A Hybrid PKI-IBC Based Ephemerizer System

Srijith K. Nair[1], Mohammad T. Dashti[2],
Bruno Crispo[1 3], and Andrew S. Tanenbaum[1]

[1] Dept. Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
`{srijith,crispo,ast}@few.vu.nl`
[2] CWI, Amsterdam, The Netherlands
`dashti@cwi.nl`
[3] DTI, University of Trento, Italy

Abstract. The concept of an Ephemerizer system has been introduced in earlier works as a mechanism to ensure that a file deleted from the persistent storage remains unrecoverable. The principle involved storing the data in an encrypted form in the user's machine and the key to decrypt the data in a physically separate machine. However the schemes proposed so far do not provide support for fine-grained user settings on the lifetime of the data nor support any mechanism to check the integrity of the system that is using the secret data. In addition we report the presence of a vulnerability in one version of the proposed scheme that can be exploited by an attacker to nullify the ephemeral nature of the keys. We propose and discuss in detail an alternate Identity Based cryptosystem powered scheme that overcomes the identified limitations of the original system.

1 Introduction

Privacy concerns have brought the question of reliable deletion of private data into sharp focus. One of the most effective tools used in privacy invasion is data recovery from persistent storage devices, even when the user had 'deleted' the data. To mitigate this risk, expensive secure hardware devices are sometimes used to keep information private and irrecoverable when deleted and extensive and meticulous process of erasure is followed to ensure that deleted files are indeed completely deleted. For example, US government specification calls for overwriting non-classified information three times [1]. Common users, however, do not have the resources nor the know-how to use these processes or equipments.

One of the techniques used to secure data is to encrypt the data on user's local storage device and to store the decryption key at a trusted remote storage. This way, both the local as well as the remote storage will have to be compromised before the data can be obtained. Furthermore, secure deletion of the decryption key renders the data unrecoverable. However, key management becomes a complicated issue in this approach. As a solution, Perlman [2] introduced the concept of an *Ephemerizer server* which is entrusted with the duty

Please use the following format when citing this chapter:

Nair, S., Dashti, M., Crispo, B., and Tanenbaum, A., 2007, in IFIP International Federation for Information Processing, Volume 232, New Approaches for Security, Privacy and Trust in Complex Environments, eds. Venter, H., Eloff, M., Labuschagne, L., Eloff, J., von Solms, R., (Boston: Springer), pp. 241–252.

to manage the availability and secure deletion of the keys used in decryption of the encrypted data.

To formulate the problem more precisely, assume that Alice wants to send a message M to her confidant Bob at time t_0 with following expectations: (1) only Bob will be able to read the plaintext and (2) after time t_1 ($t_1 > t_0$), no one including Bob can read the plaintext. Bob is assumed to be non-malicious and a willing party in the exchange, but is not trusted to have the capability to securely delete the message after t_1. An attacker, who wants to gain knowledge of the message exchanged, is assumed to have the capability to break into the computer system of any ordinary user and seize all equipments, extract any data from persistent storage that is presently stored or previously deleted or kept encrypted, including forcing the user to disclose passwords used to secure decryption keys. We also assume an open communication medium in which all transmissions can be intercepted, recorded, modified, and retransmitted.

We describe the basic working of the original Ephemerizer system, as proposed in [2], in Section 2 and then explain in detail one of the proposed versions of the scheme. We show the existence of a vulnerability in this protocol, which allows an attacker to subvert the ephemeral nature of the system. We also identify some shortcomings associated with the scheme, which we consider to be crucial for the security of the system. We then propose a modified Ephemerizer scheme based on Identity Based Public Key Cryptography (IB-PKC). As far as we are aware of, the scheme proposed in this paper is one of the very few systems that exploit the power of IB-PKC, without suffering the associated disadvantages. The cryptography primitive is briefly explained in Section 3 while the proposed scheme is presented in Section 4. In Section 5 the security aspects of the proposed scheme is discussed in detail and we show that our scheme can support richer security features than the original scheme. We conclude in Section 6.

2 The Ephemerizer System

The Ephemerizer as proposed in [2] acts as a central server that allows parties to keep data private for a finite time period and then make it unrecoverable after that. Alice and Bob executes the protocol steps with the participation of this trusted third party. This concept was later used in a system designed to provide assured delete [3].

