b and can easily compute bG and baG (which is equal to abG); using $\mathcal{A}_1()$, we can find $a = \mathcal{A}_1(G, bG, aG, abG)$. So this problem is at least as difficult as the DL problem. Hence, given the public keys, the user's private key a should be safe.

3. Rewriting any $sH_1(T_i)$ as $w_i sG$ (for some unknown w_i) and using the same argument as item 2, the problem of finding s from $\{G, sG, sH_1(T_1), sH_1(T_2), \ldots\}$ is at least as difficult as the DL problem. The server's private key s is thus safe.

4. Now, suppose a cheating receiver attempts to find $sH_1(T)$ from $sH_1(T_i)$, $T_i \neq T$. If we rewrite $H_1(T)$ as $w_i H_1(T_i)$, the problem becomes to find $sw_i H_1(T_i)$ from $H_1(T_i)$, $w_i H_1(T_i)$, $sH_1(T_i)$, which is equivalent to the Computational Diffie-Hellman Problem over a Gap Diffie-Hellman group. That is, finding a particular key update from another should be hard. However, this does not rule out the possibility that a list of key updates, say $\{sH_1(T_1), \ldots, sH_1(T_i), \ldots\}$, could help decrypt a ciphertext having a different release time T. We show in the appendix that this possibility could be neglected if we consider $H_1()$ as a random oracle.

5. A rough proof of the time locking property[9] could be as follows. Suppose we take $H_1()$ as a random oracle.[10] Hence, we could assume that a cheating receiver cannot exploit the knowledge about other key updates and from his viewpoint, $H_1(T)$, for a given release time T, is just an arbitrary point in $\mathbb{G}_1$ (The proof can be found in the appendix.). If another hash function $H_2()$ is used to match the length of the pairing to that of the message, it is also modeled as a random oracle.

   The challenge for a cheating receiver is to decrypt, without the corresponding key update $sH_1(T)$, a ciphertext with release time T not of his choice. In order to decrypt this ciphertext, he needs to compute $\hat{e}(G, H_1(T))^{ras}$ from the information he knows, namely rG, sG and a. Suppose we have an algorithm $\mathcal{A}_2()$ able to achieve this goal, that is, $\mathcal{A}_2(rG, sG, H_1(T), a) = \hat{e}(G, H_1(T))^{ras}$. We could use $\mathcal{A}_2()$ to solve the bilinear Diffie-Hellman problem as follows. Recall that the BDH problem is to find $\hat{e}(G, G)^{xyz}$ from xG, yG, and zG. We could set $H_1(T) = zG$ and $a = 1$ and give them together with xG and yG as a problem instance to $\mathcal{A}_2()$. Then, $\mathcal{A}_2(xG, yG, zG, 1) = \hat{e}(G, zG)^{xy} = \hat{e}(G, G)^{xyz}$. That is, we could use $\mathcal{A}_2()$ as a subroutine to solve the BDH problem.

   Hence, as long as the BDH problem is difficult, the receiver, even with his private key, cannot decrypt an encrypted message before its specified release time (i.e. without the needed time-bound key update from the time server) unless he colludes with the time server.

6. In TRE, the secrecy of messages of a receiver is guaranteed and it is rare that the time server can decrypt a significant amount of encrypted messages sent to a receiver. The time server could only cheat at the very beginning by giving all users its chosen generator of the form $G = H_1(T)$ for some T at which it wishes to eavesdrop messages; hence, the probability of successful wide scale eavesdropping is still negligible. In fact, this concern could be eliminated if a sender can avoid using T that would give $H_1(T) = G$. There should not be a large difference, from the sender's point of view, between using T and using T plus one second, but the resulting two $H_1()$ images could be very different.

[9] The time locking property ensures that a user cannot retrieve a message before receiving the key update of the desired release time.

[10] When we assume $H_1()$ is a random oracle, an adversary cannot evaluate it by himself but is forced to access an oracle to find any $H_1(T)$'s.

5.2 Identity-based Timed Release Encryption (ID-TRE)

This scheme is essentially the idea used in [8]. For the sake of simplicity, in the following discussion, the time server is the same entity as the trusted server assigning private key to users in an ID-based encryption scheme [4]; in real cases, it could be a different entity. Using the same notations as in TRE, the ID-TRE protocol runs as follows.

