NORDSEC 2005

Proceedings of the 10th Nordic Workshop on Secure IT Systems

October 20-21, 2005, Tartu, Estonia

Helger Lipmaa, Dieter Gollmann (eds.)

In cooperation with

ISBN 9949-11-153-6

Tartu University Press

http://www.tyk.ee

Preface

You are reading the proceedings of NordSec 2005, the 10th Nordic Workshop on Secure IT-systems. The workshop was organized by Cybernetica AS and University of Tartu, and sponsored by the United States Army Research Laboratory European Research Office and Hansapank. The organizing committee was responsible for the local organization. We thank the members of organizing committee (Jan Willemson, Uuno Puus, Peeter Laud and Liina Kamm) for all their work and for the pleasant collaboration.

A total of 32 papers were submitted of which 15 were accepted for presentations at the workshop. The program also lists invited talk by Peter Landrock ("PKI — Past, Present and Future"). Also, there was a student session, which Jan Willemson kindly agreed to chair.

During the reviewing process, every paper was reviewed by at least four members of the program committee, and papers co-authored by a member of the committee were reviewed by at least six other members. It was a pleasure for us to work with the program committee.

We are very grateful to the additional reviewers who contributed with their expertise: Jong Youl Choi, Igor Semaev, Margus Freudenthal, Magnus Almgren, Christian Probst, Steve Myers, Matthew Parker, Arne Ansper, Minaxi Gupta, Liu Yang, Alexander Kholosha, Rene Rydhof Hansen, Terkel Tolstrup, Carlos Cid, Håvard Raddum.

My work as program chair was made a lot easier by the electronic submission software written by Chanathip Namprempre for Crypto 2000 with modifications by Andre Adelsbach for Eurocrypt 2001, and by the reviewing software developed and written by Bart Preneel, Wim Moreau, and Joris Claessens for Eurocrypt 2000. We would like to thank Veiko Tõeleid for setting up all this software locally and for the help with the problems we encountered.

Finally, a thank-you goes to all who submitted papers to the workshop.

October 2005 — Helger Lipmaa and Dieter Gollmann

Program Committee

Helger Lipmaa (Co-Chair)	Cybernetica AS and University of Tartu
Dieter Gollmann (Co-Chair)	TU Hamburg-Harburg
Tuomas Aura	Microsoft Research
Ahto Buldas	Cybernetica AS and University of Tartu
Catharina Candolin	Helsinki University of Technology
Mads Dam	SICS and Royal Institute of Technology
Pasi Eronen	Nokia Research Center
Viiveke Fåk	Linköping University
Tor Helleseth	University of Bergen
Markus Jakobsson	Indiana University at Bloomington
Christian Damgaard Jensen	Technical University of Denmark
Erland Jonsson	Chalmers University of Technology
Svein Knapskog	Norwegian University of Science and Technology
Peeter Laud	University of Tartu and Cybernetica AS
Sven Laur	Helsinki University of Technology
Jussipekka Leiwo	Nanyang Technological University
Chris J. Mitchell	Royal Holloway, University of London
Hanne Riis Nielson	Technical University of Denmark
Einar Snekkenes	Gjøvik University College
Jan Willemson	Playtech and University of Tartu

Contents

PKI — Past, Present and Future

Peter Landrock
Cryptomathic

How it all started

PKI, Public Key Infrastructure, is the enabler of two significant features: Key Exchange without sharing a secret key beforehand and Digital Signatures potentially even Non-Repudiation for Transaction Security which can be verified by anybody in principle using a non-secret (consequently called Public) Key. Nothing more, nothing less per se. One of the first attempts to develop a framework architecture for the use of public key techniques was the X.509 standard, which has been with us ever since and convinced most potential users that certificates always are necessary and essential for digital signatures, even though this is not quite the case, the obvious exception being electronic banking. But X.509 was not designed to provide an architecture for Electronic Commerce (EC) however, and this is one of the main reasons why EC based on X.509 has never taken off. The legal implications are too complicated; in particular if we have to protect the user as we usually do e.g. in Western Europe. After all, he is a voter, too!

There is no doubt that you need a trusted authority of some kind, a so-called Trusted Third Party, to handle the—essential and unavoidable—registration of users, and the administration of the public keys. There is a small opposition who believe that this is not necessary. Their suggested alternative is to rely on PGP (which is basically nothing but a syntax for the use of keys, just as S/MIME) and a Trust model where you issue your own certificates and share this with your friends. This would be just as ineffective as sharing telephone numbers with your friends instead of using directories, and will only work in (very) small closed user groups with an extensive trust relationship.

PKI in the past

One of the first unfortunate PKI solutions we saw was the SET protocol, which—at least so it seems—was designed by security-nerds with no feelings for the limitations of ordinary users. Exit SET.

The announcement of the expected explosion of EC almost a decade ago created enormous activity in companies dealing in security. Many companies grew dramatically large virtually overnight, if not so much due to large sales, then at least due to heavy investment and marketing. And the dilemma these companies all faced was that their customers knew nothing about security—and had no interest—whilst their designers knew very little about commerce.

The main problem was that basically the X.509 approach was offered for all PKI-solutions, and thus the applications had to be adjusted—non-trivially—to the security architecture and not the other way around. The number of PKI-solutions exploded, but pilots quickly revealed the shortcomings and the unresolved challenges.

In fact the experience gained so far from various pilots by the end of the previous millennium were so bad, that some banks and companies declared PKI dead and useless. This of course is completely wrong. The new generation of debit- and credit chipcard (EMV) technology is based entirely on PKI and is already a success. Why? Because the PKI provided is transparent to the user.

Why Certificates?

Once you receive a certificate of another user, your first thought ought to be: Is it still valid? This is why the concept of Certificate Revocation Lists (CRL) were introduced: The idea behind CRLs comes from the method used for magnetic credit cards: At regular intervals, CRLs are submitted to subscribers, and

whenever users receive a digitally signed message with an embedded certificate, they should check if the certificate is on the list (just imagine the average user doing that!). These lists may be broadcasted on a regular basis, and the processing at the end-users becomes tedious and complicated, as the lists grow in number.

However, even worse, if a particular valuable digital signature is received, it does not suffice to rely on the last received CRL. What if the certificate was revoked after the last broadcast?

The superior solution of course is to immediately request a new version of the certificate (a so-called instant certificate) from the CA/Directory using an appropriate protocol, such as one of the PKIX CMP messages, although this is a move cumbersome solution. The user application may then have a cache available of certificates of entities he communicates with, and may even have a security profile integrated which automatically updates information on a certificate. It should be the user's responsibility to inquire, not the Directory's responsibility to broadcast. Or even better, it should be the responsibility of the user's software to inquire, because the typical user can't be bothered.

A certificate by definition is history. It is absurd to think of the expiry date as anything but the last possible expiry date. When you receive a certificate, you will never know if it *has already expired*. It would therefore make perfect sense never to include a certificate in a message, but a reference number to the public key, rather. It is then up to the receiver how often he will inquire at the appropriate directory about the status of the corresponding public key e.g. using OCSP (On-line Certificate Status Protocol), which then should be called OPKSP (On-line Public Key Status Protocol). The answer will of course be a signed statement, which we may choose to call a certificate, perhaps an *instant* certificate. Moreover, this should all be parsed by the application, not the user.

Present solutions

Meanwhile, in a much more quiet manner, the EMV-standard was developed for the use of chipcard based debit- and credit cards all over the world. It is a full-blown PKI solution, but the beauty of it is that the end-user doesn't know and doesn't have to know. He uses his card just as in the magnetic stripe days.

Other areas of large scale solutions are TPM (Trusted Platform Module) and DRM (Digital Rights Management). The main purpose of these applications is to issue various chips in various devices with public key pairs in order to be able to verify the devices properly, control installation of authorised SW and protect Intellectual Property Right. This is an area where the basic X.509 architecture works well as there are no legal implications as such of revoked keys.

Another area where we—finally—see some growth in Europe is national PKI solutions with the primary purpose of securing communication between government and citizens. Again, the legal challenges here are typically very limited, and it will help the introduction of e-government, which potentially could be very cost-efficient.

Future Solutions

One path forward here which is flexible, mobile, secure and—not to be forgotten—is to make a secure back-end available for each user, his virtual smartcard. This secure back-end is a fast tamper resistant HW-unit, which may service a number of users at the same time with each their individual key pair. Each user has secure access to his own key pair in the unit by means of a secure token. He is also responsible for his own key generation. This is particularly appealing to employees at public offices, as they are then not confined to one work station.

The basic idea is the following: When the user engages himself from any workstation connected to the Internet, he carries out his business as usual. Once he is ready to have a message or document digitally signed, a hash value is calculated and send to his secure-backend, where it is signed at his instruction using his token to provide the signature instruction. His private key NEVER LEAVES the back-end.

Obviously, as the user is carrying his business out from an insecure workstation, it would be unacceptable to use any access control mechanism via this workstation. In stead, the user is given a token which could be a traditional token based on a challenge-response protocol using a symmetric algorithm. Alternatively, he could use his mobile phone to exchange one-time passwords, thus providing a separate

secure channel to the back-end for authorization of his signature, one of the oldest and most secure tricks in the trade.

But the main feature we propose is to use the token to authorize the generation of a signature, not to authorise a transaction. So solutions are now in place, e.g. in Denmark.

In addition, in view of our past experience, the next step for adequate PKI in EC could be not to use certificates, but credentials only with a public key reference number. If I receive e.g. a signed e-mail—and I can be bothered to verify the signature, my application will automatically contact the TTP responsible for the registration of his public key and receive an instant certificate which states that at the time of request the public key had not been revoked (yet ☺)

It has been anticipated that signatures in addition will find a widespread use in e-mails, e.g. amongst citizens. We do not believe that. To use a digital signature, there must be a motivation, e.g. an application. Consequently, the way forward for governments who want to promote digital signatures is to offer interesting applications. Citizens have no other use for them.

A Security Architecture for an Open Broadband Access Network

Martin Gilje Jaatun,
Inger Anne Tøndel,
and Maria Bartnes Dahl
SINTEF ICT
Trondheim, Norway
Email: Martin.G.Jaatun@sintef.no

Thomas J. Wilke
PRZ
Technische Universität Berlin
Berlin, Germany
Email: tjw@prz.tu-berlin.de

Abstract

Europe is experiencing a rapid growth in residential broadband coverage, but due to usage patterns and cost structures, only a fraction of the available bandwidth is actually being consumed. This implies that most residential broadband subscribers have excess capacity, and the idea of the Open Broadband Access Network (OBAN) project is that this capacity can be shared with passers-by.

In order for the residential broadband subscribers to open up their networks, and for the potential wireless customers to sign up for OBAN service, the security of both parties must be ensured. OBAN needs to solve the problems posed by the fact that a visiting OBAN user and a residential access point operator have no pre-existing trust relationship. This paper describes an architecture that achieves this. In addition, the architecture ensures that all participating parties are able to prove the amount of traffic transferred in any given OBAN session. This enables a broader range of business models with respect to charging of visiting OBAN users, remuneration of residential subscribers, and cooperation between service providers. This may in turn result in new business opportunities.

Keywords—Authentication, Excess Capacity, Security Architecture, Wireless Access Networks

1 Introduction

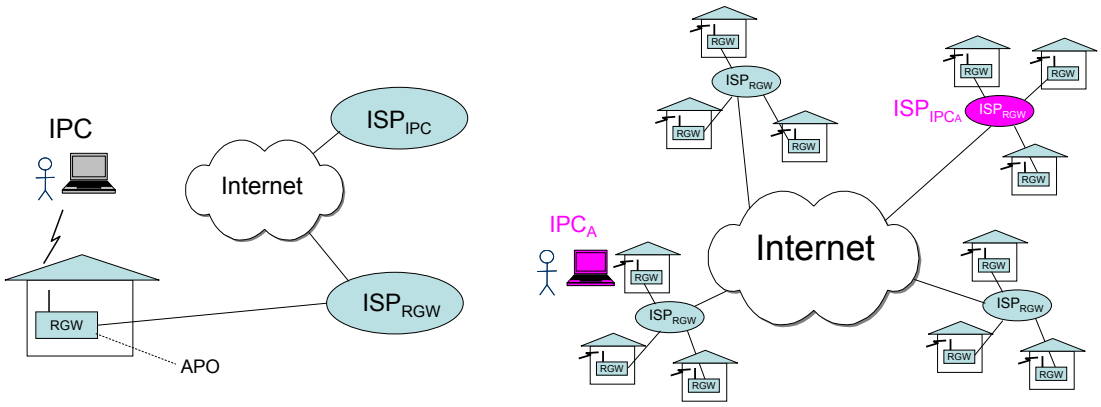
The world is experiencing increased broadband coverage in residential areas, but due to usage patterns and pricing models, only a fraction of the available bandwidth is actually being consumed [1]. The excess capacity could be put to good use, however, if residential installations were to share their bandwidth through public wireless access points [2]. In a nutshell, this is what OBAN [3] is all about. The general background for the project is discussed in detail in e.g. [4] and [5]; in this introductory section we will give a brief overview of OBAN, but otherwise concentrate on security aspects.

1.1 The OBAN Concept

The idea of OBAN is to place publicly available wireless access points in homes (and possibly business premises). These access points are operated by an Access Point Operator (APO), which may or may not be the owner of the premises. The available bandwidth is shared between residential users and visiting OBAN users, in the following referred to as IPCs (IP Customers). The bandwidth may be shared in different ways, either with a fixed amount reserved for the residential user, or by the use of various priority schemes (see [6]).

There are various possible scenarios for OBAN deployment, but a solution that will be applicable to many markets is illustrated in Figure 1(a). In this scenario, we assume that a residential broadband user is offered the use of a Residential Gateway (RGW[1]) from his Internet Service Provider (ISP_{RGW}). This RGW could be designed as a replacement for any existing broadband router, and will contain wireless

[1] Also referred to as simply "RG" in other documentation



(a) OBAN Parties

(b) An OBAN Community

Figure 1: Overview of OBAN Parties, and the Big Picture

access point functionality. The RGW will be administered by the APO, which in this scenario may be the residential user or ISP_{RGW}. Even in the latter case the residential user will always have opportunity for limited local configuration.

IPCs are required to have a subscription with a participating ISP, known as ISP_{IPC} in this context. When an IPC is in range of an RGW, a connection will be established with ISP_{RGW}, which in turn will forward the IPC's credentials to ISP_{IPC}. The ISP_{IPC} then acknowledges that the IPC is indeed a valid customer, and promises to honour any obligations made by this user while visiting this particular RGW.

Once authenticated, the IPC can use wireless IP services much like a residential user. Should the IPC wander out of the range of the current RGW, OBAN supports roaming if another RGW is within range (but see also section 2). As indicated by Figure 1(b), a single ISP_{RGW} will typically have several RGWs under its administrative control.

All ISPs participating as OBAN Service Providers will typically play two roles in the OBAN community, as illustrated in Figure 1(b). On one hand, they will serve as ISP_{RGW} to their residential users, but also as ISP_{IPC} for their users on the move.

1.2 Business Models

Many different business models can be envisaged for OBAN, both when it comes to organisation and accounting.

Regarding accounting, IPCs could be billed based on the amount of services consumed, or simply based on a flat monthly fee, to name a couple of alternatives. The solution proposed in this paper will be able to support billing based on service consumption, but this does not exclude simpler options. Flexibility in accounting is also relevant for the residential users, since the remuneration strategy chosen will depend on how these pay for their IP services. Note that there will be a significant incentive for residential users to join the OBAN concept; either increased bandwidth, reduced subscription cost, or both. It is even conceivable that in certain high-volume locations, the residential user may make a net profit on the OBAN participation, in effect getting broadband access for free *and* making extra money.

For residential users in down-town locations, usage patterns are likely to be the inverse of visiting OBAN users, i.e. visiting users are likely to be active during business hours, while residential users primarily are active after-hours. This can result in a win-win situation, where a residential user can earn money on a 100% of spare broadband capacity without suffering reduced performance.

The most interesting organisational aspects relate to the administration of the RGW; several parties are candidates for the role of APO. The simplest approach would be to let ISP_{RGW} assume this role, since the APO and the ISP_{RGW} then would be the same party. However, there are also advantages in delegating this role to other parties. This way one may have pure ISP organisations that are specialized

in providing connectivity at larger distances, while other parties may specialize in administering end-user equipment. ISPs may also see the advantage in letting the residential user perform more of the management duties, thus reducing the ISP's maintenance costs. The case where the APO is a separate entity from the ISP_{RGW} will therefore be supported.

1.3 Novel Contributions

Offering public wireless access is certainly not new, as anyone who has spent some time in major airports or hotels can verify. Using private residences as a platform for offering such services has also been done before, e.g. as implemented by LinSpot [7].

Roaming between hotspots has however only been possible on a limited scale, and certainly not between different service providers. Furthermore, public hotspots have to a large extent been vulnerable to so-called "evil twin" attacks [8], where a rogue access point may pose as a legitimate hotspot in order to steal username/password combinations or credit card information. Hype aside, it remains a fact that when arriving in a hotel in a strange city, the average user will have no way of determining whether a given access point accepting credit card information is a legitimate hotspot or a "phishing pond"[9].

Supporting roaming and secure use of public access points is an important part of the suggested security architecture, but just as important is the support of the new actor APO, which enables OBAN to handle more complex business structures. With the suggested security architecture, this can be done without sacrificing security requirements. We believe this may open up new business opportunities.

How OBAN addresses these issues will be described in the following.

1.4 Paper Outline

The rest of the paper is structured as follows:

- Section 2 sketches some mobility and QoS aspects of OBAN.
- Section 3 presents the primary security requirements for OBAN.
- Section 4 describes the relations between OBAN parties.
- Section 5 analyses threats that emerge specifically as a result of OBAN.
- Section 6 presents the security architecture for OBAN, focusing on session establishment and handover.
- Section 7 presents a discussion of our contribution.
- Section 8 concludes the paper.

2 Mobility and QoS

Mobility and quality of service (QoS) aspects of OBAN are described in [10] and [6], and while important, these will not be discussed in any detail here. However, for completeness we would like to direct the reader's attention to two specific features: Roaming to other networks, and a two-level Mobile IP [11] scheme.

2.1 Roaming

In a sparsely populated country such as Norway, any access network solution relying solely on wireless LAN access points will be unable to offer the required QoS and session mobility anywhere but select neighbourhoods – for the majority of locations, the service would degenerate to a "Hot Spot Service" as described in [7]. For this reason, the OBAN approach aims towards interoberability and roaming with other access networks, notably GSM/GPRS [12], UMTS [13] and WiMax [14] (the last, although not mentioned in [10], will be a natural extension as it becomes generally available). [10] describes how seamless mobility over heterogenous networks can be achieved; a handover to a different access network technology should not be noticeable for the OBAN user (except for reduced bandwidth when roaming

from e.g. WiFi [2] to GPRS). The preferred choice of access technology will be influenced both by available bandwidth and cost. As new wireless access technologies become available to the end-user, they will naturally find their place in the hierarchy of preferred OBAN access methods: WiFi, WiMax, UMTS, GPRS, etc.

2.2 Home Away From Home

OBAN specifies a two-level Mobile IP scheme, where an IPC is assigned a "home-away-from-home"[2] address by the ISP_{RGW} it is currently visiting (i.e. the ISP of the RGW it is currently connected to). The "home-away-from-home" address is naturally in the domain of the ISP_{RGW}, and will represent the end-point of a secure tunnel from the terminal to the ISP_{RGW}. This adds a measure of privacy for the IPC with respect to its ISP_{IPC}, since the latter will not be able to identify the specific locations the IPC has visited without the cooperation of ISP_{RGW}.

Also note that the use of Mobile IP paves the way for accountability and metering of transmitted traffic, since all traffic to the IPC is tunnelled from the IPC's home address to the home-away-from-home address in ISP_{RGW}'s network, and from there forwarded to the current care-of-address of the IPC. In contrast to traditional Mobile IP, the traffic from IPC to "the world" is tunnelled back to the Home Agent via the home-away-from-home address. This also satisfies traditional regulatory concerns regarding the origin of communications, since the traffic generated by the IPC is first tunnelled to ISP_{IPC} before being let loose on the global Internet.

3 Requirements

An OBAN implementation should fulfil some basic security requirements. The security requirements that have been considered most important in our work are listed below.

R1: It should be possible to uniquely identify each party.

R2: Each party should be able to verify the correctness of the information relevant for their activities, and should have enough information to prove their case.

R3: Each party should only get the information necessary to fulfil their particular tasks.

R4: Signalling data should be protected when it comes to confidentiality, integrity and non-repudiation.

R5: Roaming should be both secure and efficient.

R6: Personal equipment placed at the premises of the residential user should not be available for use by IPCs.

Note the conspicious absence of availability requirements – availability is considered an integral part of QoS, and is documented further in [6].

4 Relations Between Parties

For OBAN to be useful, all parties need to contribute towards the common goal of providing IP services to visitors. Consequently, the necessary trust relations and the different intentions of the parties are of high importance. Part of what makes OBAN special compared to other alternatives is the introduction of the party APO. When discussing relations between parties, the primary focus will be given to relations resulting from introducing this party. Note that in the protocol, the APO is not a communicating party per se, but will be represented by the RGW.

[2] In [10], the term "Gateway Foreign Agent" is used.

4.1 The Relation IPC – APO

The RGWs make it possible for IPCs to connect to their ISPs. In case an IPC acts illegally and/or creates technical problems the APO should be able to disconnect the terminal of this IPC. In case the IPC's behaviour results in financial losses for the APO, the APO should also be able to prove the course of events and the identities of the involved parties. On the other hand, the IPC wants privacy and anonymity. APOs should not have access to all communication of an IPC and should not know the true identity of the IPC.

4.2 The Relation APO – ISP

The APO will have a contract with an ISP that pays the APO for bridging services between terminals and this ISP. The conditions for payment may vary, but if the APO is to receive payment based on the amount of traffic that has been bridged by its RGW, they will both wish to be able to prove the amount of traffic that has been bridged.

It will also be in the APOs interest to get cost absorption confirmations from ISPs when delivering services to IPCs, since APOs normally do not have any contractual relationship with IPCs.

4.3 The Relation IPC – Residential User

Residential users have physical access to the RGW, and may also be the operator of the RGW. It should therefore be assured that residential users do not have the ability to influence the sessions of IPCs, for instance to earn more money.

4.4 The Relation APO – APO

The APOs may be paid based on the amount of traffic that is bridged between terminals and ISPs. To maximise revenue, the APOs want their RGWs to handle the traffic load as efficiently as possible. It should however be ensured that APOs cannot manipulate their RGWs in such a way the algorithm for distribution of terminals between RGWs becomes unfair, and other RGWs are excluded.

4.5 The Relation IPC – ISP

This is a traditional customer - supplier relationship. But in the case of OBAN, the IPC may also desire that the ISP is not able to determine the IPC's location while the latter is using the OBAN services.

4.6 The Relation ISP – ISP

This is a common relationship when intercommunication is required. In such cases ISPs may consume services from other ISPs and they may therefore both want to prove the amount of consumed services.

5 Threat Analysis

A fundamental premise of OBAN is that it should offer the same degree of security as seen in wired broadband connections to the Internet, and thus the threat analysis has only focused on threats specific to OBAN. This means that threats that relate to e.g. common Internet security have not been considered. The following aspects of OBAN seem to have a significant influence on the threat situation:

- Equipment is placed in the homes of individuals
- The structure of the network may be complicated, which will result in management challenges
- Wireless communication plays an important part

Threats that result from these factors will be discussed in the following subsections.

5.1 Equipment is Placed in the Homes of Individuals

The RGW will be placed in the homes of individuals, or on the premises of some enterprise. This means that no ISP is able to control the physical protection of the RGW, i.e. who has physical access to the RGW, how it is protected, etc. This results in increased probability for:

- Unauthorised theft or manipulation of the RGW
- Unauthorised access to data and operations on the RGW
- Unauthorised manipulation of the data or software of the RGW

Regarding residential users, one possible motivation for tampering with the device could be to increase the amount of traffic generated by IPCs, as seen from the ISP_{RGW}. Access to information on the communication of IPCs may also be one possible motivation for the residential users, as well as for intruders. Lack of physical control over the equipment may also result in reduced availability of service. Residential users may simply switch off the RGW, its power or its connection to the ISP_{RGW} at any time. To increase the level of protection one should consider using special protection of the most critical parts of the RGW, i.e. logs, keys, algorithms etc. Possible mechanisms to achieve this includes using tamper-proof equipment, using hardware instead of software for critical functions[3], removing interfaces that are not strictly needed, enforcing proper access control, encrypting content, and utilizing integrity checks.

Since equipment of the residential user will be connected to the RGW it is also important to protect the equipment already present, and make sure it can function as before. For instance, it should not be possible for IPCs to use a network printer of the residential user.

It may be in the interest of a rogue APO to let the RGW falsely assume the identity of an ISP. If successful, other RGWs will communicate with the RGW as if it was an ISP. This could result in the owner of the RGW getting hold of a lot of information that may be used to his advantage, for instance to earn more money at the expense of other RGW owners. Authentication and encryption are appropriate security measures also in this case.

APOs may also want to manipulate their RGW for other reasons. If APOs are paid based on the amount of traffic that is bridged between terminals and ISPs, they may wish to manipulate their RGWs in such a way that the algorithm for distribution of terminals between RGWs becomes unfair, and other RGWs are excluded. If the APO is the same person as the residential user, the APO may also have easy physical access to the RGW.

5.2 Complex Structure of the Network

Managing a network consisting of equipment placed in extremely diverse physical locations may be a major challenge for the ISPs involved. All RGWs will need updates of functionality, security features, etc. from time to time. These updates should be performed in a manner that is as automated as possible, since it is inconvenient that the users hosting the RGW should be responsible for this task. The ISPs will accordingly need to make sure the active RGWs are in working order and functioning as specified at all times. To ease this task it is important to have a good overview of the network. It should be clear who is responsible for managing the network, the network should be well documented, and network management plans should be in place.

5.3 Wireless Communication

Wireless communication is important in many other systems than OBAN, for instance within GSM and UMTS, and in regular wireless networks. As for all wireless networks, there may be problems with interference, uncontrolled resource consumption and jamming. But in addition there may be a problem with false access points. As an example a terminal of an IPC or a residential user may be used to fake an access point. Other terminals will then communicate with this terminal as if it were an access point, possibly resulting in the fake access point getting access to personal information, for instance on the OBAN subscription of the terminal owner. Among other things, this could make it possible to use OBAN services at the expense of the IPCs.

[3] This will, however, have adverse impact on the maintainability of the equipment.

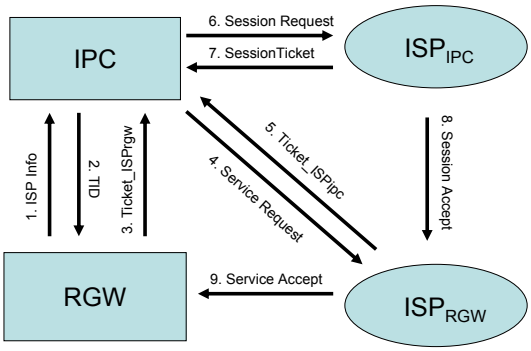


Figure 2: Session Establishment

To increase the protection from such forms of attacks, one should require authentication of access points. In addition one should encrypt any transmitted information that may be used to fake the identity of users.

5.4 Security of the Residential User Revisited

In order to convince residential users to participate in OBAN, security of the residential user's peripherals and other equipment is of paramount concern. However, the OBAN business model also depends on the residential user preserving strict access control to the "residential portion" of the wireless network. After all, who would want to run up charges on their OBAN account if there is an absolutely free residential network readily available from the same access point?

From this we may conclude that even in cases where the residential user also is the APO, steps must be taken to ensure that the RGW maintains a certain minimum of security, also with respect to the residential user. Among other things, this involves preventing the residential user from turning off encryption of the wireless traffic.

Residential users should be distinguished from IPCs, and only residential users should get access to their own local network. This can be achieved by using one Virtual Local Area Network (VLAN) for residential users and one VLAN for IPCs. As mentioned above, the VLAN of residential users need to be secured by the residential user. The use of such a VLAN solution implies that the residential user is not really an OBAN party, and is thus not considered further.

6 Security Architecture

The architecture is based on [15], but has been refined in order to concentrate on the authentication aspects.

6.1 Basic Mechanisms

The OBAN security architecture requires a Public Key Infrastrucure (PKI) where all parties are issued certificates from a universally trusted Certificate Authority[4].

Trust confirmations relayed via parties with which the recipient doesn't have a direct trust relation are transmitted in the form of *confirmation tickets*, inspired by the Kerberos authentication system [16]. However, since the use of shared symmetric keys would not be viable in an OBAN context, the tickets are instead created using a digital signature scheme. In similarity to Kerberos, we also assume the existence of "loosely syncronized" clocks.

[4]Or at least a CA universally trusted within OBAN. In theory, each $\mathrm{ISP}_{\mathrm{IPC}}$ could have operated a "unilateral" CA, relying on the direct security relationship between ISPs to generate cross signatures. We would maintain, however, that this is less maintainable and more complicated than having a single top-level CA.

1. $ISPinfo = IP_{ISP_{RGW}}, CERT_{ISP_{RGW}}, ID_{RGW}$
2. $TID_{RGW} = sign_{IPC}(E_{PK_{ISP_{IPC}}}(ID_{IPC}, subID_{RGW}), ID_{ISP_{IPC}})$
3. $Ticket_{ISP_{RGW}} = sign_{RGW}(TID_{RGW}, IP_{terminal}, ID_{RGW}, Timestamp_1)$
4. $ROT = E_{PK_{ISP_{RGW}}}(TID_{RGW}, IP_{terminal}, PK_{IPC}, Timestamp_2)$
   $ServiceRequest = sign_{IPC}(ROT, Ticket_{ISP_{RGW}}, Timestamp_3)$
5. $AST = sign_{ISP_{RGW}}(SessionKeyValidity, E_{PK_{IPC}}(SessionKey), Timestamp_4)$
   $SLD = ROT, Ticket_{ISP_{RGW}}$
   $AcceptReject = E_{PK_{ISP_{IPC}}}(AST, PK_{IPC}, SLD)$
   $Ticket_{ISP_{IPC}} = sign_{ISP_{RGW}}(MIP_{ah}, TicketValidity, AcceptReject, TID_{RGW}, Timestamp_5)$
6. $SessionRequest = sign_{IPC}(Ticket_{ISP_{IPC}}, CERT_{ISP_{RGW}})$
7. $SSC = sign_{ISP_{IPC}}(ID_{session}, Timestamp_6)$
   $SessionTicket = sign_{ISP_{IPC}}(ID_{session}, E_{PK_{IPC}}(SSC), AST, Timestamp_7)$
8. $SessionAccept = sign_{ISP_{IPC}}(AST, SessionValidity, E_{PK_{ISP_{RGW}}}(SSC), Timestamp_8)$
9. $ServiceAccept = sign_{ISP_{RGW}}(AST, SessionValidity, Ticket_{ISP_{RGW}}, E_{PK_{RGW}}(SSC), Timestamp_9)$

Figure 3: Detailed Session Initiation Messages

Certain characteristics of the RGW should be unalterable by the residential user, or any other unauthorized party. To achieve this, some sort of tamper-proof equipment should be considered for relevant parts of the RGW.

Since the RGW also bridges all traffic from the residential network, the RGW should authenticate itself to ISP_{RGW} before any traffic is accepted by ISP_{RGW}. However, this is completely analogous to the situation with current broadband routers, and is thus considered out of scope for our protocol.

Since all application traffic is tunnelled first to the home-away-from-home agent in ISP_{RGW}'s network, the protocol will negotiate a session key to encrypt the traffic in this tunnel. No special provisions are made for encrypting the application traffic between ISP_{RGW} and ISP_{IPC}; this is left to the application.

In the following, encryption with the public key of party "X" will be denoted $E_{PK_X}(\ldots)$, while creating a digital signature with the private key of "X" is denoted $sign_X(\ldots)$ (for brevity, this notation shall be interpreted to mean that both the signature and the signed data is transmitted).

6.2 Session Initiation

The session initiation process is illustrated in Figure 2, and the session initiation messages are written out in detail in Figure 3 (see Table 1 for a description of the protocol elements). As mentioned, it is assumed that the RGW has authenticated itself to ISP_{RGW} in a conventional manner before the session initiation commences, but this is not considered part of the protocol.

Note that timestamps are used to ensure freshness of messages, in addition to control expiration of tickets. Each timestamp in the protocol is thus unique and created at the time of message compilation; this is indicated by an enumeration suffix. The numbering of timestamps has no other significance.

1. When wireless connectivity to the RGW has been established, the RGW will send IPC an *ISP Information Token*, containing the IP address and public key certificate of ISP_{RGW}. For convenience, the RGW also includes its own ID in the token. The token itself is not signed, since it will be followed later by a ticket.

2. IPC replies to the RGW with a signed token containing the ID of the IPC and the ID of ISP_{IPC}. The information identifying the IPC is combined with a descriptor chosen by the IPC (so that the IPC has a unique descriptor for each RGW it has been in contact with), and encrypted with the public key of ISP_{IPC}. This prevents the RGW from learning the true identity of the IPC, and also from tracking the IPC when it roams to other RGWs, but enables the RGW to recognise the IPC as a previous visitor if the IPC should return at a later time. We refer to this identifier token sent by IPC as TID_{RGW}, since it in effect is a temporary ID for this IPC while connected to this RGW.

   Note that since the identity of the IPC is encrypted with the public key of ISP_{IPC}, no other actors have access to this information.

3. The RGW responds with a ticket, $Ticket_{ISP_{RGW}}$, which enables the IPC to contact ISP_{RGW}. This ticket is signed by RGW, but otherwise sent in clear text, since it basically only is a confirmation that the RGW has capacity to spare and is accepting connections.

4. The IPC sends a service request containg a Request Origin Token (ROT) and $Ticket_{ISP_{RGW}}$ to ISP_{RGW}. The ROT contains information about the IPC (e.g. TID, public key and current temporary IP address), and is encrypted with the public key of ISP_{RGW} in order to prevent the RGW from tracking the IPC. ISP_{RGW} can extract the ID of ISP_{IPC} from the TID in the ticket, and IPC's public key from ROT. ISP_{RGW} will use IPC's public key to encrypt the session key which later will protect the tunnel between IPC and ISP_{RGW}.

5. ISP_{RGW} transmits to the IPC a signed $Ticket_{ISP_{IPC}}$ containing among other things the home-away-from-home address of IPC (MIP_{ah}). Buried in this ticket is also an Access Service Ticket (AST), which is encrypted with the public key of ISP_{IPC}. The AST will be returned to IPC once the session establishment is approved by ISP_{IPC} (see below). Since the sensitive information in this ticket already is encrypted, the ticket itself is unencrypted.

6. The IPC sends ISP_{IPC} a signed Session Request containing $Ticket_{ISP_{RGW}}$ and the certificate of ISP_{RGW}.

7. ISP_{IPC} replies with a Session Ticket, containing a session ID and a timestamp. It also contains an encrypted and signed Service Session Close (SSC) ticket (which the IPC is to use later when closing the connection), and the Access Service Ticket (AST) which contains the session key between IPC and ISP_{RGW}. The Session Ticket itself is not encrypted, since further communication is dependent on knowledge of the session key, not the session ticket.

8. ISP_{IPC} also creates a Session Accept message by signing a combination of the AST, SSC and a timestamp. The Session Accept message is then sent to ISP_{RGW}.

9. Upon receiving the Session Accept message, ISP_{RGW} transmits a Service Accept message to the RGW, at which point the RGW allows the IPC to communicate freely toward ISP_{IPC}[5] according to its Access Service Ticket and Session Ticket.

6.3 Failure Scenarios

Note that in all steps in the above described protocol that involve some kind of verification, a failure will result in a "deny" (*NAK*) message that will abort the session establishment.

In the following we describe some examples of possible failure in session establishment. Most of these failures represent a breach of the security policy.

6.3.1 Capacity of RGW Exceeded

The session establishment proceeds normally until RGW receives the TID (step 2). Upon receiving the TID, the RGW determines that its capacity has been exceeded, and that it can no longer offer meaningful service to new customers. Instead of replying with a ticket, it thus transmits a NAK message, and terminates the connection to IPC.

In case the APO for some reason has decided that it will not do business with a certain ISP_{IPC}, it will exhibit similar behaviour when it receives a TID which names the ISP_{IPC} in question.

6.3.2 Unrecognized Customer

The session establishment proceeds normally until ISP_{IPC} receives the Session Request (step 6), and extracts the public key of IPC. This is compared with the public key ISP_{IPC} has on file for IPC, and in case of a mismatch, ISP_{IPC} transmits a NAK and closes the connection. If, on the other hand, the public key matches, the signature of the TID_{RGW} is checked. If the signature is invalid, ISP_{IPC} likewise sends a NAK and closes the connection. In both cases, it also transmits a Service Level Deny (SLD) message extracted from the ticket to ISP_{RGW}.

[5] But remember that the traffic is first tunneled to the home-away-from-home address before being forwarded to ISP_{IPC}.

Table 1: Explanation of Protocol Elements

Term	Description
$ISPinfo$	ISP information token;
TID_{RGW}	Temporary ID for an IPC at a given RGW, created by IPC
$subID_{RGW}$	"Personal" ID for an RGW, chosen by IPC
$Ticket_{ISP_{RGW}}$	Ticket allowing the ISP to communicate with ISP_{RGW}
ROT	Request Origin Ticket
AST	Access Service Ticket
SessionKeyValidity	Specification of how long a session key is valid. This is configurable by ISP_{RGW}, and may be less than SessionValidity. If a session key expires before the session itself, it will have to be re-negotiated. This will result in the generation of a new AST.
SLD	Service Level Deny
AcceptReject	Structure used by ISP_{IPC} when accepting or rejecting an attempted session initiation. In the latter case it will extract the SLD and transmit to ISP_{RGW}, otherwise the AST and public key of IPC is used to create a session ticket.
MIP_{ah}	"Home away from home" Mobile IP address. Upon completion of the protocol, traffic from the terminal will be tunneled securely to this address, and then forwarded to the Home Agent.
TicketValidity	Specification of how long a ticket is valid.
SSC	Service Session Close ticket; used by IPC to terminate session
SessionValidity	Specification of how long a session is valid. This is configurable by ISP_{IPC}.

6.3.3 Fake Access Point

This is the situation if someone should introduce a fake access point (RGW) with connection to a genuine ISP_{RGW}.

The session establishment proceeds normally until ISP_{RGW} receives the Service Request (step 4). ISP_{RGW} will examine $Ticket_{ISP_{RGW}}$, and determine that it has not been signed by an RGW with which it has a contract. ISP_{RGW} will then send a NAK and close the connection.

6.3.4 Fake Access Point and ISP

A fake access point (i.e. fake RGW), having no relationship with a real ISP_{RGW}, may try to also act as a fake ISP_{RGW}. The session establishment proceeds normally until ISP_{IPC} receives the Session Request (step 6). It will first verify the correctness of ISP_{RGW}'s certificate (also checking that it belongs to an ISP_{RGW} with which it has a contractual agreement), and then the signature of $Ticket_{ISP_{IPC}}$. If either fails, it will send a NAK and close the connection (there is no point in sending an SLD here, since the Session Request did not come via a legitimate ISP_{RGW}).

6.3.5 Expired Ticket

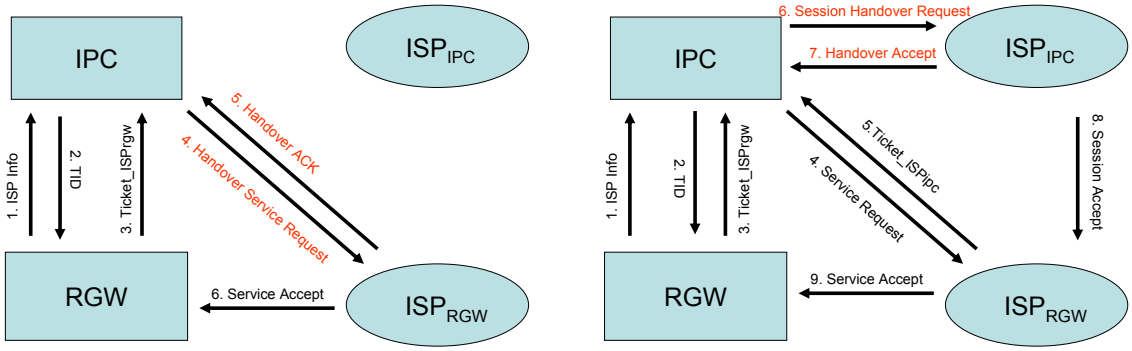
All tickets have a predefined expiration time, after which the recipient will reply to the ticket with a NAK, and close the connection. All other messages that are determined to be too old are treated similarly.

6.4 Handover

The handover process between RGWs will exhibit different characteristics depending on whether or not the old and the new RGW use the same ISP_{RGW}.

We first describe what happens when an IPC arrives at a new RGW connected to same ISP_{RGW} as the previous RGW (see Figure 4(a) and 5):

1. As with regular session establishment, the new RGW transmits the ISP Info upon completing the basic connectivity steps.



(a) Handover Within Same ISP_{RGW}

(b) Handover Involving Different ISP_{RGW}s

Figure 4: Handover

1. $ISPinfo$
2. TID_{RGW}
3. $Ticket_{ISP_{RGW}} = sign_{RGW}(TID_{RGW}, IP_{terminal}, Timestamp_{10})$
4. $HandoverServiceRequest = sign_{IPC}(ISPinfo, AST_{orig}, Ticket_{ISP_{RGW}}, Timestamp_{11})$
5. $HandoverACK = sign_{IPC_{RGW}}(AST_{orig}, Timestamp_{12})$
6. $ServiceAccept = sign_{ISP_{RGW}}(AST_{orig}, SessionValidity, Ticket_{ISP_{RGW}}, E_{PK_{RGW}}(SSC), Timestamp_{13})$

Figure 5: Detailed Handover Messages

6. $SessionHandoverRequest = sign_{IPC}(Ticket_{ISP_{IPC}}, SessionTicket_{orig}, Timestamp_{14})$
7. $HandoverAccept = sign_{ISP_{IPC}}(AST, Timestamp_{15})$

Figure 6: New Messages (with respect to Figure 3) for Handover, Different ISP_{RGW}

2. Since the new RGW doesn't really need to know that this is a handover, the IPC replies as usual with TID_{RGW}.

3. The new RGW replies with a customary ticket for access to ISP_{RGW}.

4. Since the IPC is aware that it already has a connection with a previous RGW, and can see from the ISP Information Token that the new RGW belongs to the same ISP_{RGW} as before, it then sends a Handover Service Request to ISP_{RGW}, containing the ISP token, the original AST and the new $Ticket_{ISP_{RGW}}$.

