Object Security and Personal Information Management

Rolf Blom Mats Näslund Göran Selander

Ericsson Research
SE-16480 Stockholm, Sweden

email: {rolf.blom,mats.naslund,goran.selander}@era.ericsson.se

April 27, 2001

Abstract

We propose a new security architecture, the Object Security Architecture, enabling users to securely store data on (distributed) servers, not completely trustworthy, while at the same time allowing users to share the data according to their own trust models. Moreover, it provides a corporation with means to completely outsource storage of sensitive information. The architecture seems highly suitable for personal information management, e.g. calendars and email.

1 Introduction

With the increasing use of IP and Internet technology aspects like privacy and security becomes more and more important. Certainly, many applications will never become generally accepted unless they in a convenient way provide adequate security and privacy for users and information owners. The privacy and security should be provided in a world where the users are mobile, almost always connected via different types of wireless and wired access networks, using different clients and are enabled to interact locally work off-line.

If not already today then certainly in the near future users will belong to many different information domains. In general one may assume that information in one domain is shared with other members of the same domain but that information is not shared between domains. Examples of domains a user may belong to are his employer's corporate Intranet, his family domain and domains of different associations and groups. In many instances it would certainly be convenient for a user to have a unified and transparent view of all his information in the different domains. Users would in general also like to have exactly the same information processing capabilities in the different domains. Typical

examples would be clients for Personal Information Management (PIM) such as email, address book, or calendar.

In the future, users will most likely have terminals that are considered to be personal. Their employer might provide the terminals, but then for user convenience it will be required that personal information can be stored and/or accessed as well as company information. The terminal capabilities will vary greatly in terms of computing power and the bandwidth on the access will also vary. Wireless cellular access will in relation to other access methods usually have much smaller bandwidth.

Based on the observations above a user scenario is described below. The focus is on providing access and security for corporate information while at the same time provide privacy for the users personal information.

The paper is organized as follows. To motivate the applicability of the different security architectures to PIM, we start by giving some scenarios in §4.1. After some preliminaries in §3, we describe two possible ways to maintain PIM securely; transport security and object security, see §4, §5, respectively. In §6 we show how to implement a flexible distributed object security architecture. We compare the two architectures in §7, and give some concluding remarks in §8.

2 A Scenario

The starting point for the scenario to be presented is the needs of corporate users to manage personal information in their different roles as employees and private persons. In the scenario we define different players with certain trust relations and certain needs to share information. The trust relations together with corporate policies determine where information can be stored and how it can be retrieved and processed. We illustrate the problems with email and calendar functions.

The scenario considered is small and simplified and assumes only three users, Alice, Bob, and Eve. They work for companies A, B, and E respectively. All companies have firewall protected Intranets with remote access through security gateways. Alice, Bob, and Eve have accounts for email and PIM within their respective company. They use different personal terminals. Typically they use a laptop as their main terminal and when connected inside the company they have high-speed (wired/wireless) connections while outside the company low speed wireless access is the normal case. While on the move they use both PDA's and advanced phones to access and process information.

Eve trusts her employer completely and is not afraid that some network supervisor or maintenance person will read her personal mail and calendar and disclose or use the information inappropriately. So Eve stores also her information on a network server. Alice and Bob are more suspicious and keep their personal information on a private family server. The family server service could of course be bought from a service provider. Alice and Bob compare the situation to the way they handle company internal snail mail. They trust the system

but still use different kind of envelopes and sealing techniques to ensure the confidentiality and the integrity of paper messages.

Company Intranet policies usually aim to ensure that corporate information only is accessible by authorized users. Especially outsiders should have no access to proprietary company information. This makes it difficult for Eve to share her private information with persons not employed by company E. However Alice and Bob are in complete control of their information and may share it with whomever they want. Still Eve might have problems accessing the information from inside her company.

Company Intranet policies usually also require that all company information, wherever it is stored, can be retrieved without the involvement of the user/owner of the information. So the information stored on Alice and Bob's laptops might be revealed to the company unless it is protected/ciphered in a way which makes it readable only by Alice and Bob respectively.

