The Ephemerizer: Making Data Disappear

Radia Perlman

The Ephemerizer: Making Data Disappear

Radia Perlman

SMLI TR-2005-140 February 2005

Abstract:

This paper is about how to keep data for a finite time, and then make it unrecoverable after that. It is difficult to ensure that data is completely destroyed. To be available before expiration it is desirable to create backup copies. Then absolute deletion becomes difficult, because even after explicitly deleting it, copies might remain on backup media, or in swap space, or be forensically recoverable. The obvious solution is to store the data encrypted, and then delete the key after expiration. The key is somewhat easier to manage, because it is smaller, but there is still the issue of needing to make the key reliably available for some time, and then reliably destroyed. It is difficult enough for a user to manage one key, much less different keys for different data expiration times. The user could keep each key on a tamper-proof smart card with no copies, but then the data will be lost prematurely if the user loses the smart card. And smart cards are expensive. So the idea in this paper is to concentrate all the key management expense and expertise in one place, a server we call an “ephemerizer”. The ephemerizer creates keys, makes them available for encryption, aids in decryption, and destroys the keys at the appropriate time. The design in this paper ensure that even if a client’s machine gets compromised, and everything in stable storage (including long term user keys) is stolen, any data that has expired before the compromise remains unrecoverable.

The paper starts with a description of an existing commercial scheme, and presents improvements to that scheme to eliminate the necessity for per-message state. Then it presents a new approach, based on public keys, and presents an initial design, and then a more efficient version using a new concept closely related to blind signatures, that we call “blind decryption”.

Sun Labs
16 Network Circle
Menlo Park, CA 94025

email address:
radia.perlman@sun.com

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@sun.com>.All technical reports are available online on our website, http://research.sun.com/techrep/.

The Ephemerizer: Making Data Disappear

Radia Perlman
radia.perlman@sun.com

1 Introduction

Making data robustly available is an important and difficult problem, but sometimes it is equally important for the data to become reliably unrecoverable. One example is email. One might want a certain class of emails to be readable for only some finite amount of time, say two weeks. Even if the data is explicitly deleted, copies may remain on backup media, have been captured and stored in transit, e.g., at a router or MTA (mail transfer agent), or even be forensically recoverable from disk.

One approach is to only store the data encrypted, and then it is a somewhat easier problem to delete the key. However, long term user keys can, over time, be made available through compromise or coercion. It is possible for keys to be kept in tamper-resistant smart cards, in which case it would not be feasible to covertly discover the key. Destroying the smart card certainly deletes the key. It is somewhat expensive and inconvenient to assume every user has a smart card and every computer has a smart card reader. But it is especially unrealistic for the user to simultaneously manage many keys, since there would need to be a different key for each possible data expiration time. For example, if Alice is to send an email message to Bob that will be unrecoverable after two weeks, it has to be encrypted in a key that will be guaranteed to be available for two weeks, and then reliably destroyed after two weeks. It would place a large burden in cost and expertise for Bob to create, reliably store, certify, advertise, and then reliably destroy a large number of keys.

The system in this paper instead concentrates all the expense and expertise of key management in a service, which we call an "ephemerizer", whose cost can be amortized over many users and many messages. The ephemerizer's job is to create, advertise, and destroy keys. A client needs the ephemerizer's help to read an ephemerized message. This service could be used for a message composed by Alice to be readable by Bob, or it could be used solely by Alice for management of her own data, to store data that can easily be irrevocably destroyed on demand (when an enemy captures her stored data), or upon a predetermined expiration date.

We assume both Alice (the client encrypting the message) and Bob (the client reading the message) are motivated to ensure that no copies of the data exist, so they do not save copies of the decrypted message. We also assume that Alice and Bob have special message ephemeral-encryption and ephemeral-decryption software for creating and reading messages, respectively. The decrypted message must never be in stable storage, so if the message reading system at the client is designed with that in mind, there is no danger of backup copies of the decrypted message existing, or being forensically recoverable, even if deleted. We want the system to be easy for users to use, and require sophisticated key management expertise and special hardware only at the

ephemerizer. We also want the cost to the ephemerizer of managing a key to be amortized over many users and many messages.

We assume that it is possible that Bob's machine will be compromised at some point, including his long-term private keys, and that Bob's machine was not compromised before that point. All data accessible to Bob that has not yet expired will at that point be readable by the attacker, but the system in this paper will assure that all Bob's data that has expired will be unrecoverable, even if everything in stable storage on Bob's machine, or that was ever transmitted, is seen by the attacker.

We also want to place minimal trust in the ephemerizer, and also of course, make the system cost-effective in communication, computation, and storage.

1.1 Roadmap to this paper

In section 2, we describe some existing approaches to the problem of making data disappear. Then in section 3, we present improvements to an existing commercial scheme that eliminates the necessity for per-message state. In section 4, we present a new approach, which improves the security properties by ensuring that only the authorized recipient can decrypt messages. In section 6, we present a new concept which we call "blind decryption", which, as we show in section 6, improves the security and efficiency of the general approach.