5. ISP_{RGW} replies to IPC with a Handover Acknowledge.

6. ISP_{RGW} sends a Service Accept to the new RGW.

If the IPC arrives at a new RGW that is not connected to the same ISP_{RGW} as the previous RGW, the handover process becomes more complicated, and will basically require the same amount of messages as a session initiation. The only difference is that the IPC does not need to set up a new session with its ISP_{IPC}. This can be seen in Figure 4(b). The new messages required for this type of handover are listed in Figure 6. Note that the element named $SessionTicket_{orig}$ in step 6 of Figure 6 is the original session ticket the IPC received when it first initiated the session.

6.5 Faster Handovers

Efficiency is important when it comes to handover, and the solution proposed here may be too time-consuming in many cases. Fortunately, there is room for improvement. One alternative is to do authentication in advance, prior to the actual handover. This might require the terminal to authenticate to all access points as soon as they are within range. Another alternative is utilizing delayed authentication; i.e. accepting unauthenticated connections, but tearing them down if they are not authenticated/confirmed by a set deadline. The effective time for unauthenticated communication will be limited, resulting in only a minimal risk of loss of income for the APO.

Due to restrictions in wireless network standards, terminals are not allowed to be connected to two different access points at the same time. Authentication in advance can therefore not be dependent on terminals requesting authentication at RGWs using the wireless network. However, other (as yet unspecified) mechanisms may be used for this purpose. Alternatively, RGWs may be responsible for pre-authentication. Each RGW could keep a list of neighbour RGWs which are to be contacted for pre-authentication, for instance by using the wired network. These options, and their security implications, are to be investigated in future work.

6.6 Session termination

Any party can terminate a session by sending a service session close ticket.

7 Discussion

In the following, we summarize our achievements and discuss our results in reference to other possible solutions.

7.1 Achievements

Based on the messages involved in session initiation, handover and session termination, trust between the involved parties is achieved. In general, actors who have a contractual relationship before any communication takes place will by definition trust each other. Trust establishment between these actors will therefore not be necessary, but they will need to authenticate each other.

7.1.1 IPC – RGW

The IPC does not need to trust the RGW. The only identity information provided to the RGW is a temporary ID and the ID of the ISP of the IPC. Without a proper agreement with an ISP, the RGW will not be able to handle the request and get paid. It may (in theory) function as a "man in the middle", but this will not result in any advantage. It cannot get to the confidential information that is transferred because it is encrypted, and it cannot communicate at the expense of the IPC since it does not have the necessary session key.

The RGW has the temporary ID of the IPC, and the APO will thus be able to prove the traffic sent by this IPC. In theory, if the IPC behaves badly, the RGW would later be able to recognise the IPC and deny access to communication. However, since the IPC chooses the temporary ID used for the RGW ($subID_{RGW}$), a rogue IPC would likely choose a new $subID_{RGW}$ for the next visit, and continue misbehaving with impunity. In order to effectively block misbehaving IPCs without sacrificing anonymity, some mechanism must be introduced that controls how $subID_{RGW}$ is selected. This remains an area for further study.

If the behaviour of the IPC justifies this, the APO will initiate action towards its $\mathrm{ISP_{RGW}}$, which in turn will use its contractual relationship with the relevant $\mathrm{ISP_{IPC}}$ in order to determine the real identity of the IPC. The specific procedures related to such "abuse-cases" will be subject to rules from relevant regulatory bodies.

The APO will have access to the home-away-from-home address of the IPC, MIP_{ah}, and will thus be able to track the IPC if it roams to a nearby RGW belonging to the same $\mathrm{ISP_{RGW}}$. MIP_{ah} is a dynamic address, however, and will not be reused in future sessions.

7.1.2 IPC – $\mathbf{ISP_{RGW}}$

The IPC knows which ISP_{RGW} is bridging its traffic, and knows that ISP_{IPC} accepts this ISP. The IPC is also able to prove the amount of traffic that has been sent via this ISP_{RGW}, and has confirmation that the ISP has approved the communication. The ISP_{RGW} will not know the true identity of the IPC, but will know the public key of the IPC. This means that it will be able to recognise the customer.

ISP_{RGW} has proof that the IPC is a customer of ISP_{IPC} and that ISP_{IPC} will honour any obligations made by this customer. ISP_{RGW} will also be able to prove the amount of traffic that has been sent by the customer.

7.1.3 IPC – $\mathbf{ISP_{IPC}}$

These actors will have a contractual relationship. Authentication is performed using digital signatures in accordance with the chosen PKI.

Both actors will be able to prove the amount of traffic that has been sent, but ISP_{IPC} will not know the real location of its customers.

7.1.4 RGW – $\mathbf{ISP_{RGW}}$

These actors will have a contractual relationship. Both actors will be able to prove the amount of traffic that has been bridged, and the RGW has confirmations that the IPCs that are served are accepted by ISP_{RGW}.

7.1.5 RGW – $\mathbf{ISP_{IPC}}$

These actors have no special relationship.

7.1.6 $\mathbf{ISP_{RGW}}$ – $\mathbf{ISP_{IPC}}$

These actors will have a contractual relationship. They will be able to authenticate each other based on the knowledge of private and public keys. Both actors will be able to prove what services have been offered.

7.2 Fulfilment of Requirements and Mitigation of Threats

The architecture suggested seems well suited to the task of securing all OBAN partners. The requirements stated above are fulfilled:

- Each party has an identity descriptor that can be used to uniquely identify the party. (R1)
- Each party gets the information they need to be able to perform their task and prove their case in the event of a dispute. This is described in section 7.1. Tickets and acceptance messages are signed by the issuing party to achieve non-repudiation. (R2)
- The different parties only get access to the information that is necessary to fulfill their tasks. As an example, ISP_{IPC}s do not get the exact location of their customers, they are only able to know which ISP_{RGW} the customers are connected to. Similarly, RGWs and ISP_{RGW}s do not know the true identities of IPCs. (R3)
- Signalling data is protected with signatures and encryption where needed, to achieve confidentiality, integrity and non-repudiation. (R4)
- Handover is supported, and is made efficient when roaming between RGWs connected to the same ISP_{RGW}. Handover may be cumbersome if roaming between RGWs connected to different ISP_{RGW}s, so further effort should be spent exploring the options for fast handover described in section 6.5. (R5)
- Personal equipment of residential user is protected. IPCs are only allowed to communicate via ISP_{RGW}, and are not allowed access to the local network. This local network must also be protected by other means by the residential user. (R6)

Regarding threats, protecting the core functionality of RGWs reduces the risk inherent in placing the RGW in premises that are not controlled by an ISP. At the same time, the degree of required trust in RGWs is reduced since no valuable data is sent through the RGW unencrypted. But the problem with complex configurations, and thereby complex maintenance has not been addressed. ISPs may choose to put this responsibility on residential users, if they have the role of an APO. However, residential users may not have the necessary skills to perform this task, resulting in lower quality of the service offered. Some sort of ISP involvement would therefore be beneficial.

7.3 Comparison to Common Security Mechanisms

We realize that an OBAN network has sufficiently many points of similarity with existing computer and telecommunications networks that one could have considered employing commonly available technologies for Authentication, Authorization and Accounting (AAA), as exemplified by [17] and [18]. Unfortunately, space does not permit a rigorous comparative analysis between the OBAN security architecture and commonly available alternatives, but in order to highlight some of the advantages of our architecture, we present a brief description of one concrete alternative below.

An alternative to our architecture could be to utilize 802.1X [19] and EAP-TTLS [20] for authentication. Using this solution, authentication of IPCs can still be performed. IPCs wishing access to OBAN services would send a request to the RGW. This request for service will be tunnelled to a TTLS server at the ISP_{RGW} which forwards the request to the ISP_{IPC}, which ultimately makes the decision. In many ways this is a viable solution. The problem with fake access points ("evil twins") can still be handled, the IPC will only get access to OBAN services if a relationship with a proper ISP_{IPC} is in place, and RGWs and ISP_{RGW}s will know that ISP_{IPC} will honour any obligations made by this customer. This solution has, however, some weaknesses compared to the security architecture suggested in this paper, particularly when it comes to proof of events. This is related to the introduction of the new party APO.

The advantages of the security architecture suggested for OBAN are as follows (these advantages relate to the description above, but are also relevant with respect to e.g. Diameter [18]) :

- ISP_{RGW} will not know the true identity of the IPCs that are served.
- The identities of IPCs are available in a form that can be used for identification in case of dispute.
- IPCs are able to know which RGW that has been used.
- RGWs are able to prove which IPCs have been served and how much resources have been consumed by each IPC. This can be done without relying on logs of other parties.
- All the parties will have access to all tickets and acceptances that will be relevant in case of dispute. As an example, the RGW receives an acceptance message that confirms that ISP_{RGW} has accepted the customer's request for service. In the same way the ISP_{RGW} receives an acceptance message that confirms that ISP_{IPC} has accepted the request and serves the IPC involved. These messages have been signed by the relevant party and can be saved for later use.
- The signed tickets and confirmations provide non-repudiation.
- Data is encrypted all the way to the ISP_{RGW}, not only to the access point.
- There is no need for a shared secret between ISPs and RGWs.

7.4 Business Opportunities

As a result of the work with OBAN, one now has a security architecture that is able to support a completely new party within communication services, namely the APO. This opens up new business opportunities. There is no longer a need to connect directly to some ISP to be able to roam. One can still use well known charging mechanisms, and this is possible without lowering security requirements. For ISPs this means possibilities for faster provision of higher capacity networks with a reduced need for heavy investments. Resources controlled by other parties can be utilized; this may also reduce maintenance costs. Some companies may specialize on administering end equipment like RGWs, while others may specialize in providing connectivity at larger distances. This may yield more effective communication provision, resulting in lower cost for all.

8 Conclusion

This paper has presented a security architecture for an Open Broadband Access Network. The idea of Open Broadband Access Networks is in itself appealing, since already available capacity can be used more efficiently. The architecture suggested is able to fulfil the security requirements in an OBAN environment. In addition, the architecture makes it possible to introduce a new party between the customer and the ISPs, without sacrificing security. This may again open up new business opportunities.

Acknowledgment

This paper is based on joint research in the EU 6^{th} framework programme. The authors would like to thank all the participating OBAN partners for their contribution to the discussions leading up to the security architecture.

References

[1] E. Edvardsen, T. G. Eskedal, and A. Årnes, "Open access networks." in *INTERWORKING*, ser. IFIP Conference Proceedings, C. McDonald, Ed., vol. 247. Kluwer, 2002, pp. 91–107.

[2] *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11-1999, 2003.

[3] OBAN Consortium. [Online]. Available: http://www.ist-oban.org

[4] E. Edvardsen. (2004) Fixed and Mobile Convergence. BroadBand Europe 2004. [Online]. Available: https://medicongress.be/UploadBroad/Session%2009/Paper%2009-01.pdf

[5] T.-G. Eskedal, R. Venturin, I. Grgic, R. Andreassen, J. C. Francis, and C. Fischer, "Open Access Network Concept, a B3G Case Study," in *Proceedings of 13th IST Mobile & Wireless Communication Summit*, 2003.

[6] G. Hoekstra, O. Østerbø, R. Schwendener, J. Schneider, F. Panken, and J. van Bemmel, "Quality of Service Solution for Open Wireless Access Networks," in *Proceedings of 14th IST Mobile & Wireless Communications Summit*, 2005.

[7] LinSpot. [Online]. Available: http://www.linspot.com

[8] G. Fleischmann. (2005) "My Evil Twin". [Online]. Available: http://wifinetnews.com/archives/004718.html

[9] G. Ollmann. (2004, September) "The Phishing Guide – Understanding & Preventing Phishing Attacks". NGS Software. [Online]. Available: http://www.ngssoftware.com/papers/NISR-WP-Phishing.pdf

[10] F. Steuer, M. Elkotob, S. Albayrak, H. Bryhni, and T. Lunde, "Seamless Mobility over Broadband Wireless Networks," in *Proceedings of 14th IST Mobile & Wireless Communications Summit*, 2005.

[11] C. E. Perkins, "Mobile IP," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 66–82, 2002.

[12] J. Cai and D. J. Goodman, "General packet radio service in GSM," *IEEE Communications Magazine*, vol. 35, no. 10, pp. 122–131, 1997.

[13] J. F. Huber, D. Weiler, and H. Brand, "UMTS, the mobile multimedia vision for IMT 2000: a focus on standardization," *IEEE Communications Magazine*, vol. 38, no. 9, pp. 129–136, 2000.

[14] *IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Std. 802.16-2004, 2004.

[15] T. J. Wilke and T. H. Johannessen, "Multilateral Security for IP-Service Provisioning in Open Broadband Access Networks," in *BWAN 2005, International Workshop on Broadband Wireless Access Network on fixed network*. IEEE COMSOC, June 2005.

[16] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[17] C. T. de Laat, G. M. Gross, L. Gommans, J. R. Volbrecht, and D. W. Spence, "Generic AAA Architecture," RFC 2903, August 2000.

[18] P. R. Calhoun, J. Loughney, J. Arkko, E. Guttman, and G. Zorn, "Diameter Base Protocol," RFC 3588, September 2003.

[19] *Port-Based Network Access Control*, IEEE Std. 802.1X-2001, 2001.

[20] P. Funk and S. Blake-Wilson, "EAP Tunneled TLS Authentication Protocol Version 1 (EAP-TTLSv1)," Internet-Draft (Work in progress – expired), February 2005.

A Security Analysis on JADE(-S) V. 3.2

Regine Endsuleit and Jacques Calmet
Universität Karlsruhe (TH)
IAKS
Germany

Abstract

In this paper we present a security analysis on the multi-agent platform JADE in its version 3.2 as well as on its security plug-in JADE-S. Besides a classification of possible and well-known attacks on the system we also provide a discussion on what is still missing in JADE-S and which parts of the implementation appear to be at least obscure if not insecure. We also present some Denial-of-Service attacks which we have implemented and successfully tested.

Key Words: JADE, Multi-Agent System, Security Analysis, Denial-of-Service Attacks

1 Introduction

JADE [6] is the better known and most often used platform to implement multi-agent systems. The question whether JADE is a secured platform is often asked and routinely answered as "it is not yet secure but works to this aim, such as JADE-S, are under way". Moreover, as in other systems there is still the opinion that security is achieved on another system level and that it is not the responsibility of the platform to defend against attacks. We do not agree with this opinion since only attacks from outside the system are considered in this case.

The project [3] leading to this paper did aim at assessing the JADE security features and possibly to suggest a new architecture for a new platform, better suited to security needs. The assessment part is already so involved that the latter part of the program has been postponed.

The word "security" when standing alone has a few meanings, indeed security is always with respect to well-identified threats. In most cases there is a trend to identify security with authorization and authentication. These concepts are bounded, with additional features, into firewalls. Then, cryptography delivers very efficient methods to secure messages or bases of data or knowledge. All these aspects are obviously part of agent security mechanisms but agents are sometimes mobile and this leads to specific threats. There have already been publications suggesting to define several classes of security for IT methodologies. In the context of agent technology we consider three main classes.

1. External security: This is the usual concept of security for any software system. It covers the keywords mentioned above such as firewall, authentication, authorization and encryption. In a classical architecture approach to system design, this level is shown as a security layer of the architecture. Its goal is the protection against attacks from outside the system.

2. Internal security: This refers to the integrity of the components of the software once its external security has been enforced but broken. Some facets are for instance intruder detection and/or neutralization. In a system architecture representation of agent systems, this level would translate into software modules and methodology but not into a layer of the architecture.

3. Integrity and Privacy: This class contains methods to protect the system's state's integrity. It is a kind of second layer on external security which should allow the system to degrade gracefully once the external security has been broken or malicious parties are part of the system. The necessary methods are pretty efficient given the state-of-the-art of the algorithms used in cryptography (AES,

hash functions, digital signatures). This level does not show up in any system architecture except possibly as a cryptography module.

Since we want to assess whether JADE is "secure" in the sense of those three classes, we specify security to threats specific to agents. They are listed in the body of the paper. We purposely omit the level of protocol security although work on this topic is going on in our group. At present, available methods allow checking, for instance, the integrity of circuits but are not generic enough to be included into a general classification. A motivation for this investigation is that we have designed new methods to enforce security concepts for mobile agents [4, 5] and are working on concepts of virtual knowledge societies and corporate knowledge [8] that are implemented mostly on JADE. It is required to assess how secure is the platform used to implement security features or company management concepts. It is surprising that no such assessment has been published within the last years, or at least we could not find any. The attacks we will present refer to JADE version 3.2. While writing this paper a new version has been published, but we expect the main security riks to be still existing. As the reader will see later-on, the elimination of those risks requires fundamental changes of the system' design.

The paper is structured as follows. Section 2 presents the facets of JADE that are relevant when analyzing possible attacks. Moreover, we give a classification of attacks which are possible when using JADE without JADE-S. Then, in section 3 security in JADE-S is outlined, at least what is known as of today. We continue in this section with an overview on the threats classified in section 2 that are still possible when JADE-S is installed. The following section 4 presents two attacks on JADE which we have successfully implemented. Finally, we discuss the main results of our analysis and an give an evaluation of JADE-S.

2 A Classification of Attacks on a JADE platform

Although having been announced, JADE 3.2 does not provide a security plug-in for *mobile* agents. Indeed, the implemented plug-in JADE-S supports authentication of users and agents, rights management as well as encryption and signature of messages, but exclusively in a static scenario. If one wishes to use *mobile* agents, there is no security available. However, it is worth it to make an analysis of the possible threats in JADE since mobility is a desirable and favored feature in a multi-agents system and one can count on numerous platform operators taking the risks of using JADE without the security plug-in. Therefore, this section is devoted to a classification of possible threats arising in an unprotected system.

We do not consider dependencies on Java Virtual Machines (JVM) because there is no direct connection to JADE. But one should keep in mind that running different programs in one JVM may cause security problems as those programs share their memory. Instead, we will concentrate on Java Reflections, a powerful tool used by JADE to load and execute agents. However, it enables malicious parties to spy on other agents and to manipulate them.

Before presenting possible attacks we briefly review the possible consequences of being able to access classes and interfaces of the currently running JVM [1]:

- One can determine the class of an object.
- It is easy to gain information about fields, methods, constructors and modifiers of a class and its super class.
- One can find out which constants and method declaration belong to the same interface.
- One can generate instances of a class whose names become public only at runtime.
- One can look onto and change the content of a class' field although the field's name is only known at runtime.
- It is possible to call methods of an object, whichs name is unknown at compile time.
- Arrays with unknown size at compile time can be manipulated.

JADE uses the Reflection API to load and execute agents, filters and other dynamic objects at runtime. Without any security all mechanisms mentioned above can be applied to private classes, fields and

methods. An attacker needs a reference on the specific instance only. He can get a proxy instance for the current container and read the private field `myImpl` which contains a valid reference on the attacked agent's container by calling `this.getContainerController()`. Now, the attacker may access and manipulate all data. Likewise, the call of private methods is possible. Most of the following attacks make use of this technique.

Agent vs. Platform

- *Denial of Service*: One component that is vulnerable to DoS attacks is the JADE message system. The addresses of the latest message receivers are buffered locally in the sending container. On a large platform an agent could produce a cache overflow by permanently sending messages to numerous (different) agents. In this case addresses of the receiver must be asked for at the agent management system (AMS) of the platform. This slows down the message transmission of the attacked container and also results in a distributed DoS (DDos) attack on the AMS if several attackers in different containers attack their container. In [9] it has been shown that even for undisturbed communication between different containers communication complexity is growing fast with the number of agents. Later-on in this paper we will show that a DoS attack causes significant delays in message transportation.

- *Trojan Horse*: Usually, an agent is executed with the operating system priviledges of the user that started the system. In case those rights have been set up professionally, JADE settings only can be changed and program file (`.jar` archives) can be compromised. But in case the platform is started by a user with administrator privileges (which is not unusual on a Windows PC e.g.) an arbitrary agent could use those privileges to corrupt the whole system.

- *Weed*: Weeds are agents with no meaningful functionality which could be used for a DDoS attack just by existing in large numbers and thus cause a high system load. Detecting such agents is very difficult in JADE. There is a graphical user interface available that primarily aims on searching for errors during programming. Besides the problem to decide whether an agent has a meaningful functionality, monitoring a large number of agents causes an information overload that cannot be managed by the administrator. Thus, it is not helpful in getting overview of the executing agents properties.

- *Flying Dutchman*: An agent that cannot be terminated is called a flying Dutchman. Without the security plug-in JADE-S there are no restrictions for agent mobility or cloning of agents. In JADE the state of an agent is manipulated by methods of the basic class `Agent`. For instance the method `Agent.doDelete()` contains instructions to change the agent's state and to end the agent's thread. As many other methods and fields of this class this one is not declared `final`. This implies that the agent is able to overwrite this method and neither to change his state nor to terminate itself. This attack has been successfully implemented as we will present later-on.

Agent vs. Agent This class of attacks also contains the possibility of indirect attacks which means that a malicious agent first manipulates the platform, and then attacks other agents.

- *Denial of Service*: There are two kinds of attacks

  1. Direct attacks: In this case an agent is flooded with messages and queries which bars it from following its original task. Another possibility is to modify e.g. behaviours or internal data of an agent. In case of terminating the functionality of the agent this counts as DoS attack, too.
  2. Indirect attacks: This includes to manipulate the platform in order to disturb an agent. For example it is possible to suspend or to terminate the agent.

- *Spoofing/Man-in-the-middle*: This class of attacks is difficult for parties which are outside of the platform since it is necessary to intercept message packages on the level of TCP/IP. Within the platform messages can be intercepted and manipulated by the container. This attack has been implemented and the results will be shown in section 4.

- *Takeover*: By means of Java Reflections one can add arbitrary new behaviours to an agent. Those stay part of the agent even if it migrates to another container. This implies that the agent is still under foreign control after having left the host on which it has been infected. Adding a new behaviour only requires an object reference on the attacked agent as provided by the public method `agent.addBehaviour()`. The basic class `Agent` does not check any additional requirements. A simple solution to this problem could be the restriction to allow new behaviours only within the same thread since each JADE agent is represented by one specific thread.

Platform vs. Agent

- *Code/Data Alteration*: A meaningful manipulation of an agent's code or data requires an analysis of the agent's functionality. Since JADE is based on Java this is not really problematic. Java bytecode can be nearly lossless recompiled into its source code. If one just wishes to create another functionality this is even simpler.
- *Code/Data Peeping*: These attacks are a subset of the code and data alteration. They work passively by spying on code and data and do not change anything.
- *Replay Attacks*: Messages can be delayed, deleted, stored, changed and sent repeatedly. This attack, too, will be presented in detail in section 4.
- *Malicious Routing*: In case an agent wishes to migrate he asks the responsible platform service for a transfer to the designated new host. The service may respect this wish or ignore it. When arriving somewhere the agent can ask for the identity of the current host but it must trust the integrity of the local implementation. It is very easy to mislead an agent by substituting its wished destination. Moreover, an agent using the mobility service deallocates and reinitializes components like GUI or other local resources automatically. This enables to initiate an "external" migration without the agent's knowledge. This can only be prevented by overwriting the responsible methods in his super class `Agent` and thus behaving non-compliant.
- *Misinformation/Denial of Service*: This class of attacks contains a variety of possibilities to disturb an agent in completing his task. The agent management system (AMS) could be used to suspend or terminate agents. Also disturbing an agent's communication is part of this class. Wrong information could be provided to the agent through manipulated messages or direct manipulation of its database.

3 The Security of JADE with JADE-S

As mentioned above, JADE version 3.2 provides a security add-on, JADE-S. When announced JADE-S was supposed to support mobility. In fact, it does not. Only message encryption, signing and the introduction of a security policy for users and their agents have been included. A security manager aims at preventing agents from manipulating each other or the platform. This goal is not fully reached because both JADE and JADE-S do not use signed `.jar` archives and lack an integrity check of their code. This introduces the danger of ordinary attacks compromising the JADE installation via viruses, worms or with other means. Most of the security mechanisms JADE-S offers fail to take into account internal attackers with access to the platform who run "their" host maliciously. The only thing preventing more damage in this scenario is the lack of mobility support in combination with JADE-S. But, since JADE-S is not usable together with mobility there is still no security in the mobile scenario at all. As discussed before, the latter will be the most popular since mobility is a very powerful feature. As a consequence most of the started platforms will still remain completely unsecured.

In the remainder of this section we will first discuss some existing risks that could be used by an attacker to acquire rights within the platform which are necessary for a successful attack. Then, we continue with a listing of attacks similar to that of section 2 that are still possible when using JADE-S.

3.1 Weaknesses of JADE(-S)

1. JADE(-S) archives are not signed. This implies that malicious hosts can join the platform with a manipulated installation without being detected.

2. There is a JADE login module *plaintext* which allows to store passwords unencrypted. This "feature" is meant only for the installation and test phase. But assuming a non-professional administrator it is thinkable that for reasons of simplicity, ignorance or even laziness this setting is maintained. The consequences are fatal.

3. Login is possible via command line in a shell or via plaintext storage in a configuration file. This enables an attacker to spy on passwords by having a look at the shell history or into the configuration file. Even if one uses the password modules *Kerberos, Unix* or *NT* instead of *plaintext* the passwords are temporarily unencrypted and accessible.

4. Weak passwords are a widespread problem. But as other systems do not allow more than three unsucessfull attempts to login, JADE only logs such events. It is questionable whether those log files are checked in time or even regularly.

5. The JADE rights management could cause problems in case one computer is used by several persons. This mainly happens if

   - the operating system does not enforce ownership rights as for example with FAT16/32,
   - the user is not used to handle ownership rights correctly (mainly a problem with Windows users since they are nearly forced to login with administrator rights),
   - JADE or configuration files are stored in group directories.

6. Depending on the task assigned to an agent it may be necessary to provide it with some extended rights with regard to program execution, network access and hard drive access. While the latter can be easily restricted to some specific files or directories, the right to execute programs is too dangerous to be ever granted.

7. As any other system, JADE(-S) is in danger of viruses, worms and Trojan horses. Such attacks are very common nowadays and hard to fight (see e.g. [7]).

3.2 Attacks on JADE(-S)

Agent vs. Platform

- *Denial of Service*: With JADE-S all DoS attacks which aim on flooding agents or containers with messages are still possible. There is only one restriction: The attacker must have a valid account for the platform. There is no protocol instance dealing with such security problems. This means an attack as well as its source must be detected manually which is nearly impossible in case of a distributed DoS attack.

- *Weed*: There is still no possibility to detect such attacks.

- *Flying Dutchman*: JADE-S does not check whether an agent obeys to a termination order. Still, the only possibility to force an agent to terminate itself is to put each agent in its own container and to end the container. This very "naive" solution causes a significant increase in communication complexity since for each communication the AMS must be asked for the receivers' addresses. Moreover, for each container a new JVM must be started which requires a lot of resources. The only solution would be a new implementation of the class `Agent` in which all critical system functions are declared `final`.

In version 3.2 the rights management at least prohibits cloning of agents. This prevents a part of the original attacks on JADE. In a future version of JADE-S one should not forget to refuse a migration during the termination process.

Agent vs. Agent

- *Denial of Service*: Besides an access control JADE-S has no tools against DoS attacks from within the system. For instance, there is no limitation for the bandwidth of agents. The only way to prevent from such attacks would be to prohibit communication at all.

Platform/Host vs. Agent In the previous paragraphs we could assume the JADE-S mechanisms itself as uncorrupted because possible attackers had only limited rights on the platform. Now, we have to change this view. There are two possible scenarios:

1. The container owner or host is malicious
2. The container or host have been corrupted

Both cases imply that a containers security installation may deviate from that of the platform and that configuration files, passwords or even JADE system files could be corrupted. Anyway, every attack one could think of is possible since JADE-S does not provide any means to check the system's integrity.

4 Implemented Attacks

In this section we will present in detail two attacks that we have successfully accomplished.

4.1 Manipulated Communication

Assuming either a malicious host or a successful external modification of the JADE installation the communication between all local hosts can be compromised. The only presumption necessary is either to temporarily deactivate the security manager or to be able to execute code with Reflections. How this could be achieved is described in section 3.1.

In our attack we have inserted an additional filter into the filter chain processing all communication to and from the current container. The original filter chain is depicted in figure 1.

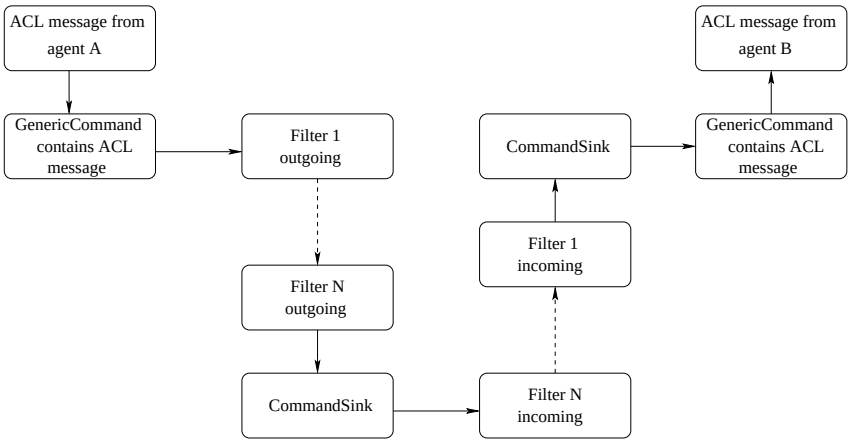


Figure 1: JADE message processing

The message encryption and signature mechanisms offered by JADE-S are implemented as filters in this chain. No message gets signed or encrypted before passing through the appropriate filter. The filter, however, will sign/encrypt whatever is passed to him. All messages reach those filters in plaintext. By inserting one filter each for both directions one can manipulate messages before they get signed/encrypted. There are no means for an agent to check whether a message has been tampered in this fashion. Therefore, the attacker has the ability to

- arbitrarily change the content of incoming messages,
- arbitrarily change the content of outgoing messages,
- drop messages,
- copy a message and resend it changed/unchanged at a later time.

We have implemented a specific filter for each possible attack in the listing above and all test messages have been recognized as authentic.

4.2 DoS Attack on Address Cache

Normally, the address of any target which is not located in the current container has to be looked up by asking the AMS. To speed up communication and lighten the load on the AMS, every container caches the last 100 communication targets. On platforms harboring a great number of agents this can be exploited for a DoS attack. A malicious agent simply has to send messages to more than 100 different agents which are located in other containers than itself (one additonal container is sufficient) to overflow the cache. In general, it will be easy to create a container with 101 agents that could be used as communication targets.

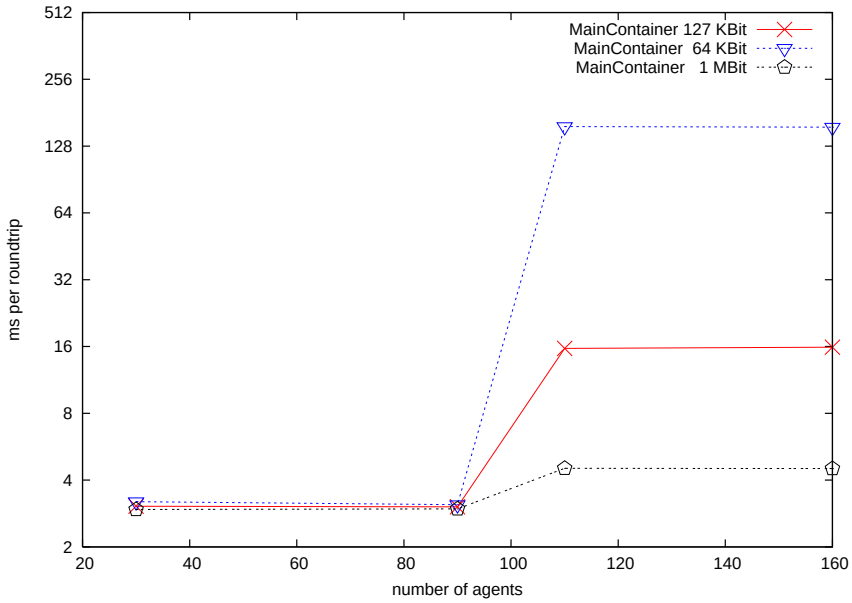


Figure 2: Measurements of the average round-trip time during the DoS attack

For our measurements we changed the official JADE benchmarks [2] in a way that a single agent is able to send messages to an arbitrary number of receivers. With this presumption we took the time between sending and receiving the messages. Besides answering messages the receiving agents have no functionality.

The tests have been performed on two Pentium 4-PCs with 1.8 and 3GHz and 512 MB and 1024 MB RAM, respectively. Both systems use Linux and are connected by 100 MBit Ethernet. To simulate the workload of the main container the bandwidth of the first computer has been reduced by the program `tc` to the values documented in figure 2 . This computer has only been used for the main container. Two other containers which contain the sending agent and the receivers, respectively, have been started on the second computer.

Measurements with a proof-of-concept implementation yielded the results shown in figure 2 (note that the scale is exponential). However, these results have to be verified in normal use because additional load on the AMS was simulated by limiting the sending bandwidth of the AMS to the indicated values. In figure 2 it is obvious that the jump in round-trip time marks the transition from less than 90 to 110 agents thereby creating a cache thrashing effect. Currently this attack is not reasonable under JADE-S with its lacking mobility support as one could only affect the container in which the sending agent was created. Technically however nothing prevents this attack under JADE-S. Moreover, one has to expect the results to worsen significantly in a real environment in which agents as well as their hosts are stressed with more processes.

5 Discussion

When we decided to analyze JADE, this decision was based on three reasons mainly:

1. JADE is an open source project and thus we thought it would be easy to gain detailed information about the systems implementation.

2. JADE is well-known, broadly used and conform to the FIPA standardization.

3. It was announced that version 3.2 (which came out in July 2004) would provide a security plug-in which supports agent mobility.

It turned out that with respect to 1. and 3. we were mistaken. While there is a usage-manual available for JADE-S, information about design and implementation is very scarce. Direct email to the developers yield some information but largely the recommendation to participate in the development and thereby gain more knowledge of the internals. In general most documentation on JADE is only about basic usage. This does not create trust in the JADE security. Concerning mobility JADE-S does not support any in this version. On the one hand a lot of security problems can thus be avoided but on the other hand mobility is a popular feature in multi-agent systems and this lack will lead to various completely unsecured platforms. Moreover, it is questionable whether secured mobility can be integrated into the existing implementation at all. In our opinion the developers have discovered this problem too and, therefore, made no efforts to do so.

Besides these two remarks we wish to draw attention on the following problems:

- *Storage of Private Keys:* When looking at the source code, JADE reveals some oversights in its implementation. According to comments in the sources it is planned that some keys for encryption or signing can be handled persistently by the platform. To prevent an extraction of these keys from either memory or disc, they should be stored encrypted. The implementation indicates that `java.security.KeyStore` will be utilized to accomplish this. However, the current (partial) implementation holds local copies of the concerned keys which defeats the purpose at least for memory protection.

- *Credentials:* Another issue which shakes the faith in the current implementation of JADE-S are repeated comments containing 'FixMe'-remarks concerning the handling of credentials. As basis of the rights-management system that JADE offers it should at least be noted in the manual or release notes, if some parts are only half-implemented.

- *Weed Agents:* JADE-containers do not enforce the termination of agents. In case its container calls the agents method `.doDelete()` the agent should be terminated. This method however is not declared `final` and can therefore be changed. Currently, the only secure way to kill an agent (even with JADE-S enabled) is to destroy the corresponding container. Agents can therefore consume computing power without easily being destroyed. If not fixed this could mean a serious problem with DoS attacks, once mobility is available for JADE-S.

Our overall evaluation of JADE(-S) is the following: Considering the security classes presented in the introduction, we must say that JADE only provides some mechanisms for external security. It seems as if intruders and, even worse, possibly malicious parties are not regarded as a real thread. Even presuming the implemented mechanisms for authentication, signing and encrypting as secure, this is far away from a system that can be used for any sensitive application.

References

[1] Campione, M., and K. Walrath, *The Java Tutorial, Second Edition: Object-Oriented Programming for the Internet*, Addison Wesley Publishing Company, 1998.

[2] Cortese, E., *Benchmark on JADE Message Transport System*. URL: `http://jade.cselt.it/doc/tutorials/benchmark/JADERTTBenchmark.htm`, 2005.

[3] Dreyer, U., *Agenten und Server – Sicher unter JADE(-S)?*, Diploma thesis, Universität Karlsruhe (TH), Institut für Algorithmen und Kognitive Systeme, March 2005,

[4] Endsuleit, R., and T. Mie, *Secure Multi-Agent Computations*, "Proc. of Int. Conf. on Security and Management", CSREA, 149–155, 2003.

[5] Endsuleit, R., and A. Wagner, *Possible Attacks on and Countermeasures for Secure Multi-Agent Computation*, CSREA "Proc. of Int. Conf. on Security and Management", 221–227, 2004.

[6] *Java Agent DEvelopment Framework*. URL: `http://jade.tilab.com/community-3rdpartysw.htm`, 2005.

[7] Keizer, G., *Unpatched PC "Survival Time" Just 16 Minutes*, InternetWeek. URL: `http://www.internetweek.com/story/showArticle.jhtml?articleID=29106061`, 2005.

[8] Maret, P., and J. Calmet, *Modeling Corporate Knowledge within the Agent Oriented Abstraction*, "Proc. of Int. Conf. on Cyberworlds", IEEE Computer Society, 224–231, 2004.

[9] Shakshuki, E., and Y. Jun, *Multi-agent Development Toolkits: An Evaluation*, "Proc. of IEA/AIE", Lecture Notes in Artificial Intelligence, vol. **3029**, Springer Verlag, 209–218, 2004.

Digital signature in automatic analyses for confidentiality against active adversaries

Ilja Tshahhirov
Tallinn University of Technology

Peeter Laud
Tartu University and Cybernetica AS

Abstract

In this article we enhance the technique of static analysis for confidentiality in cryptographic protocols with support for digital signature operations. The presented technique is an extension of cryptographic protocol analysis presented by Laud, being similar to Abadi and Rogaway — based on replacing the cryptographic operations in the protocols with constructs that are "obviously secure". The replacements are made in such a way that no insecure protocol becomes secure. The transformed protocols are then statically analysed; they should be easier to analyse than the original protocol. Handling the digital signatures is a step in exploring the general approach and making it able to handle realistic scenarios gracefully.

1 Introduction

A cryptographic protocol is expected to satisfy certain security properties. One of them is confidentiality — whether the adversary is able to gain some information about the secret messages transferred during the execution of the protocol.

There are techniques for automatic checking the confidentiality of the protocols. Our work is an extension of analysis presented by Laud [11] to the digital signature primitive. Our main contribution is extending the programming language of protocols analysed in [11] with operations for generating and verifying digital signatures, extending and adjusting the rest of the analysis appropriately. The correctness of this transformation is based on the security definition of signature scheme. This contribution gives a completely automatable method of analysing protocols for the preservation of confidentiality of secret messages, the results of the analysis are correct in the computational model, even when considering *active* adversaries.

This paper has the following structure. After reviewing some related work in Sec.2 we state the security properties of the considered signature scheme in Sec. 3 and extend the protocol language in Sec. 4. We give the security definition of the protocol in Sec. 5. In Sec. 6 we describe our main contribution — a protocol analysis extension to digital signature primitive. Last, we describe a very simple and conservative information flow analysis in Sec. 7 that is nevertheless quite successful in analysing the protocols. We give an example of analysis in Sec. 8.

2 Related Work

The results presented in this paper belong to a body of work attempting to reconcile the two main approaches for modelling and analysing cryptographic protocols — the Dolev-Yao (or "formal") model [8] and the complexity-theory-based (or "computational") approach [15]. Indeed, the semantics of our protocols is defined in terms of the computational model, while the analysis is closer to the formal model. The work in this area has been started by Abadi and Rogaway [1] who considered the relationship of formal and computational symmetric encryption under passive attacks. The same primitive and class of attacks has been further considered in [2, 14, 10, 13, 3], in these papers the language has been expanded (the constraints have been weakened) and the security definitions have been clarified.

If active adversaries are considered as well then the reconciliation approaches have mostly considered asymmetric primitives and/or integrity properties (i.e. properties of execution traces). Hence the digital

signatures are one of the most-investigated primitives in this setting. Note that it only makes sense to consider digital signatures in settings where the adversary is active because a passive adversary cannot forge anything. Several implementations of digital signatures and related functionalities (certification, etc.) have been proposed in the reactive simulatability / universal composability framework [4, 5, 6]. Both the reactive and non-reactive frameworks attempt to analyse a protocol in two steps. The first step is to translate the protocol to a form that is more similar to the Dolev-Yao model and hence more easily analysed; and the second step performs the actual analysis. The second step can provide no information to the first. In contrast, the framework that we use [11] performs both steps in parallel, thereby having the potential to achieve better precision.

3 Basic Cryptographic Notions

A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for all positive polynomials p there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $|f(n)| < 1/p(n)$. The cryptographic algorithms work with bit-strings, we denote the set of all bitstrings with Σ.

A signature scheme is a triple of polynomial-time algorithms ($\mathcal{G}$, $\mathcal{S}$, $\mathcal{T}$). $\mathcal{G}$ and $\mathcal{S}$ are probabilistic, $\mathcal{T}$ is deterministic. Here $\mathcal{G}$ is the *key pair generation algorithm*, $\mathcal{S}$ is the *signature generation algorithm*, and $\mathcal{T}$ is the *signature verification algorithm*. All algorithms take the *security parameter* n (represented in unary — 1^n) as an argument. Additionally, $\mathcal{S}$ takes two more arguments — secret signing key and message to sign. Also, $\mathcal{T}$ takes three more arguments — public test key, signature, and message to verify this signature with.

Let $msg \in \Sigma$. For all pairs (sk,pk) that may be returned by $\mathcal{G}$ and for all signatures s that may be returned by $\mathcal{S}(1^n, sk, msg)$, the algorithm $\mathcal{T}(1^n, pk, s, msg)$ must return true. This definition only states that $\mathcal{T}$ should recognize signatures, correctly generated with $\mathcal{S}$. We also want the signature system to be secure, i.e. forging the signature without possessing the secret key should be hard.

The signature oracle $\mathcal{O}_{sig}(\cdot)$ used in the security definition below is defined as follows (the security parameter n is assumed to be fixed):

- During the initialization, $\mathcal{O}_{sig}$ generates the key pair (sk, pk) by invoking the $\mathcal{G}(1^n)$, stores it in memory, and outputs the public key pk.
- Upon request to sign the message m, $\mathcal{O}_{sig}(m)$ produces the signature by invoking the $\mathcal{S}(1^n, sk, m)$ (the sk stored during the initialization is used) and returns it.

The signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{T})$ is called existentially unforgeable under adaptive chosen message attack (ACMA) if for every probabilistic polynomial-time (PPT) machine A_{sig} that interacts with the signature oracle $\mathcal{O}_{sig}(\cdot)$ based on this scheme the following holds: The probability is negligible (in n) that A_{sig} finally outputs two values m and s (meant as a forged signature for the message m) with $\mathcal{T}(1^n, pk, s, m) = \mathsf{true}$ and where m is not among the messages previously signed by $\mathcal{O}_{sig}(\cdot)$.