Alice and Eve are both members of the AE golf club. AE has a server for keeping the members' calendars and an email system to provide for message exchange between the members. The idea is that mails exchanged between members should not be visible on the Internet and that the calendar information should simplify finding partners for a round of golf. Of course each user should only publish what they want to make public to other members. Alice would of course love to have some automatic mechanism to synchronize her calendars at work, at home and at the AE golf club. But the mechanism should provide privacy. A summary of the assumptions presented above used are:

- The company requires protection of corporate data.
- Users require that their private information will be kept private and that the user should control sharing it with others.
- Users are allowed to have private protected information, stored in their personal but company owned terminal. At the same time, information concerning their work should be available to the employer.
- Firewalls and security gateways will make it difficult for non-corporate users to (use end-to-end security protocols to) access private information of a corporate user inside the Intranet.
- Users want to have clients providing a unified and transparent view of all their information.

Notice that the somewhat heterogenous security and information sharing requirements makes it difficult for the parties to co-operate and share information in a convenient and secure way. In addition the solution may depend on the different terminal capabilities, which must be taken into account, e.g. the trusted device may be a so called thin client.

3 Preliminaries

We shall use the following terminology throughout the text.

Semi-trusted storage. A server/database containing data from one or several users performing input/output and disk storage actions according to specification by the owner of the data. The server may do other things like try to read or forward the data to other users, but at least it does what the owner specified.

A semi-trusted server may be considered honest but curious; it stores data for us but at the same time may pass the data on to someone else, may try to read or manipulate copies of the data for the purpose of extracting information. The semi-trusted device is assumed not to be malicious in the sense that it performs denial of service, e.g. destruction of data. Henceforth, we do not consider malicious servers, as there is really no way to protect against them (except that they would probably very quickly be "out of business").

An example of a semi-trusted server might be a well known Internet "hotmail" server. Some parties might even consider it trusted, although a big company would probably not trust it with its corporate email without any additional protection.

Trusted proxy. A server

- performing actions and requests precisely according to specification, e.g. it does no other manipulations with the user data than it is told to do and
- satisfying reasonable confidence in the inside administration of the server and reasonable protection from outside attackers.

Trusted storage. A semi-trusted storage fulfilling the requirements of a trusted proxy.

Trusted personal device. A trusted storage administered and accessible by a single user. This is normally the case with a laptop, a Personal Digital Assistant (PDA) or a cellular phone. Even if the device is owned by the user's employer, the data in it is not available to the employer.

Thin client. A computing device with very limited computational, storage, or I/O (low bandwidth, small display etc) capabilities. Examples are Palmtop devices or cellular phones. A client may be thin in one or several of these aspects. A laptop may have thin properties e.g. if the network access bandwidth is small. However we will mainly think of the worst case in which the client that is thin in every aspect.

We will also use some common cryptographic terminology for which we refer to [3]. We assume that secure symmetric and asymmetric encryption schemes and other necessary cryptographic primitives are available.

4 Transport Security Architecture

Definition 1. By a *Transport Security Architecture* we mean an information storage system where the information is protected during *transport* between two trusted entities in the following way:

- a key agreement between the entities, and
- end-to-end encryption (and possibly other security mechanisms, such as authentication and non-repudiation) of the data using these key(s).

Trust model: We assume that data resides in trusted storage. The security mechanisms only apply during transport.

Note that

- Transport security in this sense is independent on which layer in the protocol stack it is implemented, i.e. not necessarily at transport level. Thus IPsec [4], TLS [7] and encryption on application layer here all represent transport security.
- The level of which an implementation is made may of course be of importance but we make no distinction of this here.
- We do not require data protection in the end entities since we assume trusted storage.

We will now see how the previously given scenario fits into this architecture.

4.1 The Scenario

Suppose that any of Alice, Bob, or Eve want to access their corporate email from the public Internet. Though transport security could protect incoming mail to the corporate networks, it would probably not be possible for them to retrieve mail by say, logging on from an Internet café as company firewalls are likely to prevent such access. Thus transport security does not solve the problem.