1.2 Notation

{M}K means the message M is encrypted with key K. K can be a public key or a secret key.

We assume that the plaintext of a message includes integrity protection, if needed (most likely a digital signature).

We refer to two numbers, x and y, as "exponentiative inverses" mod p to mean that anything raised to x and then raised to y will yield the original number. If p is a prime, then x and y are exponentiative inverses if they are multiplicative inverses mod p-1.

We use | to mean "concatenated with", so HMAC (T, X | Y) means applying HMAC with key T to the quantity X concatenated with Y.

2 Previous Work

There are some email systems that have a "self-destruct" feature. This is implemented purely in the mail reading client, and involves no cryptography. It just means that the copy of the email at the client is automatically deleted after reading (or perhaps not even that...perhaps merely marked as deleted). There are also commercial systems that allow setting policies that email should be

deleted automatically after some time. Again this involves no cryptography, and copies on backup media would of course not go away.

A more sophisticated system was built by a company with the wonderful name of Disappearing, Inc. [Dis]. Conceptually, one version of the system worked as follows:

1. If Alice wishes to create an encrypted message for Bob, she contacts the ephemerizer, specifying an expiration time, and requesting a key.
2. The ephemerizer chooses a random secret key K, assigns a key-ID ID_K, tells Alice: (K, ID_K), and remembers: (expiration time, K, ID_K).
3. Alice encrypts the message M with K (to obtain {M}K) and sends to Bob: ({M}K, ID_K)
4. When Bob wishes to decrypt the message, he sends the ephemerizer: ID_K
5. The ephemerizer replies with K, and then Bob can decrypt the message.
6. When expiration time occurs, the ephemerizer forgets K.

Presumably Alice and Bob talk to the ephemerizer via some protected channel such as SSL, in which they authenticate that they are indeed talking to the ephemerizer, and such that the messages between the ephemerizer and a client are encrypted.

A nice property of this scheme is that the ephemerizer can be built such that it does not see the messages (though in one implementation of the scheme the messages are stored on the ephemerizer).

The problems with this approach are:

1. Anyone that captured ({M}K, ID_K) would be able to get the ephemerizer to decrypt the message, since there is no way for the ephemerizer to authenticate Bob.
2. The ephemerizer must create and store a key for every ephemerally encrypted message.
3. The ephemerizer must communicate both when a message is encrypted (by Alice) and decrypted (by Bob).
4. Authentication of the ephemerizer by Alice and Bob depend on the non-compromise of the PKI. For example, if an attacker could trick any of the trust anchors in Alice or Bob to issue a certificate with the attacker's public key and the ephemerizer's name, it can act as a man-in-the-middle, obtain the message encryption keys, and later compromise Bob to obtain the encrypted data.

3 Improving Disappearing, Inc.'s Scheme

We present two ways of compressing the space needed at the ephemerizing server before presenting our preferred design for making data disappear in the following sections.

3.1 One secret per expiration time

It would be easy for the Disappearing, Inc.'s scheme to be modified to avoid requiring the ephemerizer to keep per-message state. We suggest the following enhanced scheme:

1. Alice chooses a per-message nonce N (or the ephemerizer chooses the nonce and informs Alice in step 3)
2. She tells the ephemerizer a desired expiration time T, and the nonce N.
3. The ephemerizer keeps a set of secrets: {expiration time, S, ID_S}. If there is an S with expiration time T (or close enough), it selects the secret S_T that has expiration time T, and calculates some hash H= h(N, S_T). If there is no suitable S_T, it creates and stores a new secret (T, S_T, ID_S) and calculates H based on the newly created S_T.
4. The ephemerizer sends to Alice: H and ID_S, and then forgets H and N.
5. Alice encrypts M with H, and tells Bob: ({M}H, N, ID_S).
6. When Bob wants to decrypt the message, he sends (N, ID_S) to the ephemerizer.
7. The ephemerizer finds S_T, the S associated with ID_S, calculates H= h(N, S_T), and tells Bob H.
8. Periodically the ephemerizer forgets all the (T, S_T ,ID_S) triples for which T has occurred.

This modification avoids the necessity for the ephemerizer to keep per-message state. It does mean that someone that compromises a single one of the ephemerizer's secret S's compromises all messages encrypted with that secret, but in the original scheme anyone who compromised the ephemerizer's data would capture all the per-message keys. This variant is probably, in practice, more secure because it would be easier to adapt this variant to having the secret keys on a tamper-resistant smart card, since the database is so much smaller. (The smart card would generate the secrets and, given a nonce and a secret's ID, calculate and output a per-message encryption key.)