The underlying idea behind the scheme's working is that Alice would send the message to Bob encrypted with a key that needs to be fetched from the Ephemerizer. The Ephemerizer would check for the expiry date associated with the key's usage before responding to the request from Bob. The original paper presented two versions of the scheme - one using triple decryption and the other using blind decryption. We discuss the triple decryption method in detail next since our analysis has identified an attack against this version of the protocol, which is presented later in Section 2.2.

2.1 Triple Encryption Method

Throughout the rest of the paper, we use $\{M\}_{K_A}$ to denote asymmetric key encryption of M with public key of entity A and $[M]_K$ to denote symmetric key encryption of M with symmetric key K. We assume the existence of a trusted public key infrastructure (PKI) from which users can obtain certified public keys of other users.

Step 1 - The Ephemerizer E creates sets of asymmetric key pairs, associate them with different expiration time and advertises tuples - (public key, key ID, expiration time).

Step 2 - Alice chooses one of the keys, K_{eph}, based on the expiration time she requires, and encrypts the data M with a random secret key S to obtain $[M]_S$. She then encrypts S with Bob's long-term public key (K_{bob}) and the result with K_{eph}.The resulting value is encrypted again with a random session key T to get $[\{\{S\}_{K_{bob}}\}_{K_{eph}}]_T$. T is then encrypted with Bob's public key and an integrity check value [4] HMAC(T,$\{\{S\}_{K_{bob}}\}_{K_{eph}}$) is calculated. Finally, Alice sends the following to Bob:

$$A \rightarrow B : \{T\}_{K_{bob}} \quad [\{\{S\}_{K_{bob}}\}_{K_{eph}}]_T \quad [M]_S \ keyID \ K_{eph} \ HMAC(T \ \{\{S\}_{K_{bob}}\}_{K_{eph}})$$

Step 3 - Bob, on receiving this message, decrypts the first part of the cipher-text using his long term private key to obtain T and uses T to decrypt and obtain $\{\{S\}_{K_{bob}}\}_{K_{eph}}$. He then verifies the HMAC value and if this check is successful, chooses a random secret key J to secure his communication with the Ephemerizer, encrypts J using K_{eph} and sends the following to the Ephemerizer:

$$B \rightarrow E : keyID \ \{J\}_{K_{eph}} \ [\{\{S\}_{K_{bob}}\}_{K_{eph}}]_J$$

Step 4 - The Ephemerizer identifies K_{eph} using *keyID*. If K_{eph} hasn't expired, the Ephemerizer uses it to decrypt and obtain J. Using K_{eph}'s private key and J, the Ephemerizer then decrypts the third part of the message to obtain $\{S\}_{K_{bob}}$ and uses J to re-encrypt the decrypted part and sends it back to Bob.

$$E \rightarrow B : [\{S\}_{K_{bob}}]_J$$

If K_{eph} has expired, Ephemerizer sends back an error message to Bob indicating the unavailability of the key.

Step 5 - Since Bob knows the value of J and his own long-term private key, he can then decrypt the message from E and retrieve the value of S which is then used to decrypt M_E to obtain M.

The Ephemerizer periodically scans its database of asymmetric key pairs and securely delete all key pairs that have an expired time value. Therefore, after the expiry of time t_1, no one will be able to recover the plaintext M if all the participants (Alice, Bob, and the Ephemerizer) have truthfully executed the protocol.

2.2 Attack Against Triple Encryption Scheme

Our analysis showed that the triple encryption version of the Ephemerizer scheme presented in [2] is susceptible to a serious oracle attack which can be exploited by an attacker to gain access to $\{S\}_{K_{bob}}$, in effect nullifying the ephemeral nature of the system. The attack plays out as follows.

The attacker captures the message sent between Bob and Ephemerizer in *Step 3* and *Step 4*. After identifying K_{eph} using $keyID$, it then generates a random key X and encrypts it with K_{eph}. The attacker encrypts the second part of the original message in *Step 3*, $\{J\}_{eph}$, with X and sends the whole message as shown below, to the Ephemerizer.

$Att \rightarrow E : keyID\ \{X\}_{K_{eph}}\ [\{J\}_{K_{eph}}]_X$

As long as this attack message is sent before the expiry of the key corresponding to $keyID$, the Ephemerizer cannot differentiate it from a genuine request from an user. Hence, it decrypts X using the private key of K_{eph} and using the relevant keys it decrypts the third part of the attack message to obtain J. This value of the random session key, used by Bob to encrypt his dialog with the Ephemerizer, is then sent back to the attacker.