Since Alice and Bob does not trust the corporate PIM system with their personal data, they would use a private server for that. Transport security may protect the contents, but will not give a very convenient way to keep track of data on two (or more) servers. For instance, Alice would probably first need to perform session key exchange with each individual server. Moreover, if Eve wants to make an appointment with either one, how does she know where to look for all calendar servers involved in keeping track of their appointments?

If the company wants to retrieve information from any of the employee's mailboxes, this is of course trivial, as data is stored in the clear on the server.

If Eve forgets her key/password protecting the data, it is likely that she can simply ask the system administrator to set a new password for her.

5 Object Security Architecture

In an Object Security Architecture (OSA) each information item ("object") is protected per se, and there are no security requirements on the transport itself. We first present a quite general definition, and then apply it to the special case of PIM.

Definition 2. By an *Object Security Architecture* we mean an information storage system where the information before transport to, and storage in, a semi-trusted entity is protected in the following way:

- applying encryption (and possibly other security mechanisms, such as authentication and non-repudiation) to the data using key(s)
- attaching information of the key enabling authorized parties to access the data.

Notice the similarity with S/MIME [5] in the case when the objects are emails.

Trust model: The only trusted devices are the trusted personal devices or the trusted proxies. The security mechanisms apply everywhere except in these.

Notice that this trust model is appealing, as all servers residing on the public Internet are essentially equal in the sense that they are all at least semi-trusted. This opens up for a distributed scenario in which many servers can be used to share data for one particular application (see §6). Also, the points of trust are much fewer; essentially the user himself and his personal devices, making security leaks less likely.

An inevitable consequence of the user being in charge of his/her own keys is the problem of key-recovery.

It is important not to confuse our notion of object security with the perhaps more common notion of "object oriented security", as for example in Java or Corba. There an "object" is an abstraction in the terminology of object oriented programming, and the goal of "object oriented security" is to protect, e.g. methods and local variables in these "objects", see e.g. [1].

How could we realize an object security architecture, in the case of "PIM objects"? The reader may think of a PIM object as e.g. an email, a calendar entry, or address book entry. There could also be cases where the entire calendar could be viewed as an object. We will see how these different ways of blocking data into objects have consequences on the granularity by which the owner can grant access to his/her data.

5.1 Realization

There may be many ways to implement this, we shall aim at a generalized version of an S/MIME-based approach applied to PIM-objects (including emails). We briefly mention S/MIME and then move on to some techniques.

5.1.1 S/MIME

On a higher level S/MIME encrypts the email with a randomly chosen key which in turn is encrypted by the public key(s) of the recipient(s). These, so called *key packages*, are appended to the encrypted email and a "header" is attached describing how to parse the message, which algorithms to use etc. When the receiver gets the S/MIME object, his mail program parses the message, performs the necessary cryptographic operations and retrieves the plain text. The same techniques can of course be extended to handle more general objects than email. This is precisely what we propose to do, with a few minor adjustments.

5.1.2 Techniques

There are differences in handling email and PIM objects in general. An email carries static information between a sender and receiver(s) at a given instant. In principle when the email has been delivered (and read) it has served its purpose. On the other hand, a general PIM object is first created and stored remotely. After that the object can be accessed (read/write/edit) many times, since a PIM object maintains dynamically changing information.

We will use the following notation:

o data object

c encrypted object

$E_k(o)$ Encryption of o with key k using symmetric encryption algorithm

$D_k(c)$ Corresponding decryption of c with key k

$V_p(o)$ Encryption of o with a public key p using asymmetric techniques (it may also mean a verification of a signature o)

$S_u(c)$ Corresponding decryption of c with a secret key u (it may also mean a signature of c)

$H(o)$ Cryptographic hash of o

u, p are the secret/public key of the owner of the object

For simplicity we thus assume one single public key scheme, which provides both encryption, decryption, digital signature and verification of signature. For the purpose of handling general PIM objects, we suggest the following object formats and procedures for storing and retrieving objects.

Object Format. Two formats are suggested:
For encryption (and optional authentication) of objects, use

$$E_k(o), \{V_{p_i}(k), h(p_i)\}_{i=0}^{n}, [\sigma] \tag{1}$$

Where $p_0 = p$ and where $p_1, \ldots, p_n$ are the public keys of the users authorized by the owner. The optional signature $[\sigma]$ equals S_u applied to everything but $[\sigma]$ in (1), see Figure 1.