Note that it might be feasible for the expiration date to be the ID, so instead of the ephemerizer needing to remember (T, S_T ,ID_S), it would only need to remember (T, S_T). But having the ID in addition to the expiration time allows there to be multiple secrets for a given expiration time. For instance, a particular user might want a key with special security properties, such as being extra long, or done with their country's national encryption standard protocol.

3.2 One secret overall

One could save even more space by having the ephemerizer roll over keys with a one-way hash function. Instead of generating a new secret for each expiration time, the secret for expiration time T might be S_T, and the secret for expiration time one time unit later would be h(S_T), and the secret for expiration time two time units later would be h(h(S_T)), and so on. The current secret, S_T, is the one about to expire. Every time unit, the current S_T is forgotten and replaced by h(S_T). This saves storage if the ephemerizer is willing to compute h^n(S_T) in order to encrypt or decrypt a message with a key n units from the current secret. Otherwise, the ephemerizer could precompute the next n secrets and store them, in which case it hasn't saved any storage. Or the ephemerizer could store S's for, say, every k time units, and then only need to compute hashes from an S close to the requested expiration time.

3.3 Backward compatibility

Either of the above enhancements might be able to be made on-the-wire compatible with Disappearing, Inc.'s scheme, and therefore could be implemented as an optimization at the server without modifying the clients. This would be done by having the ephemerizer (rather than Alice) choose the nonce, and have what appears to Alice to be the ID of the per-message secret actually be the tuple (N, ID_S).

4 Our Scheme (without blind decryption)

In this section, we present a design based on public keys, that allow messages to be encrypted without the ephemerizer's active participation (other than advertising the keys). A single public key can be used by many users, and for many messages. The ephemerizer must actively participate in decryption of the messages, though. In section 6, we present a scheme which is more efficient for the ephemerizer.

4.1 Introduction

The basic idea behind our scheme is that there will be an ephemerizer which will create and advertise public keys and expiration times, Alice will choose an appropriate public key, encrypt the message using that key, and send the message securely to Bob. When Bob wants to decrypt, he enlists the aid of the ephemerizer.

This system might be used for Alice to ephemerize her own data, in which case the step of securely sending it to Bob can be skipped.

Although the message encryption keys used by the ephemerizer are ephemeral, we assume that the ephemerizer has a long-term key that it uses to authenticate itself and/or certify the ephemeral public keys. There is no need for that key to be ephemeral. If that long-term key were to become compromised, this would be dealt with the same way that any authentication key compromise would be handled. The old key would be revoked and a new key would be advertised and certified (by whatever trusted third party had certified the original key). However, the important point is that even if the ephemerizer's authentication key is compromised, this will not cause any data to become recoverable that had expired before the ephemerizer's authentication key became compromised.

We also assume that Bob has a long term public key pair. If that key becomes compromised, and also everything in Bob's stable storage is read by the attacker, then all data that has not yet expired will be readable by the attacker, but our scheme guarantees that all data that has expired before the compromise will be unrecoverable. We also assume that Alice and Bob have special ephemerization software. Alice's must be capable of finding an appropriate ephemeral public key and encrypting with it. Bob's must be capable of communicating with the ephemerizer and doing the cryptography necessary to get the message decrypted, and not to store the decrypted message in stable storage. It is possible in many operating systems to have a process that will not be swapped out to stable storage, so the ephemerization software at the client should use that feature to ensure that the decrypted message never appear on stable storage.

4.2 Thresholding

There are two ways in which an ephemerizer can fail:
1. Prematurely forget the key, or be unavailable when needed for decryption
2. Remember the key beyond its expiration time

The solution is to use a simple thresholding scheme, where the secret is broken into n pieces, such that any k of the pieces can recover the secret, and to give each of the n pieces to each of n ephemerizers. Then as long as k of them are available before the key expires, the message can be read, and as long as n-k+1 of them forget their share of the secret when they are supposed to, the message will be unrecoverable.

Thresholding is an easy extension to any of the schemes in this paper.

4.3 Relationship to forward-secure public key cryptography

There is a concept known as "forward secure public key cryptography" [A97] which might appear to be related to this paper. That concept is focused on surviving compromise of a signature key without casting suspicion on all documents previously signed with that key (since whoever stole the private key could back-date whatever it signed). The idea is that there are a set of public signature keys, each one with an expiration date, and when the expiration date on a private key has passed and the private key is discarded, it should be impossible, even if a current private key is compromised, to recover a previously discarded private key, and therefore anything signed with a private key that expired before the owner's machine (and keys) got compromised, would still remain valid. This is conceptually trivial to accomplish if the key pairs are mathematically unrelated, but takes storage proportional to the number of existing time units to store all the keys. For instance, key j could be used to certify public key j+1. The certificate chain for public key j+1 would consist of j certificates. Or key j could directly certify the next k keys, in which case the certificate chain would be k times smaller, but the holder of the private keys would need to store k private keys.

