"On the Way of Making Data Disappear"

Spyridon Antakis, Joël Stemmer

Henri Hambartsumyan, Louisa Papachristodoulou

University of Twente
Department of Mathematics & Computer Science
Enschede, The Netherlands, P.O. Box 217, 7500 AE
{s.antakis, h.hambartsumyan, l.p.papachristodoulou}@student.tue.nl
j.stemmer@student.utwente.nl

December 20, 2009

Abstract

As digital communication continuously evolves, several newborn privacy issues are coming onto the surface. Most of the time, the transmitted data may include sensitive information that should not be processed by third parties. However, at the time there are hardly any security guarantees and as the phenomenon of Cloud Computing flourishes, a need for new protection mechanisms arises. This paper discusses three of the existing techniques that aim to ensure the data privacy: *i) Ephemerizer*, *ii) Timed-Ephemerizer* and *iii) Vanish*. It performs a comparative assessment between the security properties that these systems are able to deliver and it presents a demo implementation of the *Timed-Ephemerizer* system. Finally, the paper concludes about the efficiency, the functionality, the advantages and the disadvantages that these systems provide.

Keywords: *Ephemerizer, Timed-Ephemerizer, Vanish, security, data storage, privacy.*

1 Introduction

The fast growth of *Cloud Computing* and the new services which are offered to users, create a new demand for advanced methods that will be able to ensure the data privacy. More precisely, there is a great need for making data inaccessible under specific circumstances, since this is the only way to fulfill some of the privacy requirements. Until now, there has been done research in this area which resulted in a number of interesting solutions being proposed. This paper gives a brief introduction to the existing systems and at the same time performs an in-depth analysis on their security properties. Finally, the paper presents an implementation that was developed for one of the discussed systems, as a proof of concept.

1.1 Existing Systems

In 2005, Radia Perlman introduced the idea of the *Ephemerizer* system [1]. The Ephemerizer was a *client-server* approach that was meant to solve some of the critical privacy issues and aimed to constitute data inaccessible permanently when it was not needed anymore. The main concept was based on the destruction of cryptographic keys after a predefined time span. More precisely, it was suggested that the management of the keys used for encryption and decryption of the data was to be performed by a server called the *Ephemerizer*. The Ephemerizer was aimed to be the only one responsible for ensuring the destruction of expired cryptographic keys after a selected time period. In this way, after the passage of the desired expiration date the recovery of the data would become infeasible, regardless of the operations performed on the clients side. Of course, in order for this to be possible, a number of assumptions have

been made about the management of generated cryptographic keys and the ways that data should be stored. In practice, R. Perlman presented two main techniques in order to implement the Ephemerizer system: *i) Triple Encryption* and *ii) Blind Decryption*. However, even if the author gave a complete direction for the security properties that should be fulfilled by both approaches, the design did not provide any formal security analysis for the described protocols.

As an extension on this first proposal and based also on the findings that were published in 2007 by Nair *et al.* [2], a new variation of the Ephemerizer system was proposed. In 2009, Q. Tang presented an interesting approach for the Ephemerizer system, called the *Timed-Ephemerizer* [4]. This new system was mainly the result of a formal study of the security protocols that could be used for an Ephemerizer implementation. In practice, the Timed-Ephemerizer combined two already existing approaches, the *Ephemerizer* and *Timed-Release Encryption*, concluding in an original hybrid system. In comparison to all other conducted surveys, Q. Tang performed a security analysis on a proposed security model and he demonstrated through formalized proofs that his model is able to provide a number of security guarantees that Ephemerizer was neglecting.

In 2009, Geambasu *et al.* presented a different system, called *Vanish* [4]. This system was aimed to increase data privacy, by using a *self-destruction* technique for sensitive data. The main concept was based on the usage of symmetric key encryption in addition with Shamir's secret sharing approach. The sensitive data was encrypted with a symmetric key, then this key was divided into a number of shares and these shares were distributed among the nodes of a P2P network. The basic idea was that when a significant subset of nodes leave the P2P network then the sensitive data would become unrecoverable, since the reconstruction of the symmetric key would not be possible anymore. Even if Vanish was able to achieve a number of goals that could be useful in the modern digital world, it was missing the ability to provide a precise expiration time, which the Ephemerizer offered. Moreover, in late 2009, S. Wolchonsky *et al.* [5] have discovered ways to defeat Vanish system with *Low-Cost Sybil Attacks* and more precisely they demonstrated that Vanish is vulnerable, since the recovery of the symmetric key is feasible.

