Scalable, Server-Passive, User-Anonymous Timed Release Public Key Encryption from Bilinear Pairing

Ian F. Blake, Aldar C-F. Chan
Department of Electrical and Computer Engineering
University of Toronto, Canada
email: {ifblake, aldar}@comm.utoronto.ca

September 7, 2004

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 sender and/or receiver and are not scalable (server-based approach). Using a bilinear paring on any Gap Diffie-Hellman group, we solve this problem by giving a scalable, server-passive and user-anonymous timed release public-key encryption scheme which allows 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 anonymity of both the sender and receiver of a message and the privacy of the message. Besides, our scheme also has a number of desirable properties including self-authenticated time-bound key updates, a single form of update for all receivers, simple public-key renewal and key insulation, making it a scalable and appealing solution.

keywords: timed release encryption, bilinear pairing, anonymity

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 [12] in 1993 and further elaborated by Rivest et. al. [16].

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 9:00AM, July

27, 1973 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. Rivest, Shamir and Wagner [16] gave a number of its applications including electronic auctions, key escrow, chess moves, release of documents (like memoirs) over time, payment schedules, press releases and etc.. Bellare and Goldwasser [1] proposed the use of timed release encryption in key escrow; they suggested that 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 [1, 16, 11, 4, 9, 10] and trusted servers [12, 16, 2, 14]. 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 a lot of 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

[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.

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 [12]. In addition, the fact that the server is actively and considerably involved in the encryption or decryption of every message limits the scalability of these schemes.

It is thus fair to say that 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[2] 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 the message, remained an open problem[3]. Based on bilinear pairing over a Gap Diffie-Hellman group, in this paper we solve this open problem in the affirmative. The main contribution of this paper is the model of a completely passive time reference server in timed release encryption and the demonstration of its feasibility through designing a scalable, server-passive and user-anonymous scheme which allows a precise absolute release time specification.

In our scheme, the time server merely provides a common absolute time reference, in the form of a sequence of 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 [16], 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 he makes a query at the server (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. The security of our scheme is based on the intractability assumption of the bilinear Diffie-Hellman (BDH) problem over any Gap Diffie-Hellman groups. Our scheme is provably secure under this assumption. Besides, our 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. These include:

- The time-bound key update (from the time server) 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 that is enough for all receivers

[2]The identity-based scheme in the appendix, which is essentially the same as the one in [5], has the key escrow risk since the server has the private keys of all users.

[3]Although the server in the offline version of the Rivest's scheme [16] 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.

[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.

to recover their messages, thus making the scheme scalable.

- The key update published by the time server inherently authenticates itself. Thus, no additional overhead of a server signature is needed.
- Certifying the authenticity of a receiver's public key needs not be done by the time server and a separate un-related CA (Certificate Authority) could be used. Although the public key of a receiver is tied to that of the time server he uses, whenever he or the sender wants to use a different time server as reference, the receiver does not need to get a new certificate from the CA for the new public key; instead, anyone could use the receiver's old certified public key to verify the authenticity and validity of his new public key.
- With slight modifications, our scheme inherently satisfies the key insulation property. Instead of using his private key directly for decryption, a receiver could use a short-term key generated from his private key and the time-bound key update from the server to decrypt (for that particular time epoch) on a device vulnerable to compromise. The disclosure of a short-term key would not leak out the receiver's private key or the short-term keys for other time epochs.

We discuss the previous works 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 and 6, we give our construction of timed release encryption and discuss its properties. Finally, we conclude with a discussion of future work on the resilience against missing updates and timed release verifiable signature.

2 Related Works

There are two main approaches adopted in the previous works, 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 [13] and extended by Bellare et. al. [1] and Rivest et. al. [16]. The idea is that a secret is transformed in such a way that any machines (serial or parallel), running continuously, take 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 [1], 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 [16] 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 [4, 11, 9, 10]. Mao [11] added a zero-knowledge proof to it and constructed a time-released RSA signature. Boneh and Naor [4] introduced the notion of (verifiable) timed commitments, an extension to the standard notion of commitments, in which a potential forced opening phase permits the receiver to recover (with effort) the committed value without the help of the committer. They also showed how to use timed commitments to improve a variety of applications involving time, including timed signatures, contract signing, honesty-preserving auctions and concurrent zero-knowledge. Based on timed commitments, Garay et. al. [9, 10] constructed timed release signatures for DSA, Schnorr and RSA.