Ideally there would be a single key seed, and each subsequent key pair would be derived from a one-way function of the previous key pair, so that if the first j keys were discarded, all subsequent keys could be derived from key j+1. Although this has not been accomplished, various optimizations have been introduced over the conceptually simplest scheme of having certificate chains proportional in size to the number of elapsed time units. [BM99] [CJMM03].

This concept of forward secure public key cryptography is not directly relevant to the ephemerizer concept in this paper because the ephemerizer concept does not require archivally valid signatures. The public encryption/decryption keys for the ephemerizer only need to be valid during their lifetime. And furthermore, unlike forward secure signatures, where the signer need only keep a single private key (the current one), the ephemerizer needs to keep all currently valid encryption/decryption keys, since it might be asked to decrypt with any of the unexpired keys.

4.4 Using triple encryption

The ephemerizer will create a set of public key pairs, and advertise triples consisting of (public key, key ID, expiration time). The ephemerizer's advertised triples should be signed. We assume the ephemerizer has a long-term signature key (in addition to all the ephemeral data encryption keys) with which all the ephemeral public keys are signed. We do not need to solve the problem of forward secure signatures, since, as pointed out in section 4.3, there is no need for archival quality signatures. If the ephemerizer's long term signature key were compromised, usual PKI revocation can be done. Compromise of the ephemerizer's long term signature key would not cause any messages to become readable, if they were encrypted with an ephemeral key that expired before the compromise. Someone who has stolen the long-term ephemerizer signature key can, until the compromise is known to Alice, fool Alice into encrypting a message with a key that will not be deleted at the expiration time, or possibly a nonsensical key for which no private key exists, and the message will never be readable.

In order for Alice to encrypt a message for Bob, Alice encrypts the message with a randomly selected secret key S to obtain {M}S, and encrypts S first with Bob's key, to obtain {S}Bob, and then with Keph, an appropriate one of the ephemerizer's keys, to obtain {{S}Bob}Keph. To securely transmit {{S}Bob}Keph to Bob, Alice further encrypts {{S}Bob}Keph with Bob's long-term public key, to obtain {{{S}Bob}Keph}Bob. She sends to Bob:

- {M}S ; the message encrypted with secret key S
- key ID of Keph ; the ID of the ephemeral key Keph that Alice chose to encrypt with
- {{{S}Bob}Keph}Bob ; the message secret key S, encrypted first with Bob's public key, then with Keph, then with Bob's public key.

Bob decrypts {{{S}Bob}Keph}Bob to obtain {{S}Bob}Keph. Next the following has to happen:

- Bob needs to send {{S}Bob}Keph to the ephemerizer
- The ephemerizer has to return {S}Bob to Bob.

Bob can't directly transmit {{S}Bob}Keph to the ephemerizer, or else an eavesdropper would be able to request the ephemerizer to decrypt {{S}Bob}Keph. The attacker would still only see {S}Bob, and have to also obtain Bob's public key, but we are assuming that Bob's key is long-term and will not expire, and will be easy to obtain at some point through coercion, court order, or Bob's carelessness. Even though {M}S is not transmitted, we are assuming that anything in stable storage on Bob's machine might be retrievable at some point in the future, and we can't count on all copies of {M}S getting deleted, so it is essential that nobody other than Bob (and the ephemerizer) sees S or {S}Bob.

Therefore, Bob needs to securely convey {{S}Bob}Keph to the ephemerizer, and securely receive {S}Bob} in response.

He could, in theory, communicate with the ephemerizer using SSL, and send {{S}Bob}Keph and receive {S}Bob}, over the encrypted SSL connection. However, given that SSL does not (usually) provide PFS (perfect forward secrecy), this would mean that S would be encrypted in a non-

ephemeral key. (someone that compromised the ephemerizer's long-term key after Keph had expired and was discarded, would still be able to recover S).

We could use a protocol (such as IPSEC's IKE handshake) or a version of SSL that provides PFS. For instance, one of SSL's variants (designed for exportability) has the server certify an ephemeral RSA key with its long term key, and the session secret is encrypted with the ephemeral public key. However, we'd like Bob to know it's the same ephemerizer as Alice used.

Since Alice has already looked up Keph, the best solution is for her to send Keph to Bob, and Bob can use Keph to secure his communication to the ephemerizer. However, Keph has to be securely tied to {{S}Bob}Keph. Otherwise, if Bob can be tricked into using an attacker's public key for securing his communication to (what he thinks is) the ephemerizer, then he will be tricked into sending {{S}Bob}Keph to the attacker, who will be able to then ask the ephemerizer to decrypt it to obtain {S}Bob, and be able to, long after the message has expired, compromise Bob's long term key and decrypt the message.

