Timed Release Cryptography from Bilinear Pairings Using Hash Chains

Konstantinos Chalkias and George Stephanides

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
chalkias@java.uom.gr,
steph@uom.gr

Abstract. We propose a new Timed Release Cryptography (TRC) scheme which is based on bilinear pairings together with an S/Key-like procedure used for private key generation. Existing schemes for this task, such as time-lock puzzle approach, provide an approximate release time, dependent on the recipients' CPU speed and the beginning time of the decryption process. Additionally, some other server-based schemes do not provide scalability and anonymity because the server is actively involved in the encryption or the decryption. However, there are already protocols based on bilinear pairings that solve most of the problems referred. Our goal is to extend and combine the existing protocols with desirable properties in order to create a secure, fast and scalable TRC scheme applied to dependent or sequential events. For this purpose we used continuous hashed time-instant private keys (hash chain) in the same way the S/Key system works. Our approach decreases dramatically the number of past time-instant private keys the server stores and only two keys are needed, the last one to construct the previous keys and the first one to recursively verify the authenticity of the next keys.

Keywords: Timed-Release Cryptography, bilinear pairings, S/Key, hash chains, sealed-bid auctions.

1 Introduction

The essence of timed release cryptography (TRC) is to encrypt a message so that it cannot be decrypted by anyone, including the designated recipients, until a specific time-instance. This problem of "sending information into the future" was first mentioned by May [23] in 1993 and then discussed in detail by Rivest et al. [29]. Since its introduction, the solution to the TRC problem has been found useful in a number of real world applications. Some of the best examples are the e-voting which requires delayed opening of votes, the sealed-bid auctions in which the bids must stay sealed so that they cannot be opened before the bidding period and the Internet programming contest where participating teams cannot access the challenge problem before the contest starts. Moreover, TRC can be used for delayed verification of a signed document, such as lottery and check cashing [32] and it can also be applied to online games, especially card games, where players would be able to verify the authenticity of the result when the game ends.

H. Leitold and E. Markatos (Eds.): CMS 2006, LNCS 4237, pp. 130 – 140, 2006.

1.1 Current TRC Schemes

The existing schemes that solve the TRC problem are divided into two ways – time-lock puzzles [1, 6, 13, 15, 21, 29] and trusted servers [4, 22, 23, 29]. However, none of them are fully satisfactory. Time-lock puzzle approach is based on the required time the receiver needs to perform non-parallelizable computation without stopping. The main advantage of this approach is that no trusted server is needed, but there are also a lot of disadvantages that makes it impractical for real-life scenarios. Some of the drawbacks are that it puts immense computational overhead on the receiver, it depends on the receiver's CPU speed and it does not guarantee that the receiver will retrieve the message immediately after the sender's desired released time have passed. Still, this approach is widely used for specific applications [1, 2, 6, 13, 14, 15, 32].

On the other hand, using trusted servers relieves the receiver from performing non-stop computation and sets the time of the decryption precisely. The cost of this approach is that it requires interaction between the server and the sender or the receiver of the message, or even both. Additionally, some of the existing protocols sacrifice the anonymity of users and sometimes the server is considerably involved in the encryption or decryption process which makes these schemes less scalable. For example there are schemes where the server encrypts messages on request using symmetric encryption and then it publishes the secret key on the designated time. A different scheme was proposed by Di Crescenzo et. al [10], in which non-malleable encryption is used, the server knows nothing about the release time or the identity of the sender, while the receiver engages in a conditional oblivious transfer protocol with the server to get the clear message. Recently, there have been attempts to use bilinear map based IBE schemes for timed release cryptography [3, 4, 9, 22, 28]. Although most of them provide sufficient functionalities, there is still a need of decreasing the amount of data transferred between the users and the server. This is because in a real world application, where a time-server supports thousands or millions of users (including software-agents), there would be 'important' time instances where the majority of receivers will simultaneously try to retrieve the server's private information to read their messages. In this case, a DoS attack may occurred and it is possible that some users will gain advantage through this information.

The main contribution of this paper is to combine the schemes with desirable properties in order to decrease at the minimum the length of the private information that the server reveals and broadcasts at the specific time instance. Our aim is to avoid fairness issues arising from uncontrollable network congestion or delivery delay. Unlike other schemes [3], our approach eliminates the amount of data that the server broadcasts at the designated time and the private information is nothing more than an integer value that is recursively authenticated. Furthermore, we use a continuous hashing procedure to construct the time-instant private keys in the same way the S/Key password authentication system works. Thus, only the current private key is needed to construct the previous keys. The last property is very useful as a user can decrypt messages of previous time-instances by just getting the current private key from the server.