Both these definitions are similar to the ones used in [5], with the only difference that the maximum number of signatures ns that can be generated with key pair is passed to the signature oracle as an intialization parameter in [5]. As the number of signatures generated during the execution of the protocol subject to the analysis is bounded (one of the key assumptions in the original analysis - [11]), we assume that ns is sufficiently large and omit ns from further consideration.

One more property we require for this analysis is that the signature generation algorithm should return different signatures each time (this property is required in order to analyse all the signatures separately while replacing the signature test operations). It can be formalized as follows. Consider the oracle $\mathcal{S}(1^n, \cdot, \cdot)$. This oracle invokes the signing algorithm taking both the message to be signed and the secret key as arguments. We require that no adversary can cause (with non-negligible probability) the oracle $\mathcal{S}(1^n, \cdot, \cdot)$ to give the same answer to two queries.

A signature scheme satisfying the last requirement can be constructed. Let $(\mathcal{G}', \mathcal{S}', \mathcal{T}')$ be any signature scheme existentially unforgeable against ACMA (such systems exist under standard assumptions, see [5]). We construct the signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{T})$ as follows:

- $\mathcal{G}$ is equal to $\mathcal{G}'$.
- $\mathcal{S}(1^n, sk, m)$ first generates the signature s by invoking the $\mathcal{S}'(1^n, sk, m)$. Then, it generates a random bit-string r of length n and outputs (s, r).

- $\mathcal{T}(1^n, pk, (s, r), m)$ just returns the result of $\mathcal{T}'(1^n, pk, s, m)$. Note that r is thrown away. As we do not assume that adversary is unable to generate another valid signature for already signed message, the only function of r is to randomize the generated signature.

Obviously the probability that $\mathcal{S}$ returns the same answer twice is bounded from above by $q^2/2^n$, where q is the number of queries that the adversary makes to $\mathcal{S}(1^n, \cdot, \cdot)$. It is negligible, because q is at most polynomial in n. It is also obvious that the resulting signature scheme remains existentially unforgeable against ACMA.

4 Syntax and semantics of a protocol

Protocol, as it is understood here, is a set of programs performing certain computations over the values known to them (computed or received over the network) and exchanging messages. Additionally, there are long term secrets known or shared by participants, initialized prior to the start of the protocol. Let the set **Var** include all the variables used in the protocol.

$$
\begin{array}{rcllll}
P & ::= & k := \textbf{gen\_key} & | \; y := (x_1, \ldots, x_m) & | \; x := \pi_i^m(y) \\
 & | & x := encr_k(y) & | \; y := decr_k(x) & | \; x := \textbf{random} \\
 & | & \textbf{send}\ x & | \; x := \textbf{receive}^{\ell} & | \; \textbf{check}(x = y) \\
 & | & x := \textbf{constant}(b) & | \; x := y & | \; kp := gen\_key\_pair \\
 & | & pk := public\_key(kp) & | \; sm := sign_{kp}(m) & | \; test_{pk}(sm) \\
 & | & m := get\_signed\_msg(sm) &
\end{array}
$$

$k, x, x_1, \ldots, x_m, y, kp, pk, m, sm$ are *variables* from a set **Var** and b is a function mapping the security parameter to some bit-string.

Figure 1: Statements

A program is a sequence of statements. The statements allowed in the programs are defined in Fig. 1. The semantics of the most of the statements is trivial. $kp := gen\_key\_pair$ generates public-secret key pair, which may be used for signing the messages. $pk := public\_key(kp)$ extracts public key from the key pair. $sm := sign_{kp}(m)$ generates signature for the message and returns the signed message. Both $public\_key(kp)$ and $sign_{kp}(m)$ get stuck (the execution of correspondig participant is stopped) if kp is not valid key pair. $m := get\_signed\_msg(sm)$ extracts the message itself from the signed message; gets stuck if sm is not signed message. $test_{pk}(sm)$ gets stuck if sm was not correctly produced by $sm := sign_{kp}(m)$ (pk is $public\_key(kp)$); it also gets stuck if sm is not signed message or kp is not valid public key.

More detailed information on the operations not related to the digital signature primitive can be found in the [11]. Their *modus operandi* is similar — they get stuck if something is wrong. In particular, the i-th projection π_i^m gets stuck if its argument is not a m-tuple and $\textbf{check}(x = y)$ gets stuck if the values of x and y differ. The superscript ℓ of a **receive**-statement denotes the expected length (as a function of the security parameter) of the incoming message.

In a protocol, each variable is allowed to occur at most once at the left side of an assignment. Let $\textbf{Var}_{\text{sent}}$ be the set of all variables x, such that the statement **send** x occurs in the protocol.

The intermediate protocols resulting from the transformation may include some more statements: semaphores, case-statements, and $in()$-statements.

Semaphores (simple synchronization primitives) are implemented using statements $signal(s)$ and $wait(s)$ ($s \in \textbf{Sem}$ is a semaphore). The statement $signal(s)$ records that the semaphore s has been passed and the statement $wait(s)$ checks whether s has been passed and becomes stuck, if this is not the case. Each semaphore s may occur at most once in the statement $signal(s)$. The format and semantics of *case*-statements are defined in [11]. The $in_z(x_1, \ldots, x_n)$-statement checks whether z is equal to one of the x_i ($i = 1 \ldots n$). If $z = x_i$ ($i = 1 \ldots n$), the execution continues. Otherwise, it gets stuck.

4.1 Protocol execution

We now define what is done during the protocol execution. The emphasis is on the execution of statements related to the digital signature, other statements are described very briefly. [11] contains full details on them.

The following notions are used in the description of the protocol execution: **Parts** — set of (the names of) protocol participants; $\wp(\mathsf{pre})$ — protocol prelude, $\textbf{Parts}'$ — $\textbf{Parts} \dot{\cup} \{\mathsf{pre}\}$, $\wp$ — protocol — function mapping each element of $\textbf{Parts} \dot{\cup} \{\mathsf{pre}\}$ to a program.

The protocol runs in parallel to the adversary, which schedules the participants and relays messages between them. Let **ASt** be the set of all possible internal states of the adversary; the set **ASt** is (isomorphic to) some subset of Σ. Adversary $\mathcal{A}(1^n, \ldots)$ implements a probabilistic function $\mathcal{A}_n$ from $\Sigma_\perp \times \mathbf{ASt}$ to $\Sigma \times \mathbf{Parts}' \times \mathbf{ASt}$; the running time of $\mathcal{A}(1^n, \ldots)$ must be polynomial in n. Here n is security parameter, and $X_\perp$ is the set $X \dot\cup \{\perp\}$. The $\mathcal{A}_n$ takes the message sent to the network, and returns the message received by a participant executing $x :=$ **receive** at the next moment, as well as the name of the protocol participant that executes next.

A fixed variable $M \in \mathbf{Var}$ contains secret message before the execution of the rest of the protocol. Other variables get their values during the run of the protocol. Let p be a polynomial and let τ be an efficiently computable injective function from $\Sigma \times \Sigma$ to Σ, such that τ^{-1} is also efficiently computable. Let $(\mathcal{G}, \mathcal{S}, \mathcal{T})$ be a fixed secure signature scheme. Let keypair, publickey, and signedmessage be certain fixed bit-strings. The execution of the protocol proceeds as follows.

Initialisation.

The *execution state* E is the sequence of all the values describing the running state of the protocol. E contains the values of all initialized variables, but also other quantities like the parts of programs that are yet unexecuted, the currently executing participant, the message currently in transit (between a participant and the adversary), and the current state of the adversary. Let **ExcSt** be the set of all execution states.

During the protocol initialization the initial values to the components of E are set. It is done exactly as in the original analysis [11].

Step. During a protocol step, a statement at the head of one of the currently unexecuted parts of protocols gets executed. Let S be this statement. Here we define what happens if S is signature-related. Please refer to [11] for the rest.

- If S is $kp :=$ *gen_key_pair* then invoke $(sk, pk) := \mathcal{G}(1^n)$, set $v_{kp} := \tau(\mathsf{keypair}, \tau(sk, pk))$. Also, set i_{kp} to true. Here v_{kp} is the value of the variable kp and i_{kp} is a flag showing whether kp has been initialized.
- If S is $pk :=$ *public_key*(kp), then parse v_{kp} as $(t, x_1) := \tau^{-1}(kp)$, and $(\_, x_3) := \tau^{-1}(x_1)$. If some invocation of τ^{-1} fails or $t \neq \mathsf{keypair}$, set r_C to false. Otherwise set v_{pk} to $\tau(\mathsf{publickey}, x_3)$ and i_{pk} to true. Here C is the name of the currently running participant and r_C is a flag showing whether C has not become stuck yet.
- If S is $sm :=$ *sign*$_{kp}(msg)$ then parse v_{kp} as $(t, x_1) := \tau^{-1}(kp)$, and $(sk, \_) := \tau^{-1}(x_1)$. If some invocation of τ^{-1} fails or $t \neq \mathsf{keypair}$, set r_C to false. Otherwise, set v_{sm} to $\tau(\mathsf{signedmessage}, \tau(v_{msg}, \mathcal{S}(1^n, sk, v_{msg})))$ and i_{sm} to true.
- If S is $test_{pk}(sm)$ then parse v_{pk} as $(t_1, pk) := \tau^{-1}(v_{pk})$, v_{sm} as $(t_2, x) := \tau^{-1}(v_{sm})$, and $(m, s) := \tau^{-1}(x)$. If any invocation of τ^{-1} fails, or $t_1 \neq \mathsf{publickey}$, or $t_2 \neq \mathsf{signedmessage}$, set r_C to false. Otherwise, set r_C to the value returned by $\mathcal{T}(1^n, pk, s, m)$.
- If S is $msg :=$ *get_signed_msg*(sm), then parse v_{sm} as $(t, x) := \tau^{-1}(v_{sm})$ and $(m, \_) := \tau^{-1}(x)$. If any invocation of τ^{-1} fails, or $t \neq \mathsf{signedmessage}$, set r_C to false. Otherwise, set v_{msg} to m and i_{msg} to true.
- If S is $in_z(x_1, \ldots, x_n)$ and i_z is false, set r_C to false. Otherwise, find the first index j, such that i_{x_j} is true and $v_{x_j} = v_z$. If there is no such such j or i_{x_j} is false then set r_C to false. Otherwise, continue execution.

The steps define a probabilistic transition system over the set of execution states. For each $E \in \mathbf{ExcSt}$ there's certain probability that execution of the protocol (together with the adversary) stops at this step. This probability distribution over the **ExcSt** (together with adversary $\mathcal{A}$) is denoted by the $[\![\wp, \mathcal{A}]\!]_n$.

Fixing the random coin tosses made during the execution of each protocol statement and the adversary gives us a single state where the execution stops. If those random coin tosses are defined by $\mathbf{r}$ (as defined in [11]), this final execution state is denoted by $[\![\wp, \mathcal{A}]\!]_n^{\mathbf{r}}$.

4.2 An example

To illustrate our language, consider the simple protocol — transferring the public key and signed message with verification of the signature followed by that. Note that public key is explicitly made known to the adversary. This protocol is shown in Fig. 2.

Protocol prelude	the program for A	the program for B
$K_{AB} := \mathbf{gen\_key}^{(1)}$	$KP_A := gen\_key\_pair^{(2)}$ $PK_A := public\_key(KP_A)$ **send** PK_A $PK_e := encr_{K_{AB}}(PK_A)^{(3)}$ **send** PK_e $SM := sign_{KP_A}(M)^{(4)}$ $SM_A := encr_{K_{AB}}(SM)^{(5)}$ **send** SM_A	$m_1 := \mathbf{receive}^{l_1(6)}$ $PK_B := decr_{K_{AB}}(m_1)$ $m_2 := \mathbf{receive}^{l_2(7)}$ $SM_B := decr_{K_{AB}}(m_2)$ $test_{PK_B}(SM_B)$

Figure 2: Example protocol $\wp$ in our language

5 Security definition

A probability distribution over some finite or countable set X is a function $\mu : X \to [0,1]$, such that $\sum_{x \in X} \mu(x) = 1$. We denote the set of all probability distributions over the set X by $\mathcal{D}(X)$. We denote picking the value of the random variable x according to the distribution $D \in \mathcal{D}(X)$ by $x \leftarrow D$. The notation $\{\!|E : C|\!\}$ denotes the distribution of the random variable or expression E with the probability space defined by C, here C is a list of picking values to random variables and of defining variables.

The definition of the protocol security has not been changed from the original analysis - the protocol is considered secure if there's no non-negligible correlation between the final state of the adversary and the secret message:

$$\{\!|(E[v_M], E[AS]) \,:\, E \leftarrow [\![\wp, \mathcal{A}]\!]_n|\!\}_n \approx \{\!|(E'[v_M], E[AS]) \,:\, E, E' \leftarrow [\![\wp, \mathcal{A}]\!]_n|\!\}_n \ . \tag{1}$$

The v_M is the value of secret message and the AS is the final state of the adversary.

The exact meaning of indistinguishability between two *families* of probability distributions $D = \{D_n\}_{n \in \mathbb{N}}$ and $D' = \{D'_n\}_{n \in \mathbb{N}}$, denoted $D \approx D'$, is following: for all PPT algorithms $\mathcal{A}$, the difference of probabilities

$$\mathbf{P}[b = 1 \,|\, x \leftarrow D_n, b \leftarrow \mathcal{A}(1^n, x)] - \mathbf{P}[b = 1 \,|\, x \leftarrow D'_n, b \leftarrow \mathcal{A}(1^n, x)]$$

is a negligible function of n.

6 Analysing the protocols

The analysis of signature operations is done in several steps. First, the protocol is abstractly interpreted - each variable is assigned a term describing the construction of its value from the atomic values (keys, key pairs, public keys, signed messages, random numbers, constants, secret message, and advisory's input) - this is necessary for tracking the usage of key pairs. This part is described in the Sec. 6.1. Given the abstract interpretation, the protocol is analysed for the dead code, as described in Sec. 6.2. Then we attempt to find the key pairs which are not made available to the adversary (this check is necessary, because exposing the secret key ruins the definition of the signature scheme security). After the candidate key pair is found, we attempt to do a modification to the protocol, directly following from the security properties of the signature scheme. Given that without possessing the secret key it is computationally infeasible to generate valid signature for the message that has not been previously signed, we can make certain assumptions on the value of the message being tested. In order for $test_{pk}(sm)$ operation to succeed, sm has to be equal to one of the messages signed with the corresponding key pair, so we can supply the additional axioms. These additional axioms serve two purposes: first help us in statically analysing the outcome of $test_{pk}(sm)$ operation, and then they allow taking into the account the authenticity of the signed data (which may be useful in further extension of the analysis, for instance, to the public key cryptography, where the public key may be sent unencrypted, but signed). The axioms are introduced in the form of $in_{sm}(sm_1, \ldots, sm_n)$, where $sm_1, \ldots, sm_n$ are all the messages which have been signed with the corresponding key pair. Then, the $in()$-statement is analysed in the same way as *case*-statement - by splitting the original protocol $\wp$ into branches $\wp_1, \ldots, \wp_n$, each corresponding to one of the possible values checked against in the $in()$-statement. Finding the key pairs, as well as transforming the protocol is described in Sec. 6.3.

After the modifications are done, the information flow analysis, described in Sec. 7, is applied to the protocol(s). The process of analysis is iterative — modifications are done until either protocol can be considered secure or no more transformations are possible — in this case the protocol is considered potentially insecure.

6.1 Interpreting the protocol

Let all the statements in the protocol be labelled by elements of set **Lab**. Different statements must have different labels. For all $A \in \mathbf{Parts}'$, let l_A be the length of the program $\wp(A)$. Let $L_\wp$ (used to enumerate all the statements in the protocol) be the following set:

$$L_\wp = \{(A, i) \mid A \in \mathbf{Parts}', 1 \le i \le l_A\} \ . \tag{2}$$

Terms are used to track how the values of the variables are constructed from the "atomic" values. The definition of terms $T \in \mathbf{Trm}$ is defined in Fig. 3.

$$\begin{array}{rcllllllll} T & ::= & \underline{key}(l) & | & \underline{random}(l) & | & \underline{const}(b) & | & \underline{secret} \\ & | & \underline{tuple}^m(T_1, \ldots, T_m) & | & \underline{\pi}_i^m(T_1) & | & \underline{input}(l) & | & \underline{decr}(T_1, T_2) \\ & | & \underline{encr}(l, T_1, T_2) & | & \underline{keypair}(l) & | & \underline{publickey}(T_1) & | & \underline{signedmsg}(l, T_1, T_2) \\ & | & \underline{msg}(T_1) & & & & & & \end{array}$$

Figure 3: The set of abstract values

Let $\mathsf{I}(x) \in \mathbf{Trm}$ denote the term assigned to the variable x. We let $\mathsf{I}(M) = \underline{secret}$, where $M \in \mathbf{Var}$ is the variable containing the secret value. The rest of the terms $\mathsf{I}(x)$ are defined by examining the statement defining x. The definition of $\mathsf{I}(x)$ for signature-related terms is given in Fig. 4 (for other terms, refer to [11]). Note that there is no term for *test*(), as this statement does not affect the values of

$x := \ldots$	$\mathsf{I}(x)$ is
$gen\_key\_pair^l$	$\underline{keypair}(l)$
$public\_key(kp)$	$\underline{publickey}(\mathsf{I}(kp))$
$sign_{kp}(m)^l$	$\underline{signedmsg}(l, \mathsf{I}(kp), \mathsf{I}(m))$
$get\_signed\_msg(sm)$	$\underline{msg}(\mathsf{I}(sm))$

Figure 4: Definition of the terms $\mathsf{I}(x)$

the variables. Also note that the label of the statement is used as the term argument for those operations which depend on either random coin tosses or adversary actions. That is necessary in order to be able to analyse two instances of the protocol executing in lock-step (with fixed coin-tosses).

The semantics of the terms $T \in \mathbf{Trm}$ are defined by $\iota_n^{\mathbf{r}} : \mathbf{Trm} \to \Sigma_\perp$ (the random choices $\mathbf{r} \in \mathbf{R}$ are fixed):

- $\iota_n^{\mathbf{r}}(\underline{keypair}(l)) = \tau(\mathsf{keypair}, \tau(sk, pk))$, where (sk, pk) is generated by invocation of $\mathcal{G}^{\mathbf{r}(l,\mathsf{p})}(1^n)$. Here $\mathbf{r}(l, \mathsf{p})$ denotes the random coins in $\mathbf{r}$ intended for the legitimate participant at the statement labelled by l. These coins are used as the random coins for the key generation algorithm. Also, the generated key pair is *tagged with* keypair.
- $\iota_n^{\mathbf{r}}(\underline{publickey}(T))$ is $\perp$ (meaning that the statement defining the variable will get stuck) if $\iota_n^{\mathbf{r}}(T)$ is $\perp$. Otherwise, T is parsed as follows: $(t_1, kp) := \tau^{-1}(\iota_n^{\mathbf{r}}(T))$ and $(\_, pk) := \tau^{-1}(kp)$. If t_1 is not keypair, or any invocation of τ^{-1} fails, $\iota_n^{\mathbf{r}}(\underline{publickey}(T))$ is $\perp$. Otherwise, it is equal to $\tau(\mathsf{publickey}, pk)$.
- $\iota_n^{\mathbf{r}}(\underline{signedmsg}(l, T_1, T_2))$ is $\perp$ if $\iota_n^{\mathbf{r}}(T_1)$ is $\perp$ or is not tagged with keypair or $\iota_n^{\mathbf{r}}(T_2)$ is $\perp$. Otherwise it is equal to $\tau(\mathsf{signedmessage}, \tau(\iota_n^{\mathbf{r}}(T_2), \mathcal{S}^{\mathbf{r}(l,\mathsf{p})}(1^n, t_1, \iota_n^{\mathbf{r}}(T_2))))$ where $(\_, kp) = \tau^{-1}(\iota_n^{\mathbf{r}}(T_1))$ and $(t1, \_) = \tau^{-1}(kp)$. If any of the invocations of τ^{-1} fails, the semantics of $\iota_n^{\mathbf{r}}(\underline{signedmsg}(l, T_1, T_2))$ is $\perp$.
- $\iota_n^{\mathbf{r}}(\underline{msg}(T))$ is computed as follows: $(t_1, x) = \tau^{-1}(\iota_n^{\mathbf{r}}(T))$ and $(t_2, \_) = \tau^{-1}(x)$. If t_1 is not signedmessage, or any invocation of τ^{-1} fails, $\iota_n^{\mathbf{r}}(\underline{msg}(T))$ is $\perp$. Otherwise, it is equal to t_2.

Let $X_n^{\mathbf{r}} \subseteq L_\wp$ denote the set of statements that are actually executed (for the random choices $\mathbf{r}$), i.e. the execution of the protocol is not stopped before reaching that statement and the participant containing that statement does not get stuck at or before that statement.

Theorem 1. *Let $x \in \mathbf{Var}$ and let $l \in L_\wp$ be (the label of) the statement defining x. If $l \in X_n^{\mathbf{r}}$ or $x = M$ then*

$$[\![\wp, \mathcal{A}]\!]_n^{\mathbf{r}}[v_x] = \iota_n^{\mathbf{r}}(\mathsf{I}(x)) \ . \tag{3}$$

Theorem is proven by induction over the order of executing the statements (determined by $\mathbf{r}$). It mainly consists of comparing the definitions of the execution step and $\iota_n^{\mathbf{r}}$. Please refer to [11] for it.

There are rules for deriving the equality and inequality of two terms, denoted by $T_1 \simeq T_2$ and $T_1 \not\simeq T_2$, correspondingly. For example, getting the message body and signing the message cancel out (under certain conditions). An example of inequality is that different key pairs are unequal. Also, if the semantics of a certain term is almost always $\perp$, we denote it by $\mathsf{stuck}\, T$.

Fig. 5 presents the axioms and inference rules defining $\simeq$, $\not\simeq$ and stuck for terms introduced due to or affected by the inclusion of digital signature primitive (complete set of rules can be found in [11]). The syntax of the rules is quite simple - if the premises (above the horizontal line) are satisfied, then the rule consequent (below the horizontal line) also holds. For instance, the rule (nec) says that the type of the value that a bit-string represents can be determined from that bit-string. The rules (dif_*) say that different random bit-strings are different. The rules (inc_*) state that the corresponding operations check the types of their inputs.

$$\frac{\underline{C}, \underline{C'} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{secret}, \underline{tuple^n}, \underline{encr}, \underline{publickey}, \underline{signedmsg}, \underline{keypair}\} \quad \underline{C} \neq \underline{C'}}{\underline{C}(\ldots) \not\simeq \underline{C'}(\ldots)}(\mathrm{nec})$$

$$\frac{l \neq l'}{\underline{keypair}(l) \not\simeq \underline{keypair}(l)}(\mathrm{dif_{kp}}) \quad \frac{\underline{C} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{tuple^n}, \underline{encr}, \underline{publickey}, \underline{signedmsg}\}}{\mathsf{stuck}\, \underline{publickey}(\underline{C}(\ldots))}(\mathrm{inc_{pk}})$$

$$\frac{l \neq l'}{\underline{signedmsg}(l, T_1, T_2) \not\simeq \underline{signedmsg}(l', T_3, T_4)}(\mathrm{dif_s}) \quad \frac{\underline{C} \in \{\underline{keypair}, \underline{publickey}, \underline{signedmsg}\}}{\mathsf{stuck}\, \underline{\pi_i^m}(\underline{C}(\ldots))}(\mathrm{inc_{p2}})$$

$$\frac{\underline{C} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{tuple^n}, \underline{encr}, \underline{publickey}, \underline{signedmsg}\}}{\mathsf{stuck}\, \underline{signedmsg}(\underline{C}(\ldots), T_1)}(\mathrm{inc_s})$$

$$\frac{\underline{C} \in \{\underline{keypair}, \underline{publickey}, \underline{signedmsg}\}}{\mathsf{stuck}\, \underline{decr}(T, \underline{C}(\ldots))}(\mathrm{inc_{e3}}) \quad \frac{\underline{C} \in \{\underline{keypair}, \underline{publickey}, \underline{signedmsg}\}}{\mathsf{stuck}\, \underline{decr}(\underline{C}(\ldots), T)}(\mathrm{inc_{e4}})$$

$$\frac{\underline{C} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{tuple^n}, \underline{encr}, \underline{publickey}, \underline{keypair}\}}{\mathsf{stuck}\, \underline{msg}(\underline{C}(\ldots))}(\mathrm{inc_m}) \quad \frac{}{\underline{msg}(\underline{signedmsg}(l, T_{kp}, T_M)) \simeq T_M}(\mathrm{can_m})$$

Figure 5: Equality and inequality of terms

We say that $T_1 \simeq T_2$ is *sound at the statement* S^l if the probability

$$\mathbf{P}[l \in X_n^{\mathbf{r}} \wedge \iota_n^{\mathbf{r}}(T_1) \neq \perp \neq \iota_n^{\mathbf{r}}(T_2) \wedge \iota_n^{\mathbf{r}}(T_1) \neq \iota_n^{\mathbf{r}}(T_2) \ : \ \mathbf{r} \in_R \mathbf{R}] \tag{4}$$

is negligible in n. Similarly, $T_1 \not\simeq T_2$ is *sound at the statement* S^l if the probability

$$\mathbf{P}[l \in X_n^{\mathbf{r}} \wedge \iota_n^{\mathbf{r}}(T_1) \neq \perp \neq \iota_n^{\mathbf{r}}(T_2) \wedge \iota_n^{\mathbf{r}}(T_1) = \iota_n^{\mathbf{r}}(T_2) \ : \ \mathbf{r} \in_R \mathbf{R}] \tag{5}$$

is negligible in n. We say that $\mathsf{stuck}\, T$ is *sound at the statement* S^l if the probability

$$\mathbf{P}[l \in X_n^{\mathbf{r}} \wedge \iota_n^{\mathbf{r}}(T) \neq \perp \mid \mathbf{r} \in_R \mathbf{R}] \tag{6}$$

is negligible in n. If the statement defining the variable whose abstract value is T is above S then S is dead.

Theorem 2. *All rules in Fig. 5 are sound. I.e. if S^l is a statement of the protocol and all the premises of some rule are sound at S^l then the consequent of that rule is also sound at S^l.*

Proof.

- Rule (nec): the constructors C and C' tag the values that they construct with different tags. The probability of these values being equal is zero.

- Rule (dif_{kp}): The semantics of two terms in the consequent are two different key pairs. There is only negligible chance that they are equal (if the probability of two key pairs being equal were non-negligible then there would be non-negligible chance of guessing key pairs).
- Rule (dif_{s}): The semantics of two terms in the consequent are two different signatures. There is only negligible chance that they are equal (directly follows from the required properties of the encryption scheme).
- Rules (inc_*): clearly, the semantics of the term in the consequent of the rule is always $\bot$.
- Rule (can_{m}): pick $\mathbf{r} \in \mathbf{R}$ in such way that the event described by (4) happens for the premise (if there are any premises). Such picking disallows only a negligible fraction of possible random choices. By definition of $\iota_n^{\mathbf{r}}$, the semantics of a term in the consequence of the rule is either $\bot$ or equal to the semantics of the simpler term. □

6.2 Removing dead code

In addition to removing the cycles in the execution graph (i.e. cases where $wait(s)$ occurs before $signal(s)$) and looking for contradictions from $\simeq$ and $\not\simeq$, as described in [11], one more technique is introduced - checking the types of the arguments to $test()$-statements. The approach is the same as defined for the terms in inc_* rules, however, $test()$-statements do not produce terms, so it needs to be specified separately.

Consider the abstract interpretation of the arguments of statement $test_{pk}(sm)$. If one of the following is true, then the statement and the code following it is dead and can be excluded from further analysis.

- $\mathsf{I}(pk)$ is $C(\ldots)$ where C is one of $\underline{key}, \underline{random}, \underline{const}, \underline{tuple}^n, \underline{encr}, \underline{signedmsg}, \underline{keypair}$.
- $\mathsf{I}(sm)$ is $C(\ldots)$ where C is one of $\underline{key}, \underline{random}, \underline{const}, \underline{tuple}^n, \underline{encr}, \underline{publickey}, \underline{keypair}$.

6.3 Transforming the test-statements

The first task is to choose the key pair which can be replaced by black boxes creating signature with it and getting public key from it - i.e. this key pair may be used only for signing and for public key extraction. Choosing the key pair means selecting $l \in \mathbf{Lab}$, such that $\underline{keypair}(l)$ occurs in the values of variables.

In order to introduce the additional axioms prior the $test$ statements, we have to make sure that adversary is not able to forge signatures (i.e. generate valid signatures that pass $test$). One conservative way to do that is to make sure that the key pair does not affect the adversary view. Therefore, the rule for $\underline{keypair}(l)$ to be eligible to be chosen is: if the value of some variable x affects the adversary's view, then the $\underline{keypair}(l)$ may occur in $\mathsf{I}(x)$ only in context $\underline{signedmsg}(l, \underline{keypair}(l), T_1)$ or $\underline{publickey}(\underline{keypair}(l))$. Such variables x are:

- the elements of $\mathbf{Var}_{\text{sent}}$;
- the variables occurring in **check**()-statements for which we do not statically know whether the comparison succeeds;
- the variables occurring as ciphertexts in decryption statements;
- the variables occurring as keys in encryption or decryption statements
- the variables occurring as key pairs in signing statements (exception: $\underline{keypair}(l)$ itself may be the abstract value of $\mathsf{I}(x)$);
- the variables occurring as public keys or signed messages in $test()$-statements for which we do not statically know whether the test succeeds or not;
- the variables occurring in the right-hand side of the projection statements for which we do not statically know whether the projection succeeds or not.

While introducing the additional axioms prior to signature tests with public key corresponding to key pair $\underline{keypair}(l)$, we have to make sure that we do not test any signatures that were created with some other key pair. Since the key pair is never sent out (exception - public key component may be sent out - in this case part of the signature test statements — the ones which use public key received from the network —

will not be removed), we know exactly where it is used. Let $sm_1 := sign_{kp_1}(m_1), \ldots, sm_n := sign_{kp_n}(m_n)$ be all statements that $\mathsf{I}(kp_1) \simeq \ldots \simeq \mathsf{I}(kp_n) \simeq \underline{keypair}(l)$. If $test_{pk}(sm)$ is a signature testing statement in the protocol such that $\mathsf{I}(pk) \simeq \underline{publickey}(\underline{keypair}(l))$, then add the following code before it (assuming that tmp is a variable not occurring in the protocol):

$$tmp := get\_signed\_msg(sm)$$
$$in_{tmp}(m_1, \ldots, m_n)$$

By the security property of the signing oracle (existential unforgeable under ACMA), the chances that adversary is able to generate signature that will pass is negligible, so either the $test_{pk}(sm)$ will fail or value of the m will belong to the set of messages signed with the appropriate key pair.

The final step is removing the $in()$. This removal results in one or more protocols, such that the security of all of them implies the security of the original protocol. If the second argument of the $in()$ is an empty set, this statement is a dead code, which (along with the code following it) should be excluded from further analysis.

Let $\mathsf{IN}_1, \ldots, \mathsf{IN}_k$ be the $in()$-statements in the protocol $\wp'$. Let the second argument of the statement IN_i (set) consist of n_i items. The number of protocols $\wp'_1, \wp'_2, \ldots$ will be $n_1 \cdot n_2 \cdots n_k$; they are defined in the following way: for all $i \in \{1, \ldots, k\}$ choose $c_i \in \{1 \ldots, n_i\}$; each of the protocols $\wp'_1, \wp'_2, \ldots$ will correspond to one of such possible choices. Let $s_1, \ldots, s_n \in \mathbf{Sem}$ be semaphores not occurring in the protocol $\wp'$. To obtain a new protocol, we change the $in()$-statement IN_i in the following way:

- Add $signal(s_i)$ immediately after the statement defining x_{c_i}.
- Replace IN_i with the following fragment:

$$wait(s_i)$$
$$\mathbf{check}(z = x_{c_i})$$

Such change is applied to all $in()$-statements. The resulting protocol has no $in()$-statements. Basically, going from $\wp'$ to $\wp'_1, \wp'_2, \ldots$ just means analysing all choices of the $in()$-statements separately.

We are analyzing the branches separately, therefore we may wonder whether we are missing some implicit flows of information here. In other words, is it possible that all n programs that we get after transforming an $in()$-statement are secure, but the original program is not? Fortunately, this is not possible. In our programming language we have no means to let the value of the secret M influence, which of the branches will be taken. Therefore the choice of a branch only depends on the adversary's actions. If the adversary does not know M before one of the branches is chosen then the chosen branch does not depend on M and hence gives no information about M to the adversary.

When the described transformations are done, the resulting protocols are either submitted to the same analysis (if there still are unprocessed key pairs left) or to the information flow analysis in the next section.

7 A simple information flow analysis

We will now extend a information flow analysis from [11] to determine whether the secret variables (in our case this set is $\{M\}$) affect the view of the adversary.

First, we define the information flow. If some participant of the protocol contains a statement of the form $x{:=}E(x_1, \ldots, x_m)$, where E is any expression and $x_1, \ldots, x_m$ are all variables occurring in it, then we say that there is an *information flow* from the variable x_i to the variable x and write $x_i \Rightarrow x$.

The protocol P is deemed secure if $M \stackrel{*}{\Rightarrow} y$ holds for no y satisfying some of the following conditions:

- $y \in \mathbf{Var}_{\text{sent}}$;
- y occurs in a **check**-statement (unless we are able to determine its result from the inference rules in Fig. 5);
- y occurs as an argument to a projection (π) statement (unless we are able to statically predict whether it succeeds or fails);
- y occurs as a ciphertext in a decryption statement;

- y occurs as a key in an encryption or decryption statement;
- y occurs as an argument to the **test** statement (unless we are able to determine its result from the inference rules);
- y occurs as a key in the **sign** statement.

(here $\stackrel{*}{\Rightarrow}$ denotes the reflexive transitive closure of $\Rightarrow$). Otherwise the protocol is deemed potentially insecure.

If a protocol is secure then two instances of that protocol which differ only in the value of the secret variable M execute in lock-step, giving the same values to all public parts of the execution context. This result is proven in [11]. Extending it to digital signature operation is also trivial.

8 Analysis example

We demonstrate the developed technique by by applying it to the analysis of the security of the sample protocol from Sec. 4.2 (initial protocol is shown on Fig. 2). The abstract interpretation of that protocol is given in Fig. 6.

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
M	$\underline{secret}$
K_{AB}	$\underline{key}(1)$
KP_A	$\underline{keypair}(2)$
PK_A	$\underline{publickey}(\underline{keypair}(2))$
PK_e	$\underline{encr}(3, \underline{key}(1), \underline{publickey}(\underline{keypair}(2)))$
SM	$\underline{signedmsg}(4, \underline{keypair}(2), \underline{secret})$
SM_A	$\underline{encr}(5, \underline{key}(1), \underline{signedmsg}(4, \underline{keypair}(2), \underline{secret}))$
m_1	$\underline{input}(6)$
PK_B	$\underline{decr}(\underline{key}(1), \underline{input}(6))$
m_2	$\underline{input}(7)$
SM_B	$\underline{decr}(\underline{key}(1), \underline{input}(7))$

Figure 6: Interpretation of protocol $\wp$

Running the information flow analysis, we consider the protocol to be potentially insecure, as $SM_A \in \mathbf{Var}_{\text{sent}}$ and $M \stackrel{*}{\Rightarrow} SM_A$). Therefore we run protocol transformations in order to simplify the protocol. Since it is a "real" protocol, no dead code is found in it. The first round of transformations is replacing the encryption and decryption statements, as described in [11]. After running it, the protocol looks like shown on Fig.7.

Protocol prelude	the program for A	the program for B
$K_{AB} := \mathbf{gen\_key}^{(1)}$ $Z_1 := \mathbf{constant}(\mathbf{0}^{l_1(n)})$ $Z_2 := \mathbf{constant}(\mathbf{0}^{l_2(n)})$	$KP_A := gen\_key\_pair^{(2)}$ $PK_A := public\_key(KP_A)$ **send** PK_A $PK_e := encr_{K_{AB}}(Z_1)^{(3)}$ **send** PK_e $SM := sign_{KP_A}(M)^{(4)}$ $SM_A := encr_{K_{AB}}(Z_2)^{(5)}$ **send** SM_A	$m_1 := \mathbf{receive}^{l_1(6)}$ $PK_B := case\ m_1\ of$ $PK_e \rightarrow PK_A$ $SM_A \rightarrow SM$ $m_2 := \mathbf{receive}^{l_2(7)}$ $SM_B := case\ m_2\ of$ $PK_e \rightarrow PK_A$ $SM_A \rightarrow SM$ $test_{PK_B}(SM_B)$

Figure 7: Protocol $\wp$ after replacing encryption/decryption

Removing case-statements results in 4 sub-protocols, corresponding to all possible sub-cases (variable assignments). One of the sub-protocols (let's call it $\wp_1$) corresponds to the intended usage of the variables ($PK_B := PK_A$ and $SM_B := SM$). Other cases ($\wp_2$:$PK_B := PK_A$ and $SM_B := PK_A$; $\wp_3$:$PK_B := SM$ and $SM_B := SM$; $\wp_4$:$PK_B := SM$ and $SM_B := PK_A$) will lead to some malfunctioning of the protocol (as values are used in not-intended way).

For illustration purposes, we demonstrate the analysis of $\wp_1$ and $\wp_2$, since the analysis of $\wp_3$ and $\wp_4$ is similar to $\wp_2$.

Protocol prelude	the program for A	the program for B
$K_{AB} := \mathbf{gen\_key}^{(1)}$ $Z_1 := \mathbf{constant}(\mathbf{0}^{l_1(n)})$ $Z_2 := \mathbf{constant}(\mathbf{0}^{l_2(n)})$	$KP_A := gen\_key\_pair^{(2)}$ $PK_A := public\_key(KP_A)$ **send** PK_A $PK_e := encr_{K_{AB}}(Z_1)^{(3)}$ $signal(s_1)$ **send** PK_e $SM := sign_{KP_A}(M)^{(4)}$ $SM_A := encr_{K_{AB}}(Z_2)^{(5)}$ $signal(s_2)$ **send** SM_A	$m_1 := \mathbf{receive}^{l_1(6)}$ $wait(s_1)$ **check**$(m_1 = PK_e)$ $PK_B := PK_A$ $m_2 := \mathbf{receive}^{l_2(7)}$ $wait(s_2)$ **check**$(m_2 = SM_A)$ $SM_B := SM$ $test_{PK_B}(SM_B)$

Figure 8: Protocol $\wp_1$

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
Z_1	$\underline{const}(0^{l_1(n)})$
Z_2	$\underline{const}(0^{l_2(n)})$
PK_e	$\underline{encr}(3, \underline{key}(1), \underline{const}(0^{l_1(n)}))$
SM_A	$\underline{encr}(5, \underline{key}(1), \underline{const}(0^{l_2(n)}))$
PK_B	$\underline{publickey}(\underline{keypair}(2))$
SM_B	$\underline{signedmsg}(4, \underline{keypair}(2), \underline{secret})$

Figure 9: Interpretation of protocol $\wp_1$ (changes only)

The abstract interpretation of the protocol $\wp_1$ is given on Fig. 9. The protocol $\wp_1$ is checked for dead code - no dead is code found. At this point, the candidate key pair is found. The only key pair (KP_A) present in the protocol satisfies the criteria. Introducing the additional axioms prior to test statement corresponding to the protocol results in the protocol $\wp_1'$.

Protocol prelude	the program for A	the program for B
$K_{AB} := \mathbf{gen\_key}^{(1)}$ $Z_1 := \mathbf{constant}(\mathbf{0}^{l_1(n)})$ $Z_2 := \mathbf{constant}(\mathbf{0}^{l_2(n)})$	$KP_A := gen\_key\_pair^{(2)}$ $PK_A := public\_key(KP_A)$ **send** PK_A $PK_e := encr_{K_{AB}}(Z_1)^{(3)}$ $signal(s_1)$ **send** PK_e $SM := sign_{KP_A}(M)^{(4)}$ $SM_A := encr_{K_{AB}}(Z_2)^{(5)}$ $signal(s_2)$ **send** SM_A	$m_1 := \mathbf{receive}^{l_1(6)}$ $wait(s_1)$ **check**$(m_1 = PK_e)$ $PK_B := PK_A$ $m_2 := \mathbf{receive}^{l_2(7)}$ $wait(s_2)$ **check**$(m_2 = SM_A)$ $SM_B := SM$ $tmp := get\_signed\_msg(SM_B)$ $in_{tmp}(M)$ $test_{PK_B}(SM_B)$

Figure 10: Protocol $\wp_1'$

Removing the $in()$-statement leads us to the only possible branch, as indicated on Fig. 11.

The abstract interpretation of the protocol $\wp_1'$ with $in()$ removed is given in Fig. 12.

At this point we can conclude that the secret message does not affect adversary view in $\wp_1'$, since, given the interpretations of the SM_B and PK_B, the $test_{PK_B}(SM_B)$ always succeeds. The statement **check**$(tmp = M)$, given the abstract interpretation of tmp, always succeeds either.

Now let us consider the $\wp_2$ - the sub-protocol resulting in "not-intended" information flow. It is shown on Fig.13.

The abstract interpretation of the protocol shown in Fig.13, it is given in Fig. 14.

During the search for dead code we conclude that, given the interpretation of SM_B (which is $\underline{publickey}(\underline{keypair}(2))$), the $test_{PK_B}(SM_B)$ will always get stuck (the type check on SM_B will always fail). Considering the statements preceding it, we conclude that the $\wp_2$ is secure, since according to the information flows analysis M does not affect adversary view.

Sub-protocols $\wp_3$ and $\wp_4$ are analysed similarly to $\wp_2$, and give same result - protocol always gets stuck irrespective of the value of M. So, all sub-protocols are secure. Therefore, the original protocol $\wp$ is secure too.