We will refer to a public-key encrypted key, indexed by the hash of the public key as a *key package*.

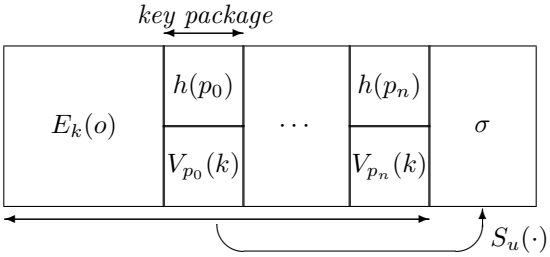

Figure 1: Principal format for secured objects.

For authentication only of objects, use

$$o, S_u(o). \tag{2}$$

It may seem strange that we allow unencrypted objects since the servers are only semi-trusted, but this will prove to be useful. We will in the following interchangingly use "object" both to denote the unprotected plaintext data and the protected version of it.

General Access Protocol. A user is supposed to request an object from a storage by sending the hash of his/her public key $h(p)$. The server is specified to check with the indices in the key packages if this string is included. If so, the server shall send the entire protected object (1) to the user.

Create Object. This operation takes the plain text object o stored or created in the trusted personal device, protects it by the format described in (1) and sends it to a specified semi-trusted storage.

Access Object (by owner or non-owner). A user, j, requests access to an object from a storage by sending $h(p_j)$ and should, if this user is authorized (has a public key among the key packages), receive the entire protected object according the protocol. The user client can now (if it indeed holds the secret key corresponding to the claimed public key p_j) decrypt the object and if desired verify the signature of the owner. Note that even if the server erroneously sends out the protected object to an unauthorized user, this user still cannot retrieve the original object.

Update Object. First access the protected object as in the previous section, edit the clear text object and create a new protected object as in the "Create" step. Note that we would normally assume that only the owner can

change an object, but any user having rights to create objects on the server can upload his/her version. Still he/she cannot forge a signature of the original owner so he cannot change the ownership of the object. In practice, an access control function should be necessary to keep control over who that can upload to a server.

Delete Object. The (claimed) owner sends a delete request containing a pointer to a protected object o and his public key p having corresponding secret key u. The server replies with a challenge c and expects a signed response $S_u(c)$. Now the server checks the signed response and also that the signature of the referred object is made with the same p. If so, the server deletes the object.

We are now ready to see how well the problems indicated in §4.1 are solved with this architecture.

5.2 The Scenario

In an OSA, Alice, Bob, and Eve can access their email and other personal data in a homogeneous way from *anywhere*; there is no need for firewalls that needs to be traversed, at least not for the purpose of protecting the data. Neither is there a principal need for performing key exchange with the servers as objects are already protected and can be transmitted "as is". (Of course, adding transport security too might be advisable in some cases, see below.)

Moreover, Alice and Bob who are a bit "paranoid" can themselves decide who can access their objects, and does not need to rely on a third party maintaining a server securely. This access can be granted at different levels on granularity, by for instance setting individual rights to entries in a calendar, rather than to the entire calendar itself.

When data objects are to be distributed on different servers, also in this case there is a problem to know where to find the locations. We discuss this below, and for the moment just note that at least, one does in principle not need to perform transport security key exchanges with each individual server.

One thing that needs to be taken care of however, is to enable a corporation to access data belonging to its employees, as even if they are stored on a company server, the data is now encrypted by user selected keys. To ensure such access, we can for instance force the user to add a key package for the system admnistrator. Thus, before storage is allowed, the server/system checks that a valid key package indeed has been added (see later discussion). If a user loses the key this mechanism is also applicable for key recovery.

What remains is to discuss how a user can create a "unified view" on data that is distributed over several servers and how one publishes information on where in this architecture other users can find data belonging to a given owner, see §6.

5.2.1 Practical Experience

We have implement an object security based architecture for corporate email. Mail created in, or sent to, the Intranet are handled by a trusted proxy that encapsulates them as secured objects, and forwards them to a semi-trusted server located outside some corporate firewalls. User can then, for instance when traveling, access their email in a convenient way, see [2] for details.