1.2 Properties of a TRC Scheme

To analyze the desirable properties of a TRC scheme, we have to describe some of the applications that require 'future decryption'. As it is referred above, one of these applications is the sealed-bid auction where bidders submit their bids in closed form to the auction board [7, 11, 17, 18, 26, 28]. Once the bidding is closed the bids are opened and the best offer wins the auction. The main problem in the auction is cheating by the auction board or by a competitor. To avoid opening the bids before the desired time, an independent and trusted time-server is needed. To lower the risk that the auction board or a competitor colludes with the time-server, the bidder can use multiple time-servers so that the 'enemy' will need to collude with all the servers to cheat. Some other basic requirements in sealed-bid auctions are that only the auction board will be able to decrypt and verify the bid after the bidding close, the auction board should not be able to disavow bid submission and the bidder should not be able to repudiate his bid.

TRC schemes can also be used to verify the authenticity of the results to the players in an online card game. In this kind of games a user plays against other players or against the gambling company itself. To avoid cheating (from the company) a TRC scheme can be applied together with an independent random generator. In this case a random generator will firstly send the encrypted sequence of cards, so that the players will have the encrypted result before the game starts. When the game ends the time-server will publish the private key for the decryption. Now the players are sure that the company hasn't changed the card sequence during the game. Our approach seems to work very well in this example as a player needs to connect to the time-server only when he stops playing. Then, he gets the private key for the last game and, using continuous hashing, he recursively constructs the private keys for the previous games.

The basic properties of our proposed TRC scheme are:

- The time-server does not interact with either the sender or the receiver.
- The time-instant private key is an integer value (not an Elliptic Curve point [3]) and is also identical for all receivers.
- The public and private key updates published by the time-server inherently authenticate themselves. There is no need of a server signature.
- A Certificate Authority could be used to verify the authenticity of the users.
- The last time-instant private key can be used to construct all the previous time-instant keys.

2 Preliminaries

In the description of our proposed scheme, we will use the following notations and definitions. To better understand the protocol and its security, we review the S/Key system, the bilinear maps and the related mathematical problems we have to face.

2.1 S/Key System

The S/Key one-time password system was proposed by Neil M. Haller [16] in 1995. It is an authentication system which applies a secure hash function multiple times to

construct the one-time passwords. The first one-time password is produced by hashing the client's processed password for some specified number of times, say *N*. The next one-time password is generated by hashing the user's password for only *N* - 1 times. Generally, If f is the hash function, *s* is the original client's password and $p_{(i)}$ is the one-time password at the *i*-th login attempt then :

$$p_{(i)} = f^{N-i}(s) \quad (1)$$

This system is secure against eavesdropping attacks as the login – passwords are always different. The eavesdropper cannot produce the next one-time password as the hash function is a one-way function. However, the last property is very useful for the verification of the next password. When the user attempts to login again using the new one-time password, the server checks that the hash product of the new password is equal to the previous password. As there are functions that their hash product is 256 or 512 bits, we can use this procedure in our scheme to construct Elliptic Curve Cryptography private keys of sufficient key-size [20].

2.2 Bilinear Pairings

Suppose $\mathbb{G}_1$ is an additive cyclic group generated by *P*, whose order is a prime *q*, and $\mathbb{G}_2$ is a multiplicative cyclic group of the same order. A map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is called a bilinear mapping if it satisfies the following properties:

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

We note that $\mathbb{G}_1$ is the group of points on an elliptic curve and $\mathbb{G}_2$ is a multiplicative subgroup of a finite field. Typically, the Weil, and Tate pairings can be used to construct an admissible bilinear pairing. For a detailed description of pairings and the conditions on elliptic curves one can see [8]. An implementation of the Weil and Tate pairing can be found at [30, 31].

2.3 Mathematical Problems

Definition 1. Discrete Logarithm Problem (DLP)
Given $Q, R \in \mathbb{G}_1$ *find an integer* $a \in \mathbf{Z}_q^*$ *such that* $R = aQ$.