So instead Alice chooses an ephemeral secret T, and she will use it to compute a cryptographic integrity check linking the encrypted per-message key S with Keph. She will send T encrypted with Bob's public key. So for efficiency, rather than sending {{{S}Bob}Keph}Bob, she will instead send {{{S}Bob}Keph}T. So, what Alice sends to Bob (Message 1) is:

Message 1: What Alice sends to Bob is:
- {T}Kbob
- {{{S}KBob}Keph}T
- {M}S
- key ID of Keph
- Keph
- HMAC (T, {{S}KBob}Keph} | Keph)

Next Bob decrypts with KBob in order to obtain T, and then with T to obtain {{S}KBob}Keph. At this point he also verifies HMAC (T, {{S}KBob}Keph} | Keph).
If the HMAC verifies, then in order to decrypt the message, Bob chooses a secret key J with which to secure his communication with the ephemerizer, encrypts J with Keph, and sends to the ephemerizer as Message 2:
- keyID
- {J}Keph
- {{{S}KBob}Keph}J

The ephemerizer uses keyID to select Keph, decrypts J, decrypts {{{S}KBob}Keph}J with J, and then decrypts {{S}KBob}Keph to obtain {S}KBob. Then the ephemerizer encrypts {S}KBob and returns to Bob as Message 3:
- {{S}KBob}J

4.5 Analysis

Why is it necessary for Alice to triply encrypt S? Why can't she send {{S}Keph}T instead of {{{S}KBob}Keph}T? What good is the inner encryption of S with KBob?

Without the inner encryption with Kbob, if the ephemerizer is really dishonest, then the ephemerizer will be able to obtain S (and therefore decrypt the message) with only eavesdropping (and not needing to compromise Bob), because a malicious ephemerizer will be able to see S and eavesdrop from the wire {M}S. A crooked ephemerizer can refuse to delete the ephemeral private keys, but with the triple encryption scheme we have presented, it will still need to compromise Bob.

Message 1 contains:

- {T}Kbob
- {{{S}KBob}Keph}T
- {M}S
- key ID of Keph
- Keph
- HMAC (T, {{S}KBob}Keph} | Keph)

An eavesdropper without knowledge of KBob will not be able to get the ephemerizer to decrypt the first item, and therefore, once Keph expires, {M}S is unrecoverable to an eavesdropper. Also, without knowledge of KBob, an attacker will not be able to substitute a different Keph, because he will be unable to compute the integrity check HMAC (T, {{S}KBob}Keph} | Keph), because the attacker does not know {S}KBob}Keph} without KBob. An eavesdropper will be able to discover keyID and Keph, but these are publicly available. Modifying them in the message will cause Bob not to be able to decrypt the message, but it will not allow the attacker to read the information.

We assume that Bob stores what he received in message 1 in stable storage, so the same argument applies to the information as stored at Bob, as the information in transit. Once Keph expires (assuming an honest ephemerizer), an attacker that reads everything Bob has in stable storage will not be able to recover M.

Messages 2 and 3 are encrypted with J, an ephemeral key that Bob creates and remembers just for the duration of those two messages. J is encrypted with Keph, so once Keph expires, no useful information can be recovered from messages 2 and 3. Although KeyID is unencrypted, and not even integrity protected, it is not of use to an attacker. Modifying KeyID is only a denial of service attack (preventing Bob from decrypting the message).

If the ephemerizer is dishonest, and retains Keph, then someone that obtains the private key for Keph, and Bob's public key, and the message sent by Alice to Bob, will be able to recover the message.

4.6 An ephemerizer implementation

Imagine an implementation of the ephemerizer in which the private keys are in a tamper-resistant smart card that also does the cryptographic operations, but the rest of the machine is a general purpose computer. If the smart card never divulges the private keys, and also decrypts J from "Message 2", and only outputs message 3 encrypted with J, then even if there were malicious software on the general purpose computer portion of the ephemerizer, it would not be able to obtain anything more than an eavesdropper would be able to obtain.

5 Blind Decryption

Blind decryption is similar in spirit to the existing concept of blind signatures [CH83]. In fact almost the same mathematics that works for blind signatures works for blind decryption. However, we will also show two additional mathematical systems that work for blind decryption that do not work for blind signatures.

Suppose there is a blind decrypter BD, with public key Keph, and Bob has a quantity {S}Keph, that he wishes to have BD decrypt. Instead of sending {S}Keph to BD, Bob will instead invent new functions "blind" and "unblind", that commute with the encryption/decryption functions associated with Keph. Bob will apply "blind" to {S}Keph to obtain B({S}Keph, and ask the blind decrypter to apply its private key. The result with be B(S), which BD returns to Bob.

We want the ability to have inverse functions "encrypt" and "decrypt", and inverse functions "blind" and "unblind", that commute, such that we can do "encrypt", "blind", "decrypt", and "unblind" and get the original message back.

5.1 Review of blind signatures

Blind signatures were invented by Chaum [CH83] using RSA public keys. Suppose the blind signer (BS) has RSA public key (e,n) with private key (d,n). Suppose Alice wants BS to blindly sign M, so she wants to obtain M^d mod n. To reduce clutter, we will leave out "mod n" and assume all arithmetic is done mod n.