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 with either the sender or receiver of a message, 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 [12] suggested that a third party can be used as a trusted escrow agent to store messages from senders and release them to the corresponding 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 [16] 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. A sender wishing to release a document at time t sends his message and its desired release time, encrypted with the server's public key, to the server. The server, after decrypting the ciphertext, encrypts this with a symmetric key encryption using the key it will release at instant t and then sends this ciphertext, encrypted with the sender's public key, to the sender which in turn decrypts it, encrypts it again with the receiver's public key and sends it to the receiver. Upon receiving this message, the receiver can get back the server's encrypted message and waits until its key is released at t. 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 future time instants, or the senders cannot freely choose the desired release time of their messages (they can only send messages into limited future). Hence, it is not scalable. Besides, a relatively large amount of computation is also needed for encryption and decryption.

In all the schemes discussed so far, the information sent by the server is not inherently authenticated; the server needs to sign them. While interaction between the sender and server is needed in [16], Di Crescenzo et. al. [6] 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, which consists of several interactive rounds of message exchanges. In the oblivious transfer, the server and the receiver engage in a 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.

When proposing the use of bilinear maps to construct identity based encryption (IBE), Boneh and Franklin [2] named time released encryption as one of its applications. Then Mont et. al. [14] 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 and know its release time and the identity of a receiver.

To eliminate the need of requiring the server to send to each user a distinct time-bound key update, as in [2, 14], we constructed an identity based timed release encryption scheme which only requires a single broadcast update for all receivers. It was later determined that the construction of this scheme had previously been mentioned by Chen et. al. [5], so we put it in the appendix for reference. When investigating the application of the additive property of a bilinear pairing for a multiple trust authority implementation in identity-based encryption, Chen et. al. [5] suggested that one of the authorities could be used for binding time information so as to give a timed press release application. This is essentially an identity-based timed release encryption scheme which only needs one key update for all receivers. A single key update for

[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.

all receivers is in fact a very desirable property, making the scheme scalable.

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 to 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 his private key and nobody else, providing the highest possible privacy.

3 Our Model

As can be seen, the existing techniques using a trusted time server either require interaction between the server and the sender [16] or receiver [6], or let the server know and deliver all the secret keys [2] or messages [12]. 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 the server is 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 "10:01:01 AM Jul 27, 2004 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 "11:01:01 AM Jul 27, 2004 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.

3.1 Timed Release Public Key Encryption

Putting pieces together, we can now state the problem of timed release public key encryption as follows:

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 should not be able to decrypt a message sent to any users[6].

4 Preliminaries

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

Let $\mathbb{G}_1$ be a cyclic additive group generated by P, 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 over $\mathbb{G}_1$
Given $P \in \mathbb{G}_1$ and aP for some unknown $a \in \mathbb{Z}_q^*$, find a.

Definition 2 Decisional Diffie-Hellman (DDH) Problem over $\mathbb{G}_1$
Given $P \in \mathbb{G}_1$, aP, bP and cP for some unknowns a, b, $c \in \mathbb{Z}_q^*$, tell whether $c \equiv ab \pmod q$.

Definition 3 Computational Diffie-Hellman (CDH) Problem over $\mathbb{G}_1$
Given $P \in \mathbb{G}_1$, aP and bP for some unknowns a, $b \in \mathbb{Z}_q^*$, find abP.

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

Definition 4 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:** $\hat{e}(P, Q) \neq 1$.
3. **Computability:** *There is an efficient algorithm to compute* $\hat{e}(P, Q)$.

Definition 5 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* $\hat{e}(P, P)^{abc}$.

The DL problem is usually assumed to be difficult (DL assumption). Both the decisional and computational Diffie-Hellman problems are usually difficult; however, if a bilinear pairing exists in the underlying 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 lead 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 non-negligible probability. $\mathbb{G}_1$ is called a Gap Diffie-Hellman group which can be found in supersingular elliptic curves or hyperelliptic curves over finite field, with the bilinear pairing derived from Weil or Tate pairing. The bilinear Diffie-Hellman problem is usually assumed to be difficult and is the basis of the security of our schemes since solving the DL or GDH problem implies solving the BDH problem. As long as the BDH assumption holds, our schemes are secure.

5 A Timed Release Public Key Encryption Construction

In this section, we will describe a simple construction of timed release encryption based on bilinear mapping. The security of this scheme is based on the hardness of the Bilinear Diffie-Hellman (BDH) Problem over a Gap Diffie-Hellman (GDH) group.

For the sake of clarity and simplicity, the Fujisaki-Okamoto Transform [8] has not been applied in the following discussion. Similar to the technique in [2], this transform can be applied directly to our scheme.

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:

$H_1 : \{0, 1\}^* \rightarrow \mathbb{G}_1^*$

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

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.

User Key Generation: Each user picks a secret key $a \in \mathbb{Z}_q^*$ and computes the public key (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[7] by checking $\hat{e}(sG, H_1(T)) = \hat{e}(G, sH_1(T))$, where (G, sG) is the server's public key.

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: $C =< rG, M \oplus H_2(K) >$.