Menezes et al. [25] show a reduction from the DLP in $\mathbb{G}_1$ to the DLP in $\mathbb{G}_2$ and they prove that DLP in $\mathbb{G}_1$ is no harder than the DLP in $\mathbb{G}_2$.

Definition 2. Decisional Diffie-Hellman Problem (DDHP)
Given $Q \in \mathbb{G}_1$, *aQ*, *bQ and cQ for some unknowns a, b,* $c \in \mathbf{Z}_q^*$ *tell whether* $c \equiv ab \pmod q$.

While DDHP is hard in $\mathbb{G}_2$, Joux and Nguyen [19] show that DDHP is easy in $\mathbb{G}_1$. Hardness of DDHP in $\mathbb{G}_2$ implies that, $\forall Q \in \mathbb{G}_1$, inverting the isomorphism that takes $P \in \mathbb{G}_1$ and computes $\hat{e}(P, Q)$ is hard [4].

Definition 3. Computational Diffie-Hellman Problem (CDHP)
Given $Q \in \mathbb{G}_1$, aQ, bQ *for some unknowns* $a, b \in Z_q^*$, *compute* abQ.

The advantage of any randomized polynomial-time algorithm $\mathcal{A}$ in solving CDHP in $\mathbb{G}_1$, is defined by the following equation:

$$Adv_{A,G_1}^{CDH} = \text{Prob}\,[\mathcal{A}(P, aP, bP, abP) = 1 : a, b \in Z_q^*] \quad (2)$$

For every probabilistic algorithm $\mathcal{A}$, Adv_{A,G_1}^{CDH} is negligible.

Definition 4. Bilinear Diffie-Hellman Problem (BDHP)
Given $Q \in \mathbb{G}_1$, aQ, bQ *and* cQ *for some unknowns* $a, b, c \in Z_q^*$, *compute* $\hat{e}(Q, Q)^{abc}$

If a bilinear pairing exists in the underlying group, the DDHP problem over it can be solved by checking if $\hat{e}(aQ, bQ) = \hat{e}(Q, cQ)$. This lead to the Gap Diffie-Hellman (GDH) assumption according to which, the DDHP on an additive group $\mathbb{G}_1$ can be solved in polynomial time, but there is no polynomial time algorithm to solve the CDHP with non-negligible probability. $\mathbb{G}_1$ is called a GDH group which can be found in supersingular elliptic curves or hyperelliptic curves over finite field. The BDHP over a GDH group is assumed to be difficult and the security of our proposed scheme is based on that assumption.

3 The Proposed TRC Scheme

In this section, we describe the proposed TRC scheme, which is a combination of the scheme proposed by Blake and Chan [3] and the S/Key password authentication system. There is an analysis of its security and an improvement/extension of the way the keys are constructed in order to be better protected against birthday attacks that can be applied in hash functions. [27, 33]. Our encryption scheme is (Gen, TGen, Enc, Dec) for four algorithms such that Gen generates the public and private keys, TGen generates the time-instant keys, Enc encrypts using the receiver's and server's public key, the time-instant public key and the sender's private key and Dec decrypts using the receiver's private key and the time-instant private key.

3.1 General Setup and Key Generation

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be an additive and multiplicative cyclic group of order q (a prime number) respectively and that $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is an admissible bilinear map. The following cryptographic hash functions are chosen: 1) H_1: $\{0, 1\}^* \rightarrow \mathbb{G}_1$, 2) H_2: $\{0, 1\}^* \rightarrow \{0, 1\}^n$ for some n. The notation $H_2^n(x)$ stands for the continuous hashing of x for n times, for example $H_2^3(x) = H_2(H_2(H_2(x)))$. If $n = 0$ then $H_2^0(x) = x$.

There are three entities in the proposed scheme, namely the server (time-server), the sender and the recipient. The server chooses a random private key $s \in Z_q^*$ and a generator of $\mathbb{G}_1$, say G. The server's public key consists of two elements: G, sG. As

for the sender, he chooses a secret private key $a \in Z_q^*$ and he publishes the public key which is: aG, asG. Similarly, the receiver chooses a secret key $b \in Z_q^*$ and he computes the public key: bG, bsG. It is easily understood that this is not an ID-based encryption scheme and a CA type of certification is needed to verify the authenticity of the public keys.