In the following sections, a description of these systems is presented and an overview of the provided security properties that each method is able to deliver is given. Moreover, the effectiveness that its design offers is discussed and the provided security guarantees are assessed. As a final outcome, a demo implementation based on the *Timed-Ephemerizer* is presented and the levels of *functionality* and *efficiency* that this application demonstrates are reported.

2 Description of the Systems

The Ephemerizer is a system that involves two entities, a *trusted server* called respectively the *Ephemerizer* and a compatible *client application* that operates on the side of each user. Likewise, the Timed-Ephemerizer system consists of these two entities that the Ephemerizer provides, but in addition includes an extra entity, the *time server*. In this way, compared with the Ephemerizer, it enables users to securely perform cryptographic operations only during a predefined time span and it constitutes a tool which can enforce information life-cycle management in outsourcing activities. On the other hand, Vanish is a totally different approach that aims to build a *zero-effort* system for the users, which will be able to constitute data unreadable after a specified time. In practice, the main purpose of Vanish targets on a global scale adoption system, which will make use of *Distributed Hash Tables (DHTs)*, in order to destroy the unneeded data. The following sections contain a brief introduction to the core operations of each system and gives a better insight on their security features.

2.1 Ephemerizer - Triple Encryption

Triple encryption is the first proposed protocol for the Ephemerizer and its security is based on the following assumptions, that R. Perlman made in [1]:

- *The Ephemerizer possesses a long-term signing key and if this key is compromised, then the system could recover by the means of a PKI revocation.*
- *The key on the receiver's side is a long-term key and thus the communicated messages between receiver and the Ephemerizer should be secured.*

- *The software which the sender and receiver are using is configured in such a way that no instance of data is stored after cryptographic operations on the users side.*
- *The Ephemerizer's private keys are stored in a tamper-resistant device, such as a smart card.*

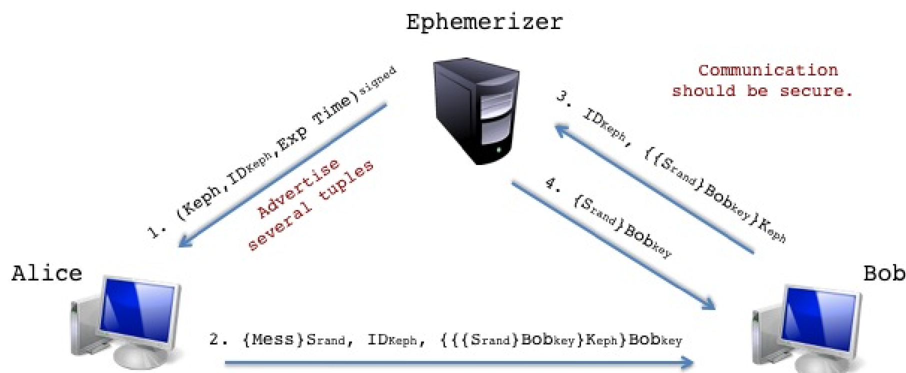

Figure 1: *Triple Encryption*

Suppose that *Alice* wants to send a message to *Bob*, by using the triple encryption protocol. As it is presented above in figure 1, *Alice* will choose one of the tuples that the Ephemerizer advertises based on the desired expiration date and she will verify if indeed the tuple is genuine, through the Ephemerizer's assigned signature. Of course, if at this point an attacker successfully compromises the long-term signature key of the Ephemerizer, then he will be able to pretend the Ephemerizer in order to fool *Alice*, until the signature key is revoked. However, the attacker will still not be able to read any of the messages that were encrypted with an ephemeral key that expired before the compromise.

After *Alice* obtains the ephemeral key, she will compute a random secret key S_{rand} and then she will use this key to encrypt the message that she wants to send to *Bob*. Then, *Alice* will triple encrypt the S_{rand} key with *Bob's* public key and the ephemeral key as follows : $\{\{\{S_{rand}\}Bob_{key}\}K_{eph}\}Bob_{key}$ and she will send it to *Bob*, together with the communicated message {Mess} encrypted with the S_{rand} key and the corresponding identity $ID_{K_{eph}}$ (figure 1 - step 2). The inner encryption performed with *Bob's* public key is very vital for the security of the protocol. If the Ephemerizer is dishonest and the public key of Bob is not used, then the Ephemerizer will have knowledge of the random secret key S_{rand} and by simply eavesdropping the communication between *Alice* and *Bob* it will be able to obtain *Alice's* original message, by decrypting $\{\text{Mess}\}S_{rand}$.