Alice gets BS to blindly sign M through the following steps:

1. Alice chooses random R
2. Alice computes R^e.
3. Alice computes $M*R^e$ and sends that to BS
4. BS raises that to d to obtain $M^d*R^{ed} = M^d*R$.
5. Alice divides by R to obtain M^d.

5.2 Encryption with blind decryption

We will have two types of "encrypt" functions in our blind decryption schemes:

1. encryption done with a public key
2. encryption with a secret function, but with blinding. In this case encryption needs to be done with the aid of the blind encrypter/decrypter BD. This requires choosing "blind" and

"unblind" functions that work with encryption as well as decryption. So to encrypt, a client of BD chooses functions B and U (blind and unblind), sends B(message) to BD with the request "please encrypt". BD performs E(B(message)), and returns the result. The client computes U(E(B(message))) to obtain the encrypted message E(M). Later to decrypt, the client again chooses inverse functions B' and U', sends BD B'(E(M)), and says "please decrypt". BD applies D and returns B'(M). The client applies U' to obtain the decrypted M.

In the next sections we describe three different types of mathematics that accomplish blind encryption/decryption. Two of them use encryption with a public key. One of them has both functions encryption and decryption as secret functions, so encryption must be done with help, and be blinded.

5.3 Blind decryption with an RSA key

This form is almost identical to blind signatures with an RSA key. The blind decrypter's public RSA key is (e,n). Again, we assume all arithmetic is done mod n.

Alice encrypts M by computing M^e.
Alice gets the BD to blindly decrypt M^e, by doing the following:

1. Alice chooses random R
2. Alice computes R^e.
3. Alice computes $M^e * R^e$ and sends that to BD
4. BD raises that to d to obtain $M^{ed} * R^{ed} = M * R$.
5. Alice divides by R to obtain M.

5.4 Blind encryption/decryption with a Diffie-Hellman public key

This form of blind decryption does not have a similar blind signature scheme. Assume that the blind decrypter's public Diffie-Hellman key is g^x mod p, where g and p are known. The private key is x. (We leave out "mod p" for clarity).

To encrypt message M with BD's public key, Alice performs the following:

1. Alice chooses random y, and computes g^y and g^{xy}. This is done by raising the publicly known base g to y, and BD's public Diffie-Hellman key g^x to y.
2. She uses g^{xy} as a secret key to encrypt M, obtaining $\{M\}g^{xy}$. She saves $\{M\}g^{xy}$ and g^y, and discards y and g^{xy}.

Again, as in the previous section, encryption is done without BD's involvement.

To get BD to blindly decrypt $\{M\}g^{xy}$:

1. Alice has $\{M\}g^{xy}$ and g^{y}.
2. Alice chooses random z, and its exponentiative inverse z^{-1}.
3. She computes $(g^{y})^{z}$, sends g^{yz} to BD.
4. The BD applies its private key (x) and sends to Alice: g^{xyz}
5. Alice raises g^{xyz} to z^{-1} to obtain g^{xy}, with which she can decrypt $\{M\}g^{xy}$.

5.5 Blind encryption/decryption without public keys

This form of blind decryption also does not have a similar blind signature scheme (and couldn't, because there is no public key with which to validate a signature). We use exponentiation mod p, a blinded version of Pohlig-Hellman [PH78]. BD does not have a public key, but rather has two secret numbers, x and x^{-1}, which are exponentiative inverses mod p. "Encrypt" will be done with x, "decrypt with x^{-1}. Blind encryption, like blind decryption, requires the involvement of BD. To get BD to blindly encrypt M:

1. Alice chooses random z, and its exponentiative inverse z^{-1}.
2. She computes M^{z}, sends it to BD, with the request to "encrypt".
3. BD applies x and returns M^{xz}
4. Alice applies z^{-1} to obtain M^{x}.

To get BD to blindly decrypt M^{x}:

1. Alice chooses random y, and its exponentiative inverse y^{-1}.
2. She computes M^{xy}, sends it to BD, with the request to "decrypt".
3. BD applies x^{-1} and returns M^{y}
4. Alice applies y^{-1} to obtain M.

Note that this scheme could be done with constant storage at the ephemerizer (as desired for the ideal forward secure scheme), by having BD's encryption exponent x for expiration time j+1 not be randomly chosen, but instead chosen as a one-way hash of the x for expiration time j. However, this would be computationally expensive.

6 Using Blind Encryption/Decryption for Ephemeral Email

It is easy to combine the ephemerizer concept with the blind encryption/decryption concept. Let's assume Alice is sending an ephemeral email to Bob. The ephemerizer makes available a set of (public key, expiration time, key ID) triples (if doing a scheme with public keys), or merely (expiration time, key ID) if doing a scheme with secret encryption and decryption functions as in section 5.5.