$E \rightarrow Att\ : [J]_X$

Since X is known to the attacker, he can decrypt this message from the Ephemerizer and obtain J, which in turn can be used to decrypt the message sent to Bob in *Step 4* to get $\{S\}_{K_{bob}}$. Thus, the purpose served by the ephemeral key K_{eph} is completely nullified, since $\{S\}_{K_{bob}}$ is now known to the attacker and is not protected by K_{eph} in a time-bound manner. Knowing $\{S\}_{K_{bob}}$, the attacker can wait as long as required to break into Bob and retrieve the long-term private key of Bob, and decrypts the value of S, even after K_{eph} has been deleted from the Ephemerizer's database.

Workaround This attack can be mitigated by using separate ephemeral keys to encrypt $\{S\}_{K_{bob}}$ and J. In *Step 1*, the Ephemerizer would generate 4-tuple (K_{eph1},K_{eph2}, $keyID$, expiry date) instead of the 3-tuple. In *Step 2* Alice would use K_{eph1} to encrypt $\{S\}_{K_{bob}}$ and in *Step 3*, Bob would use K_{eph2} to encrypt J. Thus the message sent by Bob to the Ephemerizer in *Step 3* would become

$B \rightarrow E : keyID\ \{J\}_{K_{eph1}}\ [\{\{S\}_{K_{bob}}\}_{K_{eph2}}]_J$

Since two different keys are needed to decrypt the second part and the inner encryption of part three of the message, the attack described previously will not work. However this workaround involves the generation and storage of a second asymmetric key pair for every tuple and hence is less efficient. Moreover this modified scheme still does not resolve the important shortcomings pointed out in the next section.

2.3 Shortcomings

A limitation of the proposed scheme is that Alice does not have the flexibility to define her own expiry dates for the data. Instead she has to choose an expiry date advertised by the Ephemerizer, thus constraining herself to the granularity implemented by the Ephemerizer server.

The secure working of the Ephemerizer system assumes that the recipient Bob uses volatile memory to perform all his temporary computations and that he can securely delete the symmetric key obtained from the Ephemerizer as well as the temporarily decrypted plaintext data once its use is over. This is a reasonable assumption since it is easier to securely delete data in the volatile memory than on a persistent storage device [5]. However, the schemes proposed in [2] do not provide any mechanism to *verify* that the platform used by Bob does indeed provide a secure temporary work-area for the sensitive data. Similarly, the schemes do not provide any provision for Alice to specify additional restrictions on the access of the decryption key by Bob. For example, Alice may want to restrict Bob's access to the message only when he is within, say, the company network. As such the proposed schemes do not provide any mechanism to specify additional conditions for access to the data.

We argue that these limitations cripples the system's usability and security.

In the rest of the paper we present an alternative scheme to provide the envisioned ephemeral service using IB-PKC primitive as the underlying basis. We show that this proposed scheme can address the above-mentioned limitations associated with the original scheme.

3 Identity Based Cryptography

As early as in 1984 Shamir [6] had proposed the use of an encryption scheme in which *an arbitrary string can be used as a public key.* However, it was only in 2001 that a mathematically sound and practically efficient identity-based public key cryptosystem was proposed by Boneh and Franklin [7]. An IB-PKC system, based on bilinear pairing, will be used in our proposed scheme. In this section we provide a brief introduction to the crypto-primitive.

Let P denote a generator of $\mathbb{G}_1$, an additive group of some large prime order q. Let $\mathbb{G}_2$ be a multiplicative group of the same order. A pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties:

1. Bilinear: $e(aQ, bR) = e(Q, R)^{ab} = e(abQ, R) = e(bQ, R)^a$, where $Q, R \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_q^*$.
2. Non-degenerate: $e(P, P) \neq 1_{\mathbb{G}_2}$, where $1_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.
3. Computable: There exists an efficient algorithm to compute $e(Q, R)$ for all $Q, R \in \mathbb{G}_1$

It is believed that the bilinear Diffie-Hellman problem in $\langle \mathbb{G}_1, \mathbb{G}_2, e\rangle$[1] is hard. Typically the map e is derived from either the Weil [8] or Tate [9] pairing on an elliptic curve.

An IB-PKC scheme consists of four main steps (1) **setup** in which a Key Generation Center (KGC) generates global system parameters and a system secret key, (2) **encrypt** where a message is encrypted using an arbitrary public key, (3) **extract** during which the system secret key is used by the KGC to generate the private key corresponding to the arbitrary public key chosen in the step earlier and (4) **decrypt** where the private key generated is used to decrypt the encrypted message.