3.2 Time-Instant Key Generation

In our scheme, the construction of the time-instant private keys (the keys needed to decrypt a message at a specified time instance) is based on an S/Key-like procedure. Suppose that the server needs to publish the public keys of a single day and that every key represents a different time instance of that day. To better understand the procedure let us assume that the server needs to publish 24 different public keys and that each one represents a unique hour on that day (eg. 11:00:00 PM Feb 10, 2006 GMT"). For this purpose, the time-server selects a random secret integer value $t \in Z_q^*$. To compute the private key of the first hour of the day T_1 (01:00:00 AM Feb 10, 2006 GMT), the time-server computes the $H_2^{23}(t)$ (this is the private key). The public key for that time instance is: $H_2^{23}(t) \cdot H_1(T_1)$. Similarly, the public key for the next time-instance T_2 (02:00:00 AM Feb 10, 2006 GMT) is $H_2^{22}(t) \cdot H_1(T_2)$ and the same goes for the next time-instances, until the last time instance where the value t is the private key and the point $t \cdot H_1(T_2)$ is the public key. To authenticate a time-instant public key, the server has to publish (together with the public key) the point value $H_2^n(t) \cdot sG$, where $H_2^n(t)$ is the private key for the n-th time instance. To accept a public key, the sender checks if the following equality exists:

$$\hat{e}(H_1(T_n), H_2^{24-n}(t) \cdot sG) = \hat{e}(H_2^{24-n}(t) \cdot H_1(T_n), sG) \quad (3)$$

The trusted time-server publishes the private time-instant key at the specified time.

3.3 Encryption Process

As the public keys consist of two elements we will use the notation Pub_1X to express the first element of the key that belongs to user X and Pub_2X to express the second element e.g for the recipient B : $Pub_1B = bG$ and $Pub_2B = bsG$

Given a message M, a sender's private key (a), a recipient's public key ($Pub_1B = bG$, $Pub_2B = bsG$), a server's public key ($Pub_1S = G$, $Pub_2S = sG$), a release time $T \in \{0,1\}^*$, and a time-instant public key for the time T: ($Pub_1T = nH_1(T)$, $Pub_2T = nsG$)

1. Verify that $\hat{e}(H_1(T), Pub_2T) = \hat{e}(Pub_1T, Pub_2S)$ => $\hat{e}(H_1(T), nsG) = \hat{e}(nH_1(T), sG)$ => $\hat{e}(H_1(T), G)^{ns} = \hat{e}(H_1(T), G)^{ns}$; if true → accept the time-instant public key.
2. Verify that $\hat{e}(Pub_1B, Pub_2S) = \hat{e}(Pub_1S, Pub_2B)$ => $\hat{e}(bG, sG) = \hat{e}(G, bsG)$ => $\hat{e}(G,G)^{bs} = \hat{e}(G, G)^{bs}$; if true → accept the recipients public key.
3. Choose a random integer $r \in Z_q^*$.

4. Calculate $K = \hat{e}(Pub_1T, Pub_1B)^{ar} = \hat{e}(nH_1(T), bG)^{ar} = \hat{e}(H_1(T), G)^{abnr}$. [1]
5. Send ciphertext $C = < rH_1(T), M \oplus K >$.

3.4 Decryption Process

Given a ciphertext $C = < rH_1(T), M \oplus K >$, a sender's public key (Pub_1A, Pub_2A), a recipients private key (a) and a time-instant private key n,

1. Compute the pairing $K^* = \hat{e}(rH_1(T), Pub_1A)^{bn} = \hat{e}(rH_1(T), aG)^{bn} = \hat{e}(H_1(T), G)^{abnr}$; if $K^* = K$ then the recipient is sure that the message is not corrupted and he can also verify the sender's identity and validity of the time-instant key n (receiver uses the key n and the sender's public key to compute the pairing $K^* = K$)
2. Recover M by computing $(M \oplus K) \oplus K^* = M$

3.5 A Sketch of Security

To provide a security proof, we work in the same way as Blake and Chan do in [3]. The server's private key s is safe because it is difficult to find s from G, sG (DLP). In the same way, it is difficult to find a user's private key a from G , aG , sG , asG. The argument is as follows: Suppose there exists a polynomial time algorithm $\mathcal{A}(G, aG, sG, asG)$ that finds a. This means that $\mathcal{A}$ can be used to solve the DLP in the following way: Given G, aG, we choose a random integer $b \in Z_q^*$ to compute bG and $baG = abG$; using $\mathcal{A}$, we can find $a = \mathcal{A}(G, aG, bG, baG)$. This problem is as difficult as the DLP.