Upon receiving the transmitted messages, *Bob* will have to contact the Ephemerizer in order to recover the random secret key that *Alice* used for encrypting the message. More precisely, *Bob* must decrypt the triple encrypted tuple and send the decrypted part to the Ephemerizer, together with the corresponding $ID_{K_{eph}}$. Then, the Ephemerizer will check if the private key, which corresponds to $ID_{K_{eph}}$, has expired and if not, it will decrypt the received part with the valid private key and send back to *Bob* the following part: $\{S_{rand}\}Bob_{key}$. After that, *Bob* will decrypt this part and he will obtain the random secret key that *Alice* sent. In this way, he will finally be able to recover the original message. The communication that takes place between the Ephemerizer and *Bob*, is of great importance for the security of the protocol. If an attacker eavesdrops $\{\{S_{rand}\}Bob_{key}\}K_{eph}$, then he can request from the Ephemerizer to decrypt it and thus obtain the following $\{S_{rand}\}Bob_{key}$. In this way, by the initial assumption that *Bob's* public key is a long-term key and will not expire, it becomes obvious that an attacker at some point will be able to recover S_{rand}, due to coercion or *Bob's* carelessness. A proposed way in order to solve this problem, is either to use a protocol that is able to support *perfect forward secrecy* (e.g. IPSec's IKE or SSL variant) or secure the message exchange with the ephemeral key K_{eph}, as described by R. Perlman in [1].

The fact that the Ephemerizer has to perform more than one decryption operation per message exchange and the need for applying additional security protocols in parts of communication, may constitute an efficiency drawback for a real implementation. The security that this protocol provides may be sufficient under the stated assumptions, however the computation cost on the side of the Ephemerizer should be properly considered. The triple encryption protocol involves quite expensive computations,

which will become even more severe when the number of users is relative large. Thus, in our opinion, a number of implementation experiments should be performed in order to evaluate the efficiency that this scheme is able to deliver and assess the effectiveness of the proposed security properties.

2.2 Ephemerizer - Blind Decryption

Blind decryption is the second protocol that was proposed by R. Perlman for the Ephemerizer implementation. It uses the concept of *blinding signatures* applied to encryption and decryption operations and it is possible to be implemented either with *asymmetric* or *symmetric* cryptography, by using different methods [1]. More precisely, *blind decryption* is based on the ability to have inverse functions for *encryption* and *decryption* and inverse functions for *blinding* and *unblinding*. In this way, the protocol performs *encryption* + *blinding* to secure a message and *decryption* + *unblinding* to recover the message. Blind decryption protocol is a design that makes use of this functionality and it is based on the following security assumptions:

- *The users should verify that ephemeral key is certified by the Ephemerizer's long-term private key, where the corresponding long-term public key is certified by a Trusted Third Party (TTP).*
- *There is no need for the Ephemerizer and the user to authenticate each other and there is no need to protect the integrity of the ephemeral key transmitted between the users.*
- *The software which the sender and receiver are using, is configured in such a way that no instance of data is stored after cryptographic operations on the users side.*
- *The Ephemerizer's private keys are stored in a tamper-resistant device, such as a smart card.*