Protocol prelude	the program for A	the program for B
$K_{AB} := \mathbf{gen\_key}^{(1)}$ $Z_1 := \mathbf{constant}(\mathbf{0}^{l_1(n)})$ $Z_2 := \mathbf{constant}(\mathbf{0}^{l_2(n)})$	$KP_A := gen\_key\_pair^{(2)}$ $PK_A := public\_key(KP_A)$ **send** PK_A $PK_e := encr_{K_{AB}}(Z_1)^{(3)}$ $signal(s_1)$ **send** PK_e $SM := sign_{KP_A}(M)^{(4)}$ $SM_A := encr_{K_{AB}}(Z_2)^{(5)}$ $signal(s_2)$ **send** SM_A	$m_1 := \mathbf{receive}^{l_1(6)}$ $wait(s_1)$ **check**$(m_1 = PK_e)$ $PK_B := PK_A$ $m_2 := \mathbf{receive}^{l_2(7)}$ $wait(s_2)$ **check**$(m_2 = SM_A)$ $SM_B := SM$ $tmp := get\_signed\_msg(SM_B)$ **check**$(tmp = M)$ $test_{PK_B}(SM_B)$

Figure 11: Protocol $\wp_1'$ with $in()$ removed

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
tmp	$\underline{secret}$

Figure 12: Interpretation of protocol $\wp_1'$ with $in()$ removed (changes only)

Protocol prelude	the program for A	the program for B
$K_{AB} := \mathbf{gen\_key}^{(1)}$ $Z_1 := \mathbf{constant}(\mathbf{0}^{l_1(n)})$ $Z_2 := \mathbf{constant}(\mathbf{0}^{l_2(n)})$	$KP_A := gen\_key\_pair^{(2)}$ $PK_A := public\_key(KP_A)$ **send** PK_A $PK_e := encr_{K_{AB}}(Z_1)^{(3)}$ $signal(s_1)$ **send** PK_e $SM := sign_{KP_A}(M)^{(4)}$ $SM_A := encr_{K_{AB}}(Z_2)^{(5)}$ $signal(s_2)$ **send** SM_A	$m_1 := \mathbf{receive}^{l_1(6)}$ $wait(s_1)$ **check**$(m_1 = PK_e)$ $PK_B := PK_A$ $m_2 := \mathbf{receive}^{l_2(7)}$ $wait(s_1)$ **check**$(m_2 = SM_A)$ $SM_B := PK_A$ $test_{PK_B}(SM_B)$

Figure 13: Protocol $\wp_2$

9 Conclusion

We have presented the extension of analysis given in [11] to digital signature primitive. The analysis works by first applying transformation described in Sec. 6 to the protocol, resulting in one or more subprotocols. The security of the transformed protocols can be analysed using the information flow analysis given in Sec.7. Our analysis is correct with respect to the computational semantics and (complexity-theoretical) definition of confidentiality.

There are two directions in which this work is planned to be extended:

- Adding the support for another cryptographic primitive — asymmetric encryption.
- Composing an automated prover.

The authors are in progress of doing the work in both of the mentioned directions.

10 Acknowledgement

We are thankful to the anonymous referees of NordSec 2005 conference for their comments. They certainly have improved the readability of this paper.

This research has been supported by the Estonian Information Technology Foundation and the ESF Grants No. 5645 and 6095.

References

[1] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). International Conference IFIP TCS 2000, LNCS 1872, pages 3–22, August 2000.

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
PK_B	$\underline{publickey}(\underline{keypair}(2))$
SM_B	$\underline{publickey}(\underline{keypair}(2))$

Figure 14: Interpretation of protocol $\wp_2$ (changes relative to $\wp_1$ only)

[2] Martín Abadi and Jan Jürjens. Formal Eavesdropping and its Computational Interpretation. Theoretical Aspects of Computer Software, 4th International Symposium (TACS 2001), LNCS 2215, pages 82–94, September 2001.

[3] Pedro Adão, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness of formal encryption in the presence of key-cycles. 10-th European Symposium on Research in Computer Security (ESORICS 2005), LNCS 3679, pages 374-396, September 2005.

[4] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A Universally Composable Cryptographic Library. 10th ACM Conference on Computer and Communications Security (CCS 2003), October 2003.

[5] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Low-level Ideal Signatures and General Integrity Idealization. Information Security Conference (ISC 2004), LCNS 3225, pages 39–51, September 2004.

[6] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001), pages 136–145, October 2001.

[7] Veronique Cortier and Bogdan Warinschi. Computationally Sound, Automated Proofs for Security Protocols. 14th European Symposium on Programming (ESOP 2005), LNCS 3444, pages 157–171, April 2005.

[8] Danny Dolev and Andrew C. Yao. On the security of public key protocols. IEEE Transactions on Information Theory, IT-29(12), pages 198–208, March 1983.

[9] Joshua D. Guttman, F. Javier Thayer, and Lenore D. Zuck. The Faithfulness of Abstract Protocol Analysis: Message Authentication. 8th ACM Conference on Computer and Communications Security (CCS 2001), pages 186–195, November 2001.

[10] Peeter Laud. Handling Encryption in Analysis for Secure Information Flow. 12th European Symposium on Programming (ESOP 2003), LNCS 2618, pages 159–173, April 2003.

[11] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. 2004 IEEE Symposium on Security and Privacy, pages 71–85, May 2004.

[12] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the Picture: Soundness of Formal Encryption in the Presence of Active Adversaries. 14th European Symposium on Programming (ESOP 2005), LNCS 3444, pages 172–185, April 2005.

[13] Daniele Micciancio and Saurabh Panjwani. Adaptive Security of Symbolic Encryption. 2nd Theory of Cryptography Conference (TCC 2005), LNCS 3378, pages 169–187, February 2005.

[14] Daniele Micciancio and Bogdan Warinschi. Completeness Theorems for the Abadi-Rogaway Logic of Encrypted Expressions. Workshop in Issues in the Theory of Security (WITS 2002), January 2002.

[15] Andrew C. Yao. Theory and Applications of Trapdoor Functions (extended abstract). 23rd Annual Symposium on Foundations of Computer Science, pages 80–91, November 1982.

Secure Dynamic Program Repartitioning

Rene R. Hansen and Christian W. Probst
Informatics and Mathematical Modelling
Technical University of Denmark
2800 Kongens Lyngby, Denmark
{rrh,probst}@imm.dtu.dk

Abstract

Secure program partitioning has been introduced as a language-based technique to allow the distribution of data and computation across mutually untrusted hosts, while at the same time guaranteeing the protection of confidential data. Programs that have been annotated with security types are automatically partitioned by the compiler. The main drawback in this setting is that both the trust hierarchy and the set of hosts are fixed once the program has been partitioned. This paper suggests an enhanced version of the partitioning framework, where the trust relation still remains fixed, but the partitioning compiler becomes *a part of the network* and can recompile applications, thus allowing hosts to enter or leave the framework. We contend that this setting is superior to static partitioning, since it allows redistribution of data and computations. This is especially beneficial if the new host allows data and computations to better fulfill the trust requirements of the users. Erasure Policies ensure that the original host of the redistributed data or computation does not store the data any longer.

Keywords: Secure Program Partitioning, Erasure Policies, Distributed Computation.

1 Introduction

Secure Program Partitioning [ZZNM2001, ZZNM2002] has been introduced as a language-based technique for distributing confidential data and computation across a distributed system of mutually untrusted hosts. The program to be distributed is annotated with security types that constrain permissible information flow. The resulting confidentiality and integrity policies are used to guide the partitioning across the network. The resulting communicating sub-programs not only implement the original program, but at the same time satisfy all security requirements of principals, including trust relations to other principals as well as hosts. The results reported in [ZZNM2001, ZZNM2002] with respect to the performance of the distributed code suggest that this is a feasible way to obtain secure distributed computation.

The main drawback in Secure Program Partitioning (SPP) is that the framework contains two fixed components—the *trust relation* and the *hosts in the network*. While we contend that a static trust relation essentially is necessary to ensure system integrity, the second restriction not only seems to be superfluous, but also hinders the system from moving data or computations to newly available hosts that might allow a better fulfillment of principal's requirements. Thus, by allowing the partitioning to be adjusted after a host joins or leaves the network, the overall trust of the principals in the partitioning and thus in the security of the distributed system can be increased.

After data and computations have been redistributed in the enhanced network, in an ideal world hosts should not store any references to items that have been moved to another host. The recently introduced mechanism of Erasure Policies [CM2005] allows to design systems where programmers annotate their data with policies that describe exactly this kind of behavior. Erasure Policies state explicit erasure and declassification requirements.

The contributions of this paper are:

- We extend the framework for Secure Program Partitioning to allow hosts to enter and (under certain conditions) leave the network.

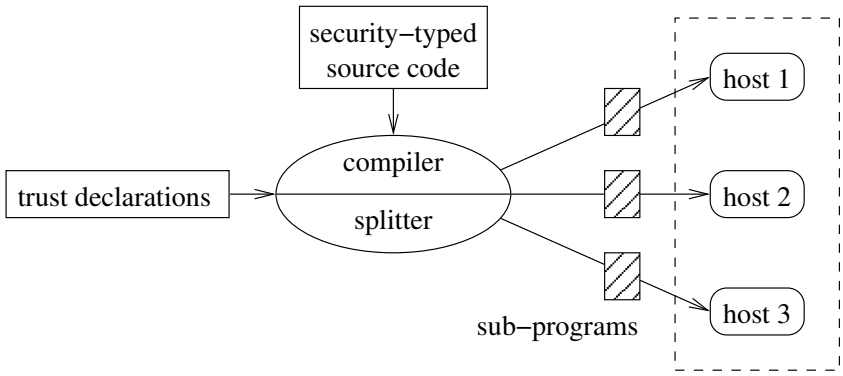


Figure 1: Structure of the Secure Program Partitioning Framework as introduced by [ZZNM2001, ZZNM2002].

- By adding Erasure Policies to SPP, the extended system assures that information is erased or made inaccessible after a node has left the network.

The rest of this paper is structured as follows. Section 2 gives an overview of related work, including Secure Program Partitioning and Erasure Policies. This is followed in Section 3 by the description of our proposed framework for dynamic repartitioning in the case of hosts entering the network. The emphasis here is on making the partitioning an *active* component of the framework, as well as using Erasure Policies to ensure that hosts make inaccessible any data that has been redistributed to another host. Section 5 concludes the paper with an overview of future work.

2 Related Work

This section gives an overview of Secure Program Partitioning [ZZNM2001, ZZNM2002] and Erasure Policies [CM2005], which form the foundation for the framework described in the rest of this paper. This Section largely follows the description in the cited papers.

2.1 Secure Program Partitioning

Information-flow policies have been used for specifying confidentiality and integrity requirements. Their success is mostly based on the ability to specify how information may be used in the system as opposed to which principals may access or modify the data. Security-typed languages [Sch2000, ABHR1999, HR1998, Mye1999, PC2000, SV1998, VSI1996, ZM2001] have been used to implement these policies in programming languages. By annotating data in programs, the programmer explicitly specifies how the flow of information allowed by the language semantics should be constrained. The benefit of these explicit annotations is that programs that violate the restrictions will be rejected either during compilation or during execution. Thus, the program itself does not have to be trusted—instead, only the reused components (compiler and run-time system) must be trusted.

The benefit of SPP is that participants do not need to fully trust each others hosts to enable the distributed execution of a program dealing with data of the principals. As Figure 1 depicts, in the original framework introduced by [ZZNM2001], the compiler receives two inputs—the program source code, which uses a security-typed language, and the trust declarations of all participants. These declarations state each principal's trust in hosts and other principals. They are used to guide the compilation and splitting of the security-typed program into sub-programs that are executed on (some of) the hosts in the network. By communicating, these sub-programs perform the same computation as the original program, however, the splitter ensures that all trust and security policies are fulfilled. As the authors state in [ZZNM2001], the splitter ensures that if a host h is subverted, only the confidentiality or integrity of data owned by principals that trust h is threatened.

The main component in the programming model used in SPP is the *principal*, who can express confidentiality or integrity concerns with respect to data. Principals can be named in information-flow

```
1     public class OTExample {
2       int{Alice:; ?:Alice} m1;
3       int{Alice:; ?:Alice} m2;
4       boolean{Alice:; ?:Alice} isAccessed;
5
6       int{Bob:} transfer{?:Alice} (int{Bob:} n)
7       where authority(Alice) {
8         int tmp1 = m1;
9         int tmp2 = m2;
10        if(!isAccessed) {
11          isAccessed = true;
12          if(endorse(n, {?:Alice}) == 1)
13            return declassify(tmp1, {Bob:});
14          else
15            return declassify(tmp2, {Bob:});
16        }
17        else return 0;
18      }
19    }
```

Figure 2: The source code for the Oblivious Transfer example taken from [ZZNM2002], based on the Oblivious Transfer Problem [Rab1981]. Alice has two values (stored in fields *m1* and *m2*), exactly one of which Bob is allowed to learn. However, Bob does not want Alice to know, which value he has requested. Fields and methods have been annotated with labels to specify confidentiality and integrity requirements of the principals Alice and Bob.

policies and also define the authority possessed by the program being executed. Security labels [ML2000] express confidentiality policies on data. A label $l_1 = \{o : r_1, r_2, \cdots, r_n\}$ means that data labeled with l_1 is owned by a principal o and that o permits readers $r_1, \cdots, r_n$ (and o) to read the data. Data can also have multiple owners, each expressing its concerns with respect to the data. E.g., $l_2 = \{o_1 : r_1, r_2; o_2 : r_2, r_3;\}$ expresses that owner o_1 allows readers r_1 and r_2 to read data labeled with l_2, and owner o_2 does so for readers r_2 and r_3. Of course each annotation must be obeyed by the system, that is only r_2 will be allowed to access data labeled with l_2. Additionally, labels may specify integrity. $l_3 = \{? : p_1, \cdots, p_n\}$ specifies which principals trust the data labeled with l_3.

To allow the splitter to partition the program across the hosts of the target network, it must know the trust relationship between the principals and the hosts. This information is specified by two components per principal and host. The *confidentiality label* $\mathtt{C}_{(h,A)}$ specifies the upper bound on the confidentiality of information that A allows to be sent to host h. Accordingly, the *integrity* label specifies whether the principal trusts information received from host h. For the example shown in Figure 2, the principals Alice, Bob, and Charlie specify the following confidentiality and integrity labels for the four hosts in the network. Alice does trust her own host A as well as the hosts T and S and also believes in the integrity of data received from these hosts. Bob trusts his own host B as well as hosts T and S, but only believes in the integrity of data received from his own host. Finally, Charlie trusts the host T. This results in the following sets `C` and `I`:

$$\begin{array}{ll}
\mathtt{C}_{(A,\text{Alice})} = \{\text{Alice} :\} & \mathtt{I}_{(A,\text{Alice})} = \{? : \text{Alice}\} \\
\mathtt{C}_{(B,\text{Bob})} = \{\text{Bob} :\} & \mathtt{I}_{(B,\text{Bob})} = \{? : \text{Bob}\} \\
\mathtt{C}_{(T,\text{Alice})} = \{\text{Alice} :\} & \mathtt{I}_{(T,\text{Alice})} = \{? : \text{Alice}\} \\
\mathtt{C}_{(T,\text{Bob})} = \{\text{Bob} :\} & \mathtt{C}_{(T,\text{Charlie})} = \{\text{Charlie} :\} \\
\mathtt{C}_{(S,\text{Alice})} = \{\text{Alice} :\} & \mathtt{C}_{(S,\text{Bob})} = \{\text{Bob} :\}
\end{array}$$

For a host h, the sets $\mathtt{C}_h$ and $\mathtt{I}_h$ are computed as the union of the according sets $\mathtt{C}_{(h,\_)}$ and $\mathtt{I}_{(h,\_)}$, respectively. Along the same lines one can derive confidentiality and integrity labels for fields and expressions in a security-typed language. Generally, the sets `C` and `I` are unified into a single label `L` and the two functions C and I are used to extract each of the subsets.

For the splitting of an application onto the hosts available in a network, the authors in [ZZNM2002]

specify constraints for fields and statements in the application. For a field f, the generated constraints are

$$C(\mathtt{L}_f) \sqsubseteq \mathtt{C}_h \text{ and } \mathtt{I}_h \sqsubseteq \mathtt{I}(\mathtt{L}_f)$$

The constraints for a statement S result from performing a simple definition-use analysis on all data used or defined in the statement. Let $U(S)$ and $D(S)$ be the set of values used and locations defined by S. Then

$$C\left(\sqcup_{v \in U(S)} L_v\right) \sqsubseteq \mathtt{C}_h \text{ and } \mathtt{I}_h \sqsubseteq I\left(\sqcap_{l \in D(S)} L_l\right)$$

For a list of additional constraints, including those regarding the program counter, refer to [ZZNM2002]. The essential property for this work is that in principal they all have the above form.

2.2 Erasure Policies

Erasure Policies (EPs) as introduced by Chong and Myers [CM2005] impose strong end-to-end requirements to enforce that information is either erased or made less accessible. They are based on a lattice of security levels. The simplest kind of a policy is a *label* l that limits how the labeled data may be used. In the setting of SPP this would be a set of confidentiality and integrity requirements. Additionally, *erasure policies* have the form $l_1 \overset{c}{\nearrow} l_2$, where l_1, l_2 are policies and c is a condition specifying that l_1 must be enforced on the labeled data, and once condition c is fulfilled, l_2 must be enforced as well, independent of the future evaluation of c. Finally there are *declassification policies* $l_1 \overset{c}{\rightsquigarrow} l_2$ stating that l_1 must be enforced on the labeled data, but once condition c is fulfilled, the data may be declassified. From thereon policy l_2 must be enforced, again independent of the future evaluation of c.

3 Dynamic Repartitioning

This section introduces our extension to the Secure Program Partitioning framework as introduced in Section 2.

Dynamic Repartitioning essentially shares all the properties of SPP as described above. The main achievement is that hosts may join or leave the network after an initial partitioning of the application has been found. This initial partitioning is important, since it ensures that the program can also be partitioned across a bigger set of hosts. We define the set of hosts used for constructing the initial partitioning as H_{init}. Hosts in this set are never allowed to leave the network, since they are needed to ensure the existence of a partitioning. Hosts that join the network later are part of the set H_{join}. Hosts in this set may freely join or leave the network since they are not needed for the initial partitioning. In contrast, in the case of hosts from the set H_{init} *leaving* the network no guarantee can be given that the source code can be partitioned across the remaining hosts.

The effect of having an additional host in the network is that some of the constraints introduced in Section 2 may be resolved to an element that is smaller with respect to $\sqsubseteq$ than the original solution. This is true for all constraints where the confidentiality (integrity) sets for the new host n ($\mathtt{C}_n$ and $\mathtt{I}_n$) are smaller (bigger) than those of the originally chosen host h. Of course there is no *guarantee* that the new host will be chosen for data or computations, just like the host S in the Oblivious Transfer example.

3.1 Enforcing Erasure of Data

If data has been repartitioned to another host as result of a host joining the network, principals will want to be assured that their data has been erased or made less accessible on the original host. Using the framework of Erasure Policies as described in Section 2, this effect can be ensured automatically *without* additional annotations by the principals.

To do so, we introduce two conditions *rem* and *loc* that model the event of data being rescheduled to a remote host and data being available locally. We assume that during the redistribution of an application no data ever is reused. As the result of a host n joining the network, data might be repartitioned from host h to the new host n. If n leaves the network again, depending on the other hosts that joined or left the network, the data or parts of it could be repartitioned to be stored on h again. In this scenario, a

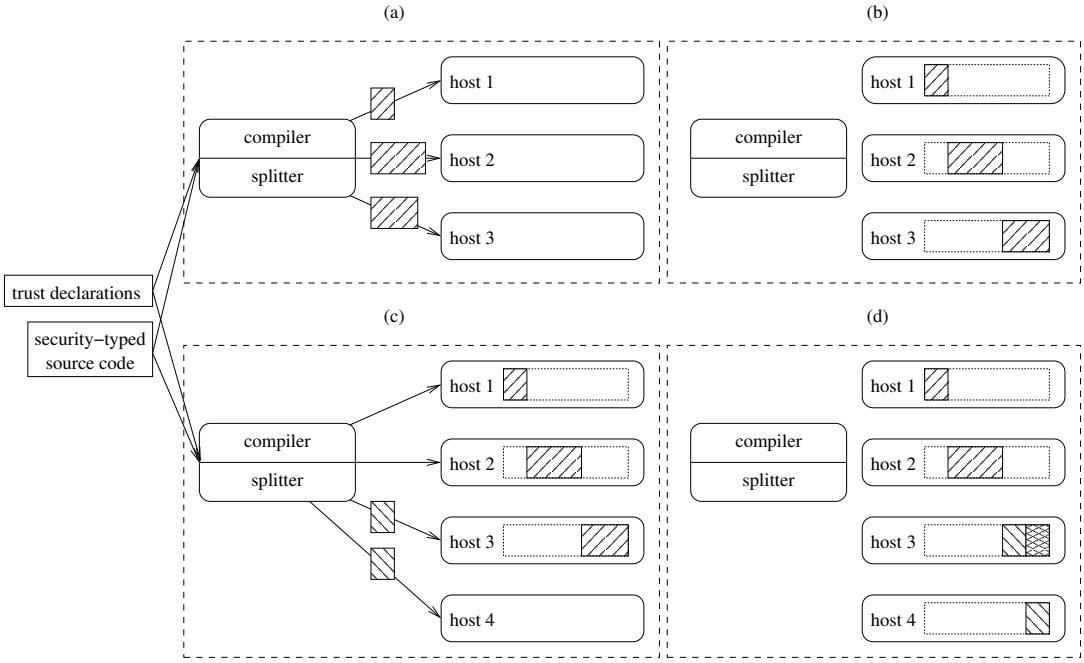


Figure 3: Secure Dynamic Program Repartitioning. The main change in contrast to SPP is that the Compiler/Splitter component is part of the network (a). The initial distribution of application code is done just like in the original framework (b). Once a new node enters the network (c), the Compiler/Splitter tries to find a new distribution that better fulfills the trust requirements specified by the principals. If the new host enables such a repartitioning, the new partitions are distributed to the hosts (d). Otherwise, the system remains unchanged. In a last step, hosts must erase the data that has been partitioned to another host (or make the data inaccessible). In (d) this is the hatched area of host 3.

fresh copy of the data would be created and stored on h. To make sure that data stored on hosts are not reused once they have been partitioned on another host, we use an erasure policy using the *rem* condition. The policy for some data d that is partitioned on a host h and in the original framework would have label l then becomes $l^{rem}\nearrow\top$. This ensures that after the d has been partitioned onto another host it can no longer be accessed on host h.

4 An Example

This section returns to the Oblivious Transfer example presented in Figure 2. Using the annotations to the source code and the confidentiality and integrity labels specified by the principals, the code can be partitioned on three hosts A, B, and T. The labels for the four available hosts are

$$\begin{array}{ll} \mathtt{C}_A = \{\text{Alice :}\} & \mathtt{I}_A = \{\text{? : Alice}\} \\ \mathtt{C}_B = \{\text{Bob :}\} & \mathtt{I}_B = \{\text{? : Bob}\} \\ \mathtt{C}_T = \{\text{Alice :, Bob, Charlie :}\} & \mathtt{I}_T = \{\text{? : Alice}\} \\ \mathtt{C}_S = \{\text{Alice :, Bob :}\} & \mathtt{I}_S = \{\text{? :}\} \end{array}$$

Figure 4 shows how the application again and how to partition it on three of the hosts, A, B, and T.

Assume that later on a host N joins the network for which the principals have specified $\mathtt{C}_N = \{\text{Alice :, Bob}\}$ and $\mathtt{I}_N = \{\text{? : Alice}\}$. Repartitioning data and computations that in Figure 4 have been partitioned to T will allow to fulfill the requirements of both Alice and Bob better than in the original partition, since the set $\mathtt{C}_T$ is larger than necessary to fulfill all constraints generated for the program (since the labels in the program do not reason about Charlie).

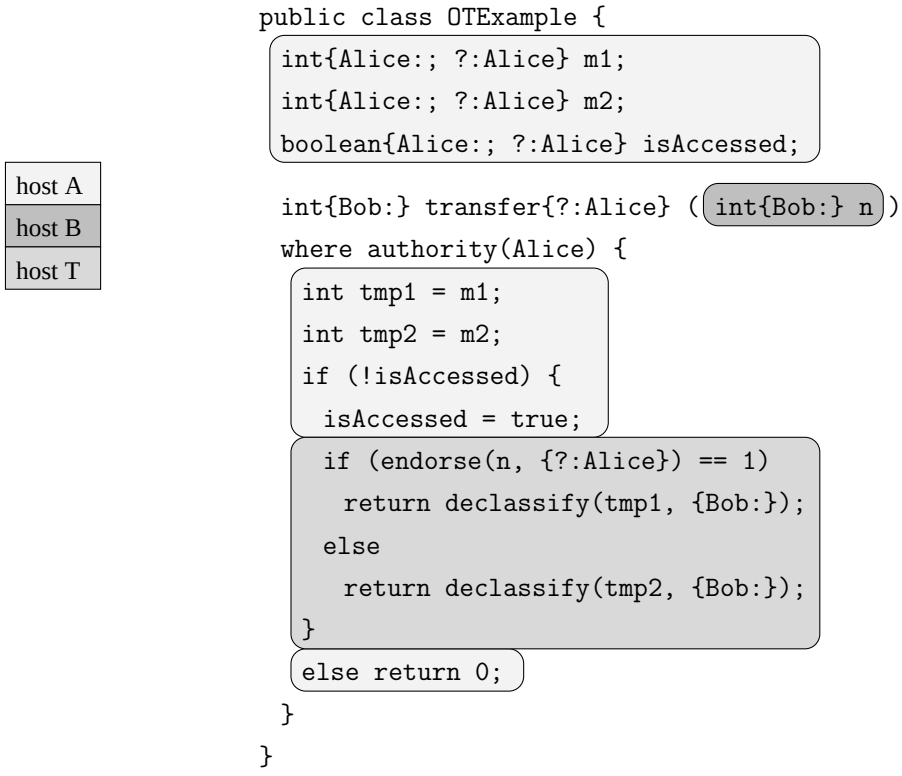


Figure 4: The Oblivious Transfer example including the partitioning of code pieces across three hosts according to [ZZNM2002].

5 Conclusions and Future Work

We have introduced a framework for Secure Dynamic Program Repartitioning. The work presented in this paper builds on prior work on Secure Program Partitioning, a framework working with a fixed sets of hosts. Our framework inherits all properties from SPP, but additionally allows hosts to join and (under certain conditions) leave the network, possibly causing a re-partitioning of the application, and uses Erasure Policies to ensure that data is made inaccessible on hosts that no longer store the data after repartitioning has taken place.

We see two limitations for our approach. On the on hand, for the time being like SPP we only investigate single-threaded programs. Repartitioning of applications can only occur when the program is not being executed. Thus, no additional communication primitives are needed on top of those introduced by [ZZNM2002]. On the other hand, theoretically the system does not know the confidentiality and integrity sets of principals for hosts joining the network. This could be avoided by having a list of hosts potentially joining the network and principals specifying policies for each of them or by specifying certain properties of a host like operating system or owner to deduct policies for each user based on preferences. While these approaches are inherently inelegant, we currently see no real alternative.

We are currently working on the formalization of the extensions described here. It will be especially interesting to model fully dynamic networks based on the available confidentiality and integrity, that is allow hosts to leave the network again based on the trust of principals into the hosts remaining in the network. In order to ensure that applications can still be partitioned in this case, the system will need to ensure that (instead of forcing the initial set of hosts to remain in the network) the hosts remaining in the network provide the same confidentiality and integrity as the initial set. Another interesting problem is to change the token model introduced by [ZZNM2002] so that repartitioning can be guaranteed to be safe also while the application is executed.

References

[ABHR1999] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 147–160, New York, NY, USA, 1999. ACM Press.

[CM2005] Stephen Chong and Andrew C. Myers. Language-Based Information Erasure. In *CSFW '05: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 241–254, Washington, DC, USA, 2005. IEEE Computer Society.

[HR1998] Nevin Heintze and Jon G. Riecke. The SLam calculus: Programming with security and integrity. In *Conference Record of POPL '98: The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 365–377, San Diego, California, 19–21 January 1998.

[ML2000] Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4), October 2000.

[Mye1999] Andrew C. Myers. JFlow: practical mostly-static information flow control. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 228–241, New York, NY, USA, 1999. ACM Press.

[PC2000] François Pottier and Sylvain Conchon. Information flow inference for free. *ACM SIGPLAN Notices*, 35(9):46–57, September 2000.

[Rab1981] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[Sch2000] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, February 2000.

[SV1998] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98)*, pages 355–364, New York, January 1998. Association for Computing Machinery.

[VSI1996] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, December 1996.

[ZM2001] Steve Zdancewic and Andrew C. Myers. Secure information flow and CPS. *Lecture Notes in Computer Science*, 2028:46–??, 2001.

[ZZNM2001] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Untrusted hosts and confidentiality: Secure program partitioning. In Greg Ganger, editor, *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP-01)*, volume 35, 5 of *ACM SIGOPS Operating Systems Review*, pages 1–14, New York, October 21–24 2001. ACM Press.

[ZZNM2002] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *ACM Transactions on Computer Systems*, 20(3):283–328, August 2002.

A Vulnerability Taxonomy Methodology applied to Web Services

Chris Vanden Berghe[1,2], James Riordan[1], Frank Piessens[2]

[1] IBM Zurich Research Laboratory
{vbc,rij}@zurich.ibm.com

[2] Katholieke Universiteit Leuven
{chrisvdb,frank}@cs.kuleuven.be

Abstract

We present a methodology for taxonomizing vulnerabilities based on the likelihood that they will be present in a certain system. It attempts to capture and formalize the intuition that allows security professionals to make predictions about likely security problems. The method exploits the realization that the vulnerabilities present in a system are related to the set of properties that define the system. By modeling it using a selection of relevant properties and correlating this with the body of knowledge on historic vulnerabilities and the systems in which they lived, we obtain a heuristic of the likelihood that these vulnerabilities will reappear in a new system. The predictive nature of this methodology serves as an early warning for systems before they are widely deployed. As an example we apply our methodology to Web Services, thereby providing a tool to focus efforts in securing Web Services.

1 Introduction

Service-oriented architectures are a major evolutionary step in the creation of distributed systems. J2EE [25] and .NET [18] have embraced this trend and provide tools and support for Web Services (WS), which are practical implementations of the principles of a Service-Oriented Architecture (SOA). The architecture introduces a service layer around software components and can be regarded as the logical next level of abstraction after the componentization of software. Such an abstraction allows for the creation of rich, highly-distributed applications by enabling dynamic discovery and loose-coupling of heterogeneous components that span organizational boundaries.

The flexibility of these highly-distributed applications enables new and interesting possibilities. At the same time it creates numerous security challenges that need to be addressed for the ultimate success of the technology. This need was quickly recognized and has resulted in numerous standards addressing various aspects of SOA security. Examples include the WS-Security family of standards [22], SAML [20], XACML [19] and the Liberty Alliance specifications [14]. Each of these specifications targets some (not necessarily disjoint) collection of the security needs in the service layer.

By contrast, security concerns regarding implementations and operation have received far less attention. This disparity is largely rooted in the youth of WS rather than in the relative importance of the two classes of security failure. There are simply too few deployed and active instances of WS to generate an adequate body of knowledge about real-world security failures and the vulnerabilities behind them.

Our methodology is predicated upon two principal assumptions: The first is that one can usefully think of security vulnerabilities as being present in the properties of a system rather than in the system itself. The second is that we will not see fundamentally new vulnerabilities in SOA but rather variations of those already present in existing systems. Based on these assumptions, we develop a general-purpose methodology for generating predictive vulnerability taxonomies and apply it to WS. Some vulnerability types are more likely to arise than others in such architectures, depending upon the properties of the individual components.

As a testcase example, we apply our methodology to WS which illustrates how the resulting taxonomy can used to reason about practical security in WS. It indicates the degree to which the WS components are prone to several classes of vulnerability, thereby providing insight into an optimal use of finite resources for securing WS.

The paper is structured as follows. In Section 2 we present the aspects of SOA and WS that are the most important ones to our work. In Section 3 we give a brief overview of practical security, which is our security layer of focus. Section 4 provides some insights into taxonomies in general and vulnerability taxonomies in particular. Section 5 lays out and explains our methodology for creating vulnerability taxonomies, using the example of WS. Section 6 details the actual results of this application to WS. Section 7 presents related work, and Section 8 concludes with a summary of our findings.

2 Web Services

In this section we present an overview of security-relevant aspects of WS. We start by outlining the principles of a Service-Oriented Architecture (SOA). This is followed by a description of WS and its relation to SOA. Finally we discuss the implementation-related aspects of WS.

2.1 Service-oriented architecture

SOA is a software development methodology aimed at improving the manageability of today's increasingly complex systems. A good understanding of the methodology's principles, and the reasoning behind them, is essential for understanding WS security, because WS is a practical implementation of the more general SOA concept.

Object-oriented and component-based software development methodologies have successively increased our ability to develop and manage complex applications. They are based upon the abstraction of *components*, which increases re-usability of software functionality. While this notion is very powerful, interfaces between components tend to be rather tightly coupled and inflexible. It therefore does not scale to distributed applications spanning organizational boundaries, as they require that developers understand and control each component in the application. The integration of components coming from a wide variety of organizations is therefore a daunting task.

SOA can be seen as an extension of the ideas of component-based software development aimed at increasing the flexibility of application integration. It introduces the concept of a service layer, which provides a framework to address the complexity of integration in highly-distributed heterogeneous systems. Conceptually, SOA represents a model in which loosely coupled pieces of application functionality are published, located, consumed, and combined over a network. Such exposed functionality is called a service. Services adhere to the following principles [17]:

- They deliver some *self-contained* and *composable* functionality that is advertised in terms of the task performed by the service. The focus is on *what* is performed instead of *how* it is performed. In addition, it should be possible to use the service as part of a larger application.
- They are *loosely-coupled* and can be *discovered and invoked dynamically*. Traditional applications depend on tight interconnections of all subsidiary components, therefore changing existing systems is exceedingly difficult. Loosely coupled systems require less coordination and allow for more flexible reconfiguration, thereby enabling the use of services across organizational boundaries.
- They have *well-defined interfaces* and *coarse-grained interactions*. Coupling systems is generally difficult, so it is important to keep interfaces simple and the number of interactions minimal. This keeps the dependencies between services manageable. In addition, it facilitates more thorough testing of the service and makes the interactions easier to understand.
- They are *network-addressable* and *location-transparent*. The ability to invoke the service over a network is key to the concept of SOA. It allows applications to use the services best suited to their needs, independently of their location.
- They are *interoperable*. This is a necessary characteristic of heterogeneous systems that communicate and cooperate. Towards this end, the service consumer and provider must agree on a mediating protocol and data-format on to which they map their platform-specific characteristics.

2.2 Web Services

WS are practical implementations of SOA principles. The term itself is not very well defined and is used both in a conceptual and in a technical sense. Conceptually, WS are XML-based SOA that use standard

Internet transport protocols[1]. In the technical sense, WS refers to specific collections of standards, tools and practices to implement the WS concept. Different styles of WS exist such as SOAP Web Services [28], REST Web Services [10] and XML-RPC [27].

SOAP WS are the most common form. Specific to SOAP WS is the use of WSDL [29] as the service descriptor, SOAP [28] as the messaging protocol and UDDI [21] as the directory protocol. WS-I [31], an industry organization created with the mission to promote interoperability among WS, recommends this style of WS (with some additional restrictions) for interoperability reasons. Many of the higher-layer WS protocols, addressing issues such as workflow, business processes and security, are built atop these core specifications. The main advantage of this particular flavor of WS is its wide adoption and availability of supporting tools. The complexity of the protocol stack is its major disadvantage.

REST Web Services and XML-RPC provide a less complex alternative to SOAP Web Services, but are also less widespread. REST Web Services adhere to the concepts of the Representational State Transfer architectural style [10]. In this model, services are seen as resources addressable by URIs. Accessing and managing these services is done exclusively through the HTTP verbs GET, POST, DELETE and PUT, where each one has well-specified semantics. Most of the data transmitted is in the form of XML.

REST Web Services are very light-weight by design because they only use URIs, HTTP and standard resource representations, such as XML, jpeg, etc. They possess additional desirable properties such as improved performance due to better caching opportunities. XML-RPC, although often considered as another light-weight protocol for WS, does not adhere to all of the aforementioned properties of a SOA. For example, XML-RPC does not publish application functionality with a service abstraction, but is merely a convenient way to open functionality to the Web.

In this paper we focus on SOAP WS because of their wide acceptance. Many of the conclusions also hold for other WS flavors, as the fundamental concepts are largely shared.

2.3 Web Services implementation

There is no universally accepted set of tools for developing WS or platform for their deployment; this would indeed defeat much of the purpose of WS. Instead, toolkits exist for many platforms. Nevertheless, it is safe to say that most WS will be developed in higher-level languages and deployed in managed-execution environments. Two concrete examples of WS implementations are J2EE [25] application servers and the .NET framework [18]. They are already widespread and provide good tool support for SOAP Web Services.

Development toolkits for J2EE and .NET provide wizard-based tools that take care of deploying the selected functionality as a WS, and even publish the WS with one or more directory services. The application servers handle message parsing, cryptographic transformations, data-type mapping and transparent dispatching. Although the specifics differ, both platforms abstract the WS details away from the applications.

Subsequent discussion of WS implementation-related properties is limited to properties shared by common implementations, as opposed to aspects of a specific implementation (thus rendering the results more widely applicable).

3 Practical Security

Our work focuses on the layer of security that addresses real-world vulnerabilities caused by a system's implementation, deployment and management. We refer to this layer as "practical security". It occupies a position complementary, yet related to, security specifications (such as those of WS). Specifications determine the way in which a system *should* behave, whereas the implementation determines the way in which it *does* behave.

Although we are concerned with such concrete and specific vulnerabilities, it makes sense to speak of them as abstractions. Any reader of a security vulnerability announcement list, such as Bugtraq [11], will be eventually struck with a certain sense of *déjà-vu*. On an average day several new vulnerabilities are published, but only very seldomly something fundamentally new is discovered. Instead, they are variations that differ only in the specifics, but not in the *implicitly-adjudged underlying reason* why the system is vulnerable.

[1]The term "transport protocol" now includes higher level protocols such as HTTP and BEEP

The foundation of our subsequent analysis is that one can think of vulnerabilities as being manifest in the implicitly-adjudged underlying reason for their existence rather than in the systems themselves. We therefore propose a model that describes these reasons in terms of a collection of properties. This is analogous to the epidemiological practice of ascribing a disease to a population rather than an individual. This epidemiology applies to our work in three distinct manners.

- Code analysis and testing resources are limited by both time and cost. As such, they should be utilized towards maximum effect. Knowledge of what sorts of problem are likely to occur in which places is thus quite valuable.
- Properties should be seen as characteristics of the complete process of design, development, use and maintenance of the system. Determination of the property sets is both important and potentially difficult. The properties that influence the security of a system are not limited to technical or implementation-oriented properties, such as which development tools are used. They also include nontechnical issues such as security-awareness of the users of the system. These properties should ultimately describe the system as completely as possible.
- The influence-map from properties to vulnerabilities is not necessarily injective: indeed, a group of properties may be required to influence the presence of a vulnerability. For example, buffer-overflows are commonly attributed to the use of a low-level programming language, whereas they actually result from the combination of low-level programming languages, lack of testing, poor development processes, et cetera. Correspondingly, we do not assume strict causality but merely correlation: the C programming language does not *create* buffer-overflows itself, it is just particularly well-suited towards creating them and, therefore, programs written in C should be checked for buffer overflows.

To predict vulnerabilities of a new technology, we assume that their recurrent nature will continue with minor variations. For example, we anticipate that code using the standard selection and predicate language for data in the hierarchical model of XML (XPath) [30] will present problems similar to those in code using the standard selection and predicate language for data in relational databases (SQL): XPath-injection is very likely to be a problem in WS.

Based on these considerations regarding practical security, we developed a methodology for predicting vulnerabilities in systems through taxonomization. Before detailing this methodology in Section 5, we introduce general concepts related to vulnerability taxonomies.

4 Vulnerability taxonomies

Although classification of items and events is a commonplace activity, there is no universal agreement on the semantics of terms surrounding classification and taxonomization. Therefore we commence by defining these concepts and by providing some general observations about classifications. We then discuss vulnerability taxonomies in general and vulnerability taxonomies for WS in particular.

4.1 Classifications and taxonomies

Classification refers to both the operation and its result. Marradi [16] defines the operation as being one of three possibilities:

- an intellectual operation, whereby the extension of a concept at a given level of generality is subdivided into several narrower extensions corresponding to as many concepts at a lower level of generality;
- an operation whereby the objects or events in a given set are divided into two or more subsets according to the perceived similarities of one or several properties; or
- an operation whereby objects or events are assigned to classes or types that have been previously defined.

The first is an *a priori classification*, in which one starts from a concept and then further refines it into subconcepts. In this definition "the extension of a concept" refers to its most direct or specific meaning. The second is an *a posteriori classification*, in which one starts from the actual objects or events and proceeds by grouping them according to properties, or taxonomic characters. Each of these two types of classification produces a hierarchy of classes. The final possibility is the actual process of assigning the objects or events to these pre-defined classes.

A *taxonomy* is a "classification, including bases, principles, procedures and rules". This definition, introduced by Simpson in his seminal work on the classification of animals [24], is widely accepted, and is also used in other vulnerability classifications [13]. The definition suggests that a taxonomy is more than a classification, in the sense that it also describes the principles according to which the classification is done and the procedures to be followed in order to classify new objects. A resulting class of a taxonomy is called a taxon (Pl. taxa).

The goal of classification is to turn chaos into regularity. Systematization is necessary to handle the large amount of information humans are confronted with. The first known attempt to perform systematic classification dates back to Aristotle (322-287 BC), who worked on the first classification of animal species. For more than two millennia systematic classification was the exclusive domain of biological sciences. This changed at the end of the 19th century, when classifications started to appear in diverse branches of science [13].

The key to building a good taxonomy is the choice of the taxonomic characters according to which the objects or events will be classified. These taxonomic characters should be relevant and readily and objectively observable in the objects to be classified [24, 13]. Lough [15] gives an overview of desirable properties when building a taxonomy. Some of the most widely accepted are: deterministic, exhaustive, mutually exclusive, objective, and useful. Some of these properties are partially contradictory, e.g., a taxonomy can be made exhaustive by adding the catch-all category *other*, but doing so generally renders it less useful.

An important, yet often overlooked, issue is that there is no such thing as the ultimate taxonomy. Rather, the form of the taxonomy should be adapted to the intended usage. In practice this means that the intended usage as well as the *scope* and *viewpoint* of the taxonomy should be stated explicitly. We define scope as the part of the universe to be included in the taxonomy. This determines what should be classified. An example of scope is *all flowers indigenous to a certain country*. Equally important is the viewpoint, which determines how one looks at the things within the scope and consequently which properties will be relevant. For example, the viewpoints of a botanist and a florist yield totally different relevant classification properties in a taxonomy of flowers.

4.2 Vulnerability taxonomies

The perceived similarities between vulnerabilities published on security mailing lists such as Bugtraq [11] are an indication for the utility of taxonomizing vulnerabilities. Although ideally every report features a distinct vulnerability instance, one can naturally abstract them into classes of vulnerability instances with similar properties. This abstraction facilitates understanding of the origins of the vulnerabilities, and allows for the development of avoidance and mitigation methods for such vulnerabilities. The key challenge in the production of a good taxonomy is the selection of properties that optimally contribute to this understanding.

As discussed, these properties depend on the scope and the viewpoint. A vulnerability taxonomy implies an implicit limitation of the scope, namely to vulnerabilities. Some examples of scopes that are further limited are:

- Vulnerabilities in UNIX operating systems [3, 6]
- Cryptographic vulnerabilities [26, 2]
- Application-level vulnerabilities

By setting the scope and the viewpoint of the taxonomy, one also determines which vulnerabilities to classify as well as how to look at them. For instance, the viewpoint of a developer, a system administrator or an attacker on a particular vulnerability are quite different.

Several vulnerability taxonomies have been proposed. Those proposed by [1, 5] had the scope of "vulnerabilities in operating systems". Subsequent taxonomies differed in scope and viewpoint [3, 4, 6, 9, 13]. A shortcoming of the aforementioned taxonomies is that they do not make their intended usage explicit, but instead aim at being general-purpose.