A message cannot be decrypted since the specified time as the receiver needs to compute $\hat{e}(H_1(T),G)^{abnr}$ from sG, aG, asG, b, $rH_1(T)$, $nH_1(T)$ and nsG. As it can been seen, the mapping $\hat{e}(H_1(T),G)^{abnr}$ does not contain the server's private key, so the sG, asG and nsG are useless. Suppose that the receiver rewrites G as $wH_1(T)$ for some unknown w, then the problem becomes to find $\hat{e}(H_1(T), H_1(T))^{abnrw}$ from $wH_1(T)$, $bH_1(T)$, $rH_1(T)$, $nH_1(T)$ and $awH_1(T)$. This problem is equivalent to the BDHP.

As it can be seen, the easiest way for a receiver to recover a message before the designated time is to solve the Bilinear Diffie-Hellman Problem (BDHP). Hence, as the BDHP problem holds, the recipient cannot gain any information of the encrypted message before its specified release time (excluding the case he colludes with the time-server).

3.6 Extended Private Key Construction

One of the problems that our scheme has to face is that the time-instant private keys are fully dependent. Even though it is very difficult for someone to extract a private key from the public key, we can assume that an attacker finally finds a private key that represents a time instance T_i. Then, by hashing that key, he can produce all the previous private keys (T_{i-1}, T_{i-2}, …, T_{i-n}). This means that he will be able to decrypt all

[1] The sender's private key is used in the encryption algorithm in order for the receiver to authenticate the sender's identity during the first step of the decryption process.

the messages sent to him (encrypted with the public keys of these time-instances). To be better protected against this threat, we chose a different procedure for the key construction. Unlike the initial approach, the time-server selects two random secret integer values $t_1, t_2 \in Z_q^*$. The private key for the time-instance T_n is $t_1 \oplus t_2$. The private key for T_{n-1} is $H_2(t_1) \oplus H_2(t_2)$, for T_{n-2} is $H_2^2(t_1) \oplus H_2^2(t_2)$ etc. Using this method of key construction, the knowledge of the private key (of a time instance) does not reveal the private keys that represent previous time-instances.

Although this approach is much safer, it comes with a cost. When the server publishes the private key for a specific time instance he needs also to publish extra information, in order for the users to construct the previous keys. As it is already referred, the main advantage of our protocol is that only the last private key is needed to construct the previous time-instant private keys. For this purpose, if for example the current private key is $H_2(t_1) \oplus H_2(t_2)$, then the server also publishes $H_2(t_2)$. Now a user can compute $H_2(t_1) = (H_2(t_1) \oplus H_2(t_2)) \oplus H_2(t_2)$. Then, he can construct the previous private key $H_2^2(t_1)) \oplus H_2^2(t_2))$.

4 Discussions

To better understand the way our protocol works, we will describe a possible scenario that it can be applied to. Through this scenario, we will discuss a number of desirable properties of the TRC scheme and we will make a comparison with other related schemes.

4.1 Scenario: 'Timed Release Clues'

Suppose a scenario where a user (Bob) wants to send some information (three clues) to a recipient Alice. According to Bob, Alice should not learn all the clues simultaneously, but she can read the clues in a sequential order; each clue at a different time-instance (e.g. after an hour). For this purpose, Bob connects to a time-server that provides public time-instant keys for every single hour of the day and gets the public keys for the time instances $T_1 < T_2 < T_3$.[2] As it was referred in the description of our scheme, the public keys are of the form $(nH_1(T), nsG)$. Bob runs the encryption process of our scheme (the $clue_1$ will be decrypted at T_1, the $clue_2$ at T_2 and the $clue_3$ at T_3) and sends the ciphertexts to Alice. Now, Bob can go offline.

As for the server's side, his first job is to select two random integers t_1, t_2 that will be used for the extended private key construction. For security reasons, the server encrypts and stores these numbers using his public key. The next step is to publish the time-instant public keys. For this purpose, he first creates the private keys and then constructs the public ones.