To ephemerally encrypt message M with public keys, Alice does the following:

1. She selects an appropriate (key=Keph, time, ID).

2. She encrypts the message using Keph. (by choosing secret key S, and calculating {M}S, and {S}Keph.
3. She must send {S}Keph securely to Bob. She might do this by encrypting with Bob's long term public key, or, if she shares a secret key with Bob, she can encrypt {M}Keph with that. Or Bob might actually equal Alice (Alice is encrypting stored files for easy shredding, for herself), in which case this step isn't necessary since Alice already has {S}Keph.
4. She also sends Bob Keph, since he needs to know Keph in order to create blinding/unblinding functions compatible with Keph. So she sends to Bob: {{S}Keph}Bob, {M}S, Keph. There is no need to encrypt or integrity protect Keph, since an attacker that replaced Keph with a different key would only cause Bob not to be able to decrpt the message.
5. Bob decrypts {{S}Keph}Bob to obtain {S}Keph. To decrypt {S}Keph, he creates blinding functions compatible with Keph, and blindly decrypts with the help of the ephemerizer. He can then decrypt {M}S and read it. His machine forgets S, M, and the blinding functions (i.e., does not store them in stable storage).

If using the secret encryption method of section 5.5, Bob needs to know the modulus p, just like Bob needs to know Keph in order to create compatible blinding functions in the public key case. Otherwise, it is a straightforward variant of the above. Alice just needs to get the ephemerizer to blindly encrypt the message. When communicating with the ephemerizer in the secret function variant, it is essential to specify whether you want it to apply "encrypt" or "decrypt" to the blinded quantity.

6.1 Authentication of the ephemerizer

Although all communication with the ephemerizer is blinded, it is still important for Alice to know she is talking to an ephemerizer she trusts. Otherwise, she could be tricked into encrypting the message with a key known to a party that is intentionally not going to discard it (or a nonexistent key, so the message will be unreadable even before it expires). In the case of public keys, the public keys can be certified with a long-term signature key of the ephemerizer (the signature key need not expire). In the case of encryption with a secret encryption function, Alice will need to authenticate who she is talking to when she presents the blinded message to be encrypted, with a protocol such as SSL.

6.2 Security properties

The ephemerizer could be built on a general purpose machine, that might wind up with infected code, but as long as the ephemeral private keys are kept on a tamper-resistant smart card, this accomplishes a secure path between Bob and the smart card. As long as trusted people supervise that the smart card is not stolen, and that it has a reliable source of power so that it knows when to discard keys, infected code on the rest of the ephemerizer will not be able to store anything away that could be used to decrypt messages.

Even if Bob's machine is compromised, and everything stored in stable storage is recovered, including Bob's long term key, messages that have expired will not be readable.

If Bob's long term key is not compromised before the ephemeral key expires, someone that sees an encrypted message for Bob will not be able to get the ephemerizer to decrypt the message.

There is no need for Bob or the ephemerizer to authenticate each other. Decryption can be done anonymously through an anonymizer [CA97].

Alice does need to verify that she has an authentic ephemeral public key from an ephemerizer she trusts, in the case of the public key variants. In the case of the secret functions variant presented in section 5.5, she must authenticate that she is indeed communicating with the ephemerizer.

The cryptography has to withstand a chosen ciphertext attack, since the ephemerizer cannot, by definition, see what it is decrypting [NY90].

6.3 Analysis

Messages appear in one of several places: on Alice's machine, in transit between Alice and Bob, on Bob's machine, in transit between Bob and the ephemerizer, inside the ephemerizer, in transit between the ephemerizer and Bob, and at Bob's machine after communication with the ephemerizer.

We assume that Alice creates the message and encrypts it using special software that does not save the message, or allow its data to be swapped out to stable storage. After transmitting the ephemerized message, Alice's machine deletes all the data.

What Alice sends on the wire to Bob is: {{S}Keph}Bob, {M}S, ID, Keph. {M}S is protected with S. Keph need not be protected. If an attacker substitutes a different Keph, it is only a denial of service attack. Bob will not be able to recover M. {{S}Keph}Bob is encrypted with Bob's public key. Only someone that knows Bob's public key before P's private key is deleted will be able to unwrap this in order to obtain {S}Keph.

Bob creates a blinding pair (B,U). What he sends on the wire to the ephemerizer is B({S}Keph). This quantity is protected because it is encrypted with an ephemeral encryption function B. B will be destroyed as soon as Bob receives the reply from the ephemerizer, after which nobody will be able to obtain any information from B({S}Keph.

The ephemerizer applies the private component of Keph to obtain B(S). S is encrypted with B, and cannot be recovered even if B(S) is saved and later Bob's machine is compromised, because B will be forgotten momentarily by Bob (and never was stored in stable storage).

Bob decrypts S and the message, but does not save these in stable storage.

If the ephemerizer really does not forget Keph, then assuming {M}S and {S}Keph are later recovered from stable storage at Bob, the message will be recoverable. This is why Alice must choose an ephemerizer she trusts, or use a thresholding scheme so that messages will be unrecoverable after the expiration time provided enough ephemerizers are trustworthy.