Interested readers are referred to [7] for a more rigorous explanation of the mathematics, protocol steps and related proof of security behind the IB-PKC cryptosystem.

4 Proposed System

In this section we describe our proposed alternative Ephemerizer scheme that uses IB-PKC to overcome the deficiencies found in the original system.

As in the original scheme, our approach is to keep only the encrypted version of the data on the persistent storage of the user and to 'store' the key needed to decrypt the data on a different machine, the Ephemerizer server. When the user needs to access the plaintext data, he retrieves the decryption key from the server and uses it to decrypt and use the data in a secure manner. The Ephemerizer server in our scheme also functions as the KGC of the IB-PKC system.

Three properties of IB-PKC are exploited by our scheme (1) an arbitrary string can be used to derive the public key of an entity (2) the private key associated with such a public key is not computed at the same time as the public key and (3) the private key is generated not by the entity that creates the public key, but by the KGC. As explained further on, we exploit these properties by letting Alice embed her finegrained access requirements into the public key of of Bob and by ensuring that the Ephemerizer, which is also the KGC of the system, computes the corresponding private key and sends it to Bob only if the embedded checks have been successfully verified.

Note that, for the ease of explanation, the scheme described here uses the *BasicIdent* version of the IB-PKC scheme. This version is not secure against an

[1] Given $\langle P\ aP\ bP\ cP\rangle$ with uniformly random choices of $a\ b\ c \in \mathbb{Z}_q^*$, compute $e(P\ P)^{abc} \in \mathbb{G}_2$

adaptive chosen ciphertext attack and hence an actual implementation of our scheme would need to use the secure *FullIdent* version [7].

We divide our scheme into five steps:

Step 1 - As in the original IB-PKC system, the Ephemerizer E generates two groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ for the groups and choose an arbitrary generator $P \in \mathbb{G}_1$. It also specifies two hash functions H_1, H_2 as:

$H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$
$H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$, n being the bit-length of data to be encrypted

Message space $\mathcal{M} = \{0,1\}^n$, ciphertext $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^n$

E then computes a set of ephemeral secret keys s_{eph} uniformly at random from $\mathbb{Z}_q^*$ and the corresponding public keys $P_{eph} = s_{eph}P$ and associates each (s_{eph}, P_{eph}) pair with an expiration time and a $keyID$. It also computes another key s_E and corresponding P_E. E finally publishes the system parameters $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_E, H_1, H_2 \rangle$ and the set of tuples $(keyID, P_{eph}, expiration\_time)$.

Step 2 - Alice chooses a random symmetric key K and encrypts the data M with it: $M_E = [M]_K$. K is then encrypted with Bob's long-term public key $K_E = \{K\}_{K_{bob}}$. She then chooses the most appropriate value of P_{eph} from the available set such that (needed expiry date < expiry date of key). She then chooses as Bob's IB-PKC public key ID_b = '$Eph|Expiry : needed - expiry - date$', where Eph is the identity of the Ephemerizer.

For example, if the set of ephemeral keys available were $(ID_1,P_1,2006-28-12-22:00)$, $(ID_2,P_2,2006-28-12-22:30)$ and $(ID_3,P_3,2006-28-12-23:00)$ and Alice wants to make the data unrecoverable after 2006-28-12-22:15, she would choose ID_2, P_2 and assign ID_b = '$Eph|Expiry : 2006-28-12-22:15$'. For the rest of the paper, we will assume she chose P_{eph}. She then computes $Q_{ID_b} = H_1(ID_b)$, chooses $r_b \in \mathbb{Z}_q^*$ and encrypts K_E by computing:

$C_b = \langle U_b, V_b \rangle = \langle r_bP, K_E \oplus H_2(g_b^{r_b}) \rangle$
where $g_b = e(Q_{ID_b}, P_{eph}) \in \mathbb{G}_2$

Alice then sends the following to B:

$A \rightarrow B : \{ID_b\ C_b\}_{K_{bob}}\ M_E\ keyID$

Step 3 - Bob uses his long-term private key to decrypt the first part of the message received from A to obtain ID_b and C_b and saves them locally with the rest of the message.