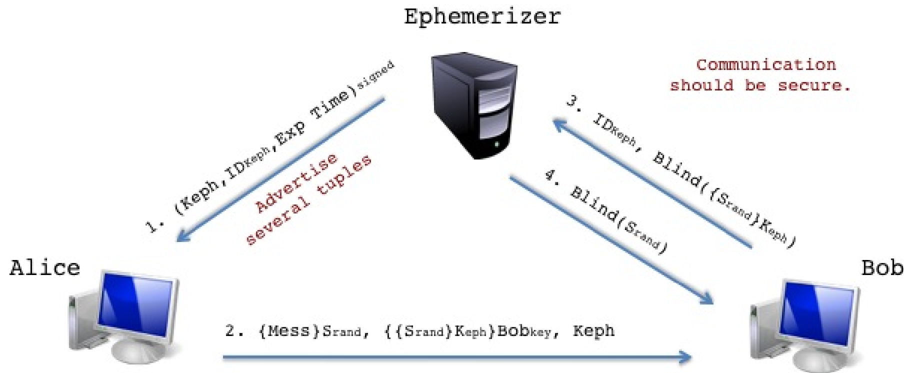

Figure 2: *Blind Decryption*

Suppose that *Alice* wants to send a message to *Bob* by using the *blind decryption* scheme, as shown in figure 2. Again, *Alice* will choose one of the tuples that the Ephemerizer advertises, based on the desired expiration time for the ephemeral key. After that, *Alice* will validate that indeed the received ephemeral key K_{eph} is genuine and then she will be ready to send her message to *Bob*. She will generate a random secret key S_{rand} and she will encrypt her message with this key. Afterwards, she will also encrypt the key S_{rand} with the obtained ephemeral key and also with *Bob's* public key. Finally, *Alice* will send to *Bob* all the encrypted messages and the ephemeral key K_{eph} transmitted in the clear (figure 2 - step 2). On the reception of the transmitted messages, *Bob* will create a *blind* and *unblind* function based on K_{eph} and he will decrypt $\{\{S_{rand}\}K_{eph}\}Bob_{key}$. Then, *Bob* will send to the Ephemerizer the decrypted part blinded as follows Blind($\{S_{rand}\}K_{eph}$), together with the corresponding $ID_{K_{eph}}$. The Ephemerizer after the reception of the messages, will certify that the ephemeral key with corresponding identity $ID_{K_{eph}}$ has not expired and then it will decrypt the blinded message and send it back to *Bob*. At this point, *Bob* will *unblind* the received message, obtaining the random secret key S_{rand} which will give him the ability to decrypt *Alice's* original message.

R. Perlman claimed that there is no need to protect the *integrity* of the ephemeral key [1]. She mentioned that if an attacker successfully modifies the transmitted ephemeral key, that could only lead to a *denial service attack* and not to the disclosure of the secret key S_{rand}. More precisely, the author reported that if the K_{eph} was not genuine, then *Bob's* request to the Ephemerizer will fail, since the Ephemerizer will not be able to decrypt the corresponding message. However, this is not entirely true and it is based mainly on the assumption that Ephemerizer can always be *trusted*. According to [3], it was shown that it is possible to recover the secret key S_{rand} if the Ephemerizer is *curious*. In practice, by eavesdropping the communication between *Alice* and *Bob* the Ephemerizer can successfully modify K_{eph} in such a way, that when he receives the blinded message from *Bob* it will be able to recover the secret key S_{rand}. The critical point in this possible vulnerability is the fact that the ephemeral key K_{eph} is only required to be certified by the private key of the Ephemerizer and a possible solution would be to directly certify the ephemeral key from a *Trusted Third Party (TTP)* [3].

In comparison to the triple encryption scheme, blind decryption may be more efficient since the Ephemerizer has only to perform one decryption. However, there are still efficiency drawbacks, since the Ephemerizer is responsible for publishing and certifying all the potential ephemeral key to the users. Considering also that a wide range of expiration times should exist, the task of providing an adequate level of efficiency becomes even more fuzzy.

2.3 Timed-Ephemerizer

As it was already mentioned in the previous sections, Timed-Ephemerizer is a variation of Ephemerizer which aims to guarantee that the data will only be available during a *predefined time span*. It includes an additional entity called a *time-server*, which has the responsibility of publishing time-stamps periodically and constitute the users able to access data during a valid time span. Of course, a trusted Ephemerizer server is still required in order to periodically publish and revoke *ephemeral public/private* key pairs. In practice, an Ephemerizer protocol is able to provide assured deletion, but not initial assured disclosure. The inclusion of the *time server* assures an initial disclosure and completes the usability of the design in a *cloud computing* environment.

At this point, someone could argue that there is no need for a time server, since the Ephemerizer could be designed in such a way that will be possible also to release the desired timestamps. This claim can be considered partially true, however as it was also mentioned in [3], by having a separation of functionalities is possible to provide a higher-level of security for the implementation. The risk for compromising either the time-server or both the time-server and Ephemerizer is reduced and by taking into account also the fact that time server does no need to interact with the other entities, the design for the Timed-Ephemerizer system could become safer.

Timed-Ephemerizer protocol ensures that even if an adversary compromises all the private keys of the system, the data will be available only during a *predefined time span*. The proposed scheme [5], applies the idea of *blind decryption* with some modifications and succeeds on constructing a functional and secure Timed-Ephemerizer protocol. More precisely, it was proposed to first encrypt data using the ephemeral public key of the Ephemerizer server and the public key of the *time server* and then re-encrypt the produced ciphertext by using the public key of the user (e.g. Alice). Afterwards, the user (e.g. Bob) would recover the data by decrypting the re-ecrypted ciphertext and obtaining in this way the ciphertext created from the public keys of the two servers. After that, the user would send this ciphertext for decryption in a *re-randomized* form based on the XOR operation to the Ephemerizer. On the reception of the decrypted data that Ephemerizer would send, the user would apply again the re-randomization and thus recover the original message.