- for T_3 private: $t_1 \oplus t_2$, public: $(t_1 \oplus t_2)H_1(T_2)$, $(t_1 \oplus t_2)sG$
- for T_2 private: $H_2(t_1) \oplus H_2(t_2)$, public: $(H_2(t_1) \oplus H_2(t_2))H_1(T_2)$, $(H_2(t_1) \oplus H_2(t_2))sG$

[2] Black and Chan [3] propose another approach where there is no need for a connection to the server to get the public keys.

- for T_1 private: $H_2^2(t_1) \oplus H_2^2(t_2)$, public: $(H_2^2(t_1) \oplus H_2^2(t_2))H_1(T_2)$, $(H_2^2(t_1) \oplus H_2^2(t_2))sG$

When the desirable time comes, the server publishes the private keys together with the needed extra information.

- for T_1 private: $H_2^2(t_1) \oplus H_2^2(t_2)$ extra info: $H_2^2(t_2)$
- for T_2 private: $H_2(t_1) \oplus H_2(t_2)$ extra info: $H_2(t_2)$
- for T_3 private: $t_1 \oplus t_2$ extra info: t_2

The extra information is important to construct the previous time-instant private keys. For example, if the private key for the time instance T_3 has been published, one can also compute the time-instant private key for the time instance T_2 by executing the following operations: get $t_1 = (t_1 \oplus t_2) \oplus t_2$, compute $H(t_1)$ and $H(t_2)$, compute the T_2 private key $\rightarrow H(t_1) \oplus H(t_2)$. Moreover, as the previous private keys can be constructed by the latest published key, the server can delete them from his database. If a user needs to decrypt a message that could have been decrypted on a previous time instance, he just has to get the current key and apply the operation discussed above.

Alice has to wait until the server reveals the private keys. In case where Alice is offline until the time instance T_3, where all the private keys have been revealed, she will get the latest published key (T_3) with its extra information to construct the keys for T_2 and T_1 respectively. This is the main advantage of our protocol. This scheme is very useful for cases where users receive a big number of messages of different time instances. Furthermore, the server does not have to keep lists of previous private keys and at each time the current private key is enough to construct all the previous ones. The communication cost at the decryption process is minimal and the server can support a bigger number of users who simultaneously request the time-instant private keys.

5 Conclusion

In this paper, we described a Timed Release Cryptography scheme that minimizes the connection cost during the receiving process. This scheme can achieve timed release decryption with a precisely specified absolute release time and is scalable enough since the public and private keys are identical for every user. We also provide a solution to missing time-instant private keys by constructing them from the latest published key. Furthermore, all the keys are recursively authenticated without the need of a server's signature.

5.1 Future Work

Currently, we are working on an efficient implementation of the proposed IBE Timed-Release Cryptography scheme. Our aim is to measure the functionality and possible defects of the proposed protocol. We focus on the time-server's resistance to Denial of Service (DoS) attacks and the problems that occur when a big number of users simultaneously request the current private time-instant key. Additionally, we are

looking for a stable and safe model that will help us to select an appropriate number for the continuous hashed keys (the length of the hash-chain). As it can be seen, we cannot have a big number of dependent keys, because the scheme will be more vulnerable to birthday attacks. A simple approximation is that if the time-quantum between two release times is very small, we can increase the length of the hash chain; otherwise we decrease it.

Another future research is to use multiple time-servers to lower the risk that a receiver colludes with a time-server. In case where the servers use the same generator G and the clients use the same private key, the problem can be solved as follows: Suppose there are three servers with private keys s_1, s_2 and s_3 respectively. The sender's public keys on each server are (aG , as_1G) , (aG , as_2G) , (aG, as_3G) and the receiver's public keys are (bG , bs_1G) , (bG , bs_2G) , (bG, bs_3G). The sender verifies all of the receiver's public keys by checking the equality $\hat{e}(bG, s_xG)= \hat{e}(G, bs_xG)$ for each server s_x. Then, the sender picks a random integer r and calculates the $K_i= \hat{e}(rn_iH_1(T), abG)$ for each $n_iH_1(T)$ (the public time-instant keys for each server). The ciphertext is: $C = < rH_1(T), M \oplus K_1 \oplus \ldots \oplus K_x >$. When the private keys are published, the receiver computes every K_i and then, he is able to decrypt the message.