Decryption: Given a ciphertext $C =< rG, \sigma = M \oplus H_2(K) >$, a receiver's private key a, and a time-bound key update $sH_1(T)$ from the server,

1. Compute the pairing $K' = \hat{e}(rG, sH_1(T))^a = \hat{e}(G, H_1(T))^{ras} = K$.
2. Recover M by computing $\sigma \oplus H_2(K)$.

5.2 A Sketch of Security Proof

1. Given G, sG, it is difficult to find s (DL problem). Hence, the server private key is safe.
2. Given G, sG, aG and asG, it is difficult to find a. The argument is as follows: Suppose we have a polynomial time algorithm $\mathcal{A}(G, sG, aG, asG) = a$ 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 $(= abG)$; using $\mathcal{A}$, we can find $a = \mathcal{A}(G, bG, aG, abG)$. So this problem is at least as difficult as the DL problem. The user private key is thus 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 private key is thus safe.
4. In order to decrypt a ciphertext, a receiver needs to compute $\hat{e}(G, H_1(T))^{ras}$. In case the receiver does not have $sH_1(T)$, he needs to compute $\hat{e}(G, H_1(T))^{ras}$ from rG, sG and a which are the only sources he has for r, s and a. Suppose we rewrite $H_1(T)$ as wG (for some unknown w), then the problem becomes to find $\hat{e}(G, G)^{wras}$ from sG, wG, rG and a, which is equivalent to the Bilinear Diffie-Hellman Problem.

[7] 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.

Now, suppose the receiver attempts to find $sH_1(T)$ from $sH_1(T_i)$ for $T_i \neq T$. If we rewrite $H_1(T)$ as $w_iH_1(T_i)$, the problem becomes to find $sw_iH_1(T_i)$ from $H_1(T_i)$, $w_iH_1(T_i)$, $sH_1(T_i)$, which is equivalent to the Computational Diffie-Hellman Problem over a Gap Diffie-Hellman group.

That is to say, if a receiver wants to recover a message before its release time, the easiest way is to solve the Bilinear Diffie-Hellman Problem. Hence, as long as the bilinear Diffie-Hellman is difficult, the receiver, even with his private key, cannot decrypt an encrypted message before its specified release time unless he colludes with the time server.

6 Discussions

In this section, we will discuss a number of desirable properties of the TRE scheme.

6.1 Time-bound Key Updates

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, 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 [3], a release time T signed by the server with its private key s.

6.2 Key Insulation Property and Applications to Key Evolution

Although simple, this 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 have been a number of works, such as [7], proposing techniques which update the private key while keeping the hardcore public key unchanged but augmented with a time index. In these schemes, given a number of private keys for different epochs, an adversary could not determine others so that compromised keys in some time epochs would not leak out those in other epochs 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 time-bound 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 if $j \neq i$.

6.3 Overhead of Changing Time Servers

In the TRE scheme, the certificate authority 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 $\mathcal{S}'$. From a first glance, the receiver might have to go through the same certification process with the CA (Certificate Authority) 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 convince others 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[8]. 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 part of the old public key of the sender which is certified), any users can tell whether the new key is really from the sender it claims to be 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.

6.4 Using Multiple Time Servers

To lower the risk that a receiver colludes with the time server so that he could receive from the time server 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 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 $\langle rG_1, rG_2, \ldots, rG_N, M \oplus K\rangle$ 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: $K_i' = \hat{e}(rG_i, s_iH_1(T))^a = \hat{e}(G_i, H_1(T))^{ras_i} = K_i$.).

[8] 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.

7 Conclusions

In this paper, we provided a solution to the problem of server-passive and user-anonymous timed release encryption. We proposed a construction that can achieve timed release encryption with a precisely specified absolute release time without needing any interaction between the server and the sender or the receiver. The scheme is scalable since only a single, identical time-bound key update for all users is needed for a certain time instant. It also has a number of useful additional properties like key insulation and simple public key renewal.

As future research, we will focus on constructions resilient to missing time-bound key updates and timed release verifiable signature schemes, which are described as follows.

7.1 Resilience to Missing Time-bound Key Updates

In the TRE scheme discussed in this paper, a server time-bound key update $sH_1(T)$ could only be used to decrypt message 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. Although in most scenarios, it is unlikely that a receiver having a timed release encrypted message would miss out the time-bound key update he needs, as future work, we still want to design schemes resilient to missing updates. A trivial way is as follows: when the sender sends out the ciphertext with release time T, he can append a number of attachments (one for each T_i) that could be used to decrypt the ciphertext using another key update for T_i where $T_i > T$. For example, in TRE, the sender could send out the list $\{X_i = \hat{e}(rasG, H_1(T) - H_1(T_i))\}$ through which the receiver having any one of the $sH_1(T_i)$'s could get back the message by computing $K = \hat{e}(rG, sH_1(T_i))^a X_i$. However, this would require an overhead linear to the number of T_i. We need a concise representation of key updates as in the recent work by Pinkas [15]. However, Pinkas's idea and other common techniques of hash chains or tree commitment do not seem to work here; more subtle techniques need to be explored.