5.3 Issues

5.3.1 Security

Confidentiality, integrity, and non-repudiation protection is clearly obtained by the above procedures.

There might be cases where, for convenience, confidentiality is not wanted. For instance, a user may by willing to publicly display which times he has appointments (without disclosing the content of the appointments). This is possible using the format (2) above.

Privacy is an issue for the owner as well as the authorized users. First of all, it seems hard to obtain privacy to the users without sacrificing the non-repudiation of the object.

Without any normal user authorization and access control to the server, anyone can essentially find out whether a given user has right to access a particular object. In practice such access control will exist, to prevent the server from being an easy target to denial of service. Since there is no transport security it is possible to determine if one user reads the same data as yesterday, two users access the same object etc. Adding transport security would prevent a passive eavesdropper from obtaining such information although the trust model of the server does not exclude the possibility of it sending the objects to anyone.

With the protocols suggested, it is not possible to create complete privacy for the users, since it is possible by trial and error to query the server, thereby learning which users that are allowed to access a certain object. However, with some minor adjustments we can obtain some degree of privacy for the owner and all users with respect to passive or active attackers that are not in league with the server, but eavesdrops or intercepts the protocol. Since the the server is only semi-trusted, we cannot hope to achieve better.

We now describe one way of doing this. The following protected object is uploaded from the owner for storage

$$E_{k_1}(o), \mathrm{MAC}_{k_2}(o), \{V_{p_i}(k), h(p_i)\}_{i=0}^{n}, [\sigma] \tag{3}$$

using the same notations as previously and where k_1, k_2 are derived from the master key, k, and $\mathrm{MAC}_{k_2}(\cdot)$ is a (keyed) Message Authentication Code using the key k_2.

Again, the optional signature $[\sigma]$ is obtained from S_u applied to the first three parts of (3). Furthermore, the access protocol is modified in this way:

Upon request of an object accompanied by an (alleged) $h(p_j)$, the server checks this with the key packages of the protected desired object. If it is included, send

$$E_{k_1}(o), \mathrm{MAC}_{k_2}(o), V_{p_j}(k) \tag{4}$$

otherwise send

$$E_{k_1}(o), \mathrm{MAC}_{k_2}(o), r \tag{5}$$

where r is randomly chosen and of appropriate length.

Note that privacy protection is not complete, as any of the authorized users may determine if another user is authorized or not, but an unauthorized user cannot.

Also, this MAC-based solution can be used to implement "corporate key-escrow" as it is now very easy to check that the system administrator has a valid key package. (Though we of course cannot guarantee that o itself is not already encrypted with another key.)

5.3.2 Scalability

There are two scalablilty issues; the number of authorized users and the number of servers on which the owner distributes the objects. We postpone the discussion of the latter until §6.

First, a large scale implementation requires a Public Key Infrastructure (PKI) or a Web of Trust, e.g. PGP. The size of the object in relation to the number of users determines the storage efficiency of this architecture. A single key package created with a reasonable security level requires about 400–1000 bits depending on the encryption technique. Although it is in principle possible to set individual rights to each individual item, in practice this will not be done for efficiency. The access to the objects may be set with a coarser granularity, for instance to an entire calendar rather than an individual item; or one key for the calendar index, stating when the owner is busy and one key for the actual calendar items. Of course, a group key distribution scheme may increase the efficiency whereby cryptographic access is granted depending on group membership.

The revocation of a previously authorized user can be done by re-encrypting all objects accessible by that user with a new key and adding key packages of the authorized users.

Adding a user is simpler, since the old key packages can be downloaded and reused, simply by adding a key package for the new user, sign and upload.

5.3.3 Thin Client

To make things more realistic (and complicated) we assume that the trusted personal device is a thin client. Since all clear text operations as well as encryption and signing is done in the trusted device, such an assumption implies significant constraints on the implementation of the actual scheme, on feasible applications and also on the number of users. Here are some examples of research areas that have a direct relevance for this case.