In section 3, a demo implementation that was developed based on the Timed-Ephemerizer proposal, will be presented and the operation of this protocol will be further analyzed. A description of a possible real design will be given and all the security properties that must be considered during an implementation phase will be discussed in detail. In this way, the reader will be able to comprehend in more depth the features that this system is able to deliver.

2.4 Vanish

Vanish is a system that aims to automatically destruct any data that is no longer useful. The basic idea behind the Vanish implementation is based on the *peer-to-peer (P2P)* infrastructures and in particular

in the large-scale *Distributed Hash Tables (DTHs)*. By comparing Vanish with an Ephemerizer-based design someone is able to reveal several differences. In [4], the authors stated that Vanish is a better approach, since users with Vanish put their *trust* in an entire P2P network and not only in one entity, such as the Ephemerizer. This assumption may be true, however the Vanish design lacks some important features that Ephemerizer is able to provide.

By using an Ephemerizer-based design, it is possible to determine the desired expiration time for the sensitive data, however with Vanish you cannot specify the precise time of expiration. The fact that the recoverability of the symmetric key is based on the participation of P2P nodes, leads to two major concerns with respect the balance between the *privacy* and the *availability* of the sensitive data. Imagine a case in which the P2P nodes disconnect from the network earlier than what was expected. In such a case, the sensitive data would become *unavailable* or *unrecoverable* even if they are still needed and thus the property of availability will not be preserved. On the other hand, if we suppose that the P2P nodes disconnect from the network later than what was expected, then in this case the sensitive data would be *recoverable* or *available*, instead of being destroyed. Another consequence of relying in the P2P nodes for recovering the symmetric key through the means of threshold cryptography, is the fact that in case of a *Denial of Service (DoS)* attack on the P2P network itself, the Vanish system will become *unavailable* and probably some of the sensitive data *unrecoverable* before the desired time. In contrast, Ephemerizer-based designs are better protected against DoS attacks, since they are based on trusted entities (e.g. Ephemerizer) and thus, such attacks are much harder to be successfully applied.

Finally, maybe the most important drawback for choosing the Vanish approach for ensuring the privacy of sensitive data, is that the system was proved vulnerable in *sybil attacks*, against the used DTHs [5]. More precisely, Geambasu *et al.* showed that Vanish in its current form cannot withstand sybil attacks, which are referring to the existence of few malicious nodes inside the P2P network that create a large number of identities in the DTHs. Usually, this kind of attacks depend on how easily sybils can be generated and accepted as an input from other entities, when they do not have a chain of trust linking them to a trusted entity. In the case of Vanish, was proved that sybils attacks are able to successfully recover the symmetric key used and thus constitute the sensitive data accessible from unauthorized entities.

Summarizing, we could say that Vanish at its current form is unable to provide the needed security guarantees that such kind of systems are requiring. Moreover, in a general comparison with ephemerized-based schemes, it seems much more unreliable and not ready to manage the delicate operation of destroying sensitive data, according to the desired privacy needs that each user defines. However, it is a promising idea that is worth to be researched further, taking into account all the recommendations that have already been proposed for additional improvement.

3 A proof of concept

To demonstrate how an implementation of the Timed-Ephemerizer would work we have created a proof of concept. Each of the applications will require at least one asymmetric key pair. The private keys need to be kept safe and the public key will have to be published to each party. This involves a lot of key management which falls out of the scope of this project. Therefore, the demonstration applications either use a simplified form of asymmetric encryption or none at all. All applications will log the various actions that take place, such as new connections being setup or messages being encrypted.

3.1 Time server

The time server has the responsibility of publishing timestamps which will be used by the all the application as the time reference. It sole purpose is to publish accurate timestamps upon request. The time server should have a public and private key pair which is used for signing every timestamp published. Other parties can then verify that the timestamps were in fact generated by the time server. In this proof of concept, the key pair generation and signing of messages is omitted for the sake of simplicity.

The response to a valid time query will always contain the current timestamp. In addition, if the timestamp supplied in the request, is a timestamp in the past (e.g. $RequestedTimestamp \leq CurrentTime$) then the time server will respond with this timestamp. If it is a timestamp in the future, then a 0 will be returned. See commands $C2$ and $C3$ in Table 1.