Sever Key Generation: The server randomly picks a generator of $\mathbb{G}_1$, say G, and then picks $s \in \mathbb{Z}_q^*$ as his private key and computes sG. G and sG are made public.

User Key Generation: For a user i with ID_i, the server computes and gives him $sH_1(ID_i)$, to be used as his private key, and his public key is his identity ID_i.

Time Server Broadcast: The time server periodically publishes a signed piece of information $sH_1(T)$ (the time-bound key update), indicating the current time $T \in \{0,1\}^*$. Its authenticity is inherent as in TRE.

Encryption: Given a message M, a receiver identity ID_i and a release time $T \in \{0,1\}^*$,

1. Compute $K_E = H_1(ID_i) + H_1(T)$
2. Pick a random $r \in \mathbb{Z}_q^*$.
3. Compute $K = \hat{e}(sG, K_E)^r = \hat{e}(G, K_E)^{rs}$.
4. Compute the ciphertext $C = \langle rG, M \oplus H_2(K) \rangle$.

Decryption: Given a ciphertext $C = \langle U, V \rangle$, a user's private key $sH_1(ID_i)$, and the needed time-bound key update $sH_1(T)$ for release time T,

1. Combine the private key $sH_1(ID_i)$ with the key update $sH_1(T)$ to get the decryption key: $K_D = sH_1(ID_i) + sH_1(T) = sK_E$.
2. Compute the pairing $K' = \hat{e}(U, K_D)$.
3. Recover M by computing $V \oplus H_2(K')$.

If C is a correct ciphertext with release time T, then $U = rG$ and $V = M \oplus H_2(K)$ with $K = \hat{e}(G, K_E)^{rs}$. The decryption works because:

$K' = \hat{e}(U, K_D) = \hat{e}(rG, sK_E) = \hat{e}(G, K_E)^{rs} = K$

and $V \oplus H_2(K') = M \oplus H_2(K) \oplus H_2(K) = M$.

A Sketch of Security Proof for ID-TRE

1. Regarding the security of the server's and receiver's private keys, points 1, 3, and 4 for TRE apply here.
2. When a cheating receiver does not have $sH_1(T)$, whether he can decrypt a ciphertext before the release time would depend on whether he can compute $\hat{e}(G, H_1(T))^{rs}$ from sG and rG. This is because the needed pairing $\hat{e}(G, H_1(ID) + H_1(T))^{rs}$ can be decomposed as a product of two pairings, i.e. $\hat{e}(G, H_1(ID))^{rs}\hat{e}(G, H_1(T))^{rs}$, and the first half is computable by the user. Applying the same argument used for TRE, this problem is at least as difficult as the Bilinear Diffie-Hellman (BDH) Problem. As long as the BDH problem is difficult, a receiver could not decrypt an encrypted message (sent to him) before its release time unless he colludes with the time server.

5.3 Discussions

We will discuss a number of desirable properties of TRE.

5.3.1 A Single, Self-authenticated Time-bound Key Update

As can be seen, only a single time-bound key update for release time T, in the form $sH_1(T)$, is needed for all receivers, making this TRE scheme considerably scalable — no matter how many users there are, only one time-bound key update for each release time T is needed. Besides, no secure channel is needed for delivering the key updates.

The time-bound key update is self-authenticated and the time server does not need to create an additional signature to convince the users of its authenticity. In fact, its form $sH_1(T)$ is equivalent to the short signature in [5], a release time T (in the form of a string of arbitrary length) signed by the server with its private key s.

5.3.2 Generalization to a Policy Lock

In TRE, the time server essentially sends out a signed message on $T \in \{0,1\}^*$ which could be of the form "It is now time t", and this signed message is one of the ingredients needed to decrypt a time-lock encryption along with the receiver's private key. It is possible to generalize it to a more general policy-lock encryption mechanism as follows. The server could be treated as a witness known to the sender and receiver. He could sign messages of an arbitrary form $C \in \{0,1\}^*$ describing the conditions (specified by the sender) under which the receiver can recover a locked message. Possible forms of C are "It is an emergency", "The receiver has completed his requested task X", etc..