Incremental Cryptography. This is based on the idea of being able to sign/encrypt only recently updated blocks/parts of a large file/object. This seems to be a useful technique since it may save bandwidth during transfers and lighten the computational burden on the client. See the references for some techniques in this area.

Server-Assisted Cryptography. Here the idea is for the thin client to get help from a (semi-trusted) server to perform heavy calculations. e.g. public key operations. The problem is to avoid that the server learns sensitive information either of the object or of the cryptographic keys (and of course not actually do more work in the client to assure this). A number of techniques exist, see References.

Operations on Encrypted Data. Suppose that an object contains a number of sub-objects, i.e. actual appointments in a calendar-object. How do we determine whether a certain string occurs in his emails. We would typically need to download the entire mailbox, decrypt, and then search the plain text data. However, it is in some cases possible to have the server performing operation on the encrypted data without supplying him with the keys, searching being one example of this. In fact it can be done without even disclosing the search pattern. This is done with special encryption techniques see [6].

It would be interesting to look into these areas further.

6 Distributed Architecture

We now address the question of how to manage personal information with distributed storage. We can think of a calendar with items on several semi-trusted servers. The questions we address are: –How can the owner get a unified view of the entire calendar? –How can an authorized user get access of the owner's various calendars? We have already seen how the owner can restrict access to the content of certain appointments, now we also discuss how the owner can restrict access to/knowledge of specific calendars to specific users.

We have previously discussed the option to have access control lists in the server. If the server was completely trusted, we could count on him checking authorization before granting access to users, but we are only assuming semi-trust. Here is a way to solve this problem, again by object security.

We know that Alice has at least three calendars: work, private, and golf club. Suppose that Eve and Alice are not only members of the same golf club, but also has a business relation. Alice might want to share her golf- and work calendars with Eve, but perhaps not her private. To accomplish this, since they are both members of the golf club, Alice creates an access control list stored at the golf club server which contains a link to the server handling her work calendar, encrypted by Eve's public key. When Eve accesses the golf calendar, she will also be given this protected link by the server. Thus, Eve can get a calendar view, consisting of the union of the golf- and work calendars of Alice.

Of course Alice can still restrict access to the individual calendar items in both calendars as described above. For instance, she might want to publish the time intervals (treating them as objects) when she is busy, but not what she is doing.

Alice could have many friends and calendars, and she can by treating the links to other calendars as objects, completely control who sees what.

There is of course some work to do when a person should become included or excluded from certain calendars. A user friendly administrative software running in Alice's trusted personal device is absolutely necessary.

6.1 Privacy

The goal with the above distributed architecture is to achieve user-controlled privacy. However, complete protection seems difficult. Even if the links between calendars are encrypted, traffic analysis might establish, or give hints to, how data is distributed among servers.

7 Architecture Comparison

There are two main strands in the choice of security architecture. In a transport security architecture (TSA) a data link is set-up between client and the server, and to protect the link against attacks, well-known security protocol like IPsec, TLS, or SSH are used. In an object-based architecture (OSA) each information item is protected per se, and there are in principle no security requirements on the transport itself. Only the intended recipient(s) is able to see the contents, which remains encrypted. The security technology can be provided by a generalization of S/MIME.

There are several pros and cons for both architectures. In both cases the data is only protected while not residing in a trusted server, but the trusted environments are different. In TSA, security is coarse-grained and a client has, after authentication, access to the whole domain attached to the other end of the link. The domain itself is usually protected by a strong perimeter. For example, virtual private network (VPN) tunnels into an Intranet extend the Intranet to the user's client. To reduce risks, before introducing VPN, some Intranets are restructured in order to restrict resource access. All this implies more costs and complexity and affects the corporate security policies and firewalls.

By contrast, in OSA, fine-grained security is employed and since the information items are secured, they can be transported over any type of network and stored anywhere, even outside the trusted domain, on untrusted servers allowing for data to be mobile. This simplifies the access problem, but adds complexity to the protocols that manipulate the data. Since information is encrypted, only the intended recipient can manipulate it and thus operations like searching, sorting and adaptation may need to be performed in the client. This might be a problem on thin clients or over narrow access channels. Such operations can possibly be off-loaded to trusted proxies, or applying server-assisted

crypto techniques that aid the client. The challenges here are to find techniques that entirely remove the need for a trusted proxy or to design a proxy that minimizes the risk of exposing confidential data. In the latter case, we also need mechanisms for temporal delegation of rights from the user to the trusted proxy.