3.2 Ephemerizer server

The Ephemerizer server is responsible for encrypting and decrypting user messages. It will wait for any incoming requests and process them. The Ephemerizer Server must be configured to use a time server, otherwise it will refuse to do any *encryption* or *decryption*. When correctly configure and started, the server will handle two types of requests, namely an encryption request and a decryption request.

3.2.1 Encryption

See commands $C4$ and $C5$ in Table 1. An encryption request consists of a disclosure timestamp, an expiration timestamp and a plaintext message. The Ephemerizer will first validate the timestamps. The expiration timestamp must be larger than the disclosure timestamp. If the expiration timestamp is in the past, the encryption request will be denied and an error message will be returned. The Time server will be queried each time the Ephemerizer needs the current timestamp. Because of delays in the network, slight timing errors might occur. In extreme cases this could lead to encryption requests being denied because the expiration time has been reached during the request. If the Ephemerizer is satisfied that the timestamps are valid, it will use these timestamps to either generate a new key pair or use an existing key pair. These asymmetric key pairs will be associated with the disclosure and expiration timestamps. In practice, this means that every unique *disclosure/expiration* combination will result in a new key pair being generated. Once they keys have been created, they are stored in a database along with their corresponding timestamps. The plaintext message supplied in the request will then be encrypted using the public part of this key pair. Finally an encryption response will be send back to the user containing the ciphertext.

3.2.2 Decryption

See commands $C6$ and $C7$ in Table 1. When the Ephemerizer receives a decryption request it will first try to look up the corresponding key pair associated with this ciphertext. If no keys are found we assume that they have expired and were deleted, the user is informed accordingly with an error message. If however the database did contain this key, the corresponding timestamps are retrieved. With the help of the Time server, these timestamps are checked. In the case that the disclosure time has been reached and the expiration time not yet, the message is decrypted and the result is send back to the user. In the case that the disclosure time has not yet been reached, an error message will be returned. Finally, there is the possibility that the keys have expired. Even though expired keys are regularly purged from the database, a key could have expired right after one of these checks. The Ephemerizer will delete this entry from the database and respond to the client with an error message.

As described earlier, we have somewhat simplified the encryption and decryption. This way we will not have the difficulty of generating, managing and publishing all the keys used during the operation of the Ephemerizer Server. Since the database and protocol changed a lot during development we needed an easy way to quickly encrypt or decrypt messages. We have therefore decided on the following dummy encryption and decryption methods:

Keys are denoted as PK_xyz or SK_xyz for respectively the public part or secret part of an asymmetric key. Encryption encloses the given plaintext in curly brackets and adds the key used for encryption to the end, as shown in Equation 1. Decryption will remove the curly brackets again, but only if the key supplied to the function is the inverse key of the one used for encryption, as shown in Equation 2.

$$\mathit{encrypt}(\texttt{"message"}, \texttt{PK_a}) \Rightarrow \texttt{"\{message\}PK_a"} \quad (1)$$

$$\mathit{decrypt}(\texttt{"\{message\}PK_a"}, \texttt{SK_a}) \Rightarrow \texttt{"message"} \quad (2)$$

Consequently, by using these dummy methods, the Ephemerizer server can easily find the corresponding keys for a ciphertext when handling a decryption request. However, when real encryption and decryption is used, the protocol will have to be modified to include extra information about the key that was used for encryption. Otherwise, the Ephemerizer would not be able to find the keys and corresponding timestamps for the given ciphertext.

3.3 Client

The Ephemerizer client will be the actual application that will be distributed to all the users. Each client will require its own key pair, of which the public part will be published to the other users. As mentioned earlier, the usage of the asymmetric keys and the actual encryption is omitted. The Client application allows the user to configure what Ephemerizer server will be used. It is important that the users who will communicate use the same server.

The client has two input areas where data can be entered. On the left side is the data encryption input, where the user can enter any text. The user must also indicate the preferred disclosure time and date and the preferred expiration time and date. In this case, each selected timestamp will be accepted by the server as long as they are valid. But it is probably more likely in a real world application that the server uses buckets instead of exact timestamps, or uses a fixed set of timestamps, in which case the user must be informed of the actual disclosure and expiration timestamps. When the user is satisfied with his message and selected timestamps, the *Encrypt* button will initiate a new connection to the configured Ephemerizer Server and send an encryption request as described in Section 3.2.1. When successful, the ciphertext from the encryption response will be put in the right hand input field. This field is also used for decryption. The user can enter any ciphertext in that field and press the *Decrypt* button. This will again initiate a connection to the Ephemerizer Server and send a decryption request. If successful, the resulting decrypted message will be put in the input field on the left. A messagebox will be shown when an error has occured, describing the nature of the problem.