When Bob needs to decrypt and obtain M, he creates an arbitrary public key ID_e for E, a random key J and computes $Q_{ID_e} = H_1(ID_e)$ and computes

$C_e = \langle U_e, V_e \rangle = \langle r_e P, (ID_b|J) \oplus H_2(g_e^{r_e}) \rangle$
where $g_e = e(Q_{ID_e}, P_E) \in \mathbb{G}_2$

Bob then sends the following to E:

$B \rightarrow E : ID_e \ keyID \ C_e$

Step 4 - When the Ephemerizer receives the message from Bob, it first makes sure that *keyID* has not expired. If the key has expired, s_{eph} and P_{eph} are deleted from the secure database and an error message is sent back to Bob. Periodically, E also scans this database on its own and deletes all expired tuples. If the key associated with Bob's request is still valid, E calculates $Q_{ID_e} = H_1(ID_e)$, $d_e = s_E Q_{ID_e}$ and $V_e \ominus H_2(e(d_e, U_e))$, which yields '$ID_b|J$'. E then examines ID_b to check for the expiration time that A has specified. Only if this expiration time is also valid would E compute $Q_{ID_b} = H_1(ID_b)$ and $d_b = s_{eph} Q_{ID_b}$, where s_{eph} corresponds to the *keyID* specified and generated in Step 1. E then sends to Bob:

$E \rightarrow B : [d_b]_J$

Step 5 - Since Bobs knows the value of J, he uses it to decrypt the message sent by E and obtain d_b, which is then used to calculate $V_b \oplus H_2(e(d_b, U_b)) = K_E$. He then uses his long-term private key to decrypt K_E to obtain K which is then used to decrypt M_E to finally obtain M. M, J, d_b and K are deleted securely by Bob after use.

Note that since the output of H_2 is used as a one-time pad to encrypt data, n used in the definition of H_2 should be at least |maximum length of ID_b | + |maximum length of key J|. The security of the proposed system relies on the security of the original IB-PKC system and interested readers are referred to [7] for detailed analysis.

5 Discussion

5.1 Security

IB-PKC has not gained widespread usage mainly because it suffers from an inherent key escrow problem. The KGC, knowing the secret s_{eph}, can compute the associated private key. Though several solutions have been proposed to counter the inherent key escrow problem, including threshold key issuing using multiple KGCs [7], generating the private key using multiple independent private keys issued by multiple KGCs [10], certificate-based encryption [11] and certificateless public key encryption [12], none of them can be directly used in our scheme. On one hand the schemes proposed in [11] and [12] do not allow arbitrary strings as public key and hence prevents Alice from specifying her own expiration time, while on the other, proposals like 1[10] require Bob to authenticate with multiple KGCs every time he needs to decrypt the data.

This is the reason why we do not completely replace the traditional PKI based system in favor of a system solely based on IB-PKC. Instead we use a hybrid scheme using each cryptosystem's strength to provide specific security requirements.

By encrypting K with Bob's traditional long-term public key the system completely side-steps the key escrow problem. A malicious Ephemerizer will not be able to obtain M with just its knowledge of d_b, the private key corresponding to ID_b. It will also have to compromise Bob's machine to obtain his long-term private key. This provides the same security setup as the original Ephemerizer scheme of Perlman. The use of IB-PKC allows Alice to specify her own finegrained expiration time and also help extend the scheme fairly easily, as discussed further on.

In *Step 2*, ID_b is sent to Bob encrypted with his long-term public key. This prevents an attacker from knowing the value of ID_b and using the knowledge to obtain d_b.

In *Step 4* d_b is sent by the Ephemerizer to Bob encrypted using a symmetric key J that is randomly selected by Bob just for that session of the protocol run and deleted afterward. Thus even if an attacker captures the message sent to Bob in *Step 4*, it cannot decrypt and obtain M, since J is deleted immediately after *Step 5*.

The Ephemerizer will send d_b to B only if both s_{eph} and the expiration time specified in ID_b are valid. Thus Alice is able to specify her expiry date at whatever granularity she prefers (as long as she chooses the right s_{eph}). Additionally, once Ephemerizer finds out that a particular s_{eph} has expired, it is permanently deleted from its secure database. Thus, even if the Ephemerizer and Bob is compromised after the expiry of s_{eph}, the attacker would not be able to use s_{eph} to obtain M.