There will probably never be a vulnerability taxonomy as universally accepted as Linnæus's original classification is in biology for several reasons. The scope of vulnerabilities is highly dynamic, since the modes of daily computer usage are evolving rapidly. In addition, the computer vulnerabilities themselves are inherently difficult to describe, and most often negative terminology is used in describing them. This is related to the fact that they exist where conceptual models break down. Finally, the way in which one describes a vulnerability is strongly tied to the viewpoint.

There are at least three realizations common to vulnerability taxonomies:

- Generating a good taxonomy is difficult (see discussion in Section 4.1).
- A taxonomy depends not only on the vulnerabilities themselves but also on the viewpoint of the taxonomy creator; this viewpoint is generally determined by the intended use of the taxonomy.
- Related vulnerabilities often manifest themselves in packages that share some common property.

Generating a vulnerability taxonomy with the scope of WS incurs the additional difficulty that the WS themselves are not yet fully deployed. However, it is our central assumption that there is a sufficient body of knowledge regarding related systems to allow accurate prediction of the classes of vulnerability that are most likely to be problematic in WS. The key benefit of such a predictive taxonomy is that it can guide the investment of limited resources in securing WS.

5 Proposed Methodology

Our methodology produces predictive vulnerability taxonomies based on correlation of properties of system components and their adjudged influence on a selection of historical vulnerabilities. It is predicated on the two principal assumptions introduced in Section 3. Namely that it is meaningful to think of vulnerabilities as being present in the properties of the system in addition to being present in a particular version of a particular program and that vulnerabilities in new systems are mostly variants, in an abstract sense, of vulnerabilities existing in older systems.

We begin with a discussion of correlative properties and the process by which they are selected. Two inputs drive this process. The first is an architectural refinement of the system being analyzed into functional components. These components need to be of sufficiently fine granularity to express possible attack scenarios. They need also be uniform with respect to the selected properties for each component. The second is a representative collection of vulnerabilities, associated with the selected properties by the adjudged influence of the properties on the vulnerabilities. Naturally, the selection of properties, refinement of architecture, and collecting of vulnerabilities are iterative interdependent steps in the methodology. These inputs allow for the correlation of components with vulnerabilities through common properties. The result is a table detailing the likelihood of a vulnerability variant being present in a given component.

In the remainder of this section we detail each of the general steps of the methodology and illustrate them using the example of WS. Details that go beyond clarification of the methodology are provided in Section 6.

5.1 Architecture and selection of properties

The predictive capabilities of the produced taxonomy are determined by the accuracy with which the selected properties describe the system with respect to vulnerabilities. The selection of the properties is therefore paramount. While there is no predefined and exact means of selecting properties, we target those with influence on the security of the system. The task is facilitated by the fact that we merely need coverage rather than a perfect selection. Selection of irrelevant properties is not a problem, as they will have no effect in the correlation. Selection of strongly correlated properties is more problematic as it may lead to multicollinearity, resulting in an unproportional influence of these properties. In this work we endeavor to avoid the problem through utilization of the basis of this work: we attempt to use our security experience to select properties that are not causally related. While this heuristic does not have statistical rigor, it does have the advantage of being tractable.

A complex system is comprised of different components with diverse properties. In order to capture this with our methodology, it is necessary to refine the analyzed system into components for which the properties are well defined (e.g., true, false or irrelevant, rather than dependent upon the subcomponent). In a complex system, different components are prone to different vulnerabilities. The refinement offers the advantage that we capture not only the sorts of problems the complex system is apt to have, but also where within the system they are most likely to appear.

It is also useful to list pairs of components as connections, which indicate the composition or simultaneous presence of two components. These connections have security-relevant properties themselves, since properties on both the physical connection (e.g., use of encryption) and the logical connection (e.g., trust relationship) can render the system more or less secure.

We clarify the selection of properties using the example of WS. WS is a complex, highly-distributed architecture, but as an input into the methodology, a simple refinement suffices. As depicted in Figure 1,

our WS refinement is comprised of a service requester, a directory service, a service provider, backend systems and the connections between these components. Intermediaries are not explicitly modeled, but are seen as a service provider acting as a service requester.

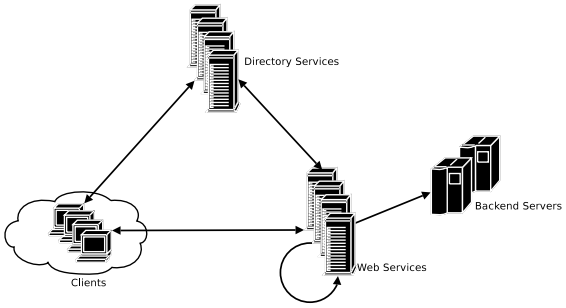


Figure 1: WS Architectural Refinement

The next step is the selection of the most relevant properties describing these components. We divide them into two categories: paradigm-related and implementation-related.

The first category contains properties related to the underlying concepts of WS and SOA. Examples of the properties in this category include support for dynamic discovery and binding of services, XML-based messaging, and cross-platform support.

The second category involves properties related to actual implementations. These properties are equally important but less uniformly applicable and more difficult to find; different implementations have different properties. In order to ensure general applicability of the produced taxonomy, it is important to choose properties common to most implementations. For example, most WS implementations use high-level languages and server application containers, which provide system-level services such as memory and transaction management. As such we consider "use of high-level languages" a shared property.

The result of this property selection step is a matrix [properties $\times$ architecture]. The rows contain the different properties and the columns contain the different components of the architecture. For every property and component there is a cell describing the degree to which the property is present in the component. The different values are "the property is present", "the property is not present" or "the opposite property is present".

5.2 Selection and assessment of vulnerabilities

The next major step of our methodology is the selection of vulnerabilities. Recalling the assumption that new vulnerabilities will largely be variants of existing vulnerabilities (as opposed to fundamentally new vulnerabilities), it is necessary to identify a representative base of existing vulnerabilities. With several "new" vulnerabilities discovered on an average day, it is necessary to discriminate. We set the following three criteria: coverage, relevance and availability of information.

Coverage in this context refers to the need for a broad and complete range of vulnerabilities. The goal is to estimate the likelihood that variants of the selected vulnerabilities will appear in the new system. Therefore, it is necessary to ensure that the selection is not too limited. While buffer-overflows account for a major part of the published vulnerabilities, there is no need for proportional representation of this class of vulnerabilities in the selection.

Our second criterion is *relevance*. The vulnerabilities should also be relevant to the security layer of interest. If the layer of interest is practical security for applications, the selected vulnerabilities will be centered around the application-level. Likewise, vulnerabilities from similar systems will have a higher chance of being relevant to the new system. For example, vulnerabilities in CORBA [12] bear greater relevance to DCOM [8] than those of an architecturally dissimilar system.

The final criterion is *availability of information* about the particular vulnerability. The best sources for obtaining vulnerability information are security databases and mailing lists, such as Bugtraq [11]. Nevertheless, the available information is often sparse or obscured, inhibiting understanding of the vulnerability. Hence, it is best to select the vulnerabilities with the most complete information available.

This selection step yields a list of vulnerabilities that needs to be assessed, in order to determine the influence that each property has on each vulnerability. Many properties will not influence the likelihood that a vulnerability appears whereas others will either increase it or decrease it.

This assessment is necessarily partially biased, as our method aims to capture the experience of a security professional. In many cases, the assessment will be trivial as the property is a prerequisite or conversely an inhibitor of the vulnerability. In other cases, there is no relation whatsoever between the property and the vulnerability. In cases where the influence is less clear, different professionals may come to different conclusions. This can be mitigated by having a more fine-grained scoring system and averaging results or a questionnaire resulting in a standardized score.

The result of this vulnerability selection and assessment step is a matrix [vulnerabilities × properties]. Each cell qualitatively describes the influence of the property on the vulnerability. Later, this qualitative description will be mapped onto a quantitative value in order to perform correlation.

In our WS example, we begin with the values "positive", "none" and "negative", meaning respectively more likely, no influence and less likely. For instance, text-based communication, which facilitates understanding and manipulation of the transmitted data, increases the likelihood of input validation vulnerabilities; its influence is therefore positive. Similarly, the use of high-level programming languages and managed computing environments diminishes the likelihood of buffer-overflow vulnerabilities; their influence is therefore negative.

5.3 Correlation of properties and vulnerabilities

The purpose of this step is to compute an estimate of the likelihood that a variant of a vulnerability will be present in the different components of the system, by combining the information resulting from the previous steps; namely the matrices [vulnerabilities × properties] and [properties × architecture].

To obtain a quantitative estimate, we first assign numerical values to the two matrices. In our WS example, for the first matrix, we map the three values ("the property is present", "the property is not present" and "the opposite property is present"), to the values 1, 0 and -1 respectively. Similarly, for the second matrix, the three values ("positive", "none" and "negative") are mapped to 1, 0 and -1.

It is now possible to determine the likelihood that a particular vulnerability is present in a particular component. We do so by means of a simple linear composition with equal weights, obtained by the multiplication of the two matrices [vulnerabilities × properties] × [properties × architecture]. This results in the desired description matrix [vulnerabilities × architecture].

The values in this matrix are not mathematical probabilities (e.g., they are not bounded) but rather indications of likelihood, with higher values indicating higher likelihood. For simple components, the values indicate the most relevant vulnerabilities for each component. Recall that, the simultaneous presence of components are modeled as connections between components. In this case, the meaning of the values is different: they indicate the likelihood that a vulnerability may be exploited over the link between the components.

As an example, input received over a trusted link is less apt to be validated than that received over an untrusted link. Therefore the likelihood of an exploit of an input validation vulnerability over the trusted link is greater. This is especially important for WS, where the most obvious attack flow (user attacks WS) is not the only realistic attack scenario.

5.4 Leveraging existing taxonomies

We now use a similar construction to combine the results of the previous correlation step with existing vulnerability taxonomies. This combination may be viewed as:

- A summarization of results of the previous step with respect to the viewpoint of the existing vulnerability taxonomies.
- A specialization of existing vulnerability taxonomies into a vulnerability taxonomy for the system under study.

A taxonomy can again be represented by a matrix, namely [vulnerabilities × categories]. In this matrix the rows represent particular vulnerabilities and the columns represent the taxonomy's different classes. The cells of the matrix describe whether the vulnerability is an instance of the class; values are therefore "true" or "false". Vulnerabilities generally belong to one and only one class, although this is not uniformly the case.

By mapping the values "true" to "1" and "false" to "0" and normalizing with a weight vector, we can use a simple matrix multiplication to obtain the desired result: [architecture × vulnerabilities] × [vulnerabilities × categories] × [normalizing vector] = [architecture × categories]. The normalizing vector accounts for the different number of vulnerabilities per class.

For our WS example we used the Bugtraq classification which we selected following our use of Bugtraq as the source of vulnerability data. It is rather coarse-grained, having only 7 classes, but is reasonable for our purposes.

6 Applying our methodology to WS

In this section, we detail application of our methodology to WS and the salient results obtained. We focus on the details, as the coarse-scale process was already explained in the previous section. We begin with the architectural model and properties, and continue with the selection of the vulnerabilities. We present the produced result matrix and touch on how these results can be validated. Finally, we highlight the most interesting findings.

6.1 Architecture and Properties

Our WS architecture model was introduced briefly in Section 5.1 and is depicted in Figure 1. It is a coarse-grained model by design, with four functional components and their connections:

- Clients: the service requester such as an ordering system.
- Directory services: the service locator such as UDDI or DNS.
- WS providers: the actual providers of a service (can be clients of other WS providers).
- Backend systems: the backend systems behind the WS providers such as a database.
- Links: components have logical links between them, e.g., WS requesters communicate directly to WS providers and directory services, but only indirectly to backend systems.

The loopback connection in Figure 1 represents the ability of a WS provider to make requests to other WS providers, aggregate the results and return them to the service requester.

The property selection follows from the architecture model. We selected 17 properties divided in two categories: properties related to the WS paradigm itself, and properties related to implementations of WS systems.

The properties of the first category, derived from the definition of SOA and WS (discussed in Section 2.1 and Section 2.2), are:

- WS are designed to enable interactions between diverse systems and are therefore considered *cross platform*.
- WS allow for the *dynamic discovery* of WS providers through directory services.
- WS support *dynamic binding* between service requester and provider.
- WS are an implementation of SOA and therefore are *service oriented*.
- WS use *XML-based messaging* for communication.
- WS, in contrast to web applications, are especially well suited for *machine to machine interaction*.
- WS are *message-centric*; messages describe what should be performed instead of how it should be performed.
- WS are not limited to a particular transport protocol and are therefore *transport agnostic*.

The second category of properties, related to the implementation of WS, is less clearly defined (as they are, by definition, implementation dependent). We selected the following properties shared by the most popular WS implementations:

- WS are typically implemented in systems with a *managed execution environment*, such as Java or .NET.
- WS have a highly *layered* structure.
- WS make heavy use of *general-purpose libraries and components*.
- Creation, configuration, and deployment of WS is often done via *wizard*.
- Interaction between the WS components typically takes place over *stateless synchronous transport protocols*, e.g., HTTP.
- WS are *not particularly efficient*, e.g., extensive XML parsing.

- The WS standards are *highly complex*.
- Different connections in the WS architecture have *different trust-levels*.

For every property and architectural component described above, we assessed the property's presence: present, not present, or the opposite property is present.

6.2 Vulnerabilities

The second step in the methodology is the selection of the vulnerabilities that are representative of the body of knowledge of historical vulnerabilities. We used Bugtraq as the source of the vulnerabilities as it is one of the largest publicly available vulnerability databases. We selected a suitable subset using the criteria set in Section 5.2: coverage, relevance and availability of information.

Availability of information was the first criterion used to create our subset. It is important as vulnerability descriptions are often incomplete and inconsistent. Therefore, we discarded vulnerabilities whose descriptions did not provide adequate information to understand root causes and thus to assess property influences.

From the set of vulnerabilities with adequate information, we discarded those that are not relevant to WS. We therefore focus on software vulnerabilities and, in particular, those in distributed systems. We discarded, for example, vulnerabilities in physical security systems, such as locks.

From the remaining vulnerabilities we selected a subset so as to retain coverage; representative vulnerabilities from each different class need to remain in the subset. This resulted in a final set of 54 vulnerabilities. For each of these vulnerabilities, we categorized the influence of each selected property on the vulnerability as positive influence, negative influence or no influence.

6.3 Result matrix

The final step of the methodology is the combination of the [vulnerability $\times$ properties] and the [properties $\times$ architecture] matrices via linear correlation to obtain the [vulnerability $\times$ architecture] matrix. We applied the technique described in Section 5.4, using the Bugtraq classification, to group the results according to the [classification $\times$ architecture]. The result matrix is depicted in Figure 2.

The columns of the result matrix represent the component architecture depicted in Figure 1. Recall that the functional components are client, web service, directory and backend components, as well as all the possible connections between these components. The rows represent the different vulnerability classes as defined by Bugtraq: access validation error, boundary condition error, input validation error, design error, failure to handle exceptional situations and unknown.

6.4 Validation

The predictive nature of our methodology poses some specific challenges for its validation. One possible approach entails applying it to a well-established technology, without using prior knowledge of the vulnerabilities in this particular technology. Comparison of our results with the historical vulnerabilities discovered in this technology allows for validation of our methodology. This approach is, however, out of scope for this paper as the authors' interests in this work lies in the early understanding of applied security problems in WS.

To date, the publicly available data on vulnerabilities in WS is not adequate to validate our approach. Nevertheless, this technology is maturing rapidly and we expect to have more data available soon.

In the next section we will discuss the result matrix which indicates both anticipated and unanticipated, yet retrospectively clear, outcomes. We thereby perform a functional validation although fully acknowledge that this is by no means complete.

6.5 Discussion of results

There are a number of conclusions one may draw from the result matrix. We focus our discussion on the most salient of these in the context of real-world WS deployment.

WS often provide an interface that allows for direct interaction with core business processes. Traditionally these interfaces have been closed to the outside world and consequently run in a trusted, but not necessarily trustworthy, environment. WS changes this situation by allowing direct interaction with

	client	WS	directory	backend	client x directory	client x WS	WS x directory	WS x backend	WS x WS
Access Validation Error	0.5	0.4	0.4	0.3	1.8	0.4	1.8	1.3	1.9
Boundary Condition Error	-1	-2	-2	2	1.8	-0.2	1.8	2	1.8
Input Validation Error	2.1	1.2	1.2	2.2	4.2	2.3	4.2	3.3	4.3
Design Error	1	1	1	0.2	1.2	0.3	1.2	0.9	1.2
Failure to Handle Exceptional Conditions	-0.2	-0.7	-0.7	1.2	2	0.3	2	2	2
Configuration Error	1	1	1	0.3	1.3	-0.2	1.3	1.3	1.3
Unknown	0	-1	-1	3	4	2	4	3	4

Figure 2: [classification × architecture]

the core systems, thereby exposing them to a significantly larger range of threats. Therefore, previous assumptions need be reevaluated.

The result matrix shows consistently high values in the input validation class over all the components and connections between the components. This indicates that input validation errors are likely in all WS components and can be exploited over all the connections between these components.

We discuss two subclasses of input validation: input format and input origin. We conclude with a discussion of attack flows and the implications derived from the matrix values describing the connections between components.

6.5.1 Input Format

Unfounded assumptions regarding the format of the input can lead to vulnerabilities. These vulnerabilities are commonly known as input format validation errors, but input validation is only part of the reason why these errors exist.

The loosely coupled and composable nature of WS requires input validation at each of the various stages of the complete WS process. Unfortunately, the fact that form data has different meanings in different layers of the WS implies that input must also be validated in the different layers. Proper validation requires data normalization, and order is important: data must be normalized and then checked (the Nimda worm exploited a series of vulnerabilities that stemmed from the fact that a validation check was performed before the character encoding normalization was applied).

Good support for input validation is essential in both tools and libraries; currently such support is limited. For example, the best method for avoiding SQL-injection (the archetypical input validation error) is the use of prepared statements which has the effect of enforcing separation of control and data channels. The equivalent in the context of WS is XPath-injection, and there is currently no standard support from "prepared XPath".

6.5.2 Origin of data

Other unfounded assumptions relate to the origin of the input, as opposed to its format. The induced vulnerabilities include the direct spoofing of origin, corruption of directory services leading to an incorrectly contacted party, and cross site scripting. These are caused by inadequate input validation. The likelihood of each of these vulnerability subclasses is increased by the composable nature of WS.

We would make special mention of the subclass of vulnerabilities involving bounce attacks wherein the attacker tricks a trusted party into making a bad request (including cross site scripting). The likelihood of these is greatly exacerbated by the use of standard libraries for XML handling that support inclusion of external data sources (such as XInclude and external entities). For example, the attacker may set his name to the string `<xi:include href="passwords.txt" xmlns:xi="http://www.w3.org/2003/XInclude"/>` which will be interpreted as the *instruction* to include a password file.

6.5.3 Attack flows

Recall that the matrix values for the connections between components, unlike the values for components themselves, do not indicate the likelihood of the connection having a vulnerability but rather indicate how easily a certain vulnerability can be exploited over that connection. In the same way that a property or a combination of properties can render a vulnerability more or less likely, there are properties that make the exploitation of a vulnerability over a certain connection more or less difficult.

An example property is the nature of the relationship between the communicating components. When this is a trust relationship, the received input will typically also be trusted and is thus often not subjected to the proper access controls and not sufficiently validated.

The notion of exploitability of an attack over a certain connection naturally leads to the concept of attack flows. Before an attack on a component of a system can be staged, the component needs to be vulnerable to the attack *and* the path from the attacker to the victim needs to permit the attack to be launched. This raises the question: from what components can a certain component be attacked?

Consider the boundary condition error category as an example. These errors are typically caused by a buffer's reserved memory being exceeded by unexpectedly long input, which can lead to the execution of arbitrary code by an attacker. Programs written in C or C++ are prone to this variety of vulnerability.

WS components are generally written in higher-level languages and executed in managed environments, and are thus generally not vulnerable to boundary condition errors. One might there expect that WS, as an entire system, would be similarly invulnerable. This is not the case. The backend components of the WS architecture are generally written in C or C++ and are hence prone to boundary condition errors, as one can observe in the result matrix shown in Figure 2. While the WS component itself may not be vulnerable, it may pass on an attack to the vulnerable backend component.

If we assume that the client is the attacker, then determination of whether such a vulnerability in the backend system is exploitable, involves determination of whether there exists a suitable path from the client to the backend. By suitable we understand "having properties that make exploitation feasible". As there is no direct path from the client to the backend system, the attack would have to be staged indirectly through one or more of the other components. The most obvious scenario is to connect to the WS that, although not vulnerable for the attack in itself, can pass the attack on to the vulnerable backend.

The data in Figure 2 suggests that this most obvious scenario may not be the most problematic, since the properties of the connection between the client and the WS make it difficult to exploit this type of vulnerability over this connection. This is due to a combination of properties, of which one is the untrusted nature of the connection.

Although more difficult to stage, a scenario in which the attacker uses a path through the directory service or in which the directory service itself is the attacker is more likely to be successful. This follows from the relatively high values for these connections in table .

An important realization stemming from this analysis is that one's initial expectations concerning the vulnerabilities of a system are not always accurate. One must consider not only the component-wise vulnerabilities but also the paths over which they can be exploited. Furthermore, it is important to consider all attack flows rather than only that in which the client is the attacker and/or the web service component is the target.

7 Related work

The key difference between our approach and existing taxonomies (see Section 4) is the taxonomic character used and the systematic methodology for deriving new taxonomies. Our taxonomic character is the likelihood that vulnerabilities will appear in a system. This implies that our taxonomy is predictive, whereas previous work focused on classifying existing vulnerabilities. Our methodology is systematic: the classes are not selected *ad-hoc*, but through a well defined process.

Orthogonal Defect Classification [7] is a methodology for analysis of software defects, serving as an early indicator of the health of the software development process. It is related to the methodology proposed in this paper as it also involves a systemic analysis of software defects. However, both the techniques applied and the purpose differ widely. From certain attributes of the detected defects, e.g., the *defect type* or *defect trigger*, statistics are created that are compared to the expectations on defects in a certain phase of the development process.

Our test case of WS is, to our knowledge, the first taxonomy focusing on WS as a system in contrast to web applications, which have been addressed as a component by OWASP [23]. The OWASP vulnerability classes are more specialized (towards web applications) and therefore finer-grained. For example, the input validation error class of the Bugtraq classification is split into several classes, such as SQL injection, encoding errors, etc. However, the similarities between WS and web applications lead us to expect many of these classes to be also relevant for WS.

8 Conclusions and Future Work

In this work we developed a methodology for predicting vulnerabilities in systems. Our methodology systematizes the *déjà-vu* feeling a security expert has when confronted with a new system. It assesses the likelihood that variants of historic vulnerabilities will appear, based on the combination of the properties describing the system. The use of an architectural refinement provides not only an indication of which vulnerabilities will appear, but also where they are likely to appear. This allows one to reason about the security of a system before it is widely deployed and to thereby address security problems at an early stage.

We applied this methodology to WS. This has practical significance because of the importance of WS despite the current lack of wide deployment. The results of our WS example are promising and present a first step towards full validation of our methodology.

The test case application of our methodology revealed two critical steps. On one hand, the selection of the properties is difficult. There is no well-defined minimal set of properties that describes a system in all its aspects. Further, for any particular aspect, balance must be achieved between selecting a set of properties which is rich enough to describe the system yet compact and clear enough as to remain tractable. On the other hand, the selection of vulnerabilities is not trivial. Information in vulnerability databases is often incomplete, hindering understanding of the root causes of the described vulnerabilities.

One limitation of our methodology is the use of a linear mapping with equal weights to derive likelihoods. Linearity is a strong simplification implying that all properties are equally important. Alternative weighing schemes would provide more accurate models, but at the cost of higher complexity.

Nevertheless, the methodology provides a valuable heuristic for identifying the most likely sources of insecurity in a system.

References

[1] R.P. Abbott, J.S. Chin, J.E. Donnelley, W.L. Konigsford, S. Tokubo, D.A. Webb, and T.A. Linden. Security analysis and enhancements of computer operating systems: The RISOS project. Technical Report NBSIR 76-1041, Institute for Computer Sciences and Technology, National Bureau of Standards, US, April 1976.

[2] Ross Anderson. Why cryptosystems fail. *Proceedings of the ACM Conference in Computer and Communications Security*, pages 215–227, November 1993.

[3] Taimur Aslam. *A Taxonomy of Security Faults in the UNIX Operating System.* PhD dissertation, Purdue University, US, August 1995.

[4] Taimur Aslam, Ivan Krsul, and Eugene Spafford. Use of a taxonomy of security faults. *Proceding of 19th NIST-NCSC National Information Systems Security Conference*, pages 551–560, September 1996.

[5] R. Bisbey and D. Hollingworth. Protection analysis: Final report. Technical report, Information Sciences Institute, University of Southern California, US, May 1978.

[6] Matt Bishop. A taxonomy of unix system and network vulnerabilities. Technical Report CSE-9510, University of California, Davis, US, May 1995.

[7] Ram Chillarege, Inderpal Bhandari, Jarir Chaar, Michael Halliday, Diane Moebus, Bonnie Ray, and Man-Yuen Wong. Orthogonal defect classification a concept for in-process measurements. *IEEE Transactions on Software Engineering*, 18:943–956, November 1992.

[8] Microsoft Corporation. Distributed Component Object Model (DCOM). Webpage at `http://www.microsoft.com/com/tech/DCOM.asp`.

[9] Richard DeMillo and Aditya Mathur. A grammar based fault classification scheme and its application to the classification of the errors of TEX. Technical Report SERC-TR-165-P, Purdue University, US, November 1995.

[10] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD dissertation, University of California, Irvine, US, 2000. chapter 5: REpresentational State Transfer (REST).

[11] Security Focus. Bugtraq mailing list. Webpage at `http://www.securityfocus.com/archive/1`.

[12] Object Management Group. Common Object Request Broker Architecture (CORBA). Webpage at `http://www.omg.org/`.

[13] Ivan Krsul. *Software Vulnerability Analysis*. PhD dissertation, Purdue University, US, May 1998.

[14] Liberty Alliance. Liberty alliance project. Webpage at `http://www.projectliberty.org/`.

[15] Daniel Lowry Lough. *A taxonomy of computer attacks with applications to wireless networks*. PhD dissertation, Virginia Polytechnic Institute and State University, US, April 2001.

[16] Alberto Marradi. Classification, typology, taxonomy. *Quality and Quantity*, 2:129–157, May 1990.

[17] James McGovern, Sameer Tyagi, Michael Stevens, and Sunil Mathew. *Java Web Services Architecture*. Morgan Kaufmann, April 2003.

[18] Microsoft Corporation. Microsoft .NET. Webpage at `http://www.microsoft.com/net/`.

[19] Oasis. eXtensible Access Control Markup Language (XACML). Webpage at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

[20] Oasis. Security Assertion Markup Language (SAML). Webpage at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`.

[21] Oasis. Universal Description, Discovery and Integration of Web Services (UDDI). Webpage at `http://uddi.org/`.

[22] Oasis. Web Services Security (WSS). Webpage at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

[23] OWASP. The Open Web Application Security Project. Webpage at `http://www.owasp.com/`.

[24] George Gaylord Simpson. Principles of animal taxonomy. Technical report, Columbia University, New York, US, 1961.

[25] Sun Microsystems, Inc. Java 2 Platform, Enterprise Edition (J2EE). Webpage at `http://java.sun.com/j2ee/`.

[26] Paul Syverson. A taxonomy of replay attacks. *Proceedings of the Computer Security Foundations Workshop*, 1994.

[27] Userland. XML Remote Procedure Call (XML-RPC). Webpage at `http://www.xmlrpc.com/`.

[28] W3C. SOAP. Webpage at `http://www.w3.org/TR/soap/`.

[29] W3C. Web Services Description Language (WSDL). Webpage at `http://www.w3.org/TR/wsdl`.

[30] W3C. XML Path Language (XPath). Webpage at `http://www.w3.org/TR/xpath`.

[31] WS-I. Web Services Interoperability Organization (WS-I). Webpage at `http://ws-i.org/`.

Vulnerabilities in Online Banks

Thomas Tjøstheim and Vebjørn Moen
Department of Informatics
University of Bergen, Norway

Contact author: Thomas Tjøstheim, thomast@ii.uib.no

Abstract

This paper describes some attacks on online banks that authenticate each customer through the use of a unique user identifier and a Personal Identification Number (PIN). Many user identifiers contain structure which make them easy to generate on a computer. Given a generated set of identifiers it is possible to do a brute-force attack on the PINs. A general attack model is described and some example attacks against a Scandinavian online bank are discussed.

Keywords: online bank, brute-force attack, DoS attack.

1 Introduction

Online banks have thrived with the explosive growth and availability of the Internet. A wide variety of services are offered to the customers. Paying a bill, checking the account balance, or applying for a loan can now be comfortably done from one's own home or office. However, the new possibilities introduced with Internet banking have also resulted in new security challenges. It is difficult to create both user friendly and secure Internet banking solutions. Can customers really trust that an attacker will not be able to break into their accounts?

Online banks claim that they are secure as they have many security features like firewalls, Public Key Infrastructure (PKI), Intrusion Detection Systems (IDSs), money auditing systems, and Secure Socket Layer (SSL). The average customer seems to be satisfied with the level of security in Internet banks. However, security is complex and not some magic potion that you add to your system to make it secure. Security should be considered from the start of the system development phase. A careful analysis of the environment is necessary to determine the needed security services and to determine how to implement these services correctly. Analysis of a security protocol is very difficult, due to the many ways an attacker can take advantage of the protocol environment.

In this paper we show that online banks authenticating each customer through an N-digit PIN in combination with a structured user identifier are vulnerable to both brute force and Denial of Service (DoS) attacks. There are at least three online banks in Norway that use or have used this form of customer authentication. However, the authors have decided not to explicitly name any banks, as this has been requested from one of the banks, and the fact that some of the banks are still vulnerable to the attacks described in this paper.

The rest of this paper is organized as follows: Section 2 presents the general attack model, Section 3 describes structure and generation strategies when Social Security Numbers (SSNs) and account numbers are used as unique identifiers, Section 4 discusses some example attacks on a real Internet bank in Scandinavia, and Section 5 concludes the paper.

2 Attack model

A common way of authenticating customers in online banks is to require a unique identifier for each customer together with a secret that only the customer knows. This section describes a general attack

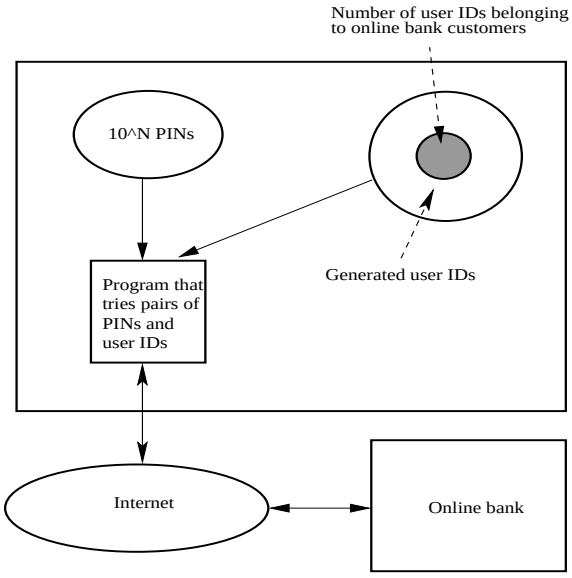


Figure 1: Attack model

model against online banks where each customer has a unique user ID and an N-digit PIN as their secret. The PIN can be either static or dynamic. A static PIN stays the same while a dynamic PIN is changed for each login; it can for example be generated by a PIN calculator.

A customer gains access to an account only by entering a valid user ID and PIN combination. Typically, there will be a limit on how many times (often three or five) a wrong PIN can be entered for a given customer's user ID. The objective of this limit is to prevent a brute-force attack against the customer's PIN. A customer will temporarily lose access to the account if the limit is exceeded and must contact the bank in order to receive a new PIN.

2.1 Brute-force attack

Figure 1 depicts the attack model for the brute-force attack. The generated set of user IDs will contain a subset of the user IDs belonging to the customers of an online bank (we will see examples of how to generate such sets in Section 3). A program logs into the online bank's web pages and automatically enters different user ID and PIN pairs. One can observe that it does not matter if the PIN is dynamic or not. The only difference is that if a PIN is dynamic the attacker would have to attack the account at the moment a valid PIN and user ID combination is found.

Running the attack from only one host is not realistic, as this most likely will be detected by the bank's IDS. A distributed attack could be done by for example spreading a virus that contains the brute-force program in addition to having a control program that gets feedback from the zombie machines. Furthermore, it is possible to avoid the IDS by spreading the attack over several days in order to hide the number of tried logins in the anticipated natural traffic from legitimate users accessing the bank.

The probability of accessing at least one account is

$$\begin{aligned} P(\text{At least one}) &= 1 - P(\text{none}) \\ &= 1 - (1 - P_{success})^Y \end{aligned} \tag{1}$$

where Y is the number of online bank customers in the generated set of user IDs. The anticipated number of cracks is

$$\mu = Y \times P_{success}. \tag{2}$$

The probability that an attacker cracks a random customer's account is

$$P_{success} = \frac{X}{10^N}. \tag{3}$$

Here, X is the maximum number of allowed wrong entries for a PIN, and N is the number of digits in the PIN.

Given a generated user ID, an attacker must first consider whether it is valid and belongs to an online bank user, or not. An attacker would ideally like to maximize the amount of customer user IDs in the initially generated set. This would give the most effective and *silent* attack.

2.2 DoS attack

If an attacker can acquire *many* user IDs, there is also a possibility of doing a distributed DoS attack against the bank's customers. The login scheme of online banks simplifies an application layer based DoS attack. An attacker can temporarily shut down access to accounts by entering X incorrect PINs for each valid user ID.

2.3 Combined DoS and brute-force attack

The probability in (3) assumes a combined DoS and brute-force attack. It can be discussed if the attacker is better off guessing $X-1$ times since customers are not denied access this way, and it might take longer before the bank's IDS detects the attack. However, if the attacker controls a network of "zombie" machines, it is very difficult to both identify the attacker and stop the attack from all the machines in the controlled network. The chaos created by the combined attack could also be an advantage for the attacker. For more information on how to execute a distributed DoS attack please consult [1].

3 Generating user IDs

This section describes two real cases of user IDs being used in online banks. We will study structure and generation strategies for Norwegian SSNs and account numbers. The arguments that apply to the Norwegian user IDs are similar for other countries. In particular, we have verified that the same generation strategies, with minor modifications, can be applied to both Swedish and Danish user IDs.

3.1 Norwegian SSNs

Norwegian SSNs are not confidential. Given a reasonable documented need, the SSNs can be requested from a national register. Many public institutions like hospitals, banks, and tax authorities have legal access to people's SSNs, but it can be difficult for private persons to argue the need for many SSNs. Therefore it might be better for an attacker to generate a set of SSNs. Most SSNs have a specific structure which make them easy to generate.

3.1.1 Structure of Norwegian SSNs

The Norwegian SSN [2] consists of 11 digits: $x_1x_2x_3x_4x_5x_6i_1i_2i_3c_1c_2$.

$x_1x_2x_3x_4x_5x_6$ is the birth date of the individual on the form *ddmmyy*.

$i_1i_2i_3$ is called the individual number and is used to separate people born on the same date. The national register distributes SSNs in the order they receive birth messages. They start with the highest available valid individual number for that day and proceeds downwards for each new birth message. The individual number is based on the century the person is born in, as shown in Table 1. It is also possible to distinguish boys from girls by looking at i_3, which is odd for boys and even for girls.

c_1c_2 are control digits that are calculated as weighted sums of the first 9 and 10 digits, respectively.

$c_1 = 11 - (3x_1 + 7x_2 + 6x_3 + x_4 + 8x_5 + 9x_6 + 4i_1 + 5i_2 + 2i_3 \pmod{11})$.

$$c_2 = 11 - (5x_1 + 4x_2 + 3x_3 + 2x_4 + 7x_5 + 6x_6 + 5i_1 + 4i_2 + 3i_3 + 2c_1 \pmod{11}).$$

If either c_1 or c_2 is calculated to be 10 (mod 11) the SSN is discarded, and if c_1 or c_2 is equal to 11 then it is set equal to 0. Let's assume that c_1 and c_2 are approximately independent, then an SSN will be discarded with the following probability:[1]

$$\begin{aligned} p(c_1 \cup c_2) &= p(c_1) + p(c_2) - p(c_1 \cap c_2) \\ &= \frac{1}{11} + \frac{1}{11} - (\frac{1}{11})^2 = \frac{21}{121}. \end{aligned}$$

Individual number	**Year in birth date**	**Born**
500–749	>54	1855–1899
000–499		1900–1999
500–999	<55	2000–2054

Table 1: Correspondence between individual number and birth date

3.1.2 SSN generation strategies

How can an attacker maximize the ratio of customer SSNs in the initially generated set? Four different strategies will be discussed.

Strategy 1: The simplest strategy is to generate SSNs in such a way that all of the online bank's customers are covered. If for example the online bank only has customers in the age group 18–100, one could generate all possible SSNs for this group. With this scenario, all customers are born in the 20th century and are therefore given individual numbers in the range 000–499. Hence, for each day in the year we get 500 possible SSNs, but an estimate of 21/121 will be invalid numbers. The number of possible SSNs for people that are 18–100 is $500 \times 365 \times 83 \times (100/121) \approx 12.5$ million. Let Z denote the total number of customers in an online bank. The proportion of customer SSNs would then be

$$R_{customers} = \frac{Z}{12.5 \text{ million}}. \tag{4}$$

The drawback, from the attacker's point of view, is of course the huge number of SSNs that has to be checked. A large portion of the SSNs will neither belong to *real* people nor to online bank users. There is also a high probability for the bank's IDS detecting the attack because of the big workload.

Strategy 2: A strategy for increasing the concentration of customer SSNs is to focus on a specific age group that has a high percentage of online bank customers. For example, 34 % of customers in pure online banks are males in the age group 26–35 [3]. The number of generated SSNs for this particular group would be $250 \times 365 \times 10 \times (100/121) \approx 754,132$. The ratio of customer SSNs would then be

$$R_{customers} = \frac{Z \times 0.34}{754,132}. \tag{5}$$

Strategy 3: However, one still has to generate a lot of SSNs belonging to non-existing people. This problem can be avoided by taking advantage of the chronological assignment of SSNs to newborn and immigrants. Instead of generating all valid SSNs for one day, it is possible to use population statistics to reduce the amount of generated SSNs. As an example, one can look at the period corresponding to males in the age group 26–35. There is an average of about 33,343 SSNs assigned per year for this group [4]. This gives an average of $33,343/365 \approx 91.4$ people per day. Let S be the number of assigned SSNs for a particular day. This number will of course vary from day to day. To get an idea of how S varies one

[1] To control the assumption of stochastic independence, a computer program was written that generated all the possible SSNs for the 20th century and counted the number of discarded SSNs. The results from the computer program gave the probability estimate 21/121 down to the 5th decimal.

can make the simplifying, but only approximately true assumption, that each day in the year is equally probable for the assignment of an SSN. This gives a binomial probability distribution with $n = 33,343$ (the average for a year) and the probability $p = 1/365$ (ignoring leap years) that a person is assigned an SSN for a particular day. The standard deviation for a random variable V having a binomial distribution is

$$Sd(V) = \sqrt{np(1-p)}. \tag{6}$$

From (6) we have $Sd(S) \approx 9.5$. To get an estimate of how S varies for each day one can construct an interval with ± 3 standard deviations. The probability that S lies in the interval is

$$\begin{aligned} &P(91.4 - 3Sd(X) \leq S \leq 91.4 + 3Sd(X)) \\ &\approx P(62 \leq S \leq 120) = 0.9974, \end{aligned}$$

since a a binomial distribution can be approximated with a normal distribution.

If the attacker generates 120 SSNs for each day, then:

$$P(120 \leq S \leq 120 + m)$$

is the probability that the attacker looses between 0 and m SSNs for that day. This probability is limited to ≤ 0.0013 since ± 3 standard deviations are used. The number of generated SSNs would be $120 \times 365 \times 10 = 438,000$. We can assume that almost all of the online bank's customers in the age group 26–35 are covered. An approximated ratio of customer SSNs is then:

$$R_{customers} = \frac{Z \times 0.34}{438,000}. \tag{7}$$

SSNs that do not belong to *real* persons can be minimized if the attacker for example generates 62 SSNs for each day. This way the attacker will loose some SSNs belonging to real people, but will with probability $P(S \leq 62 - m)$ which is ≤ 0.0013 generate between 0 and m too many SSNs for a particular day. The number of generated SSNs would then be $62 \times 365 \times 10 = 226,300$. This will correspond to approximately generating 0.68 (226,300/333,430) of the total number of SSNs for the age group 26–35. If we assume a uniform distribution of online customers among the assigned SSNs, an estimate of the customer ratio is then:

$$R_{customers} = \frac{Z \times 0.34 \times 0.68}{226,300}. \tag{8}$$

Strategy 4: Another possibility is to filter out the SSNs belonging to online bank customers by trying to exploit response information from the bank's web pages. Two filtering examples are shown in Section 4.2.

3.2 Norwegian account numbers

An account number is a unique identifier for a customer's account, and can be generated in much the same way as an SSN. This is not hard when the structure is known.

3.2.1 Structure of Norwegian account numbers

A Norwegian account number [5] consists of 11 digits: $b_1b_2b_3b_4a_1a_2a_3a_4a_5a_6c_1$

$b_1b_2b_3b_4$ indicates which bank the account belongs to. Each bank has a set of serial numbers that identify the particular bank.

a_1a_2 is the type of account, e.g. a salary account or a high interest account. There is no standard for which numbers have to be used, each bank can define its own system.

$a_3a_4a_5a_6$ are digits that uniquely identify a customer's account. When a new account is created the smallest available 4-digit number is chosen.

c_1 is a control digit that is calculated as a weighted sum of the first 10 digits:
$c_1 = 11 - (5b_1 + 4b_2 + 3b_3 + 2b_4 + 7a_1 + 6a_2 + 5a_3 + 4a_4 + 3a_5 + 2a_6 \pmod{11})$
However, if c_1 is calculated to be $10 \pmod{11}$, the account number is discarded.

3.2.2 Account number generation strategies

The strategies for generating account numbers are easier than for SSNs. An attacker can find out which serial and account type numbers a particular bank uses. Given one of the bank's serial numbers and an account type, an attacker can generate the next four digits $a_3a_4a_5a_6$ in such a way that it gives a valid account number. Note that there are only $10,000 \times 10/11 \approx 9,090$ valid combinations.

An attacker can also take advantage of the fact that the smallest available account number is always chosen. It is also likely that an attacker can filter out valid account numbers by guessing incorrectly X times for a given account number and observing the response. Given this, it is possible to generate an interval of account numbers that the attacker knows belongs to customers.