3.4 Protocol

The applications communicate with each other using a simple protocol. A single request can be made for each connection. This means that after a TCP connection is made, a single line of text can be send. Then the response will be send back, after which the connection will be closed again. The protocol itself is done in ASCII text only, on a single line. All commands and parameters are separated by a single space, with the exception of an error message (see Table 1). Data which contains multiple lines of text or binary data will require to be encoded in Base64. If something goes wrong at any point during the communication an error message will be send describing the problem. This can be due to several problems such as an unknown command, a malformed message, unable to query the time server, unable to encrypt or decrypt, et cetera.

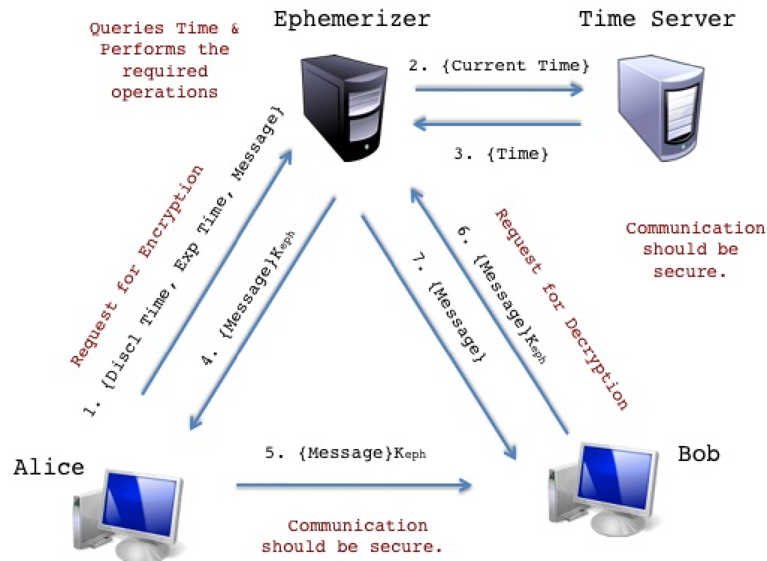

Figure 3: *Protocol example*

In order for the reader to better understand how the various parts of the system work together, we take a closer look at the protocol as presented in Figure 3. Suppose *Alice* wants to send a message to

Bob that he is only able to read at a specific time span. *Alice* starts by creating a new message and configuring the disclosure time and expiration time. She then sends an encryption request (step 1) to the Ephemerizer server. The Ephemerizer in turn will accept this request and check the current time with the time server (steps 2 & 3). If the expiration time is in the future, the server will look for an existing key pair for these particular timestamps or create a new pair if one does not exist. The message from *Alice* is then encrypted using the public part of this key and the resulting ciphertext is send back to *Alice* (step 4). *Alice* can now send *Bob* the encrypted message (step 5). When *Bob* is ready to decrypt the message, he will send a decryption request to the Ephemerizer containing the encrypted message (step 6). The Ephemerizer will then lookup the key used for encryption and the associated timestamps. After querying the time server again (steps 2 & 3) and verifying that the disclosure time has been reached and the expiration time has not been reached, it will decrypt the message. Finally, *Bob* will receive a reply containing the original message (step 7).

	Command and parameters	Description
C1	`error <message>`	This is an error message. The `message` contains a description of the error.
C2	`time <timestamp>`	This is a time request. `timestamp` must be a positive number.
C3	`timestamp <current> <timestamp>`	This is a time response. `current` contains the current time, `timestamp` contains the requested timestamp if and only if this timestamp is not in the future, otherwise it is 0.
C4	`encrypt <disclosure> <expiration> <data>`	Encryption request. `disclosure` contains the timestamp when the message can be disclosed. `expiration` is the timestamp when the decryption keys will be destroyed, making the message unrecoverable. `data` contains the entire message, encoded in Base64.
C5	`encrypted <ciphertext>`	This is the encrypt response. `ciphertext` is the ciphertext of the original message, encrypted with the keys associated with the timestamps given in the encryption request.
C6	`decrypt <ciphertext>`	The decryption request. `ciphertext` is the encrypted message.
C7	`decrypted <plaintext>`	The decryption response. If a key pair associated with this message has been found and the disclosure time has passed, then `plaintext` will contain the original message. Note that no keys will be found for an expired message, since these will have been deleted.