References

1. M.Bellare and S. Goldwaaser. Encapsulated key escrow. *MIT LCS Tech. Report MIT/LCS/TR-688,* April 1996.
2. M.Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Proc. Of Asiacrypt '00, Lecture Notes in Computer Science, Vol. 1976,* 2000.
3. I. F. Blake and A. C-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. *http://eprint.iacr.org/2004/211/*, 2004.
4. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in Cryptology – Crypto '01, Springer-Verlag LNCS vol. 2139, pages 213-229*, 2001.
5. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proc. Of Asiacrypt '01*, 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. Brandt. Fully private auctions in a constant number of rounds. In *Proceedings of the 7th Annual Conference on Financial Cryptography (FC)*, 2003.
8. J. Cha and J. Cheon. An id-based signature from gap-diffie-hellman groups. In *Public Key Cryptography – PKC 2003*, 2003.
9. L. Chen, K. Harrison, D. Soldera, and N. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *Proceedings InfraSec 2002, Springer LNCS 2437, pp 260-275*, 2002.
10. G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Advances in Cryptology –- Eurocrypt '99*, *Springer-Verlag LNCS vol. 1592, pages 74-89*, 1999.
11. M. K. Franklin and M. K. Reiter. The design and implementation of a secure auction service. In *Proceedings of 1995 IEEE Symposium on Security and Privacy, pp. 2-14, Oakland, California*, 1995.

12. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Proceedings CRYPTO 1999, Springer LNCS 1666, pp 537 - 554*, 1999.
13. J. Garay and M. Jakobsson. Timed release of standard digital signatures. In *Financial Crypto '02*, 2002.
14. J. Garay and C. Pomerance. Timed fair exchange of arbitrary signatures. In *Financial Crypto '03*, 2003.
15. J. A. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *Financial Cryptography '02*, 2002.
16. N. Haller. The S/KEY One-Time Password System. *http://www.rfc-archive.org/getrfc.php?rfc=1760*, 2005.
17. J. T. Harkavy and H. Kikuchi. On cheating in sealed-bid auctions. In *EC'03*, 2003.
18. J. T. M. Harkavy and H. Kikuchi. Electronic auctions with private bids. In *3rd USENIX Workshop on Electronic Commerce, Boston, Mass., pp. 61–73*, 1998.
19. A. Joux and K. Nguyen. Separating decision diffie-hellman from diffie-hellman in cryptographic groups. Available from *http://eprint.iacr.org/2001/003/*, 2001.
20. A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. In *Proceedings PKC 2000, Springer- Verlag LNCS 1751, pages 446-465,* 2000.
21. W. Mao. Timed-release cryptography. In *SAC '01, Springer-Verlag LNCS vol. 2259, pages 342-357*, Aug. 2001.
22. K. H. Marco Casassa Mont and M. Sadler. The hp time vault service: Exploiting IBE for timed release of confidential information. In *WWW2003*, 2003.
23. T. May. Timed-release crypto. Manuscript, *http://www.hks.net.cpunks/cpunks-0/1560.html*, Feb. 1993.
24. R.C Mercle. Secure communications over insecure channels. *Communications of ACM, 21(4):294-299*, April 1978.
25. A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *IEEE Transactions on Information Theory IT-39, 5 (1993), 1639–1646*, 1993.
26. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of ACM Conference on Electronic Commerce, pp. 129–139*, 1999.
27. P. van Oorschot, and M. Wiener. A Known Plaintext Attack on Two-Key Triple Encryption. In *Advances in Cryptology – Eurocrypt '90. New York: Springer-Verlag, pp. 366-377*, 1991.
28. I. Osipkov, Y. Kim, and J. H. Cheon. A Scheme for Timed-Release Public Key Based Authenticated Encryption. Available from *http://citeseer.ifi.unizh.ch/709184.html*, 2004.
29. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and time-released crypto. In *MIT laboratory for Computer Science,MIT/LCS/TR-684*, 1996.
30. Shamus Software Ltd. Miracl: Multiprecision integer and rational arithmetic c/c++ library. Available from *http://indigo.ie/mscott/*.
31. Marcus Stögbauer. Efficient Algorithms for Pairing-Based Cryptosystems. *Diploma Thesis: Darmstadt University of Technology, Dept. of Mathematics*, Jan 2004.
32. P. F. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *1998 IEEE Computer Security Foundations Workshop (CSFW11)*, 1998.
33. G. Yuval, How to Swindle Rabin. *Cryptologia 3, pages 187-189*, Jul. 1979.