4 Example attacks on a real Internet bank

Let Bank B denote the Norwegian branch of a Scandinavian bank that specializes in online banking services. The security solution was changed in 2004. In this section we will look at some theoretical example attacks on the Norwegian bank B, both before and after the change of security solution.

4.1 Bank B before 2004

A customer in bank B is supposedly authenticated by having a valid SSN, a $N = 4$ digit PIN, and a *personal* certificate. A new certificate must be downloaded for each new host used to connect to the bank. Before 2004 a customer downloaded a new certificate by entering a valid PIN and SSN pair. With this scenario an attacker could try to brute force the PIN by attempting to download a new certificate. An attacker had ($X = 3$) tries at guessing the correct PIN.

4.2 Brute-force attack

How does the brute-force attack described in Section 2.1 apply to bank B? Given the first strategy in Section 3.1.2, all SSNs for people in the age group 18–100 are generated. It is realistic to assume that nearly all of B's customers are covered in this SSN generation. In Norway, bank B had more than $Y = 220,000$ customers in 2003. From (1), the following probability can then be obtained

$$P(\text{At least one crack}) = 1 - (1 - \frac{3}{10^4})^{220,000} \approx 1,$$

and from (2), the anticipated number of cracks are

$$\mu = 220,000 \times \frac{3}{10^4} = 66.$$

The ratio of customer SSNs is from (4) $220,000/12.5$ million ≈ 0.018.

The second strategy was to only generate SSNs belonging to males in the age group 26–35. The expected number of B's customers in this age group is $Y = 220,000 \times 0.34 = 74,800$. This gives the following probability:

$$P(\text{At least one crack}) = 1 - (1 - \frac{3}{10^4})^{74,800} \approx 1.$$

The anticipated number of cracks when checking all SSNs one time is

$$\mu = 74,800 \times \frac{3}{10^4} \approx 22.$$

From (5), the ratio of customer SSNs is $74,800/754,132 \approx 0.099$.

The third strategy was to exploit the chronological ordering of SSNs combined with the use of population statistics. We apply the strategy to both the age group of 26–35 and 18–100. Table 2 shows some different results when the average number of SSNs ± 3 standard deviations is generated per day for the two groups. In particular, the table lists the total number of SSNs generated and the approximated number of B's customers covered.

Strategy 3						
Variation	**SSNs per day**	**Age**	**Total SSNs**	**≈ B SSNs**	**Ratio SSNs**	**Cracks**
1	148	18–100	4,483,660	220,000	0.049	66
2	84	18–100	2,544,780	160,180	0.063	48
3	120	26–35	438,000	74,800	0.171	22
4	62	26–35	226,300	50,864	0.225	15

Table 2: Overview of results for strategy 3, when the average number of SSNs ±3 standard deviations is generated each day for the two age groups

In Section 3.1.2 a binomial probability distribution was assumed, and the two cases of generating 120 SSNs and 62 SSNs each day for the group 26-35 were considered. In the first case one can expect to almost cover all of the 74,800 customers in B and obtain about the same probabilities as when we generated all the valid SSNs for the same age group. The ratio (7) of customer SSNs would be $74,800/438,000 \approx 0.171$. With 62 SSNs generated each day the following estimate of the ratio of B SSNs is obtained from (8) to be $50,864/226,300 \approx 0.225$.

The birth statistics [4] for men and women in the age group 18–100 show that there is a total of 3,495,131 people (SSNs) and this gives an average of $3{,}495{,}131/83 \times 365 \approx 115.4$ per day. The standard deviation is calculated from (6) to be ≈ 10.7.

From Table 2 we see that the anticipated number of accounts cracked is dependent on the number of SSNs generated. There are different attack variations depending on how the bank will react to the attack. For instance, the most effective attack would be to try and verify the 226,300 SSNs generated with variation 4 in Table 2. Depending on how B would react to the first attack, the same attack could be repeated with about 15 anticipated cracks each time. On the other hand, if the attacker only gets one chance at verifying the SSNs and the number of SSNs does not matter, then variation 1 in Table 2 yields the best attack.

The fourth strategy was to try to filter out the SSNs belonging to online bank customers. Two different approaches were discovered for the case of bank B:

1. When a valid SSN is combined with a wrong PIN the following error message is returned: "You have entered the wrong SSN or PIN." After three incorrectly entered PINs, and only if this is an SSN belonging to a customer, will the bank respond that the customer has been denied access to the bank. This means that an attacker can guess three times for each SSN, and not only find valid PIN and SSN combinations, but also filter out which of the SSNs belong to B's customers.

2. It is also possible for an attacker to filter an SSN by trying to register a new customer. When registering, bank B only verifies the correspondence between the SSN and the name. Fake email, phone numbers and so on can be entered. Only when entering an SSN that belongs to a customer will a specific error report be sent: "There was en error with registration. Please contact..." Otherwise the person with this SSN will be registered as a new customer. A disadvantage is that an attacker will register a *large* amount of new customers and this will probably be detected. The advantage of this method compared to the first is that an attacker can filter the SSNs before executing the brute-force attack.

4.3 Bank B in 2004

The certificate downloading scheme in bank B was altered in 2004. In addition to entering a valid PIN and SSN combination, a customer must also enter a valid *one-time password* that is sent either to the customers' mobile phone or mailbox. If the password is sent as an SMS it has 15 minutes validity, while a password in the regular mail is valid for 14 days.

The brute-force attack described in Section 4.1 is still possible. An attacker that finds valid PIN and SSN combinations can decide how the one-time password shall be delivered. If the password is delivered by SMS, the attacker could try to *sniff* the password with an interceptor [6]. However, it is much easier and cheaper for the attacker to get the password from the mailbox. Given a valid SSN it was possible to find the matching name and the address. This could for instance be done by using a Norwegian pension

fund web site [7]. This site authenticates members using only SSNs. When a member logs in, the site displays the name and address corresponding to the SSN. If the attacker chooses password delivery by mail, he decides when the password shall be delivered, and will have a good indication of when to steal the mail.

5 Conclusions

This paper shows that online banks authenticating customers through the use of a PIN in combination with an SSN or an account number are vulnerable to both brute-force and DoS attacks at the application level. The degree of exposure to brute-force attacks depends on the number of digits in the PIN. Whether the PIN is static or is changed for each login makes no difference in this case.

In Section 4 it was shown that bank B is vulnerable to a brute-force attack as the PIN only has 4 digits. The PKI solution in bank B is of limited value, since a new certificate and corresponding private key can be downloaded given a valid PIN and SSN combination and the one-time password.

An easy countermeasure against the attacks described in this paper would be to use user IDs that do not contain any structure, so that they would be difficult to generate automatically. The reason the banks have not done this, might be that it simplifies customer handling to use SSNs or account numbers as unique customer identifiers. Another suggestion is a fully functional PKI solution that require customers to meet in person with the Registration Authorities (RAs) when opening an account. This would enable stronger user authetication and possibly solve many of the vulnerabilities in online banks.

Much of the security in online banks relies on IDSs and money auditing systems. The problem with this approach is that it deals more with detection than attack prevention. A combined brute-force and DoS attack is hard to protect against with the current security schemes. The online banks have little other protection than temporarily closing down service for customers. The potential damage to the bank's reputation and the loss in revenue could be substantial.

References

[1] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher "Internet Denial of Service, Attack and Defense Mechanisms." Pearson Education 2005.

[2] Ernst S. Selmer "Personnummerering i Norge: Litt andvendt tallteori og psykologi". Nordisk matematisk tidsskrift 12, Oslo 1964 (in Norwegian).

[3] Monica Hjorth. "En kartlegging av nettb@nkkunders holdninger". Hovedfagsoppgave i Informasjonsvitenskap, Universitetet i Bergen, November 2002 (in Norwegian).

[4] See `http://statbank.ssb.no/statistikkbanken/` (in Norwegian). Last visited August 29th 2005.

[5] See `http://www.ecbs.org/Download/TR201/norway.pdf`. Last visited August 29th 2005.

[6] See `http://www.spylife.com/digital_interceptor.html`. Last visited August 29th 2005.

[7] See `http://www.pensjonskassa.no/elaan/Elaan` (in Norwegian). Last visited August 30th 2005.

Exponentiation to the power p in $\mathbb{F}_{p^k}$ using Variants of Montgomery Modular Arithmetic

Christophe Negre,
Équipe DALI - Université de Perpignan

Abstract

Koblitz curves are special elliptic curves which give efficient implementation of ECC protocols. Scalar multiplication, the most important operation in ECC protocols, requires in the case of Koblitz curves over $\mathbb{F}_{p^k}$ efficient Frobenius evaluation, i.e., efficient exponentiation to p. In this paper we present two algorithms to exponentiate to the power p in fields $\mathbb{F}_{p^k}$. We note that these algorithms can be used to implement randomized arithmetic, and thus, prevent side channel attacks.

1 Introduction

Koblitz [7] and Miller [8] proposed in 1985 the elliptic curve cryptosystem (ECC). Regarding efficiency and security ECC seems to be a good alternative to RSA [10] cryptosystem. The security of ECC is based on the difficulty of the discrete logarithm problem in the group of points of elliptic curves.

The most costly operation of ECC protocols is the scalar multiplication of a point on the curve. Precisely, given an integer m and a point P on the curve, we have to compute mP.

For general elliptic curve, the scalar multiplication is usually done by a chain of doubling and addition of points on the curves, using the so-called *Double-and-Add* method (cf. Cohen *et al.* [3]). Each point doubling and addition require several multiplications, additions, and one possible inversion of point coordinates. In the case of Koblitz curves, the scalar multiplication can be done with a chain of additions and Frobenius evaluations (cf. Smart [11]). In each case the efficiency of the protocols is deeply related to the arithmetic of the underlying field.

For both software and hardware implementation of ECC we have to prevent side channel attacks. As explain by Joye in [6] different strategies are possible: we can use balanced curve arithmetic, randomized finite field arithmetic, and randomized curve representation. The so-called Montgomery multiplication [9] gives simple way to randomize finite field arithmetic using a random multiple of the modulus. In this paper we investigate a method to exponentiate to p in characteristic p inspired by Montgomery multiplication. We extend this method to a recent version of the Mongemery multiplication [2] which seems to be well suited for hardware implementation. This gives finite field arithmetic which improves the security of ECC implementation over Koblitz curve.

This paper is organized as follows: in the first section we recall some basic facts on elliptic curve over finite field, point representations and Koblitz curves. In the second section we briefly present the different strategy to implement finite field arithmetic and the possible advantage of Montgomery methods to prevent side channel attack. In the third section we recall polynomial Montgomery multiplication (i.e. the polynomial version of Montgomery integer multiplication [9]) and then we extend the method to get a modular exponentiation to the power p (algorithm 3 and 4). In the third section we recall the CR representation of field $\mathbb{F}_{p^k}$ and the CR multiplication algorithm presented in [2]. We then present the main result of this paper: we extend previous CR multiplication algorithm of [2] to get an algorithm to exponentiate to the power p (algorithm 9). At the end we give a small example and we conclude.

2 Cryptographic context

In finite group G with underlying difficult discrete logarithm problem (DLP) and efficient group law, one could use it to implement cryptographic protocols such that ElGamal encryption [5] or Diffie-Hellman

key exchange [4].

As mentioned by Koblitz an Miller, elliptic curves have a group structure which gives efficient group arithmetic and difficult DLP. Indeed, let $\mathbb{F}_{p^k}$ be a finite field, with $p > 3$, an elliptic curve $E(\mathbb{F}_{p^k})$ over $\mathbb{F}_{p^k}$ is the set of pair $(x, y) \in \mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$, which fulfill an equation of type $Y^2 = X^3 + aX + b$, plus special point: point at infinity $\mathcal{O}$. We have a group law on the set of points of E: given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on E, we compute the addition $P_3 = (x_3, y_3) = P_1 + P_2$ by

$$\begin{cases} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_3 - x_1) + y_1 \end{cases} \quad \text{where} \quad \lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq \pm P_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P_1 = \pm P_2 \end{cases}$$

Before proceeding, we note that the formulas of the group law above need inversion in $\mathbb{F}_{p^k}$, which is a relatively expensive operation compared to multiplication. It is possible to avoid inversion in addition and doubling formulas when using projective coordinates, in this case the most used field operation is the multiplication. A projective point (X, Y, Z) can be set in an affine form, by one inversion and several multiplications. The are different types of projective coordinates which give different efficiency for the doubling and the addition formulas

Projective $(X/Z, Y/Z) = (x, y)$,

Jacobian $(X/Z^2, X/Z^3) = (x, y)$.

For more on these coordinates, we refer to the work of Cohen *et al.* [3].

For field $\mathbb{F}_{p^k}$ of small characteristic $p < 67$ a special family of curves, called the Koblitz curves, have special property which provides efficient scalar multiplication. These curves E have their equation $Y^2 = X^3 + aX + b$ with $a, b \in \mathbb{F}_p$ and thus admit a curve endomorphism: the Frobenius endomorphism

$$\begin{array}{rccc} \sigma : & E(\mathbb{F}_{p^k}) & \to & E(\mathbb{F}_{p^k}) \\ & P = (x, y) & \mapsto & \sigma(P) = (x^p, y^p). \end{array}$$

In [11] Smart has shown that each integer m, more precisely the curve endomorphism $P \to mP$, can be written in a σ-expansion of the form

$$m = \sum_{i=0}^{\ell} m_i \sigma^i \text{ where } |m_i| \leq \frac{p+1}{2} \text{ and } \ell \leq \lceil 2 \log_p(m) \rceil + 3.$$

If an integer is given by its σ-expansion we can use algorithm 1 to multiply the point P by m.

Algorithm 1 Scalar multiplication on Koblitz curve

Input: $m = \sum_{i=0}^{\ell} m_i \sigma^i$ the σ-expansion of an integer m and $P \in E(\mathbb{F}_{p^k})$ where E is a Koblitz curve
Output: mP

```
Precomputations. For i = 1 to (p+1)/2 do P_i = iP
P' = 0
for i = ℓ to 0 do
  if m_i ≥ 0 then
    P' ← σ(P') + P_{m_i}
  else
    P' ← σ(P') − P_{|m_i|}
  end if
end for
return P'
```

This method is efficient as soon as $\sigma(P)$ can be computed efficiently, i.e., as soon as the exponentiation to the power p can be done efficiently.

Computing the σ-expansion of an integer m is a costly operation. So, for practical applications, Koblitz curves are advantageous when integer m are already expressed in σ-expansion (or can be randomly constructed like in key exchange or El Gamal encryption), or when the scalar multiplication by integer m is done more than once.

3 Preventing side channel attack

Generally finite fields $\mathbb{F}_{p^k}$ are constructed as quotient $\mathbb{F}_{p^k} = \mathbb{F}_p[X]/(M)$ where M is an irreducible polynomial of degree k in $\mathbb{F}_p[X]$. One can speed up field multiplication by using special irreducible polynomial M. Generally the best strategy is to get an M as sparse as possible to obtain simple reduction modulo M [16, 12]. Another strategy consist to use normal bases. These bases provide efficient exponentiation to p [15], but for random normal bases, multiplication is highly inefficient (the complexity is in average equal to $O(n^3)$ multiplications and additions in $\mathbb{F}_p$). To get efficient multiplication it is thus recommended to carefully choose the normal basis, i.e. to use *low complexity* normal basis [17, 13].

The other method to implement finite field arithmetic consist to use the Montgomery's Method [9]. This method is well suited for a large choice of modulus M (even not irreducible). We can use this fact to randomize field arithmetic: instead of computing modulo M we compute modulo $N = MM'$ where M' is randomly chosen polynomial with small degree. As noticed by Joye in [6] it is an important task to have randomized arithmetic in the field, because this provides a protection of ECC against side channel attack.

Precisely, side channel attack ([14], [18]) are based on the property that every point arithmetic is done exactly in the same way at different moment.

For a fixed integer m, side channel attack measure some difference in point operation in the loop for of algorithm 1 to determine the point added P_{m_i} and thus the coefficient m_i.

For example, timing analysis on elliptic curve record the time required for a full scalar multiplication with algorithm 1 and also record additions $\sigma(P') + P_{m_i}$ and substraction $\sigma(P') - P_{m_i}$. Then it use statistic technique to determine the coefficient m_i. So if the arithmetic is randomize, the timing for both scalar multiplication and point operations will be erratic, and then the attack fails. As noticed by Joye [6], to prevent SPA one has to use balanced arithmetic, but for DPA it is also crucial to use both balanced arithmetic and and randomized arithmetic.

In summary, Montgomery methods provide quite less efficient field arithmetic than arithmetic with sparse polynomial or normal basis, but they clearly improve security level. This fact motivated our research on the extending the Montgomery method to exponentiate to the power p.

4 Montgomery Arithmetic

In this section, we will recall the original Montgomery's algorithm for modular polynomial multiplication. Then we present an algorithm to exponentiate to the power p in $\mathbb{F}_{p^k}$ inspired by Montgomery multiplication.

4.1 Montgomery multiplication

Let A, B be two polynomials in $\mathbb{F}_p[X]$ such that $\deg A, \deg B < \deg N$ and let $\Phi \in \mathbb{F}_p[X]$ such that $\deg \Phi \geq \deg N$ and $\gcd(\Phi, N) = 1$. The polynomial version of the Montgomery algorithm [9] computes $AB(\Phi)^{-1} \mod N$ instead of $AB \mod N$ In practice we generally take $\Phi = X^k$.

If we denote by U div V the Euclidean division of U by V, then we can express the calculations in two following steps: first we compute $Q = -A \times B \times N^{-1} \mod \Phi$ and then we compute $R = (A \times B + Q \times N)$ div Φ. The division by Φ is an exact division since $A \times B + Q \times N = 0 \mod \Phi$. Thus when $\Phi = X^k$ this is done by shifting by k of the coefficients of $A \times B + Q \times N$, which is a really simple operation.

Algorithm 2 Montgomery Multiplication

Input: A polynomial $N \in \mathbb{F}_p[X]$ of degree k, two polynomials $A, B \in \mathbb{F}_p[X]$ such that $\deg A, \deg B \leq k-1$ and $\Phi \in \mathbb{F}_p[X]$ with $\deg \Phi \geq k$ and $\gcd(\Phi, N) = 1$

Output: $AB\Phi^{-1} \mod N$

Step 1. $Q \leftarrow -A \times B \times N^{-1} \mod \Phi$

Step 2. $R \leftarrow (A \times B + Q \times N)$ div Φ

return R

Complexity. The complexity of this algorithm was computed in [2] in term of operations in $\mathbb{F}_p$: it requires $2k(k-1)$ multiplications and $2k(k-1)$ additions in $\mathbb{F}_p$.

Exact computation of $AB \mod N$ can be done using algorithm 2 by first compute $R = AB\Phi^{-1} \mod N$ and then, if $C = \Phi^2 \mod N$ is precomputed, by doing

$$R \times C \times \Phi^{-1} \mod N = (AB\Phi^{-1}) \times \Phi^2 \times \Phi^{-1} \mod N = AB \mod N.$$

A method to avoid this second multiplications consist to use the so-called Montgomery representation: a polynomial A of degree smaller than $(k-1)$ is represented by $A_M = A \times \Phi \mod N$. The algorithm 2 compute $A_M B_M \Phi^{-1} \mod N = AB\Phi \mod N$ which is equal to C_M the Montgomery representation of $C = AB \mod N$. Thus, as long as we have to do multiplication or addition, we can stay in Montgomery representation and thus avoid superfluous operations.

4.2 Montgomery exponentiation to the power p

This section is devoted to present an algorithm to exponentiate to the power p in $\mathbb{F}_{p^k}$ using a method inspired by Montgomery multiplication.

Let A be a polynomial in $\mathbb{F}_p[X]$ such that $\deg A < \deg N$, we want to compute $A^p \mod N$. Let Φ be a polynomial of degree bigger or equal than $\deg N$ and such that $\gcd(\Phi, N) = 1$, in practice we will take $\Phi = X^k$. The following algorithm computes $A^p(\Phi^p)^{-1} \mod N$ in a similar way as Montgomery multiplication does: in a first step, we compute $Q = A^p N^{-1} \mod \Phi^p$, and then we compute $R = (A^p - QN) \text{ div } (\Phi^p)$ which is an exact division.

Algorithm 3 Montgomery exponentiation to the power p

Input: A polynomial $N \in \mathbb{F}_p[X]$ of degree k, two polynomials $A \in \mathbb{F}_p[X]$ such that $\deg A \leq k-1$ and $\Phi \in \mathbb{F}_p[X]$ with $\deg \Phi \geq k$ and $\gcd(\Phi, N) = 1$
Output: $A^p(\Phi^p)^{-1} \mod N$
Step 1. $Q \leftarrow A^p \times N^{-1} \mod \Phi^p$.
Step 2. $R \leftarrow (A^p - QN) \text{ div } \Phi^p$.
return R

Let us check that R is equal to $A^p(\Phi^p)^{-1} \mod N$. From the first step, we have $A^p - QN = 0 \mod \Phi^p$, and since $R = (A^p - QN) \text{ div } \Phi^p$ we have

$$(A^p - QN) = \Phi^p R$$

If we reduce this expression modulo N and then multiply by $(\Phi^p)^{-1}$ modulo N we get the required result

$$R = A^p(\Phi^p)^{-1} \mod N.$$

It is easy to check that $\deg R < \deg N$.

In the case $\Phi = X^k$ and $p > k$ it is possible to simplify algorithm 3. First, if we remark that $\deg A^p \leq (k-1)p < kp$ we can see that $R = (-QN) \text{ div } (\Phi^p)$ (here the division is not exact).

Secondly, the major drawback of algorithm 3 is due to the large degree of the polynomials $\widetilde{N} = N^{-1} \mod \Phi^p$ and Q which have degree equal to $kp-1$ in average. In fact, we need only to know a small part of the coefficients of Q to be able to compute R. Precisely, we have only to know the coefficients of Q of degree between $kp-k$ and $kp-1$ to compute the coefficient of $R = (Q \times N) \text{ div } \Phi^p$ since N is of degree k . If we express $\widetilde{N}$ and A^p in radix-X^p

$$\begin{array}{rcll} \widetilde{N} & = & \sum_{i=0}^{k-1} \widetilde{N}_i(X) X^{ip} & \text{with } \deg \widetilde{N}_i(X) < p, \\ A^p & = & \sum_{i=0}^{k-1} a_i X^{ip}, & \end{array}$$

since, $\Phi^p = X^{kp}$, we have

$$Q = A^p \widetilde{N} \mod \Phi^p = \sum_{i=0}^{k-1} (\sum_{j=0}^{i} a_j \widetilde{N}_{i-j}(X)) X^{ip}.$$

To compute $R = (Q \times N) \text{ div } \Phi^p$ we multiply this last expression of Q by N and then divide it by $\Phi^p = X^{kp}$

$$\begin{aligned}(QN) \text{ div } X^{kp} &= \left(\sum_{i=0}^{k-1}(\sum_{j=0}^{i}(a_j\widetilde{N}_{i-j}(X)N)X^{ip}\right) \text{ div } \phi^p \\ &= \left(\sum_{j=0}^{k-1}(a_j\widetilde{N}_{i-j}(X)N)\right) \text{ div } X^p\end{aligned}$$

since for $i \leq k-2$ we have $\deg \widetilde{N}_{i-j}(X)NX^{ip} \leq p-1+k+ip < kp$.

To conclude, if we note $\widehat{N}_i = \widetilde{N}_{i-j}(X)N \text{ div } X^p$, the computation of $A^pX^{-kp} \mod N$ can be computed by the following algorithm.

Algorithm 4 Simplified Montgomery exponentiation to the power p

Input: A polynomial $N \in \mathbb{F}_p[X]$ where $\gcd(N, X) = 1$ and $\deg N = k$ and a polynomial $A \in \mathbb{F}_p[X]$ of degree less than $k-1$, and the set $\{\widehat{N}_0, \ldots, \widetilde{N}_{k-1}(X)\}$ defined by $N^{-1}(X) \mod X^{kp} = \sum_{i=0}^{k-1}\widetilde{N}_iX^{ip}$ and $\widehat{N}_i(X) = \widetilde{N}_i \times N \text{ div } X^p$

Output: $A^pX^{-pk} \mod N$

$R \leftarrow 0$

for $j = 0$ **to** $k-1$ **do** $R \leftarrow R + a_j\widehat{N}_{k-1-j}(X)$ **end for**

return R

Complexity. Let us evaluate the complexity of algorithm 4. The operations done consist to the k multiplications $a_i\widetilde{N}_{k-1-j}$, i.e., a constant polynomial by an element of $\mathbb{F}_p$, and $k-1$ additions of polynomials of degree $k-1$. Thus, the total amount of computation is equal to k^2 constant multiplications in $\mathbb{F}_p$ and $k(k-1)$ additions in $\mathbb{F}_p$. The cost of an exponentiation to the power p with the algorithm 4 is thus roughly equal to the cost of a multiplication.

We have to deal with a last problem: the previous algorithm does not preserve the Montgomery representation. Indeed let $A_M = A \times \Phi \mod N$, then the output of the algorithm 4 is equal to $A_M\Phi^{-p} \mod N = A^p \times \Phi^p\Phi^{-p} \mod N = A^p \mod N$. So, to go back to obtain the Montgomery representation of $A^p \mod N$, it is necessary to use Montgomery multiplication to compute $A^p\Phi = (A^p\Phi^2)\Phi^{-1} \mod N$.

5 Chinese remainder representation

In this section we will recall the results presented in [1, 2] about a version of the Montgomery algorithm which use a chinese remainder representation. Next we will establish the main result of this paper concerning the exponentiation to the power p in CR representation. We give details concerning the version in [1, 2] of Montgomery multiplication, since all tools (cf. Chinese Remainder Theorem and Lemma 1) will be used to get the algorithm of exponentiation to the power p.

Let us go back to the original Montgomery multiplication, i.e., to algorithm 2. This algorithm is clearly well suited to the case $\Phi = X^k$, but, as we will see in the sequel, we could take advantages of a different type of Φ. For differents Phi the exact division by Φ in algorithm 2 is generally not so simple . To avoid this division, we can replace the costly exact division by Φ with a modular multiplication by the inverse of Φ. Precisely if Φ' is a polynomial of degree bigger than $k = \deg N$ and such that $\gcd(\Phi, \Phi') = 1$ then we can compute R by doing

$$R = (A \times B + Q \times N) \times \Phi^{-1} \mod \Phi'.$$

Using this strategy we get the following generalized version of Montgomery's polynomial multiplication.

Algorithm 5 Generalized Montgomery Multiplication

Input: An polynomial $N \in \mathbb{F}_p[X]$ of degree k, two polynomials $A, B \in \mathbb{F}_p[X]$ with $\deg A, \deg B \leq k-1$ and $\Phi, \Phi' \in \mathbb{F}_p[X]$ such that $\gcd(\Phi, N) = \gcd(\Phi, \Phi') = 1$ and $\deg \Phi, \deg \Phi' \geq k$

Output: $AB\Phi^{-1} \mod N$

Step 1. $Q \leftarrow -A \times B \times N^{-1} \mod \Phi$

Step 2. $R \leftarrow (A \times B + Q \times N) \times \Phi^{-1} \mod \Phi'$

return R

The efficiency of this algorithm is related to arithmetic modulo Φ and Φ'.

In [1] we proposed to use generalized Montgomery multiplication with Φ and Φ' which are products of degree one polynomials. Precisely if $N \in \mathbb{F}_p[X]$ is of degree k and if we suppose $2k \leq p$, we define two disjoint subsets of $\mathbb{F}_p$

$$\mathcal{E} = \{e_1, \ldots, e_k\} \text{ and } \mathcal{E}' = \{e'_1, \ldots, e'_k\}.$$

The polynomials Φ and Φ' are defined as follows

$$\Phi = \prod_{i=1}^{k}(X - e_i) \text{ and } \Phi' = \prod_{i=1}^{k}(X - e'_i). \tag{1}$$

The arithmetic modulo such polynomials Φ and Φ' can be done efficiently by using the following Theorem.

Theorem 1 (Chinese Remainder Theorem). *Let $\Phi = \prod_{i=1}^{r} \phi_i \in \mathbb{F}_p[X]$ with $\gcd(\phi_i, \phi_j) = 1$ for $i \neq j$. Let $A \in \mathbb{F}_p[X]$, if we denote $|A|_{\phi_i}$ the remainder of A modulo ϕ_i, then the following application is an isomorphism*

$$\begin{array}{ccc} \mathbb{F}_p[X]/(\Phi) & \rightarrow & \mathbb{F}_p[X]/(\phi_1) \times \mathbb{F}_p[X]/(\phi_2) \times \cdots \times \mathbb{F}_p[X]/(\phi_r) \\ A & \mapsto & (|A|_{\phi_1}, |A|_{\phi_2}, \ldots, |A|_{\phi_r}) \end{array} \tag{2}$$

Moreover if $A \in \mathbb{F}_p[X]$ is such that $\deg A < \deg \Phi$, and if we denote $a_j = |A|_{\phi_j}$, $\Phi_j = \prod_{i \neq j} \phi_i$ and $\rho_j = |\Phi_j^{-1}|_{\phi_j}$, we obtain

$$A = \sum_{j=1}^{r} |a_j \rho_j|_{\phi_j} \Phi_j \tag{3}$$

We can use the previous theorem to get a simpler expression of the multiplication modulo $\Phi = \prod_{i=1}^{k}(X - e_i)$. Let A and B two polynomials of degree smaller than $\deg \Phi$. From the isomorphism of equation (2), if we know the evalutaion of A and B at e_i for $i = 1, \ldots, k$

$$\begin{array}{ll} a_i & = A(e_i) = A \mod (X - e_i), \\ b_i & = B(e_i) = B \mod (X - e_i), \end{array}$$

then the evaluation of the polynomial $C = (AB \mod \Phi)$ at e_i is equal to $a_i b_i = AB \mod (X - e_i)$. This property motivates the following representation of polynomials.

Definition 1 (CR representation). *Let $A \in \mathbb{F}_p[X]$. The CR representation of A relatively to $\Phi = \prod_{i=1}^{k} \phi_i$ is the vector*

$$CR_\Phi(A) = (a_1, \ldots a_k)$$

such that $a_i = A \mod \phi_i$.

If in algorithm 5 the elements are given in a CR representation relatively to Φ and Φ', the two steps of algorithm become really simple and highly parallelizable: for the first step if $a_i = A(e_i), b_i = B(e_i)$ and $\widetilde{n}_i = N^{-1} \mod (X - e_i)$ then for $i = 1, \ldots, k$

$$q_i = Q \mod (X - e_i) = a_i b_i \widetilde{n}_i.$$

For the second step, if we know the following CR representations relatively to Φ'

$$\begin{array}{rlrl} CR_{\Phi'}(A) & = (a'_1, \ldots, a'_k), & CR_{\Phi'}(B) & = (b'_1, \ldots, b'_k) \\ CR_{\Phi'}(Q) & = (q'_1, \ldots, q'_k) & CR_{\Phi'}(Q) & = (n'_1, \ldots, n'_k) \\ CR_{\Phi'}(\Phi^{-1}) & = (\beta_1, \ldots, \beta_k), & & \end{array}$$

then we compute the coefficients of the CR representation relatively to Φ' of R by doing

$$r'_i = (a'_i b'_i + q'_i n'_i)\beta_i \text{ for } i = 1, \ldots, k.$$

We have to deal with a last problem: at the end of the first step we know the CR representation of Q relatively to Φ, but we do not know the CR representation of Q relatively to Φ'. A similar remark can be done for R. The following lemma gives us a method to change CR representation.

Lemma 1. *Let $\Phi = \prod_{i=1}^{r} \phi_i$ and $\Phi' = \prod_{i=1}^{r} \phi'_i$ be two polynomials of $\mathbb{F}_p[X]$, such that $\deg \phi_i, \deg \phi'_j$ are all equal, and let A be a polynomial such that $\deg A \leq \deg \Phi, \deg \Phi'$. Let $CR_\Phi(A) = (a_1, \ldots, a_r)$ be the CR representation of A relatively to Φ. If $\Gamma_{\Phi,\Phi'}$ be the $r \times r$ matrix where the coefficients are defined by*

$$Coeff_{i,j}(\Gamma_{\Phi,\Phi'}) = \left| |\Phi_j^{-1}|_{\phi_j} \Phi_j \right|_{\phi'_i},$$

then $CR_{\Phi'}(A) = (a'_1, \ldots, a'_r)$ the CR representation of A relatively to Φ verifies

$$\begin{bmatrix} a'_1 \\ \vdots \\ a'_r \end{bmatrix} = \Gamma_{\Phi,\Phi'} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_r \end{bmatrix}. \tag{4}$$

In the sequel we will call $\Gamma_{\Phi,\Phi'}$ the base change matrix from Φ to Φ'.

Proof. The lemma is a simple consequence of the chinese remainder theorem: since $\deg A \leq \deg \Phi$ we know from equation (3) that

$$A = \sum_{j=1}^{r} a_j |\Phi_j^{-1}|_{\phi_j} \Phi_j, \text{ where } \Phi_j = \prod_{\ell \neq j}^{r} \phi_\ell \text{ and } \phi_j = (X - e_j).$$

We obtain the coefficient a'_i of $CR_{\Phi'}(A)$ by reducing this last expression modulo ϕ'_i. But this is nothing else than the expression given in equation (4). □

In our case where Φ and Φ' are defined by $\Phi = \prod_{\ell=1}^{k}(X - e_\ell), \Phi = \prod_{\ell=1}^{k}(X - e'_\ell)$, the coefficients of $\Gamma_{\Phi,\Phi'}$ the base change matrix from CR_Φ to $CR_{\Phi'}$ are equal to

$$\gamma_{i,j} = \left| |\Phi_j^{-1}|_{\phi_j} \Phi_j \right|_{\phi'_i} = \prod_{\ell=1, \ell \neq j}^{k} \frac{e'_i - e_\ell}{e_j - e_\ell}. \tag{5}$$

Similarly the coefficients of $\Gamma_{\Phi',\Phi}$ are equal to

$$\gamma'_{i,j} = \prod_{\ell=1, \ell \neq j}^{k} \frac{e_i - e'_\ell}{e'_j - e'_\ell}. \tag{6}$$

Thus, if we use expression (4) to compute the CR representation of Q relatively to Φ' and the CR representation of R relatively to Φ we obtain the following algorithm to multiply in CR representation relatively to Φ and Φ'.

Algorithm 6 CR Multiplication

Input: Two polynomials A, B of degree less than k given by their CR representations relatively to Φ, Φ' $CR_\phi(A) = (a_1, \ldots, a_k), CR_{\phi'}(A) = (a'_1, \ldots, a'_k)$ and $CR_\Phi(B) = (b_1, \ldots, b_k), CR_{\Phi'}(B) = (b'_1, \ldots, b'_k)$. And for $i = 1, \ldots, k$ the elements $\widetilde{n}_i = N'^{-1} \mod (X - e_i)$ and $\beta_i = \Phi^{-1} \mod (X - e_i)$ relatively to Φ and Φ'

Output: $AB\Phi^{-1} \mod N$ in CR representation.

Step 1. **for** $i = 1$ **to** k **do** $q_i \leftarrow a_i b_i \widetilde{n}_i$ **end for**
Step 2. **for** $i = 1$ **to** k **do** $q'_i \leftarrow \sum_{j=1}^{k} q_j \gamma_{i,j}$ **end for**
Step 3. **for** $i = 1$ **to** k **do** $r'_i \leftarrow (a'_i b'_i + q'_i n_i) \times \beta_i$ **end for**
Step 4. **for** $i = 1$ **to** k **do** $r_i \leftarrow \sum_{j=1}^{k} r'_j \gamma'_{i,j}$ **end for**
return $CR_\Phi(R) = (r_1, \ldots, r_k), CR_{\Phi'}(R) = (r'_1, \ldots, r'_k)$

The complexity of this algorithm is equal to $2(k+1)^2$ multiplications by constants, $2k$ multiplications and $2k(k-1)$ additions in $\mathbb{F}_p$. For a detailed study of the complexity of this algorithm we refer to [1].

5.1 CR Exponentiation to the power p

As in the case of Montgomery multiplication we can give a generalized version of Montgomery exponentiation to the power p, i.e., of algorithm 3. This algorithm was well suited to the case $\Phi = X^k$, because the division by X^p is in this case simple. If we want to use a more general Φ, we can replace the division by Φ^p by a multiplication by $(\Phi^p)^{-1}$ modulo a polynomial Φ'^p.

We get the following generalized version of Montgomery's polynomial exponentiation to p.

Algorithm 7 Montgomery exponentiation to p

Input: An polynomial $N \in \mathbb{F}_p[X]$ with $\deg N = k$, $A \in \mathbb{F}_p[X]$ of degree less than k and $\Phi, \Phi' \in \mathbb{F}_p[X]$ such that $\deg \Phi, \Phi' \geq k$ and $\gcd(\Phi, N) = \gcd(\Phi, \Phi') = 1$
Output: $A^p(\Phi^p)^{-1} \mod N$
Step 1. $Q \leftarrow A^p N^{-1} \mod \Phi^p$
Step 2. $R \leftarrow (A^p - QN)(\Phi^p)^{-1} \mod \Phi'^p$
return R

This algorithm has the same disadvantage as algorithm 3, some polynomials are too big: $A^p, (\Phi^p)^{-1} \mod \Phi', \widetilde{N} = N^{-1} \mod \Phi^p$ and Q have a degree roughly equal to $kp-1$. But if we take Φ and Φ' as in the previous section $\Phi = \prod_{i=1}^k (X - e_i)$ and $\Phi' = \prod_{i=1}^k (X - e'_i)$ we could take advantage of the fact that $\Phi^p = \prod_{i=1}^k (X^p - e_i)$ and $\Phi'^p = \prod_{i=1}^k (X^p - e_i)$ to reduce the degree of these polynomials: we will use a CR representation relatively to Φ^p and Φ'^p.

Let us rewrite algorithm 7 with modular arithmetic modulo Φ^p and Φ'^p expressed with the CR representations relatively to Φ and Φ'.

Step 1 in CR_{Φ^p} representation. We first rewrite the polynomials used in step 1. Since $2k \leq p$ we have $\deg A \leq k-1 < p$, and this implies $A \mod (X^p - e_i) = A$. Consequently the CR representation of A relatively to Φ^p is $CR_{\Phi^p}(A) = (A, A, \ldots, A)$. Let $\widetilde{N} = N^{-1} \mod \Phi^p$, and let $CR_{\Phi^p}(\widetilde{N}) = (\widetilde{N_1}, \ldots, \widetilde{N_k})$ be its CR representation relatively to Φ^p. From the chinese remainder theorem the CR representation relatively to Φ^p of Q is given by $Q_i = A^p \times \widetilde{N}_i \mod (X^p - e_i)$, for $i = 1, \ldots, k$,. In this case the polynomials $A, \widetilde{N}_i$ and Q_i have all a degree less than p.

Step 2 in $CR_{\Phi'^p}$ representation. Let $CR_{\Phi'^p}(Q) = (Q'_1, \ldots, Q'_k)$ and $CR_{\Phi'^p}((\Phi^p)^{-1}) = (\beta_1, \ldots, \beta_k)$ be the CR representation of Q and $(\Phi^p)^{-1} \mod \Phi'^p$ relatively to Φ'^p. As in the case of CR representation relatively to Φ^p since $\deg A, \deg N \leq k < p$ we have $CR_{\Phi'^p}(A) = (A, \ldots, A)$ and also $CR_{\Phi'^p}(N) = (N, \ldots, N)$. We finally get

$$R'_i = A^p - Q'_i \times N \mod (X^p - e'_i). \tag{7}$$

Base change matrix Γ_{Φ^p,Φ'^p} and $\Gamma_{\Phi'^p,\Phi^p}$. For the base change matrix, we will see in the lemma bellow that the base change matrices Γ_{Φ^p,Φ'^p} and $\Gamma_{\Phi'^p,\Phi^p}$ between CR_{Φ^p} and $CR_{\Phi'^p}$ have all coefficients in $\mathbb{F}_p$ and are equal respectively to $\Gamma_{\Phi,\Phi'}$ and $\Gamma_{\Phi',\Phi}$ the base change matrices between CR_Φ and $CR_{\Phi'}$.

Lemma 2. *Let Φ and Φ' the polynomials defined in equation 1. Then we have*

$$\Gamma_{\Phi^p,\Phi'^p} = \Gamma_{\Phi,\Phi'} \quad \textit{and} \quad \Gamma_{\Phi'^p,\Phi^p} = \Gamma_{\Phi',\Phi}.$$

Proof. We give only the proof for Γ_{Φ^p,Φ'^p}. From lemma 1 the coefficients of Γ_{Φ^p,Φ'^p} are equal to $\left| |(\Phi_j^p)^{-1}|_{\phi_j^p} \Phi_j^p \right|_{\phi_i'^p}$. But we have

$$|(\Phi_j^p)^{-1}|_{\phi_j^p} = \left(\prod_{\ell=0, \ell \neq j}^{k} (X^p - e_\ell) \right)^{-1} \mod (X^p - e_j) = \prod_{\ell=1, \ell \neq j}^{k} \frac{1}{e_j - e_\ell},$$

and similarly $(\Phi_j^p \mod \phi_i'^p) = \prod_{\ell=1,\ell\neq j}^k (e'_i - e_\ell)$. Thus the coefficients of Γ_{Φ^p,Φ'^p} are finally equal to $\prod_{\ell=1,\ell\neq j}^k \frac{e'_i - e_\ell}{e_j - e_\ell}$ which are nothing else than $\gamma_{i,j}$ the coefficients of $\Gamma_{\Phi,\Phi'}$. □

Finally in the CR representation relatively to Φ^p and Φ'^p of algorithm 7 becomes algorithm 8 below.

Algorithm 8 CR exponentiation to the power p

Input: Let $N \in \mathbb{F}_p[X]$ a polynomial of degree k, $A \in \mathbb{F}_p[X]$ with $\deg A < k$. Let $\widetilde{N}_i = N^{-1} \mod (X^p - e_i)$ and for $i = 1, \ldots, k$.
Output: $AB\Phi^{-1} \mod N$ in CR representation.
Step 1. **for** $i = 1$ **to** k **do** $q_i \leftarrow A\widetilde{N}_i \mod (X^p - e_i)$ **end for**
Step 2. **for** $i = 1$ **to** k **do** $Q'_i \leftarrow \sum_{j=1}^{k} Q_j \gamma_{i,j}$ **end for**
Step 3. **for** $i = 1$ **to** k **do** $R'_i \leftarrow (A^p + Q'_i N_i) \times \Phi^p)^{-1} \mod (X^p - e'_i)$ **end for**
Step 4. **for** $i = 1$ **to** k **do** $R_i \leftarrow \sum_{j=1}^{k} R'_j \gamma'_{i,j}$ **end for**
return $CR_{\Phi^p}(R) = (R_1, \ldots, R_k), CR_{\Phi'^p}(R) = (R'_1, \ldots, R'_k)$

5.2 Improved CR Exponentiation

In this section we will modify the algorithm 8 such that

- it takes the input A in CR representation relatively to Φ and Φ' ;
- and such that the algorithm output the CR represenation relatively to Φ and Φ' of the element R.