Note however that there could be a time gap between the expiry specified in ID_b and that of s_{eph}, between which an attacker could compromise E and gain access to s_{eph}. We argue however that since the time gap can be made as small as possible by judicious choice of *seph*, this risk is as acceptable as the chance of Ephemerizer server being compromised before the expiry date specified by Alice. Furthermore, the access to s_{eph} does not mean that the attacker can successfully generate d_b. If the whole computational process at the Ephemerizer's end, including the IB-PKC equivalent of **decrypt** as specified in *Step 4*, is implemented in a secure coprocessor [13], the comparison between the current time and the expiration time specified in ID_b can be performed in a secure and tamper resistant manner. Thus, even if s_{eph} is valid, the computation of Q_{ID_b} will fail due to expiration time check and by extension so will the computation of d_b. As the Ephemerizer server is a dedicated machine operated solely for the purpose of managing the ephemeral keys, its use of tamper-resistant hardware is not a far-fetched assumption. A similar reasoning applies to attacks aimed at changing E's system clock.

5.2 Supporting Richer Access Control

The scheme proposed above does not explicitly describe how Alice can specify further conditions for release of d_b to Bob. This was consciously done to keep the basic protocol simple and easy to explain. In this section we explain how the basic scheme can be extended to support extra conditions that Alice may like to impose.

In the basic scheme proposed in Section 4, ID_b was used by Alice to specify her fine-grained expiration time. This was checked in the later stages of the protocol run, by the Ephemerizer, to ensure its validity before d_b was sent to Bob. Extending this to include other checks is straight forward.

In *Step 1*, along with the publication of the system parameters, the Ephemerizer could state which other conditional checks are offered by it as a service to the users. These extra checks could include, for example, IP address access check, secure system check etc.

Implementing the IP address check would be as simple as Alice stating the allowed IP address or range of addresses in ID_b. For example ID_b = '$Eph|Expiry : 2006 - 28 - 12 - 22 : 15|IP : 132.168.2.*$' would mean that Alice wants to allow Bob to access the data M only until 22:15 hrs of 2006-28-12 and only from the address range 132.168.2.1 - 132.168.2.254. In Step 4 of the protocol run, when the Ephemerizer retrieves the value of ID_b from C_e, it checks the IP address that Bob is using along with the validity of the expiry date and proceeds only if both the checks succeed. The assumption is that a spoof-proof method to determine the IP address of a remote host exists, maybe using techniques like [14]. The support for wildcards would require the use of the identity-based encryption with wildcards cryptoprimitive [15].

The original Ephemerizer scheme as well as the one proposed in this paper depends on the ability of Bob to securely delete the temporary plaintext data as well as all the data that he receives from Ephemerizer. To increase the users' faith in the system, Alice should be able to verify that Bob is indeed using a secure system when accessing the secret data. The original scheme had no mechanism to check for the presence of such a system. However our proposed scheme can be extended fairly simply to support this verification. For this extension to work, Bob would need to use special hardware like the Trusted Platform Module [16], which has the ability to perform *remote attestation* [17] or a richer semantic remote attestation [14]. Alice would specify the Platform Configuration Register (PCR) that she trusts to be that of a secure system in ID_b, ID_b = '$Eph|Expiry : 2006 - 28 - 12 - 22 : 15|PCR1 : 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12$'. It is assumed that Alice has some mechanism to find out the correct PCR value of a trusted secure system.

Once the Ephemerizer receives the decryption request from Bob, and has decrypted the value of ID_b, it initiates a protocol similar to the 'Integrity Challenge Protocol' of [18] to verify the integrity of Bob's system.

$E \rightarrow B : ChReq(n, PCR1)$
$B \rightarrow E : sig\{PCR1, n\}_{AIK}$

The Ephemerizer sends Bob a PCR challenge request, along with a nonce to prevent replay attacks. Bob's machine's TPM uses its Attestation Identity Key (AIK) to sign the PCR value requested and then sends the signed PCR value back to the Ephemerizer along with the nonce sent with the request. At the Ephemerizer, the AIK signature is first verified and then the reported PCR value can be used in calculating Q_{ID_b} and d_b. Thus a wrongly reported PCR would create a wrong ID_b and hence a wrong Q_{ID_b} and d_b, preventing Bob from accessing the protected data.

In general, the proposed scheme can be extended to support any number of extra restrictions by specifying them appropriately in Bob's public key ID_b, as long as the Ephemerizer has the ability to perform these checks.