Table 1: Communication protocol commands

3.5 Future Work

We have learned several things in the process of creating this proof of concept. The most obvious is the lack of proper key management. The *time server* and *client* application are relatively easy to be modified since they only require a single key pair. However, the Ephemerizer requires further investigation and a proper key management proposal. The current solution is not scalable, since a new key pair is generated for every new combination of disclosure and expiration times. This could quickly lead to a performance bottle neck. It could also be the target of a Denial of Service attack. It is probably better to let the Ephemerizer generate key pairs with a variety of disclosure and expiration times and publish these together with their respective timestamps. The added benefit is that the client will then be able to encrypt his own messages, by using the published public key for the timestamps he requires. This releases the Ephemerizer from the encryption process and once a proper key management strategy has been created, the time server should be updated to sign all published timestamps with his secret key. This allows anyone to verify the timestamps. Finally, the users themselves will have to have their personal key pairs so that messages can be exchanged securely between them. It will also enable the use of triple encryption, so the content of the messages will not be exposed to the Ephemerizer.

While developing these demonstration applications, we have decided to use a certain protocol. In retrospect, the protocol started to resemble the way the *Hyper Text Transfer Protocol (HTTP)* is used. Therefore it is probably a good idea to use HTTP in future applications. This in fact means the time server and ephemerizer can be offered as a web service. All data could then be exchanged using XML, which also supports encrypted documents. There are a wide variety of applications, programming languages and frameworks to create these web services. This will allow you to focus more on the functionality itself and move away from the low level part of creating your own connections and communication protocol. The communication in this proof of concept was deliberately not secured for testing purposes. One could rely on VPN or SSH tunneling to set up secure connections, but when creating a web service you would have the ability to make use of SSL for secure channels.

4 Conclusions

This paper discussed the most important conducted researches which were focusing on ways of data destruction for privacy reasons. More precisely, it presented three innovated systems, namely the *Ephemerizer*, *Timed-Ephemerizer* and *Vanish*. In the last section, the paper described a demo implementation which was based on Timed-Ephemerizer and showed all the difficulties that arise when you have to apply the theory in a functional application.

The needs of the digital world are changing rapidly and new privacy concerns arise. The already existing systems seem promising, nevertheless several concerns that must be evaluated are coming into the surface. Apart from an effective security scheme, such systems should also be *scalable* and *efficient*. As in every cryptographic implementation, the biggest challenge is to perform an efficient key management and define the proper trust relations among the involved entities. All of the described systems included a number of assumptions that in a real life implementation may be difficult to be fulfilled or sometimes impossible. In practice, a prototype proposal should be always strictly connected to its possible transformation into a real implementation. Most of the time, implementation requirements hide things that cannot be predicted when a system is theoretical designed. A thorough investigation is always needed and only then there is good possibility to develop an implementation that would be embraced by the users' community.

In conclusion, the idea of destroying the cryptographic key after a predefined timescale, in order to constitute data *unrecoverable*, is really promising. However, there are many issues that should be re-considered before these designs could be formally applied. In our opinion, a reference implementation and additional research is needed with respect to both areas of *security* and *effeciency* in a real life scenario. Only then would these systems have a chance to be adopted and evolve through time.

References

[1] R. Perlman, *"The Ephemerizer: Making Data Disappear"*, Technical Report TR-2005-140, Sun Microsystems, Inc., 2005.

[2] S. K. Nair, M. T. Dashti, B. Crispo, and A. S. Tanenbaum, *"A Hybrid PKI-IBC Based Ephemerizer System'*, Proceedings of the IFIP TC-11 22nd International Information Security Conference (SEC 2007) ,Volume 232 of IFIP, pages 241252. Springer, 2007.

[3] Q. Tang, *"Timed-Ephemerizer: Make assured data appear and disappear"*, In Proceeding of Public Key Infrastructure, 5th European PKI Workshop: Theory and Practice (EuroPKI 2009).

[4] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy, *"Vanish: Increasing Data Privacy with Self-Destructing Data"*, In Proc. of the 18th USENIX Security Symposium, 2009.

[5] S. Wolchoky, O. S. Hofmanny, N. Heninger, E. W. Felten, J. A. Hademan, C. J. Rossbach, B. Waters, and E. Witchel, *"Defeating vanish with Low-Cost sybil attacks against large DHTs"*, Technical report, University of Texas, 2009.