6.4 Secure channel to the ephemerizer's smart card

A way of building a cheap and secure ephemerizer is to have the private keys in a tamper-proof smart card, and connected to a general purpose machine. Assume that the machine might have malicious code, and that malicious people might have access to the machine, including with the smart cart connected. However, also assume that the smart card can be physically protected, not removed from the building (at least without being detected), and that it has a reliable source of power so that it will know when to delete keys. Perhaps it erases its memory if deprived of power.

With our ephemerizer scheme, there is nothing that can be gained by storing information transmitted over the wire, or seen by the general purpose computer portion of the ephemerizer. There is a secure path between the client (Bob) and the smart card. Destruction of the smart card, or overwriting its contents, can be done in a supervised way. Someone might steal the smart card, but that would at least be known. There is no way to covertly compromise the system.

Without a reliable source of power or access to a reliable clock, it might not be easy for the smart card to automatically delete the private key at the right time. However, we assume there are some honest people at the ephemerizer company that can supervise timely destruction of the keys.

6.5 Comparison between blind decryption and triple encryption schemes

Both have the same security properties, provided that the ephemerizer keeps its ephemeral keys on a tamper-resistant smart card that performs the cryptographic operations. However, the blind decryption scheme is more efficient for the ephemerizer, because it only needs to apply its private key once in order to decrypt a message. In the triple encryption scheme, the ephemerizer needs to do two private key operations per message decryption; once to decrypt {J}Keph, and once to decrypt {{S}KBob}Keph.

7 Conclusions

We presented a scheme in which a service known as an ephemerizer creates encryption and decryption functions with expiration times. The purpose is to concentrate the expense and expertise of secure key management in one place, and the expense of key management will be amortized over many users and many messages. The ephemerizer's functions must work together with blinding functions that Bob will create in order to decrypt an ephemerized message. Ephemeral encryption can be done with one of two public key schemes, or with a scheme in which the ephemerizer's encryption function is a secret function, and encryption must be done with the ephemerizer's help (and using blinding functions chosen by Alice, the entity creating the encrypted message). If the clients, when creating and reading messages, use special software that does not store decrypted messages in stable storage, expired messages will be unreadable once the ephemerizer deletes expired keys.

8 Acknowledgements

We would like to thank Charlie Kaufman, Hilarie Orman, and Gideon Yuval for their comments, which helped improve both the technical content and readability of this paper.

Bibliography

[A97] Anderson, R., “Two remarks on public-key cryptology”, Invited lecture, Fourth ACM Conference on Computer and Communications Security, April, 1997.

[BM99] Bellare, M., and Miner, S.K., “A Forward-secure digital signature scheme”. Advances in Cryptology - CRYPTO ‘99 Lecture Notes in Computer Science, 1999.

[CA97] Camp, L. J. (1997, February). Web security & privacy: An American perspective. ACM SIGCAS CEPC ’97 (Computer Ethics: Philosophical Inquiry).

[CH83] Chaum, D., “Blind signatures for Untraceable payments”, Advances in Cryptology - proceedings of Crypto 82, 1983.

[CJMM03] Cronin, E., Jamin, S., Malkin, T., and McDaniel, P, “On the performance, feasibility, and use of forward-secure signatures”, Conference on Computer and Communications Security, 2003.

[DH76] Diffie, W., and Hellman, M., “New directions in cryptography”, IEEE Transcations on Information Theory”, 1976.

[Dis] Disappearing, Inc., web site: http://www.specimenbox.com/di/ab/hwdi.html

[NY90] Naor, M., and Yung, M., “Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks”, 22nd Annual ACM Symposium on Theory of Computing, 1990.

[PH78] Pohlig, S., and Hellman, M., “An Improved Algorithm for Computing Logarithms in GF(P) and Its Cryptographic Significance,” IEEE Transactions on Information Theory, v. 24, #1, Jan 1978.

[RSA78] Rivest, R., Shamir, A., and Adleman, L., “A method for obtaining digital signatures and public-key cryptosystems”, Communications of the ACM, 1978.

Author Biography

Radia Perlman is a Distinguished Engineer at Sun Microsystems Laboratories. Her work has had a profound impact on the field on computer networking. She designed the algorithms that make link state routing protocols robust, efficient, and manageable, the spanning tree algorithm used by bridges/switches, routing in the presence of Byzantine failures, and many other contributions to network security and routing. She is the author of "Interconnections: Bridges, Routers, Switches, and Internetworking Protocols" and coauthor of "Network Security: Private Communication in a Public World", both textbooks used in many universities as textbooks, and as references by engineers. She holds about 80 issued patents, a PhD in computer science from MIT, and an honorary doctorate from KTH. She was named 2004 Inventor of the Year by SVIPLA (Silicon Valley Intellectual Property Law Association).