The client must handle information from different domains and should provide a unified and transparent view. This can be done in the client itself, provided it is powerful enough. The client gets the views from the distributed domains and then integrates them. The other way is to have the domains synchronize against a single untrusted server, to which various clients have access.

8 Conclusions

Object security and transport security architectures solve different problems. A security architecture which works under the assumptions in the scenarios summarized above must have elements of object security to guarantee protection of user data.

The OSA opens up for an easy way to distribute and share personal information securely according to a personal security policy set by the user himself.

A lot of intriguing research problems, e.g. different kinds of server assisted and incremental cryptography would have direct practical implications in this setting.

References

[1] Blakely B.: *Corba Security.* Addison-Wesley, 1999.

[2] Gehrmann C.: *Flexible Corporate S/MIME Security Architecture.* Proceedings, Nordsec '99, pp. 85–96, Dept. of Computer and Systems Sciences, Stockholm University, 1999.

[3] Menezes A., van Oorschot P., Vanstone S.: *Handbook of Applied Cryptography.* CRC press, 1997

[4] RFC 2401: *Security Architecture for the Internet Protocol.* IETF.

[5] RFCs 2630, 2631, 2632, 2633: *S/MIME Proposed Standard.* IETF.

[6] Song D. X., Wagner D., Perrig A.: *Practical Techniques for Searches on Encrypted Data.* Proc. of IEEE Security and Privacy Symposium, May 2000.

[7] RFC 2246: *The TLS Protocol Version 1.0.* IETF.

Server-Assisted Cryptography

[8] Anderson R. J.: *Attack on Server-Assisted Authentication Protocols.* IEE Electronics Letters **28**(15), p. 1473, 1992.

[9] Burns J., Mitchell C. J.: *Parameter Selection for Server-Aided RSA Computation Schemes.* IEEE Trans. On Computers **43**(2), 163–174, 1994.

[10] Horng G.: *An active attack on protocols for server-aided RSA signature computation.* Information Processing Letters, **65**(2), 71–73, 1998.

[11] Horng G.: *A Secure Server-Aided RSA Signature Computation Protocol for Smart Cards.* J. Inf. Sci. and Eng. **16**, 847–855, 2000.

[12] Merkle J.: *Multi-Round Passive Attacks on Server-Aided RSA Protocols.* Proceedings, CCS '00, pp. 102–107, 2000.

[13] Merkle J., Werchner R.: *On the Security of Server-aided RSA Protocols.* Electronic Colloquium on Computational Complexity (ECCC), TR97-027, 1997. http://www.eccc.uni-trier.de/eccc/

Incremental Cryptography

[14] Bellare M., Goldreich O., Goldwasser S.: *Incremental Cryptography: The Case of Hashing and Signing.* In "In Advances in Cryptology – Crypto 1994", Lecture Notes in Computer Science, Vol. 839, pp. 216–233, 1994.

[15] Bellare M., Goldreich O., Goldwasser S.: *Incremental Cryptography and Application to Virus Protection.* Proceedings of 27th ACM STOC, pp. 45–56, 1995.

[16] Bellare M., Micciancio D.: *A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost.* In "Advances in Cryptology – Eurocrypt 97 Proceedings", Lecture Notes in Computer Science Vol. 1233, pp. 163–192, 1997.

[17] Fischlin M.: *Incremental Cryptography and Memory Checkers.* In "Advances in Cryptology – Eurocrypt '97", Lecture Notes in Computer Science, Vol.1233, pp. 393–408, 1997.

[18] Fischlin M.: *Lower Bounds for the Signature of Incremental Schemes.* Proceedings, 38th IEEE FOCS, pp. 438–447, 1997.

[19] Micciancio D.: *Oblivoius Data Structures: Applications to Cryptography.* Proceedings, 29th ACM STOC, pp. 456–464, 1997.