5.3.3 Key Insulation Property and Applications to Key Evolution

Although simple, the TRE scheme has the property of very good key insulation to provide resilience against key exposure. In most cryptosystems, the private key is directly used in decryption; so when the decryption is done on an insecure device, key exposure could be the biggest threat. There has been work, such as [10], proposing techniques which update the private key while keeping the public key unchanged but augmented with a time index. In these schemes, the lifetime of a public key is divided into a number of epochs in each of which a different private key is used. Given a number of private keys for different epochs, an adversary could not determine others so that compromised keys in some epochs would not leak out those in others or lead to a total break. In fact, the TRE scheme proposed here achieves the key insulation goal for free.

In TRE, we could avoid using the secret key a directly in decrypting any ciphertexts; instead we could use $K_i = aH_1(T_i)$ as the key for an epoch between time instants T_i and T_{i+1} to do all the decryption on the relatively insecure device. The original secret key a could be stored in a safe device (say a smart card) or derived from a certain human-memorable password (using a good cryptographic hash function). When a new key update for instant T_i is received from the time server, the user computes $aH_1(T_i)$ in a safe device and then stores it in the relatively insecure device; this computation could be done in a micro-controller-based smart card. In case a human-memorable password is used to derive the secret key a, the computation of $aH_1(T_i)$

could be done on the insecure device and all the intermediate results are deleted once the computation is completed. Due to the security guarantee in TRE, any compromised K_i would not leak out K_j, $j \neq i$.

5.3.4 Overhead of Changing Time Servers

In TRE, the certificate authority (CA) and the trusted time server need not be the same entity. In fact, as long as the aG part of a public key is certified, the validity of its asG part could be verified easily. In TRE, it is the receiver who chooses the time server. In some cases, the sender may not trust the time server $\mathcal{S}$ chosen by the receiver and request the receiver to give him a public key using another time server $\mathcal{S}'$. At a first glance, the receiver might have to go through the same certification process with the CA in order to convince others this is his new public key with Server $\mathcal{S}'$. But in fact, the receiver does not need to get a new certificate in order to achieve this because using the original public key, other users can verify the authenticity of the new public key as follows: Suppose (aG, asG) is the original, CA-certified public key and $(aG, as'G)$ is the new public key[11]. By verifying that $\hat{e}(G, as'G) = \hat{e}(s'G, aG)$ (where $s'G$ is the public key of the new time server $\mathcal{S}'$ and aG is a part of the old receiver public key which is certified), any users can tell whether the new key is really from the claimed receiver since aG has been certified in the original public key and only people knowing a (the private key) can create $as'G$ satisfying the above condition.

5.3.5 Using Multiple Time Servers

To lower the risk that a cheating receiver colludes with the time server so that he could receive a time-bound key update before its dedicated release time, the sender could use multiple time servers so that the receiver now needs to collude more servers to cheat. Suppose there are N time servers (specified by the sender) each using a secret key s_i and a generator $G_i \in \mathbb{G}_1$, where $1 \leq i \leq N$, their corresponding public keys and time-bound key updates for T are then (G_i, s_iG_i) and $s_iH_1(T)$ respectively. To encrypt a message M, the sender asks the receiver to give him a new public key of the form $K_{new} = a\sum_{i=1}^{N} s_iG_i$. Using the same trick as above, the sender could verify the validity of the new receiver public key and send a ciphertext of the form $\langle rG_1, rG_2, \ldots, rG_N, M \oplus H_2(K)\rangle$ where $K = \hat{e}(rK_{new}, H_1(T))$. Arranging terms, K is equal to $\prod_{i=1}^{N} \hat{e}(G_i, H_1(T))^{ras_i}$. The receiver now needs to get all $s_iH_1(T)$, each from one of the N servers, together with his private key a to compute K which is needed for decryption (Note that if we denote $K_i = \hat{e}(G_i, H_1(T))^{ras_i}$, where $1 \leq i \leq N$, we could compute K_i in the following way: $K_i' = \hat{e}(rG_i, s_iH_1(T))^a = \hat{e}(G_i, H_1(T))^{ras_i} = K_i$.).

Although the properties of ID-TRE is not as appealing as TRE and key escrow is inherent in it (as in all other identity-based schemes), the time-bound key update for a particular time instant T in ID-TRE is still identical for all receivers; the time server just needs to broadcast a single key update for each T, thus offering good scalability.