6 Conclusion and Future Work

In this paper we analyzed the Ephemerizer system proposed by Perlman [2] and used in [3] as a system that allows parties to securely share data, by keeping the encrypted data and the decryption key in physically separate entities, for a finite time period and then making it unrecoverable after that. However, as noted in this paper, the schemes do not allow the parties to specify more detailed and flexible usage restrictions on the data. In addition one of the version of the original scheme suffers from a fatal oracle attack as described in the paper.

We proposed an alternate scheme to implement the Ephemerizer system with none of the identified flaws and functional constraints. Our scheme exploits the properties of Identity Based cryptosystem and is one of the few systems that utilise the power of the crypto-primitive without suffering the associated disadvantages. The security of the proposed scheme and its ability to support flexible usage restrictions were then discussed in detail.

We plan on developing a formal proof of correctness of our scheme as well as an implementation of the same to prove that the scheme is indeed secure and implementable.

Acknowledgment

We thanks Chandana Gamage for his invaluable inputs. This work has been supported by NWO project ACCOUNT 612.060.319 and partially by the EU project S3MS IST-STREP-27004.

References

1. United States Department of Defense (2006) National Industrial Security Program Operating Manual. DoD 5220.22-M

2. Perlman R (2005) The Ephemerizer: Making Data Disappear. Journal of Information System Security, Vol. 1 (1), pp. 51–68
3. Perlman R (2005) File System Design with Assured Delete. Third IEEE International Security in Storage Workshop, pp. 83–88, USA
4. Bellare M, Canetti R, Krawczyk H (1996) Keying Hash Functions for Message Authentication. Advances in Cryptology - Crypto 96, LNCS 1109, Springer-Verlag, pp. 1–15
5. Crescenzo GD, Ferguson N, Impagliazzo R, Jakobsson M (1999) How to Forget a Secret. International Symposium on Theoretical Aspects of Computer Science, LNCS 1563, Springer-Verlag, pp. 500–509
6. Shamir A (1984) Identity-based Cryptosystems and Signature Schemes. Advances in Cryptology - Crypto 84, LNCS 196, Springer-Verlag, pp. 47–53
7. Boneh D, Franklin F (2001) Identity-based Encryption from Weil Pairing. Advances in Cryptology - Crypto 2001, LNCS 2139, Springer-Verlag, pp. 213–229
8. Lang S (1973) Elliptic Functions. Addision-Wesley
9. Frey G, Muller M, Ruck H (1999) The Tate Pairing and the Discrete Logarithm Applied to Elliptic Curve Cryptosystems. IEEE Transactions on Information Theory, 45(5)L1717-9
10. Chen L, Harrison K, Smart NP, Soldera D (2002) Applications of Multiple Trust Authorities in Pairing Based Cryptosystems. InfraSec 2002, LNCS 2437, Springer-Verlag, pp. 260–275
11. Gentry C (2003) Certificate-based Encryption and the Certificate Revocation Problem. Advances in Cryptology - Eurocrypt 2003, LNCS 25656, Springer-Verlag, pp. 272–293
12. Al-Riyani S, PatersonK (2003) Certificateless Public Key Cryptography. Advances in Cryptology - Asiacrypt 2003, LNCS 2894, Springer-Verlag, pp. 452–473
13. Dyer J, Lindemann M, Perez R, Sailer R, van Doorn L, Smith SW, Weingart S (2001) Building the IBM 4758 Secure Coprocessor. IEEE Computer Vol. 34, no. 10, pp. 57–66
14. Haldar V, Chandra D, Franz M (2004) Semantic Remote Attestation: A Virtual Machine Directed Approach to Trusted Computing. USENIX Virtual Machine Research and Technology Symposium, pp. 29–41
15. Abdalla M, Catalano D, Dent AW, Malone-Lee J, Neven G, Smart NP (2006) Identity-Based Encryption Gone Wild. Automata, Languages and Programming: 33rd International Colloquium, LNCS 4052, Springer-Verlag, pp. 300–311
16. Trusted Computing Group (2006) http://www.trustedcomputinggroup.org
17. Trusted Computing Group (2006) Trusted Platform Module Main Specification, Part 1: Design Principles, Part 2: TPM Structures, Part 3: Commands, Version 1.2, Revision 94. http://www.trustedcomputinggroup.org
18. Sailer R, Zhang X, Jaeger T, van Doorn L (2004), Design and Implementation of a TCG-Based Integrity Measurement Architecture. 13th Usenix Security Symposium, USENIX, pp. 223–238