7.2 Timed Release Verifiable Signature

Another problem for future research is the notion of timed release verifiable signature in which a signature can be verified by the designated receiver only but he cannot prove to any third parties its authenticity (that the signer has really signed the document) before the specified release time has passed. A timed release verifiable signature is analogous to the concept of post-dated cheques in the real world, except that in the latter, all people know who has signed a particular cheque whose instructions cannot be exercised before the specified date, whereas nobody would be convinced that a timed release verifiable signature was actually signed by the signer (not the receiver) before the release time. This notion first appeared in [11, 4, 9] (which is slightly different from the notion of our considerations). However, these schemes used the time-lock puzzle approach to implement the timed release mechanism, which is both computationally inefficient and unable to give a precise release time.

In our model, there are three parties — signer, receiver and time server (excluding the public). The signer, using his secret, the server's public key and the receiver's public key, signs a certain document M with a specified release time T. Using his private key and the server's public key, the receiver can be convinced that the signature σ is really the signer's signature for the message M and can become universally verifiable (i.e. verifiable by anybody else) at or after T. However, without the time-bound key update for the time instant T from the time server, the receiver cannot prove to anyone else σ is an authentic signature of the signer (even if he disclose his private key) since the receiver, using his private key, is capable of creating the same signature indistinguishable from that of the signer. After getting the required time bound key update, the receiver can convert σ into a universally verifiable signature that everyone can verify its authenticity using the signer's public key. Both the signer-and-receiver verification and the signature conversion should be non-interactive.

Acknowledgement

The authors would like to thank Nigel Smart for pointing out that the idea they used to construct the identity-based variant of timed release encryption in the Appendix is the same as that in [5].

References

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

[2] 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.

[3] 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.

[4] D. Boneh and M. Naor. Timed commitments (extended abstract). In *Advances in Cryptology — Crypto'2000, Springer-Verlag LNCS vol. 1880*, pages 236–254, 2000.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. Garay and M. Jakobsson. Timed release of standard digital signatures. In *Financial Crypto'2002, Springer-Verlag LNCS vol.*, 2002.

[10] J. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *Financial Crypto'2003, Springer-Verlag LNCS vol.*, 2003.

[11] W. Mao. Timed-release cryptography. In *SAC'01, Springer-Verlag LNCS vol. 2259*, pages 342–357, August 2001.

[12] T. May. Time-release crypto. *Manuscript, http://www.hks.net.cpunks/cpunks-0/1560.html*, February 1993.

[13] R. C. Merkle. Secure communications over insecure channels. *Communications of ACM*, 21(4):294–299, April 1978.

[14] 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.

[15] B. Pinkas. Efficient state update for key management. *Proceedings of the IEEE*, 92(6), June 2004.

[16] 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: ID-based Timed Release Encryption (ID-TRE)

Here we will consider an ID-based Timed Release Encryption. For the sake of simplicity, the time server is the same entity as the trusted server assigning private key to users in an ID-based encryption scheme [2] in the following discussion; in fact, it could be a different entity.

Suppose $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive and multiplicative cyclic groups of order q (prime) respectively and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a bilinear map. Assume ID_i is the identity of a particular user i and T is an arbitrary release time. Given the following cryptographic hash functions:

$H_1 : \{0,1\}^* \to \mathbb{G}_1^*$

$H_2 : \mathbb{G}_2^* \to \{0,1\}^n$

The ID-TRE protocol runs as follows.

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.

For a user i with ID_i, the server computes and gives him $sH_1(ID_i)$, to be used as user i's private key, and the corresponding 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), which indicates the current time. Anyone using the server's public key can verify its authenticity because $\hat{e}(sG, H_1(T)) = \hat{e}(G, sH_1(T))$.

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 $C = \langle rG,\ \sigma = M \oplus H_2(K)\rangle$, a user's private key $sH_1(ID_i)$, and a time-bound key update for release time T, which is $sH_1(T)$,

1. Combine the static private key $sH_1(ID_i)$ with the time-bound 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 $K' = \hat{e}(rG, K_D) = \hat{e}(rG, sK_E) = \hat{e}(G, K_E)^{rs} = K$.
3. Recover M by computing $\sigma \oplus H_2(K)$.

A Sketch of Security Proof

1. Regarding the security of the server's and the receiver's private keys, points 1 and 3 for the first TRE scheme apply here.
2. When a 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. Applying the same argument for the first TRE scheme here, 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.

Although the properties of ID-TRE is not as appealing as TRE, 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 a good scalability.