6 Conclusions

In this paper, we provide a solution to the problem of server-passive and user-anonymous timed release encryption. We give two constructions that can achieve timed release encryption with a precisely specified absolute release time without needing any interaction between the server and the sender or receiver. The schemes are scalable since only a single, identical time-bound key update for all users is needed for any particular time instant.

In the schemes discussed in this paper, a key update from the time server, $sH_1(T)$, could only be used to decrypt messages with release time T, but not any $T_i < T$. In this paper, we assume that the server posts all the old, published updates at a public place (say on a webpage) for the users to look up, so missing an update would not cause a problem. As future work, we wish to design schemes resilient to missing updates. A possible approach could be using the hierarchial identity based encryption in a way similar to forward secure encryption [7].

Acknowledgement

The authors would like to thank Nigel Smart for pointing out that the idea they used to construct the identity-based timed release encryption was proposed in [8]. They also thank the anonymous reviewers for the useful comments and making them aware of the references [1] and [13].

References

[1] D. Beaver and N. So. Unpredictable bit generation without broadcast. In *Advances in Cryptology — Eurocrypt'1993, Springer-Verlag LNCS vol. 765*, pages 424–434, 1993.

[2] M. Bellare and S. Goldwasser. Encapsulated key-escrow. *MIT LCS Tech. Report MIT/LCS/TR-688*, Apr. 1996.

[3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *proc. ACM Conference on Computer and Communication Security (CCS'93)*, pages 62–73, Nov. 1993.

[11] For simplicity in discussion, we assume the new time server uses the same generator G. In fact, the generator of the new time server needs not be the same as the old one. Even if a different G is used, the same discussion applies because we can re-write the new generator G' as $G' = xG$ for some unknown $x \in \mathbb{Z}_q^*$ and take $s'x$ as the new private key in the discussion.

[4] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology — Crypto'2001, Springer-Verlag LNCS vol. 2139*, pages 213–229, 2001.
[5] D. Boneh, B. Lynn, and H. Shacham. Short signatures from Weil pairing. In *Advances in Cryptology — Asiacrypt'2001, Springer-Verlag LNCS vol. 2248*, pages 514–532, 2001.
[6] D. Boneh and M. Naor. Timed commitments (extended abstract). In *Advances in Cryptology — Crypto'2000, Springer-Verlag LNCS vol. 1880*, pages 236–254, 2000.
[7] R. Canetti, S. Halevi, and J. Katz. A forward secure public key encryption scheme. In *Advances in Cryptology — Eurocrypt'2003, Springer-Verlag LNCS vol. 2656*, pages 236–254, 2003.
[8] L. Chen, K. Harrison, N. P. Smart, and D. Soldera. Applications of multiple trust authorities in pairing based cryptosystems. In *InfraSec 2002, Springer-Verlag LNCS vol. 2437*, pages 260–275, 2002.
[9] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and time-release encryption. In *Advances in Cryptology — Eurocrypt'99, Springer-Verlag LNCS vol. 1592*, pages 74–89, 1999.
[10] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *Advances in Cryptology — Eurocrypt'2002, Springer-Verlag LNCS vol. 2332*, 2002.
[11] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology — Crypto'99, Springer-Verlag LNCS vol. 1666*, pages 537–554, 1999.
[12] J. Garay and M. Jakobsson. Timed release of standard digital signatures. In *Financial Crypto'2002, Springer-Verlag LNCS vol.*, 2002.
[13] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelop. In *ACM Symposium on Principles of Distributed Computing (PODC)2003*, pages 182–189, July 2003.
[14] W. Mao. Timed-release cryptography. In *SAC'01, Springer-Verlag LNCS vol. 2259*, pages 342–357, Aug. 2001.
[15] T. May. Time-release crypto, Feb. 1993. Manuscript.
[16] R. C. Merkle. Secure communications over insecure channels. *Communications of ACM*, 21(4):294–299, Apr. 1978.
[17] 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 *HP Lab. Report HPL-2002-243*, 2002.
[18] T. Okamoto and D. Pointcheval. REACT: Rapid enhanced-security asymmetric cryptosystem transform. In *Cryptographers' Track RSA Conference CT-RSA 2002, Springer-Verlag LNCS vol. 2271*, pages 159–175, 2002.
[19] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. *MIT LCS Tech. Report MIT/LCS/TR-684*, 1996.