For the first point we will use the following Lemma.

Lemma 3. *Let Φ et Φ' be the two polynomials defined in page 76 equation 1 and Φ^p, Φ'^p their p power. Let A be a polynomial of degree smaller than $k-1$. The following identity holds for $i = 1, \ldots, k$*

$$A^p \mod (X^p - e_i) = A \mod (X - e_i) = A(e_i).$$

Proof. Let $CR_\Phi(A) = (a_1, \ldots, a_k)$ be the CR representation relatively to Φ of a polynomial A such that $\deg A < k$. Using the chinese remainder theorem we have

$$A = a_1 \frac{\Phi_1}{\Phi_1(e_1)} + a_2 \frac{\Phi_2}{\Phi_2(e_2)} + \ldots + a_k \frac{\Phi_k}{\Phi_k(e_k)} \text{ where } \Phi_i = \prod_{\ell=1, \ell \neq i}^{k} (X - e_\ell).$$

If we exponentiate A to the power p we get

$$A^p = a_1 \frac{\Phi_1^p}{\Phi_1(e_1)} + a_2 \frac{\Phi_2^p}{\Phi_2(e_2)} + \ldots + a_k \frac{\Phi_k^p}{\Phi_k(e_k)} \text{ where } \Phi_i^p = \prod_{\ell=1, \ell \neq i}^{k} (X^p - e_\ell).$$

We then reduce this expression modulo $(X^p - e_i)$ we get

$$A^p \mod (X^p - e_i) = a_i \frac{\Phi_i^p}{\Phi_i(e_i)} \mod (X^p - e_i).$$

The lemma is finally a consequence of the following identity

$$\Phi_i^p \mod (X^p - e_j) = \begin{cases} 0 \text{ if } j \neq i, \\ \Phi_i(e_i) \text{ if } i = j. \end{cases}$$

□

Lemma 3 expresses that if $CR_\Phi(A) = (a_1, \ldots, a_k)$ then $CR_{\Phi^p}(A^p) = (a_1, \ldots, a_k)$, this means that we have nothing to do to compute the CR representation of A^p relatively to Φ^p.

For the second point we only need to compute the CR representation of R relatively to Φ', since the CR representation of R relatively to Φ can be computed by a simple change-basis. So let us see how to get the CR representation relatively to Φ' of R. We use the fact that for every polynomial U we have

$$U \mod (X - e'_i) = (U \mod (X^p - e'_i)) \mod (X - e'_i) \quad \text{for each } i = 1, \ldots, k. \tag{8}$$

This means that from a CR representation $CR_{\Phi'^p}(U) = (U_1, \ldots, U_k)$ relatively to Φ'^p of a polynomial U, we compute the CR representation relatively to Φ' of U by reducing each coordinate u_i modulo $(X - e'_i)$. So if we apply this fact to R we get that if $CR_{\Phi'^p}(R) = (R'_1, \ldots, R'_k)$ then $CR_{\Phi'}(R) = (R_1 \mod (X - e'_1), \ldots, R'_k \mod (X - e'_k))$. But from equation (7) we have $R'_i = (A^p - Q'_i N)(\Phi^p)^{-1} \mod (X^p - e'_i)$, if we reduce this expression modulo $(X - e'_i)$ we obtain

$$r'_i = R'_i \mod (X - e'_i) = (a'_i - q'_i n'_i)\beta_i \tag{9}$$

where

$$\begin{array}{lll} a'_i & = A^p \mod (X^p - e'_i) = A \mod X - e'_i, \\ n'_i & = N \mod (X - e'_i), \\ \beta_i & = (\Phi^p)^{-1} \mod (X^p - e'_i) = \Phi^{-1} \mod (X - e'_i), \end{array}$$

and where

$$q'_i = \left(\sum_{j=1}^{k} a_j \widetilde{N}_j(X) \gamma_{i,j} \right) \mod (X - e'_i) = \sum_{j=1}^{k} a_j \lambda_{i,j} \gamma_{i,j} \quad \text{with } \lambda_{i,j} = \widetilde{N}_j(X) \mod (X - e'_i)).$$

Finally we obtain algorithm 9 below for the exponentiation to the power p in CR representation relatively to Φ and Φ'.

Algorithm 9 CR Exponentiation to the power p

Input: The $CR_\Phi(A), CR_{\Phi'}(A)$ of A a polynomial such that $\deg A \leq k - 1$.
Output: $CR_\Phi(R), CR_{\Phi'}(R)$ where $R = A\Phi^{-p} \mod N$.
Step 1. **for** $i = 1$ **to** k **do** $q'_i = \sum_{j=1}^{k} a_j \lambda_{i,j} \gamma_{i,j}$ **end for**
Step 2. **for** $i = 1$ **to** k **do** $r'_i = (a'_i - q'_i n'_i)\beta_i$ **end for**
Step 3. **for** $i = 1$ **to** k **do** $r_i = \sum_{i=1}^{k} r'_j \gamma'_{i,j}$ **end for**
return $CR_\Phi(R), CR_{\Phi'}(R)$

Complexity. Let us express the complexity of this algorithm in function of the number of operations in the field $\mathbb{F}_p$. We have $2k^2$ multiplications in $\mathbb{F}_p$ and $k(k-1)$ additions in $\mathbb{F}_p$ for the computation of the q'_i. Next we have $3k$ multiplications and k additions for the computation of the r'_i. And finally we have k^2 multiplications and $k(k-1)$ additions for the computation of the r_i. This give a total of $3k(k+1)$ multiplications and $2k(k-1) + k$ additions in $\mathbb{F}_p$.

Remark 1. We remark that we could decrease the cost of the algorithm if we precompute the products $\lambda_{i,j}\gamma_{i,j}$. In this situation the cost would become $2k^2 + k$ multiplications and $2k(k-1) + k$ additions thus a complexity slightly smaller than the complexity of algorithm 9.

We did not use such precomputations because the $\gamma_{i,j}$ are used in CR multiplication algorithm.

It could be necessary to go back to a Montgomery representation when the input A is given in Montgomery representation, which is not the case of the previous algorithm. But as in it was notice 4.2 we can get back to a Montgomery exponentiation by one multiplication by $(\Phi^2 \mod N)$ using the CR multiplication algorithm.

Example 1. Let us present a small example to illustrate the process of algorithm 9. We choose $p = 17, k = 4$ and $N = X^4 + 2X + 3$ and let us take Φ and Φ' as follows

$$\begin{array}{lll} \Phi & = & \prod_{i=0}^{3}(X - 2i), \text{ here } e_i = 2(i-1), \\ \Phi' & = & \prod_{i=0}^{3}(X - (2i+1)), \text{ here } e'_i = 2(i-1) + 1. \end{array}$$

The CR representation relatively to Φ' of polynomial $\Phi^{-1} \mod \Phi'$ is here as follows

$$CR_{\Phi'}(\Phi^{-1}) = (9, 2, 9, 6),$$

and the CR representation of N relatively to Φ' is equal to $CR_{\Phi'}(N) = (6, 5, 9, 4)$.

The base change matrix $\Gamma_{\Phi,\Phi'}$ and $\Gamma_{\Phi',\Phi}$ defined in the lemma 1 are equal here to

$$\Gamma_{\Phi,\Phi'} = \begin{bmatrix} 12 & 2 & 5 & 16 \\ 1 & 8 & 8 & 1 \\ 16 & 5 & 2 & 12 \\ 5 & 13 & 1 & 16 \end{bmatrix}, \quad \Gamma_{\Phi',\Phi} = \begin{bmatrix} 16 & 1 & 13 & 5 \\ 12 & 2 & 5 & 16 \\ 1 & 8 & 8 & 1 \\ 16 & 5 & 2 & 12 \end{bmatrix}.$$

For the computation of Λ we compute $CR_{\Phi^p}(N^{-1}) = (\widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_3, \widetilde{N}_4)$ and we deduce the coefficients of Λ by computing $\Lambda_{i,j} = \widetilde{N}_j \mod (X - e'_i)$. We get

$$\Lambda = \begin{bmatrix} 7 & 3 & 14 & 11 \\ 15 & 14 & 9 & 9 \\ 3 & 0 & 6 & 10 \\ 0 & 3 & 4 & 13 \end{bmatrix}.$$

We will now apply algorithm 9 to compute $A^p\Phi^{-p} \mod N$ where $A = 11X^3 + 7X^2 + 15X + 3$. First we must compute the CR representation of A relatively to Φ and Phi'

$$CR_\Phi(A) = (3, 13, 12, 1), \;\; CR_{\Phi'}(A) = (2, 0, 13, 8).$$

Now we can compute the CR representation of $Q = A^p N^{-1} \mod \Phi^p$ relatively to Φ' with $CR_\Phi(A), \Lambda$ and $\Gamma_{\Phi,\Phi'}$. We obtain $CR_{\Phi'}(Q) = (3, 11, 0, 15)$.

Next we compute $CR_{\Phi'}(R) = (9, 9, 15, 11)$ with expression 9 and then we get the CR representation of R relatively to Φ by a base change, $CR_\Phi(R) = (12, 3, 8, 11)$. Finally it is easy to see that $CR_\Phi(R)$ is the correct CR representation of

$$(A\Phi^{-1})^p \mod N = 11X^3 + 8X^2 + 12X + 12.$$

$\diamond$

6 Complexity comparison

The complexity of the algorithms stated in this paper are summarized in the table below. They are express in term of additions and multiplications in $\mathbb{F}_p$. Since in practice it is often possible to optimize multiplication by constant element, we count separately multiplications with non-constant operands (Mult) and multiplication with constant operands (C. Mult). For comparison we mention also the complexity for other methods which use fixed representation (polynomial basis modulo binomial, and optimal normal basis).

Method	Multiplication			Exponentiation to p	
	# Mult.	# Add.	# C. Mult.	# Add.	# C. Mult.
Montgomery (algorithm 2 and 4)	$2k(k-1)$	$2k(k-1)$	0	$k(k-1)$	k^2
CR method (algorithm 6 from [2] and 9)	$2k$	$2k(k-1)$	$2k(k+1)^2$	$k(2k-1)$	$3k(k+1)$
Random Normal Bases	k^2	k^3	k^3	0	0
Optimal Normal Basis (cf. [17])	k^2	k^2	0	0	0
Polynomial basis modulo Binomial (cf. OEF[12])	k^2	k^2	$(k-1)$	0	k

We compare here only methods which can be randomized, as others are clearly more efficient for both multiplication and exponentiation to p and also because we are interested only on *secure finite field arithmetic*.

Let us first compare CR arithmetic to Montgomery arithmetic. CR arithmetic has a better multiplication since in CR multiplication most of the multiplications are constant. In other word the exponentiation is more efficient using Mongomery exponentiation. But for hardware implementation, CR arithmetic could take advantage of common chipset for multiplication and exponentiation to the power p since both algorithm use the same base-change matrices.

Now if we compare Montgomery or CR method to random normal basis, we see that randomized normal basis multiplication is highly inefficient. Even with a nearly free exponentiation to p the hight costly of multiplication prevent the use of such bases.

7 Conclusion

In this paper we have presented different algorithms to exponentiate to the power p in $\mathbb{F}_{p^k}$. These algorithms do not take advantage of the form of the polynomial used to construct the field $\mathbb{F}_{p^k}$. Thus, they can be used to implement field arithmetics with randomized modulus. After careful comparison to usual methods, we show that our algorithms are efficient with a balanced security/efficiency point of view (and not uniquely efficiency). In other word, our algorithms allow implementations of ECC over Koblitz curves which prevents side channel attack.

References

[1] J.-C. Bajard, L. Imbert, and C. Negre. Modular multiplication in GF($\mathrm{p^k}$) using Lagrange representation. In *INDOCRYPT*, pages 275–284, 2002.

[2] J.-C. Bajard, L. Imbert, C. Negre, and T. Plantard. Efficient multiplication in GF(p^{k}) for elliptic curve cryptography. In *IEEE Symposium on Computer Arithmetic*, pages 181–187, 2003.

[3] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT'98: International Conference on the Theory and Application of Cryptology*, volume 1514 of *Lecture Note in Computer Science*, pages 51–65. Springer-Verlag, 1998.

[4] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 24:644–654, 1976.

[5] T. El Gamal. A public key cryptosystem and signature scheme based on discrete logarithms. *IEEE Transaction on Information Theory*, IT-31:469–472, 1985.

[6] M. Joye. Elliptic curves and side-channel analysis. *ST Journal of System Research*, 4(1), 2003.

[7] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[8] V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology, proceeding's of CRYPTO'85*, volume 218 of *Lecture Note in Computer Science*, pages 417–426. Springer-Verlag, 1986.

[9] P.L. Montgomery. Modular multiplication without trial division. *Mathematic of computation*, 44(170), april 1985.

[10] L. Adleman R.L. Rivest, A. Shamir. A method for obtaining digital signature and public key cryptosystem. *Comm. ACM*, 21:120–126, 1978.

[11] N.P. Smart. Elliptic curves over small fields of odd characteristic. *J. Cryptology*, 12(2):141–151, August 1999.

[12] D.V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. *Lecture Notes in Computer Science*, 1462:472, 1998.

[13] I.F. Blake D.W. Ash and S.A. Vanstone. Low Complexity Normal Bases. *Discrete Applied Mathmatics*, 1989.

[14] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Lecture Notes in Computer Science*, 1109:104–113, 1996.

[15] J.L. Massey and J.K. Omura. Computational Method and Apparatus for Finite Field Arithmetic, 1986.

[16] E.D. Mastrovito. *VLSI architectures for computations in Galois fields.* PhD thesis, Dep. Elec. Eng.,Linkoping Univ, 1991.

[17] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson. Optimal Normal Bases in GF(p^n). *Discrete Appl. Math.*, 22(2):149–161, 1989.

[18] J. Jaffe P. Kocher and B. Jun. Differential power analysis. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, London, UK, 1999. Springer-Verlag.

Algebraic Test Case Generation of Security Policies in Communication Networks

Mohamed Hamdi, Jihène Krichène, and Noureddine Boudriga

CN&S Research Lab., University of the 7th of November at Carthage, Tunisia

Abstract

Security policies (SPs) play an important role in the protection of networked information systems. They can be seen from two perspectives. As a product, a SP determines the rules that govern the protection of a system. As a specification, a SP describes the constraints that a security solution should comply with. The validation of a SP is therefore essential for an efficient protection. This paper presents a formal technique for the generation of a complete test cases derived from the algebraic specification of SPs. Two properties are considered for the convergence of the generation process, the uniformity and regularity.

1 Introduction

Because computer system technologies are rapidly spreading from academic research to industrial applications, many security issues have been raised. This need for security is driven by the increasingly large proportion of losses caused to the enterprises by various security incidents. Security attacks may disturb the operation of the system, entail loss of secrets, and privacy, and become a risk to the national security and economy. An intriguing fact that has been pointed out by the statistical studies that have addressed communication network attacks is that many organizations already have had, at the moment they were attacked, some protection mechanisms. Effectively, most of security threats are not due to the lack of security equipments, but instead are due to breaches at the planning level. Clearly, there should be a strategic security plan for each organization. In fact, security controls are rarely acquired within the frame of a global security program. To alleviate this problem, enterprises should consider computer system security as a means to achieve their business objectives. Security should be subjected to a documentation activity just as is done for normal production processes. Therefore, using policies to regulate the security management program is among the most important aspects the enterprise should address. Developing Security Policies (SPs) is a sensitive task because the policy itself can be a security weakness if it does not conform to the security requirements. Furthermore, SPs are generally proportionally complex with respect to the protected system making their verification arduous. Hence, appropriate techniques should be used to check whether or not a SP verifies the target properties. Formal methods have been presented as an alternative to represent and to prove the correctness, with regard to a set of defined requirements, of SPs. Many approaches have been proposed to formally model SPs, and prove their correctness. Nonetheless, very little research has been directed towards SP testing, which is an important issue. The existing approaches focused on developing correct SP specification but did not demonstrate how to know whether a SP implementation corresponds to a formal representation. In fact, validating the SP abstract model is not sufficient because faults can also appear at the low-level representation of the policy.

This paper presents an approach to test a SP based on its specification. We also provide a way to check whether or not a candidate security solution is conforming to a SP. We introduced algebraic specifications as a useful tools to represent the assets to be protected, the operations that can be performed on these resources, and the properties that these operations should respect. Algebraic specifications provide a means to rigorously reason on SPs due to the richness of their semantics. Conditional axioms can translate most of the properties the security administrator can think of to a mathematical language. Moreover, in our approach, SP implementations are viewed as many-sorted algebras that model a given

specification. The major problems that we have attempted to cope with consist in: (1) stating whether the algebra correctly translates the specification reasoning to the concrete context, (2) detecting SP faults according to their reaction to automatically generated test sets, and (3) providing a framework to assess candidate SPs for a specific situation. To this end, we developed a test generation technique that consists in executing the SP for a set of selected inputs (called test set) and interpreting the resulting outputs. We used a technique borrowed from software engineering to generate test sets from the axioms of an algebraic specification. The main shortcut of such automated test generation methods is that they often return infinite test sets that are practically impossible to execute. We defined two hypotheses (uniformity and regularity) that are shown to reduce the size of the test set while preserving its mandatory properties (i.e., validity, unbiasedness). Illustrative examples are given to provide the reader with real situations of testing security. In these examples we give the algebraic specification of the policy of interest and we show how to extract test sets using our methodology. Reduction process is then performed, if necessary, to reduce the exhaustive test set to a finite one keeping the properties of tests, i.e. completeness and correctness.

The rest of the paper is structured as follows. Section 2 discusses the definition of the term "security policy" and shows the importance of validating and testing security in telecommunication networks. Section 3 introduces algebraic preliminaries and applies them on security policies. Specific examples are given in this section for illustration purpose. Test derivation from algebraic specification of SPs is addressed by Section 4 where exhaustive test set generation is presented. Hypotheses (uniformity and regularity) are also introduced to generate finite test sets. A case study illustrating these theoretical concepts is given in Section 5 as a real application to what has been stated formally. Finally, Section 6 concludes this paper.

2 Security policy fundamentals

2.1 Defining the security policy

Finding a precise meaning to the security policy turns out to be very arduous as it is used to refer to numerous disparate aspects of information systems' security [6, 7]. In the following we give some examples highlighting the fact that the definition of a security policy is narrowly related to the context in which it is used.

- **Network security policy:** The communication infrastructure is often used to carry out various attacks (e.g., flooding, e-mail spoofing). Therefore, the system should prevent network nodes from forwarding suspicious traffic. Networked assets should behave securely to cover the existing weaknesses. The network SP consists of a set of rules indicating how data should be transmitted across the network. It consists of two components: the preventive network SP, and the reactive network SP. The first attempts to cover vulnerabilities so that attacks do not occur while the second aims at limiting the damage resulting from the occurrence of a security incident.

- **Access control security policy:** Due to numerous security threats that exploit weaknesses at the operating system (OS) level, a set of protection mechanisms should be implemented to plug up such vulnerabilities. The totality of the protection mechanisms related to an OS is called Trusted Computing Base (TCB). They concern the various resources of the computer system (e.g., hardware, software, processes). The most relevant example consists of the access control policy which is enforced by secure OSs to protect the objects they handle. Obviously, for consistency and completeness purposes, those mechanisms should abide by a set of rules, which form the SP. The reference monitor is an entity that mediates accesses to objects by subjects. Among those accesses, only those that conform to the SP are allowed. The reference monitor basically guarantees that the OS respects several pre-defined security principles such as least privilege and continuous protection [6].

- **Key management security policy:** To establish a secure tunnel using the IPSec protocol suite, two end-points should agree upon a set of mutually acceptable cryptographic parameters called Security Association (SA). These security parameters are managed according to local security policies which are set in each end-node. For example, when creating a new SA in order to modify

an older one, "*deletion of the old SA is dependent on local security policy*". Besides, a standard has been recently developed to administrate IPSec security policies; it defines the concept of IP Security Policy (IPSP) [6].

The examples listed above present the security policy seen from different angles. To unify all these views, the security policy has been defined in [6] as "*a set of rules that determine how a particular set of assets should be secured*". This definition may appear to be too general, but it has the merit to extend to most communication network contexts.

2.2 Specifying, validating, and testing security policies

Various languages can be used to express security policies. The choice of the language depends on the context addressed by the policy. Referring to the examples discussed in the previous subsection, it can be pointed out that a myriad of implementations are provided. Each implementation is characterized by a language permitting to express the SP. These languages can be more or less abstract (i.e., formal). The reader would refer to [7] for a more detailed idea about these languages. For instance, access rules to the various objects handled by an operating system (e.g., files, processes) are exported to the kernel language, such as the C language in the case of the Linux system. For key exchange in an encrypted connections, both open-source and industrial solutions implement most of the protocols that address this consideration. Some of the available implementations are written in the C language (e.g., FreeSwan) and several others use proprietary languages.

To make the use of these languages efficient, three important issues should be addressed: specification, validation, and testing. Typically, the SP is written in a human natural language. When it is applied, the SP is translated into another language that is suitable to the secured process. To illustrate this idea, consider a policy describing the security of a networked system. To implement it, the administrator should configure the firewall, using its proper command set, in such a way that it will execute the SP. Even if the original SP substantially achieves its objectives, an error made by the administrator might make it deviate from these goals. Henceforth, the main problem at this stage is to prove that an expression of a SP in a given language conforms to another expression of the same SP in a different language. This issue is analogous to the software development process where many specifications corresponding to different levels of abstraction can be considered. These specifications that deal with the same problem are derived from each other by decreasing the abstraction level at each refinement. This encompasses the use of different more or less abstract languages to specify SPs. Specification languages have to support techniques allowing to verify the conformance with the defined security requirements. This functionality is called validation. It is performed at the specification level through the use of logical tools and deduction systems [3, 1].

Unfortunately, validation is not sufficient to state that a SP implementation is correct with regard to the basic security requirements, another key consideration is to test the conformance of each specification to the one it was derived from. This feature is useful to check whether or not the implementation of a SP fulfills the criteria defined by its specification (which is supposed to be valid). Generating test sets constitutes an important step in SP engineering because it permits to detect the security properties that are not fulfilled by the developed solution. In addition, the testing processing allows to compare a set of candidate SPs with respect to their conformance to the target ideal solution.

3 Algebraic specification of SPs

3.1 Algebraic preliminaries

An algebraic specification is composed of two main parts: a syntactic part, called the signature, and a semantic part, modeled by a set of axioms [2]. A many-sorted signature typically consists of a non-empty set S of sort names, and a $S^\star \times S$-sorted finite set of operation names.

Let $\Sigma = \langle S, \Omega \rangle$ be a signature where S and Ω denote respectively the set of sorts and the set of operations of the signature Σ. Let $\chi = \bigcup_{s \in S} \chi_s$ be a set of variables on this signature (i.e., χ_s is an s-sorted set for every s in S), where χ_s is a set of variables of sort s. The set of terms $T(\Sigma, \chi)$ is defined as follows:

- $\chi_s \subseteq T(\Sigma, \chi_s)$,
- if $o :\rightarrow s$ is a zero-arity operation of Ω, then $o \in T(\Sigma, \chi_s)$,
- if a$o : s_1 \times ... \times s_k \rightarrow s$ is an operation of Ω and $t_i \in T(\Sigma, \chi_{s_i})$ for $i \in \{1, ..., k\}$, then $o(t_1, ..., t_k) \in T(\Sigma, \chi_s)$.

In addition, we consider the positive conditional logic L and the decidable set $L(\Sigma)$ of ε-sentences that express the properties that the signature verifies.

The set $L(\Sigma)$ contains conditional Σ-equations having the following form

$$\forall X. t_1 = t'_1 \wedge ... \wedge t_n = t'_n \Rightarrow t = t', \tag{1}$$

where X is a Σ-sorted set of variables and

$$\begin{cases} t, t' \in |T_\Sigma(\chi)|_s \\ t_i, t'_i \in |T_\Sigma(\chi)|_{s_i}, \ i = 1, .., n \end{cases},$$

$|T_\Sigma(\chi)|$ being the term algebra corresponding to Σ and generated by applying the operations of Ω to variables belonging to the sorts S.

A many-sorted algebra assigns a concrete aspect to a many-sorted signature by associating a set of data to each sort and a function to each operation. In other terms, if $\Sigma = \langle S, \Omega, \Pi \rangle$ is a many-sorted signature, a Σ-algebra α assigns:

- a set $|\alpha|_s$ to each sort $s \in S$, called the carrier set of the sort s,
- a function $|\omega|_\alpha : |\alpha|_{s_1} \times ... \times |\alpha|_{s_k} \rightarrow |\alpha|_s$ to each operation $(\omega : s_1 \times ... \times s_k \rightarrow s) \in \Omega$,
- a predicate $|\pi|_\alpha : |\alpha|_{s_1} \times ... \times |\alpha|_{s_k}$ to each predicate symbol $(\pi : s_1 \times ... \times s_k) \in \Pi$.

The class of all Σ-algebras is denoted $Alg(\Sigma)$.

Let $\Sigma = \langle S, \Omega, \Pi \rangle$ be a signature, χ a corresponding set of variables and α a Σ-algebra. An assignment of χ for α is a family $f = (f_s)_{s \in S}$ of functions such that

$$f_s : \chi \rightarrow \alpha.$$

Therefore, an assignment is simply a mapping from a set of Σ-variables to a Σ-algebra. The value of a Σ-term $t \in T(\Sigma, \chi)$ under an assignment f, denoted $[t]$, is computed using the following induction rules:

1. if $t = x$ with $x \in \chi_s$, $s \in S$, then $[t] = f_s(x)$,
2. if $t = c$ with $\omega = (c :\rightarrow s) \in \Omega$, then $[t] = \omega_\alpha$,
3. if $t = g(t_1, .., t_k)$ with $\omega = (g : s_1 \times .. \times s_k \rightarrow s) \in \Omega$, $k \geq 1$, $t_i \in T(\Sigma, \chi_{s_i})$, then $[t] = \omega_\alpha([t_1], .., [t_n])$.

The first rule means that a ground term has a fix assignment value in the Σ-algebra α.

To state about the conformance of a specific algebra to a set of requirements, expressed through the use of Σ-formulas (i.e., Σ-sentences), we use a binary relation $\models \subseteq Alg(\Sigma) \times L(\Sigma)$ called the satisfaction relation. For instance, when dealing with traditional equational logic, a Σ-algebra α satisfies a formula $(t_1 = t_2) \in L(\Sigma)$ if and only if $[t_1] = [t_2]$ for all assignments $f : \chi \rightarrow \alpha$.

Let Φ be a set of Σ-sentences, an algebra α satisfies Φ if and only if it satisfies every formula $\varphi \in \Phi$. In other terms, we can define a binary relation $\models^\star \subseteq Alg(\Sigma) \times L(\Sigma)^\star$ such that $\alpha \models^\star \Phi$ iff $\varphi \in \Phi \Rightarrow \alpha \in \models .\varphi$.

An interesting property of the satisfaction relation is given in the following equation: $\models^\star .(\Phi \cup \Psi) = (\models^\star .\Phi) \cap (\models^\star .\Psi)$, meaning that a set of equations can be enriched generally by axioms that are satisfied by the algebra.

A formula φ is said to be a logical consequence of Φ (denoted by $\Phi \vdash \varphi$) if and only if every algebra α satisfying Φ also satisfies φ. Formally, this can be expressed using the condition $\Phi \vdash \varphi \Rightarrow \models .\varphi \subseteq \models^\star .\Phi$. Practically, the relation $\vdash$ can be seen as the union of some elementary relations defined on the set of Σ-sentences called inference rules $\vdash = \bigcup_n \prod_{i=1}^{n} (\vdash_i)$, where $\vdash_i$ is an inference rule belonging to the deduction system (e.g., reflexivity rule, congruence rule) and $\prod_i$ denotes the indexed relational product. Informally speaking, this means that deriving an equation consists in applying iteratively the rules of the deduction system.

3.2 Algebraic specification of security policies

Because the SP should address all the security requirements of the enterprise (e.g., authentication, access control), it should be split into multiple components. RFC 2196 defined a list including the major components, i.e. computer technology purchasing guidelines, privacy policy, access policy, accountability policy, authentication policy, availability statement, information technology system & network maintenance policy, and violations reporting policy.

We found that all of these policy components can be specified similarly even though they use distinguished security techniques. Abstracting away from its context, a SP representation should contain the protected asset, the operations modeling their interaction, and the security properties that must be followed. Algebraic specifications, that have been defined in the previous sub-section, allow to build an abstraction reasoning about the SP. Many-sorted signatures [5, 2] can be used to handle resources and operations, while conditional axioms can model the security requirements. According to this view, the assets of the protected infrastructure are categorized into sorts. For example, to establish a connection between two machines, we need three sorts (i.e. host, port, protocol).

For the sake of parsimony, we will not treat all the components listed by RFC 2196. Our analysis will be limited to a single example (i.e., network SP) that shows how sort, operations, and axioms can efficiently model a SP.

Example: Network access SP

This example considers two simple firewall rules within a specific network.

- Allow all connections to the web (IP_1) and the DNS (IP_2) servers.
- Allow connections from the local network to external machines using the tcp protocol.

The algebraic specification of the above-mentioned policy is given in the following:

$pres\,\Pi_{FW}$	$Sorts$		$actions$, ip, $protocols$, $ports$
	$Opns$		$deny :\rightarrow actions$
			$allow :\rightarrow actions$
			IP_1, $IP_2 :\rightarrow ip$
			udp, $tcp :\rightarrow protocols$
			80, 8080, 443, 53 $:\rightarrow ports$
	$Preds$		$inLAN : ip$
			$connect : protocols \times ip \times ports$
	$Axioms$	$\forall x : ip,\, p : ports.$	$(p = 80 \vee p = 8080 \vee p = 443) \wedge \neg inLAN(x) \wedge$
			$connect(tcp, x, IP_1, p) \Rightarrow allow$
		$\forall x : ip,\, p : ports.$	$(p = 53) \wedge \neg inLAN(x) \wedge connect(udp, x, IP_2, p)$
			$\Rightarrow allow$
		$\forall x, y : ip,\, p : ports.$	$inLAN(x) \wedge \neg inLAN(y) \wedge connect(tcp, x, y, p)$

The operations of the algebraic signature correspond to the elementary connection establishment parameters (e.g., ports, IP addresses) and to the firewall responses (i.e., allow, deny). The predicates *inLan* and *connect* respectively state whether a specific host belongs to the local network and whether a given IP address has requested a connection.

4 Deriving security policy test cases from algebraic specification

This section addresses the main goal of this paper, i.e. automatically generating test cases from algebraic specification of security policies. Tests considered here are expected to be finite, complete and correct. To this end, we have adapted the techniques used by Gaudel in [4], especially those used to reduce the exhaustive test set. We also define a methodology for deriving security tests from the formal specification of security policies.

4.1 Basic definitions

Let $SP = (\Sigma, \Phi)$ be such the specification of a security policy P. Given a ground Σ-term t, we note t_P the result of its computation by P. Given a Σ-equation ε, and a policy P providing an implementation for every Σ-operation

- a *test* for ε is any ground instantiation $t = t'$ of ε;
- a *test experiment* of P against $t = t'$ consists of the evaluation of t_P and t'_P and the comparison of the resulting values.

In the following, a test experiment is said to be successful if it concludes to the satisfaction of the test by P, and we note it $P \models \Gamma$ where $\Gamma$is the test.

Given a specification $SP = (\Sigma, \Phi)$, the *exhaustive test set* for SP, denoted $Exhaust_{SP}$, is the set of all well-sorted ground instances of all the Σ-axioms.

$$Exhaust_{SP} = \{\Phi_\sigma | \Phi \in Ax, \sigma = \{\sigma_s : var(\Phi)_s \rightarrow T_{\Sigma_s} | s \in S\}\}.$$

An exhaustive test of P against SP is the set of all the test experiments of P against the formulas of $Exhaust_{SP}$.

Because it is practically impossible to consider the exhaustive test space due to its infiniteness, we introduce the concept of hypotheses that is shown to considerably reduce the test set size. Moreover, we define some properties that must be considered to guarantee the correctness and completeness of test cases.

4.2 Need for hypotheses

Let $Exhaust_{SP}$ be the exhaustive test set for the policy specification SP resulting from replacing variables by the associated ground terms. Even though it covers all the test space, this test set is practically not useful to assess SP implementations. In fact, it is often infinite because, for a specific axiom, it considers all the states of the precondition. This means that the whole carrier set, in the corresponding algebras, should be inspected for erroneous SP behavior. Furthermore, and as it considers all axiom preconditions, $Exhaust_{SP}$ also includes false ones. Although they are meaningless from the SP testing perspective, conditional ground instances of the tested axioms belong to $Exhaust_{SP}$ leading to a substantial waste of computing resources. In addition, conditional tests in $Exhaust_{SP}$ may be redundant with the equational tests as they can be seen as conjunctions and their preconditions are tested by other equational tests. In other terms, $Exhaust_{SP}$ contains both simple and composite equations (i.e., conjunctions), and this makes some tests be performed more than once.

Hence, when testing a SP, only a subset of $Exhaust_{SP}$ may be sufficient. However, some properties should be guaranteed by this subset. Eliminating infiniteness, false preconditions, and redundancies must be done according to a procedure that preserves the properties of $Exhaust_{SP}$. The two major requirements consist in the fact that the selected test set should be *valid* and *unbiased*; meaning that incorrect SP implementations should be discarded, and that all correct SP implementations are accepted. To this end, we use selection hypotheses aiming at reducing the exhaustive test set to a finite test set T.

Regularity and uniformity of hypotheses

Two hypotheses will be considered: uniformity and regularity. These hypotheses are considered on a per-axiom basis. Hypotheses basically consist in assertions about the behavior of an axiom with respect to its precondition space. For instance, uniformity hypotheses states that the test result can be generalized to a whole domain if the test is performed at a single point of this domain. This corresponds to the determination of sub-domains of the variables where the program is supposed to have the same behavior. Assuming that, it is no more necessary to have all the ground instances of the variables but only one by sub-domain. Such criteria are modeled in our framework by uniformity hypotheses. Formally, this is expressed by the following definition.

Definition 1. ***Uniformity hypothesis.*** *Given a formula $\Phi(X)$ where X is a variable, a* uniformity hypothesis *on a sub-domain D for a program P is the assumption:*

$$(\forall t_0 \in D)\,(P \models \Phi(t_0) \Rightarrow (\forall t \in D)\,(P \models \Phi(t))).$$

Another type of hypothesis, called the regularity hypothesis, relies on testing the axiom for several variables that do not exceed a defined 'size'. This notion of size can be customized to represent multiple aspects of SP objects.

Definition 2. ***Regularity hypothesis.*** *Given a formula $\Phi(X)$ where X is a variable, and a function of interest $|t|$ from ground terms into natural numbers, a* regularity hypothesis *for a program P is the assumption:*

$$((\forall t \in \Sigma)\,(|t| \leq k \Rightarrow P \models \Phi(t))) \Rightarrow (\forall t \in T_\Sigma)\,(P \models \Phi(t)).$$

4.3 On hypotheses properties

We introduce the notion of a *testing context* which is a pair (H, T) of a set of hypotheses and a set of tests and we define some important properties which are required for testing contexts.

Definition 3. *Given a specification $SP = (\Sigma, Ax)$, a testing context (H, T) is* valid *if, for all Σ-testable program P,*

$$H \Rightarrow (P \models T \Rightarrow P \models Exhaust_{SP}).$$

Definition 4. *Given a specification $SP = (\Sigma, Ax)$, a testing context (H, T) is* unbiased *if, for all Σ-testable program P,*

$$H \Rightarrow (P \models Exhaust_{SP} \Rightarrow P \models T).$$

Assuming H, validity ensures that any incorrect program is rejected and unbias prevents the rejection of correct programs.

5 Case study

This section discusses the application of the algebraic test case generation method to Kerberos 5, which is a cryptographic authentication protocol. Kerberos 5 is specified as shown in Figure 1.

1. Client to Kerberos: c, tgs
2. Kerberos to client: $\{K_{c,tgs}\}\,K_c, \{T_{c,tgs}\}\,K_{tgs}$
3. Client to TGS: $\{A_{c,s}\}\,K_{c,tgs}, \{T_{c,tgs}\}\,K_{tgs}$
4. TGS to client: $\{K_{c,s}\}\,K_{c,tgs}, \{T_{c,s}\}\,K_s$
5. Client to server: $\{A_{c,s}\}\,K_{c,s}, \{T_{c,s}\}\,K_s$

Where:

c = client	K_x = x's secret key
s = server	$K_{x,y}$ = session key for x and y
a = client's network address	mK_x = m encrypted in x's secret key
v = beginning and ending validity time for a ticket	$T_{x,y}$ =x's ticket to use y
t = time-stamp	$A_{x,y}$ = authenticator from x to y

Figure 1: Kerberos 5 messages.

5.1 Kerberos protocol algebraic specification

To define an algebraic specification of the kerberos 5 protocol, four Finite State Automata (FSA) are proposed. The switching conditions, that allow to move from one state to an other, are established. In other terms, each entity among the four aforementioned ones (i.e., client, server, TGS, Kerberos) will interact with the three remaining ones through the corresponding FSA. According to this reasoning, the axiom section of the algebraic specification is easily deduced from these triggering conditions. Figure 2 illustrates the four modules.

In the following, we consider each module by giving the transition conditions and the related axioms.

5.1.1 The Kerberos server module

The kerberos server is characterized by three states: (0) Kerberos is waiting for client's request, (1) Kerberos is checking client's identity, and (2) Kerberos is generating a session key and a TGT. We give here after the conditions for each transition.

$01 : receive(c, tgs)$
$12 : indb(c)$
$10 : \neg indb(c)$
$20 : send(encrypt(sessionkey(c, tgs), kc) \wedge encrypt(tgt(c, tgs), ktgs))$

We thus obtain a set of axioms describing the Kerberos server module.

Φ_{01} $kerberosstate(S) = waitingforclreq \wedge receive(c, tgs) \Rightarrow$
$nextkerberosstate(S) = checkingclid$
Φ_{12} $kerberosstate(S) = checkingclid \wedge indb(c) \Rightarrow$
$nextkerberosstate(S) = generatingkeyandTGT$
Φ_{10} $kerberosstate(S) = checkingclid \wedge \neg indb(c) \Rightarrow$
$nextkerberosstate(S) = waitingforclreq$
Φ_{20} $kerberosstate(S) = generatingkeyandTGT \wedge send(encrypt(sessionkey(c, tgs), kc) \wedge$
$encrypt(tgt(c, tgs), ktgs)) \Rightarrow nextkerberosstate(S) = waitingforclreq$

5.1.2 The client module

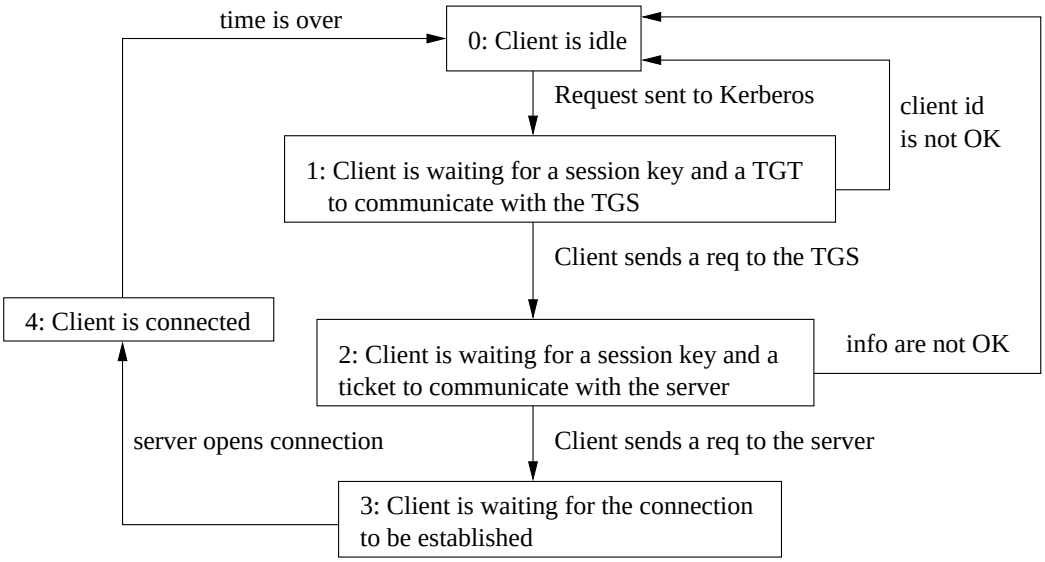
The client is characterized by five states: (0) the client is idle, (1) the client is waiting for a session key and a TGT to communicate with the TGS, (2) the client is waiting for a ticket and a session key that he shares with the server for communication purposes, (3) the client is waiting for the session to be opened by the server, and (4) the client is connected.

The set of axioms describing the client module are given in the following.