Appendix

In the paper, we consider a simpler adversary model in which the adversary $\mathcal{A}_2()$ views $H_1(T)$ as an arbitrary point in $\mathbb{G}_1$. Considering $H_1()$ as a random oracle, we show here that this assumption is valid and the knowledge of a limited number of other key updates will not help decrypt a ciphertext with a particular release time. We achieve this by showing how $\mathcal{A}_2()$ can be implemented with another adversary $\mathcal{A}_3()$ which, with the help of other key updates, can decrypt a ciphertext whose release time is of his choice with a non-negligible probability ε of success. This equivalently means if solving the problem for $\mathcal{A}_2()$ is hard, so is that for $\mathcal{A}_3()$. As shown in the paper, the task $\mathcal{A}_2()$ solves is at least as hard as the BDH problem, then so is that $\mathcal{A}_3()$ solves, which is the very problem of breaking the TRE time lock.

We will show how $\mathcal{A}_2()$ could simulate the adversary environment $\mathcal{A}_3()$ works with to get it help solve its problem. Suppose now $\mathcal{A}_2()$ receives the following problem instance: Given rG, sG, and $Q \in \mathbb{G}_1$, find $\hat{e}(G,Q)^{rs}$. (Without loss of generality, a is dropped here.)

When we model $H_1()$ as a random oracle, $\mathcal{A}_3()$ is forced to query an oracle under full control of $\mathcal{A}_2()$ in order to evaluate $H_1()$. $\mathcal{A}_2()$ simulates the $H_1()$ queries as follows. All the previous queries are kept in a list; if the current query T_i is in the list, return the previous reply. If not, it generates a new one by first picking a random number $b_i \in \mathbb{Z}_q^*$ and then returning $H_1(T_i) = b_iQ$ with probability δ or $H_1(T_i) = b_iG$ with probability $(1-\delta)$. The new entry $(T_i, b_i, H_1(T_i))$ is added to the existing $H_1()$ query list. It is clear that $\mathcal{A}_3()$ cannot distinguish the query output from a random one.

Since $\mathcal{A}_3()$ can have access to other key updates $sH_1(T_i)$, $T_i \neq T$ (with T being his choice of challenge yet to be decided). $\mathcal{A}_2()$ can simulate the update query as follows: If T_i is in the $H_1()$ query list, it retrieves it, otherwise generates a new one and adds it back to the list. Then, if $H_1(T_i) = b_iQ$, this run of $\mathcal{A}_2()$ fails, otherwise it returns $sH_1(T_i) = sb_iG$ (which can be computed by $\mathcal{A}_2()$ from sG and b_i).

$\mathcal{A}_3()$ is then asked to pick a challenge T. $\mathcal{A}_2()$ does the same procedures for any further $H_1()$ queries. If $H_1(T) = bG$, this run of $\mathcal{A}_2()$ fails, otherwise (i.e. $H_1(T) = bQ$) $\mathcal{A}_2()$ gives $\mathcal{A}_3()$ the following ciphertext: $C = \langle rG, X \rangle$ where X is randomly picked. $\mathcal{A}_2()$ keeps on replying any $H_1()$ and update queries from $\mathcal{A}_3()$. At the end, $\mathcal{A}_3()$ should return a plaintext of the form $M = X \oplus H_2(\hat{e}(G, H_1(T))^{rs}) = X \oplus H_2(\hat{e}(G,Q)^{rsb})$ with a probability of success ε. Suppose $\mathcal{A}_3()$ has made at most q_u key update queries; then a run of $\mathcal{A}_2()$ does not abort with probability $\delta(1-\delta)^{q_u}$. That is, we could find $H_2(\hat{e}(G,Q)^{rsb})$ with probability $\delta(1-\delta)^{q_u}\varepsilon$. If $H_2()$ is an invertible function, we could get $\hat{e}(G,Q)^{rs}$ (what $\mathcal{A}_2()$ needs to find) as b is known. If $H_2()$ is a hash function and modeled as a random oracle, we randomly pick an entry in the list of previous $H_2()$ queries and would still have a good probability in picking the desired preimage of $H_2()$. It could be observed that $\mathcal{A}_3()$ should not be able to distinguish the simulation played by $\mathcal{A}_2()$ from a real adversary environment.