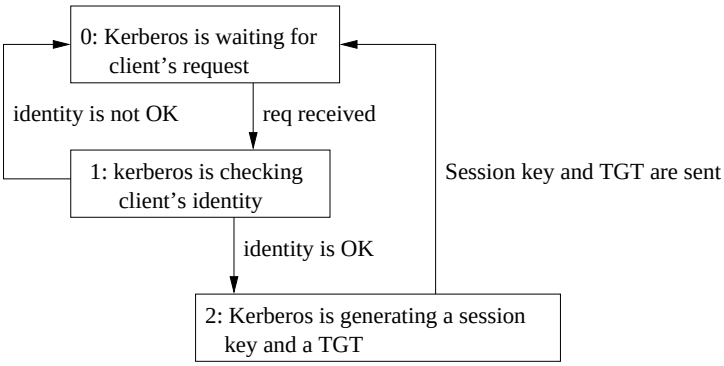
Φ_{01} $clientstate(S) = idle \wedge send(c, tgs) \Rightarrow nextclientstate(S) = waitingforkeyandTGT$
Φ_{12} $clientstate(S) = waitingforkeyandTGT \wedge receive(encrypt(sessionkey(c, tgs), kc) \wedge$
$send(enkrypt(auth(c), sessionkey(c, tgs)), encrypt(tgt(c, tgs), ktgs))) \Rightarrow$
$nextclientstate(S) = waitingforkeyandticket$
Φ_{23} $clientstate(S) = waitingforkeyandticket \wedge$
$receive(encrypt(sessionkey(c, s), sessionkey(c, tgs)), encrypt(ticket, ks)) \wedge$
$decrypt(encrypt(sessionkey(c, s), sessionkey(c, tgs))) \wedge$
$send(enkrypt(auth(c), sessionkey(c, s)), encrypt(ticket, ks)) \Rightarrow$
$nextclientstate(S) = waitingforcx$
Φ_{34} $clientstate(S) = waitwetheringforcx \wedge openconnection(c, s) \Rightarrow$
$nextclientstate(S) = clientisconnected$
Φ_{40} $clientstate(S) = clientisconnected \wedge \neg validity(openconnection(c, s)) \Rightarrow$
$nextclientstate(S) = idle$
Φ_{10} $clientstate(S) = waitingforkeyandTGT \wedge \neg indb(c) \Rightarrow nextclientstate(S) = idle$
Φ_{20} $clientstate(S) = waitingforkeyandticket \wedge receive() \Rightarrow nextclientstate(S) = idle$

5.1.3 The TGS module

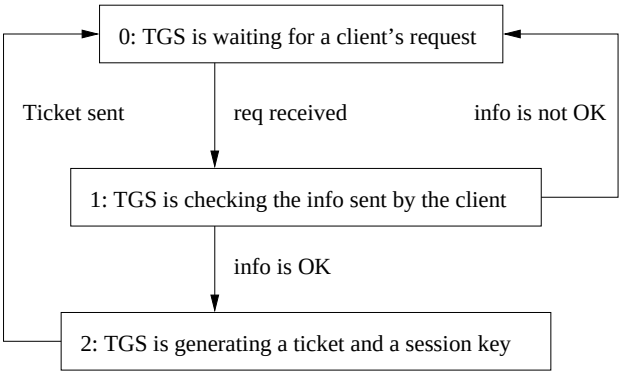
The ticket granting server is characterized by three states: (0) the TGS is waiting for a client's message comporting the session key and the TGT, (1) the TGS is checking the information sent by the client,



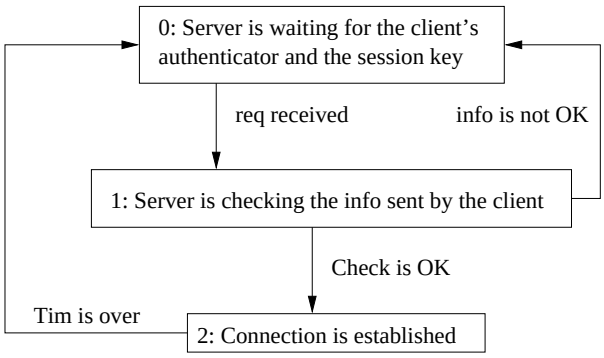
(a) Client module



(b) Kerberos module



(c) TGS module



(d) Server module

Figure 2: Kerberos 5 protocol modules.

and (2) the TGS is generating a ticket and a session key to be shared by the client and the server.

The axioms describing the behaviour of the TGS module are given below.

Φ_{01} $tgsstate(S) = waitingforclreq \wedge$
$receive(enkrypt(auth(c), sessionkey(c, tgs)), encrypt(tgt(c, tgs), ktgs)) \Rightarrow$
$nexttgsstate(S) = checkinginfo$

Φ_{12} $tgsstate(S) = checkinginfo \wedge aresimilar(decrypt(tgt(c, tgs), ktgs), auth(c)) \wedge$
$(timestamp \simeq currenttime) \Rightarrow$
$nexttgsstate(S) = generatingticketkey$

Φ_{10} $tgsstate(S) = checkinginfo \wedge (\neg aresimilar(decrypt(tgt(c, tgs), ktgs), auth(c)) \vee$
$\neg(timestamp \simeq currenttime) \vee (alreadiexist(decrypt(tgt(c, tgs), ktgs), timestamp)))$
$\Rightarrow nexttgsstate(S) = waitingforclreq$

Φ_{20} $tgsstate(S) = generatingticketkey \wedge$
$send(encrypt(sessionkey(c, s), sessionkey(c, tgs)), encrypt(ticket, ks))$
$\Rightarrow nexttgsstate(S) = waitingforclreq$

5.1.4 The server module

The server is characterized by three states: (0) the server is waiting for the client's authenticator and the session key that they will share, (1) the server is checking the information sent by the client, and (2) the server opens a connection to for the client.

The server module operaions are executed in conformance with the following axioms.

Φ_{01} $serverstate(S) = waitingforclreq \wedge$
$receive(enkrypt(auth(c), sessionkey(c, s)), encrypt(ticket, ks)) \Rightarrow$
$nextserverstate(S) = checkinginfo$

Φ_{20} $serverstate(S) = checkinginfo \wedge aresimilar(auth(c), ticket) \Rightarrow$
$nextserverstate(S) = connected$

Φ_{10} $serverstate(S) = checkinginfo \wedge \neg aresimilar(auth(c), ticket) \Rightarrow$
$nextserverstate(S) = waitingforclreq$

Φ_{20} $serverstate(S) = connected \wedge \neg validity(openconnection(c, s)) \Rightarrow$
$nextserverstate(S) = waitingforclreq$

5.2 Test case selection

This section aims at selecting the test cases from the algebraic specifications afore mentioned. Uniformity and regularity hypotheses are applied to reduce the exhaustive test sets.

5.2.1 Testing the Kerberos server module

The exhaustive test set for this wethermodule is defined as follows:

$$Exhaust_{kerberos} = \{\Phi_{\Pi_{kerb}} | \Phi \in Ax, \Pi_{kerb} = \Pi_s : var(\Phi)_s \rightarrow T_{\Pi_s} | s \in \{clients, TGS, states\}\}$$

Test set reduction

As test set reduction is considered, we apply hypotheses to each axiom separately. For instance, for the first axiom (Φ_{01}), we apply the uniformity hypothesis according to the following manner:

(H1) Consider some representative clients such that two of them do not belong to the same network. Consider also a number of ticket granting servers.

$$(\forall t_0 \in clients, \forall u_0 \in TGS)(P \models \Phi_{01}(t_0, u_0) \Rightarrow (\forall t \in clients, \forall u \in TGS)(P \models \Phi_{01}(t, u)))$$

5.2.2 Testing the client module

The exhaustive test set for this module is defined as follows:

$$Exhaust_{client} = \{\Phi_{\Pi_{cl}} | \Phi \in Ax, \Pi_{cl} = \Pi_s : var(\Phi)_s \rightarrow T_{\Pi_s} | s \in \{servers, TGS, states, time\}\}$$

Test set reduction

Similarly to the Kerberos module, we apply hypotheses to each axiom within the client policy. For instance, for the axiom Φ_{40}, we apply the uniformity hypothesis according to the following manner:

(H1) Consider some representative servers such that they belong to different networks and provides different services. Consider also a number of ticket granting servers.

$$(\forall t_0 \in servers, \forall u_0 \in TGS)(P \models \Phi_{40}(t_0, u_0) \Rightarrow (\forall t \in servers, \forall u \in TGS)(P \models \Phi_{40}(t, u)))$$

In addition, we apply the regularity hypothesis to the validity variable (time) such that we consider an interval of time greater than the validity interval (H2).

5.2.3 Testing the TGS moduwetherle

The exhaustive test set for this module is defined as follows:

$$Exhaust_{TGS} = \{\Phi_{\Pi_{TGS}} | \Phi \in Ax, \Pi_{TGS} = \Pi_s : var(\Phi)_s \rightarrow T_{\Pi_s} | s \in \{clients, TGS, time, states\}\}$$

Test set reduction

We give here the test reduction hypothesis applied to the axiom Φ_{12}. Here also, we consider the uniformity and regularity hypotheses.

(H1) Consider representative clients belonging to different networks and some ticket granting servers.

(H2) Consider current time and timestamp values such that they belong to a given interval and try to chose values that are equal in some cases and sometimes different.

5.2.4 Testing the server module

The exhaustive test set for this module is defined as follows:

$$Exhaust_{server} = \{\Phi_{\Pi_{srv}} | \Phi \in Ax, \Pi_{srv} = \Pi_s : var(\Phi)_s \rightarrow T_{\Pi_s} | s \in \{clients, time, states\}\}$$

Test set reduction

The uniformity hypothesis can be applied for instance to the axiom Φ_{01} as follows:

(H1) Consider clients belonging to different networks.

$$(\forall t_0 \in clients)(P \models \Phi_{01}(t_0) \Rightarrow (\forall t \in clients)(P \models \Phi_{01}(t)))$$

6 Conclusion

Algebraic specifications constitute a useful tool to generate test cases following a formal process. This paper has proved the efficiency of algebraic specifications in modeling SPs and deriving associated test sets. Using the proposed approach, we can verify the conformance of a SP to its specification, detect implementation errors in SPs, and assess SP efficiency. Two hypotheses, uniformity and regularity, have been proposed to prevent the generation of infinite test sets.

References

[1] F. Siewe, A. Cau and H. Zedan, "A Compositional Framework for Access Control Policies Enforcement," ACM Conference on Computer Security, FMSE'03, pp. 32-42, Washtington, D.C., 2003.

[2] J. Loeckx, H.D. Enrich and M. Wolf, "Specification of Abstract Data Types," Wiley Teubner, 1996.

[3] L. Cholvy and F. Cuppens, "Analyzing Consistency of Security Policies," IEEE Symposium on security and privacy, Oakland, USA, 1997.

[4] M.C. Gaudel, "Testing can be formal too," TAPSOFT95, LNCS. Springer Verlag, Vol. 915, pp. 82-96, May 1995.

[5] M. Hamdi and N. Boudriga, "Algebraic Specification of Network Risk Management," ACM Workshop on Formal Methods in Security Engineering, pp. 52-60, Washington, D.C., 2003.

[6] M. Hamdi, N. Boudriga, M.S. Obaidat, "Security Policy Guidelines," Handbook of Information Security, H. Bidgoli, Editor, 2005.

[7] S. Barman, "Writing Information Security Policies," New Riders, 2002.

Attack on Sun's MIDP Reference Implementation of SSL

Kent Inge Fagerland Simonsen, Vebjørn Moen, and Kjell Jørgen Hole
Department of Informatics
University of Bergen, Norway

Contact author: Vebjørn Moen, moen@ii.uib.no

Abstract

Key generation on resource-constrained devices is a challenging task. This paper describes a proof-of-concept realization of an attack on Sun's reference implementation of the Mobile Information Device Profile (MIDP). It is known that the MIDP implementation has a flaw in the generation of the premaster secret in SSL. Our attack exploits the flaw to recover the symmetric keys used in an SSL session.

1 Introduction

Running Java programs on resource-constrained devices like cellular phones and personal digital assistants require a specialized run-time environment. The Connected Limited Device Configuration (CLDC) [1] provides a set of Application Programming Interfaces (APIs) and a virtual machine for this environment. Together with a profile such as the Mobile Information Device Profile (MIDP) [2], it provides the possibility to develop Java applications to run on devices with limited memory, processing power, and graphical capabilities.

MIDP is a collection of APIs building on CLDC, providing some more advanced capabilities. Applications that comply with this standard are called MIDlets. Many companies have been involved in the development of MIDP, including Ericsson, NEC, Nokia, Palm Computing, Research In Motion (RIM), DoCoMo, LG TeleCom, Samsung, and Motorola.

MIDP has support for the Hyper Text Transfer Protocol (HTTP), where the information is sent in the clear, and secure HTTP, denoted HTTPS, which supports authentication, confidentiality, and integrity. The security of HTTPS is provided by Secure Socket Layer (SSL), or its successor Transport Layer Security (TLS).

As with many other cryptographic protocols, the security of SSL and TLS depends on generating secret key material. The randomness used in the process of generating the key material decides the strength of the resulting keys.

The first version of SSL in Netscape was shown to create key material using time [3] as input to a Pseudo-Random Number Generator (PRNG); this input is called a *seed*. Seeding with time is a common mistake, since it is difficult to get access to a good seed on a general purpose computer. Creating truly random numbers on a deterministic device such as a computer is impossible. We need to access a hardware source to get some randomness—strong sources of randomness include thermal noise and a radioactive decay source. Creating good random numbers in a constrained environment such as a cellular phone is truly a challenge, but the security in SSL and most other crypto systems depend on a source for randomness.

It has previously been commented [4] that the reference implementation of MIDP provided by Sun generates the *premaster secret*, from which the message authentication and encryption keys in SSL are derived, with a PRNG seeded with current time. Sun offers the reference implementation, but it is intended that every manufacturer of MIDP devices should port the implementation to their products. We describe an implementation of an attack on an SSL session between a server and a client using Sun's MIDP reference implementation which successfully recovers the SSL premaster secret, and consequently the authentication and encryption keys used in the SSL session.

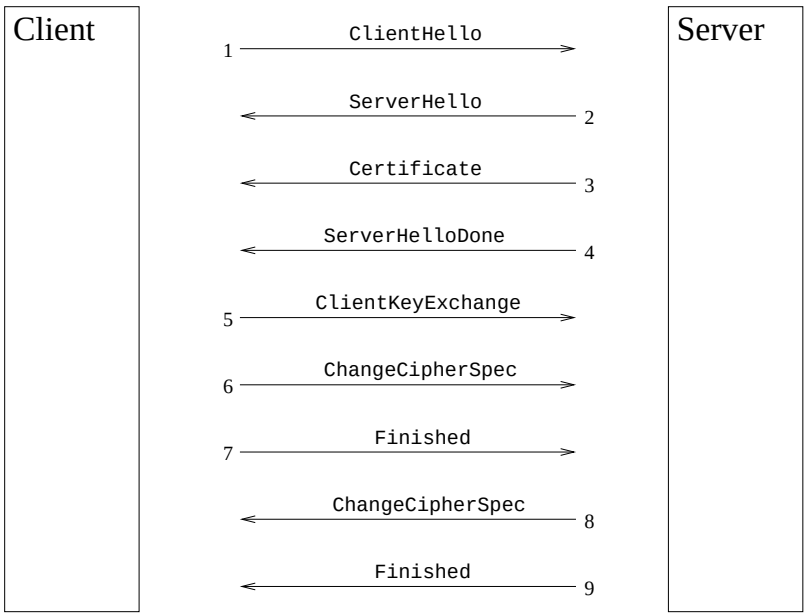


Figure 1: The 9 messages that SSL uses to establish an encrypted communication channel.

In Section 2 we give a brief introduction to SSL, Section 3 considers randomness, Section 4 describes the attack on SSL in MIDP, as well as the implementation of the attack, and Section 5 concludes the paper.

2 SSL

This section is not meant to give a complete description of the SSL protocol; for a complete description [5] is recommended. We will consider the simplest case of SSL, namely establishing an encrypted communications channel.

The situation is that a client wants to establish a secure session with a server. To do this the client and server exchange SSL messages. Figure 1 shows the SSL handshake used to establish a shared secret.

1. `ClientHello`: The client asks the server to begin the negotiation of the security services used by SSL. This message contains fields for a version number (3.0 for SSLv3 and 3.1 for TLS), and a 32-byte nonce used as seed in the generation of the premaster secret. The SSL specification suggests that 4 of these 32 bytes contain the time and date to avoid client reuse of this 32-byte random number. A session ID to identify the specific SSL session, a list of cryptographic primitives that the client can support, and some more fields not mentioned here are also a part of the `ClientHello`.

2. `ServerHello`: The server responds to the `ClientHello`. This message contains fields for a version number, a 32-byte nonce where 4 bytes are used for time and date, a session ID number, a CipherSuite field which determines the cryptographic parameters, such as algorithms and key sizes. The `ServerHello` also contains some more fields not discussed here.

3. `Certificate`: The server sends a certificate containing the public key information.

4. `ServerHelloDone`: Tells the client that the server is finished with the initial negotiation messages.

5. `ClientKeyExchange`: The client generates the premaster secret, encrypts it with the public key received in the server certificate and sends the result to the server.

6. `ChangeCipherSpec`: This message tells the server that from now on any message received from the client will be encrypted with the agreed algorithm and key.

7. `Finished`: This message from the client to the server allows the server to verify that the negotiation has been successful. It contains a hash of key information, and contents of all previous SSL handshake messages exchanged by the client and server. Also notice that this message is encrypted.

8. `ChangeCipherSpec`: This message tells the client that from now on all messages from the server will be encrypted.

9. `Finished`: The client can now verify that the SSL negotiation has been successful. Just as for the finished message from the client it contains a hash of key information, and contents of all previous SSL handshake messages, and it is also encrypted.

After finishing the above protocol the client and the server share symmetric keys for message authentication and encryption, and using the certificate received from the server in message 3 the client can verify that it is talking to the correct server. Note however that the described SSL negotiation does not allow the server to authenticate the client. Observe also that "`Finished`" messages can be used by the server and the client to verify that the other part has the correct key.

3 Randomness and PRNGs

The security of SSL rests on the infeasibility of testing all possible keys used for encryption. If the key space is too large, then the brute-force attack will take too much time. But if an attacker can reduce the number of keys to be tested, she might be able to crack the key.

Many applications use easily available sources of randomness to create an initial value, or seed. This seed is then used as input to a PRNG. The PRNG expands the seed into a longer, random-looking bit stream. For a non-security application the seed only needs to change every time the program runs, but when we use it to generate cryptographic keys, the seed also needs to be as unpredictable and unguessable as the key itself for an attacker.

Consider a system using 128-bit keys. A brute-force attack on such a system would need to check on average 2^{127} keys, which is a huge number and clearly infeasible on a modern computer. What happens if these 128 bits are generated with a PRNG? Assuming that all the details about the PRNG are known to the attacker, the security of the cryptographic key now depends upon the seed. In other words, the number of possible seeds gives the number of possible cryptographic keys. If the PRNG is seeded with milliseconds since midnight, January 1, 1970 in the GMT timezone, and the attacker knows which year the seed is created, she only needs to check $365 \cdot 24 \cdot 3,600 \cdot 1,000 = 31,536,000,000 \approx 2^{35}$ different keys, which is a relatively small task for a modern computer.

Using PRNGs to create cryptographic keys requires that there exists at least as many equally likely seeds as possible keys, to avoid that the PRNG reduces the effective key length.

3.1 Creating a seed

The seed is essential for the security of the system. RFC1750 [6] gives some recommendations for security in randomness. Essentially there are two strategies: either use a reliable hardware source of randomness or use a mixing function to combine several more or less random inputs to create a "pool" of random data, e.g. Yarrow [7] and */dev/random* in GNU/Linux.

Radioactivity decay source, Gaussian white noise and spinning disks [6, 8] are all examples of hardware sources of randomness. A small addition in hardware and software to access these sources, could solve the seed problem.

The */dev/random* in GNU/Linux is an RNG which collects environmental noise from devices and other sources into an entropy pool, and keeps an estimate of the number of available bits in the entropy pool. When random numbers are requested they are created from the pool. Gutmann [9] describes some practical solutions of how to create random numbers for use in cryptographic protocols and for key material.

4 The Attack

The source code for Sun's reference implementation of MIDP is available for download from Sun, but it does not contain the source code for SSL and the PRNG. By decompiling the SSL.jar which comes with the compiled version of MIDP we obtained the Java byte code, and from that we discovered how the seeding of the PRNG is implemented.

The PRNG is seeded with the current time in milliseconds and 16 static bytes. The PRNG also allows manual seeding, but this is not used in the reference implementation. First, we give a brief overview of how the PRNG works and what the idea of the attack is, then more details are given in the remainder of the section.

The PRNG uses the MD5 hash function to mix input and the current state, and it is reseeded with current time and the previous seed for each block of data that is generated. The entire MD5 output is used, which gives a block size of 16 bytes.

During the SSL handshake a PRNG object is constructed on the client. The PRNG object generates a 32-byte nonce sent in the clear, as well as a 48-byte premaster secret which is sent encrypted. The first two bytes of the 48 bytes used for the premaster secret are discarded to make room for some version information.

The PRNG is seeded 5 times with time in milliseconds, and one can be certain that all the time seeds come in proximity of each other. Since the nonces are sent in the clear, it seems reasonable to split the process in two parts. First, the time seeds used to create the client nonce are found so that we can synchronize our clock with the clock on the device, and then we guess the next three time seeds that lead to the premaster secret.

For each suggestion for the premaster secret we need to generate the encryption/decryption and message authentication keys, decrypt a package and check the Message Authentication Code (MAC) value.

4.1 The PRNG

The handshake procedure uses the same PRNG object to create the nonce and the premaster secret. The pseudo code version of the decompiled PRNG from Sun's reference implementation of MIDP is shown in Figure 2.

When the PRNG is constructed it initializes the MD5 digest and the `updateSeed()` method is called, where a time seed together with a constant are used to create the first state. The `updateSeed()` method feeds the current state and the current time in milliseconds in that order and calls the `doFinal()` method whose output is the next state. The digest is reset after every `doFinal()`.

The `generateData()` method writes the pseudo random data to an array (which it takes as an argument). When it runs out of random data (every 16 bytes) it digests the current state and calls the `updateSeed()` method. The data resulting from hashing the current state is said to be the pseudo random data, and is written to the array until it is full, or more data is needed. Note that `randomBytes` is a global array.

The generation of the nonce and premaster in the MIDP SSL is illustrated in Figure 3. The client generates 5 different 16-byte values with this PRNG, the first two outputs are used for the known nonce and the three next outputs are used for the unknown premaster secret. To generate the first 16-byte, a 16-byte constant found in the decompiled Java byte code and current time in milliseconds are hashed and the output is the first state, which again is hashed to yield the first 16-byte of output. At the same time the state and current time in milliseconds are digested and the output is the next state. The next four outputs needed to create the nonce and premaster secret, are generated in a similar manner; digest the state to get the output, and digest the state together with current time to get the next state.

4.2 The attack step-by-step

1. Sniff an SSL session and record the starting time.
2. Retrieve the client nonce and the server nonce. These are sent in the clear in the `ClientHello` and `ServerHello` messages.
3. Decide the start and stop time, i.e., in which time interval did the client seed the PRNG.

```
constructor() {
  initialize digest;
  updateSeed();
}
updateSeed() {
  digest.update(seed);
  digest.update(currentTimeMillis);
  seed = digest.doFinal();
}
generateData(byte[] buf, int off, int len) {
  int i = 0;
  int byteAvailable = 16;
  while(true) {
    if(bytesAvailable == 0)  {
      randomBytes = digest.doFinal(seed);
      updateSeed();
      bytesAvailable = 16;
    }
    while(bytesAvailable > 0) {
      if (i == len)
        return;
      buf[off+i] = randomBytes[--bytesAvailable];
      i++;
    }
  }
}
```

Figure 2: Parts of the code of the PRNG from Sun's reference implementation of MIDP.

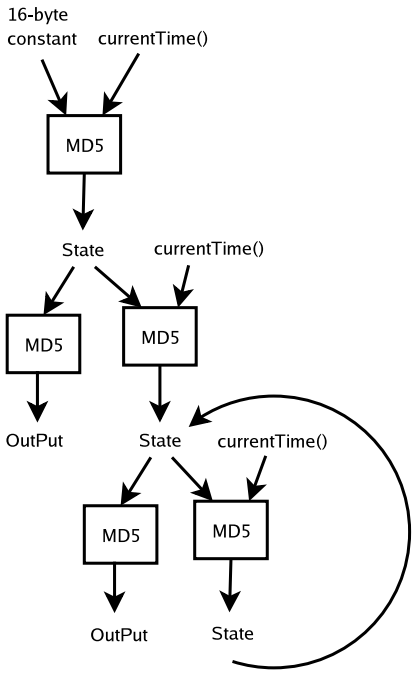


Figure 3: How MD5 is utilized to generate the pseudo random data used for nonce and premaster in the reference implementation of MIDP SSL. The 16-byte constant is known from the decompiled Java byte code.

4. Since the client nonce is sent in clear, we know the first and second output of the PRNG. Find the value between start and stop time that was used to create the first 16 bytes of the client nonce by trying all possible values.

5. When the time seed that were used to generated the first 16 bytes is found, the PRNG can be set in the correct state. Then try all possible time seeds from the start time until the stop time, until the next 16 bytes of the nonce is found.

6. We now know exactly when the client's nonce was created according to the clients internal clock. Using this information we try to find the premaster secret which the client generates a short time after creating the nonce. Exactly how short this time is, is determined by the client device, its load, the speed of the network connection and many such factors. The amount of uncertainty about the time period in which the premaster secret is generated affects the complexity of the search for the premaster secret. Use the time seeds found in step 4 and 5 to set the state of the PRNG, then generate all possible values for the next three time seeds. Then use the suggested values together with the client nonce and server nonce to generate a candidate for the premaster secret and check if it is correct.

```
for each t1 in time interval
  for each t2 in time interval ≥ t1
    for each t3 in time interval ≥ t2
      premaster = generatePreMasterCandidate(
                    PRNG_state,t1,t2,t3)
      check(premaster)
```

4.3 Checking the premaster

There are several approaches to check if the suggested premaster secret is correct. One good suggestion is to create the keys used in SSL (encryption and message authentication keys) based on the premaster secret. Then we decrypt a package and attempt to verify the MAC. If the MAC verifies, we have a suggestion for the premaster secret. Any false positives can be eliminated by using more packets and MACs.

One other method is to use the `Finished` packets in the SSL handshake protocol, which contain a hash of the key material together with other known data. Yet another method could be a known plaintext attack on an SSL connection.

4.4 Time complexity

Given a start time t_{start} and finished time t_{stop} then $\Delta t = t_{stop} - t_{start}$ denotes how many milliseconds the SSL handshake takes on the device we are attacking. Using the client nonce and guessing the first time seed of the PRNG takes $\mathcal{O}(\Delta t)$ time, since we can compare the first output of the PRNG with the first half of the client nonce, and similarly guessing the second time seed also takes $\mathcal{O}(\Delta t)$ time. Notice that this step allows us to synchronize with the device since finding one of the seeds tells us the local time on the device. We can use this to calculate an exact $\widehat{t}_{start}$, $\widehat{t}_{stop}$ for the key generation and $\Delta\widehat{t} = \widehat{t}_{stop} - \widehat{t}_{start}$ where $t_{start} \leq \widehat{t}_{start} < \widehat{t}_{stop} \leq t_{stop}$.

We need to guess three time seeds to generate a suggestion for the premaster secret, which have time complexity $\mathcal{O}\left(\left(\Delta\widehat{t}\,\right)^3\right)$. However, since the time seeds are generated sequentially with approximately the same amount of work between each generation, it is possible to implement the attack so that it divides $\Delta\widehat{t}$ into three time-slots and searches the first time-slot for the first time seed, and so on... Estimated time complexity for the search for the premaster secret is $\mathcal{O}\left(\left(1/3 \cdot \Delta\widehat{t}\,\right)^3\right) = 1/27 \cdot \mathcal{O}\left(\left(\Delta\widehat{t}\,\right)^3\right)$. Resulting in a total time complexity of:

$$2 \cdot \mathcal{O}(\Delta t) + \frac{1}{27} \cdot \mathcal{O}\left(\left(\Delta\widehat{t}\,\right)^3\right).$$

This way of implementing the attack might be too optimistic, considering a pre-emptive kernel, and a more conservative estimate would be:

$$2 \cdot \mathcal{O}(\Delta t) + \cdot\mathcal{O}\left(\left(\Delta\widehat{t}\,\right)^3\right).$$

4.5 Implementation

The attack was tested with a simple SSL client MIDlet written in J2ME and a simple SSL server implemented in J2SE. We used Ethereal [10] to sniff the traffic between the two programs and recover one encrypted SSL package. The attack code guessed keys and decrypted the package and checked the MAC value, utilizing methods from TinySSL [11] for key generation, decryption and MAC calculation.

The MIDlet first ran on a Nokia 6600 and a SonyEricsson P900 over GPRS. However, we were unable to recover the time from the client nonce, which led to the conclusion that Nokia and SonyEricsson have made their own implementation of the PRNG, or the seeding of the PRNG.

The same MIDlet was then tested on the emulator in Sun J2ME Wireless Toolkit 2.1 over the loop back interface, where the attack successfully recovered the shared premaster secret.

4.5.1 How long to find the keys?

On average an SSL handshake took approximately 20–30 seconds over GPRS with both the SonyEricsson P900 and the Nokia 6600; the timings include the time it took to enter user input requested by the phones during an SSL connection.

Remember that $\Delta\widehat{t}$ corresponds to the exact time it takes to execute the SSL handshake depicted in Figure 1. When we tested the attack on the emulator, it was found that $\Delta\widehat{t}$ was less than 200 milliseconds. To simulate the uncertainties associated with a real mobile phone, the initial Δt was set equal to 40 seconds, allowing for very course guesses of the start time $t_{start}$and stop time t_{stop}. The attack recovered the premaster secret in less than a second on a laptop with an Intel Pentium M processor running at 1600MHz. It is likely that the attack on an SSL connection between a real phone and a server will take more time, since all the seeds to the PRNG were created within 25 milliseconds on the emulator.

5 Conclusion

We have shown that Sun's reference implementation of SSL in MIDP is vulnerable to a key recovery attack because of a poor choice of seed to the PRNG. As far as we know the described attack has not been implemented before. However, we have not been able to find any real mobile devices with MIDP that are vulnerable to this attack. Since very little information is publicly available, we can only speculate whether the manufacturers of mobile devices have found a good solution to the difficult problem of selecting a good seed, or if an insecure technique is used to generate the seed.

Today, games are the most common applications on mobile phones. Most of the games do not communicate with a server. However, the security of SSL is essential for the creation of many client-server applications, such as online banking and e-commerce applications. All these applications need a good seed. The problem with creating good seeds could be solved by adding hardware RNG to mobile devices, which could also offer true randomness to 3rd party software developers.

References

[1] Connected Limited Device Configuration (CLDC), `http://java.sun.com/products/cldc/`, last visited: September 6, 2005.

[2] Mobile Information Device Profile (MIDP), `http://java.sun.com/products/midp/`, last visited: September 6, 2005.

[3] Ian Goldberg and David Wagner, "How Secure is the World Wide Web?," Dr. Dobb's Journal, January 1996, pp. 66–70.

[4] Dean Povey, "Wireless Java Security," `http://jdj.sys-con.com/read/37377.htm`, last visited: September 6, 2005.

[5] Stephen Thomas, "SSL and TLS Essentials: Securing the Web," Wiley Computer Publishing, 2000.

[6] D. Eastlake, S. Crocker, J. Schiller, "RFC 1750, Randomness Recommendations for Security," December 1994, `http://www.ietf.org/rfc/rfc1750.txt`

[7] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudo-Random Number Generator," Sixth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August 1999, pp. 13–33.

[8] Don Davis , Ross Ihaka , Philip Fenstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives," Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology, August 21-25, 1994, p.114–120.

[9] P. Gutmann, "Software generation of practical strong random numbers," Proceedings of the Seventh USENIX Security Symposium, 1998, pp. 243–257.

[10] Ethereal network protocol analyzer, `http://www.ethereal.com,` last visited: September 6, 2005.

[11] TinySSL, `http://www.xwt.org/javadoc/javasrc/org/xwt/util/SSL_java.html`, last visited: September 6, 2005.

ESAF - an Extensible Security Adaptation Framework

Andreas Klenk, Marcus Masekowsky, Heiko Niedermayer, Georg Carle
Computer Networks and Internet, University of Tübingen, Germany

Abstract

The Extensible Security Adaptation Framework (ESAF) is designed to make configuration more flexible and avoid protocol-dependent application development. Among its features are the seamless integration of new protocols, exchangeability of corrupt protocols and utilization of the best protocol available for communication. This choice is based upon security policies specified by the administration, the user, and the applications. The security support is end-to-end and layer-independent. It also includes transport layer and quality of service requirements since the transport layer is transparent for applications using ESAF. High-level policies provide a fine-grained support for defining requirement levels. Requirements are therefore not binary, but scalar.

Keywords Security Adaptation, Virtualization, Security Context Negotiation, XML Security Policies, Quality of Protection.

1 Introduction

During the last decades the number of computers drastically increased as well as the demand for communication. This trend raises the issue on how to handle and guarantee security in those systems, considering their vast complexity. Traditional manual security management approaches reach their limits of applicability. Huge human resources are necessary to setup the systems to communicate securely and keep them running in the face of a constantly changing network environment. This administrative overhead makes it difficult to introduce new innovative services into the network and sometimes even impossible to use them at all.
As more and more computers are introduced into the network the problem gets even worse. Especially in ubiquitous scenarios where computers and other electronic helpers are embedded everywhere in the environment, it is not feasible to make a manual configuration for any communication and any device. The complexity and the unforeseeable number of possible interactions and communication interfaces require a self-configuration capability for the security and communication functions of the devices. A solution to this problem is offered by the virtualization of the communication mechanisms.

1.1 Technologies

During the last decade numerous security technologies evolved which are able to guard the user from attacks and intrusions. Security protocols like IPSec, SSL, TLS, SRTP are thoroughly evaluated but they require a careful configuration to guarantee security. Users without expert knowledge are often unable to make the right decisions and introduce severe vulnerabilities. Most large-scale networks that can be found in companies and universities are insecure due to misconfiguration of a few individual workstations. We propose a framework that can take the burden of making the right configuration decision away from the user and still offer experts the flexibility to tune specific configuration details.

1.2 The Configuration Problem

Protocols like IPSec suffer under the configuration complexity as Bruce Schneier[1] already stated in his survey on this protocol "Even though the protocol is a disappointment – our primary complaint is with its complexity – it is the best IP security protocol available at the moment." Hence, the challenge today is not to design a secure protocol but to configure a protocol to be secure!

Security configuration today can be done either at system level, for example by managing security policies or associations for IPSec, or at application level. Configuration at the system level is usually done to reflect the highest demand for security and therefore must use the protocol and cryptographic algorithm which offers the best security guarantees. The choice to use the "best" security protocol reflects only badly the true security needs at a given time. Which level of security is required depends a lot on what the user wants to do. During a session only certain data need strong protection. Other data like, for example, background images require less security. Another example for reduced security requirements is communication in trusted environments. The *security requirements* are volatile and change frequently even during the runtime of an application. It is obvious that security configuration needs to be done at a finer granularity and must be dynamic in contrast to the state today with static configuration at system level.

Some applications try to deal with the configuration of security protocols on their own and are therefore forced to support the protocol interface. Such applications break if old protocols become unavailable even if new security protocols are introduced into the system as a replacement. The ESAF deals with this problem by offering the application virtualization of communication. Virtualization makes it possible to introduce new communication and security protocols transparently. The application utilizes an abstract communication interface to hide the particular protocol. However, in case of ESAF the application can still control the communication context by specifying *high level policies*.

In contrast to the needs of applications an administrator prefers to have a single control point where a security policy can be specified and enforced. ESAF can provide this by using obligatory *system policies* which define the minimal security level. These system policies defined by the administrator must always be fulfilled for any communication. Application specific policies can only request higher security levels. As an option the ESAF framework can keep an audit of the communication contexts and inspect the applied configurations in detail. This audit helps to detect possible attacks and provides a mean to check the correctness of the decisions of the ESAF system.

The heterogeneity of today's network topologies and the vastly differentiating available security protocols at the end systems introduce additional challenges for protocol configuration. The prime task is to identify possible security configurations supported at the end systems. In a next step a consensus must be reached about the configuration that meets the requirements for the hosts best. This exchange during the *security context negotiation* must be sufficiently secured and designed with care to reduce the risk of vulnerabilities.

1.3 Structure of the document

The structure of the document is as follows. First, we present Related Work in Section 2. Then, the design goals which led to the development of the Extensible Security Adaptation Framework (ESAF) are discussed in Section 3. In Section 4 we finally introduce ESAF with its architecture and components. We deal with security considerations in Section 5 and finally summarize our approach in the Section 6.

2 Related Work

Several projects strive to provide flexibility for communication. We take a closer look at some research endeavors. The terminology of [2] helps us to categorize the systems.
The literature shows two main directions how to solve the adaptivity issue. One is to provide coordinated distributed adaptation functionality inside the networks like Yarvis proposed with the Conductor framework[3]. Conductor intercepts the communication and redirects it to the framework. Agents inside the network adapt the data streams according to a plan made by the Conductor framework at the data source. Although conductor provides some flexibility by altering the data streams it only provides rudimentary security services. Unspecified encryption algorithms are used to provide security services.
The other approach is to support the virtualization at the end-systems. The Generic Security Service Application Program Interface(GSS-API) [4] is standardized by the IETF and aims at providing a generic interface to use end-to-end security independent of the security mechanism and the communi-

cation protocol. The GSS uses tokens for the establishment of the security context and the protection of the communication. The application is responsible for the token exchange via a suitable transport protocol. GSS-API is well-established as an interface for authentication and key exchange. However the token based approach introduces a GSS dependent overhead for each message. Secure communication protocols, for instance SSL/TLS, cannot be accessed through this interface due to the peculiar message protection mechanism. Quality of protection can be achieved at the prize of loss of transparency of the underlying security mechanism. Furthermore, the API only covers security services and cannot provide for an abstraction of communication mechanisms. The communication and hence the context negotiation must be handled by applications itself.
The Common Data Security Architecture (CDSA)[5] is such an end-to-end cryptographic framework for creating security-enabled applications for client-server environments. The Framework implements its own security services at application layer to protect data transfers at run time. Hence it is not possible to use standardized, widely-used security protocols at different protocol layers. Policy based security configuration is not considered.
The Iceberg project[6] is similar to the Transformation, Aggregation, Caching and Customnization (TACC) architecture[7]. Thus it is a pure proxy system, specially designed for mobile devices with few resources. Proxy systems provide their services to legacy applications but fail to adapt to specific requirements of the applications. Secure protocols like IPSec or SSL are not supported and it is difficult to adapt the project to utilize these protocols.
The Chisel Framework[8] can be described as a reflective, application- and user-aware, end-system-based architecture. It is possible to specify adaptation policies to influence the adaptation process. Their reflective approach lets them change technical behavior without modifying their adaptation system. Security concerns were not in focus of this project and hence the framework fits only badly for the provisioning of security services.
In [9] Stiller introduced the DaCaPo++ system which uses protocol composition based on module chains. It is an application-aware system for the use on end-systems. Applications can influence their Quality of Service by selecting suitable protocols among a set of predefined protocols. This makes applications dependent on the supported protocols and future changes may cause problems with legacy applications. Security functionality is provided by proprietary security modules, thus security of the communication is difficult to proof because non standard protocols are utilized.
The Celestial Project[10] was developed at the North Carolina State University and is an application-aware and protocol-oriented security policy management system. The core component is the Security Management Agent(SMA), which performs the security protocols configuration and exchanges configuration messages with other Celestial nodes for connection setup. The SMA is implemented as a kernel module and offers additionally a simple socket interface for applications. SMNAs also reside at middleboxes on the data path and influence the connection setup by adding their own security policies to the message. Security services can be provided by those middleboxes and are therefore not end-to-end. Thus a high level of trust is required in the unprotected link with the middleboxes and of course the trustworthiness of the middleboxes must be assured. Celestial introduced a proprietary protocol called ISCP for the exchange of configuration messages and uses undisclosed algorithms and encryption functions for protecting the message exchange. The management of the security policies is done at the application level without an option for the administrator to influence or confirm the configuration. Therefor Celestial lacks the capability to enforce a system wide security policy. If a critical bug of a security protocol is exposed Celestial cannot easily adapt its policies.

3 Architecture Design Goals

While trying to provide communication and security virtualization it is important to state the requirements such a system must fulfill. Two basic principles are always visible in the design: a) be as autonomous as possible during connection establishment b) provide control at all stages and transparency of the state for the system.

We consider authentication to be out of scope for this document. That it can be provided with other means, say out-of-band or using a PKI infrastructure.

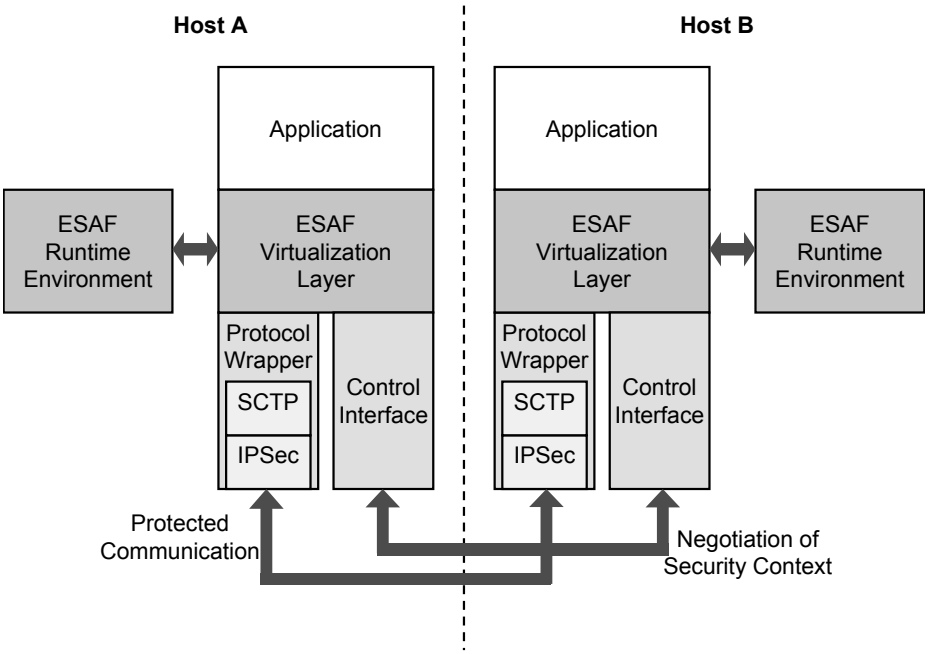


Figure 1: Extensible Security Adaptation Framework

- Self-Configuration: The communicating entities must find a configuration set that fulfills at least the minimal security and communication requirements of all participating entities. A consensus must be reached about which protocols and which configuration fits the requirements best. Self-configuration must be performed in a secure manner.

- Intrusion Handling: Security mechanisms sometimes comprise flaws and if these are disclosed a responsible administrator must react. Currently, one has to wait till an update becomes available to fix the flaw. It is usually no option to disable the protocol because a lot of applications would stop working. The mechanism must be able to disable protocols without harming the functionality of the system.

- Communication Virtualization: Communication interfaces must be abstracted and generalized to allow the exchangeability of underlying protocols. The abstract interface must provide all functionalities necessary for communication. It shall be possible to take control over the communication but must be able to work autonomously with self configuration.

- Context Adaptation: A secure perimeter requires maybe less security measures than a hostile environment. The computational resources that are available determine what kind of protocol is applicable at all. The context determines the communication requirements.

- Large Degree of Control: Different stakeholders take interest in the configuration of the connection. It must be possible for all participants to express their configuration demands. Administrators want to enforce a minimum security level whereas users take a large interest in the performance of the system. Applications can optimize the performance by adapting the security to the actually performed task.

- Ease of Control: The control must not only be possible but also be easy to implement. The method to express the requirements must be straight forward to formulate and to modify. The requirements

must be human understandable to allow administrators to make sure that the security context offers adequate protection. It must be possible to trace the state of the system during configuration for each message exchange.

- System Enhancement: Systems undergo many changes in the course of their lifetime. Protocols are added and configuration changes frequently. The installed applications should be able to take advantage of new system capabilities without being redesigned.

- Security: The architecture itself must be designed with security in mind. Possible weaknesses of the architecture must be detected to avoid introducing new security holes in the system.

4 Extensible Security Adaptation Framework

The ***E**xtensible **S**ecurity **A**daptation **F**ramework (ESAF)* was designed to provide applications with a novel interface that provides virtualization especially for security. Applications can take control over the security protocols without the need to know anything about the parameters and interfaces of the protocol at all. The decision which protocol and which configuration should be used has to be derived directly form the security and communication requirements of the different stakeholders in the system: user, application, system, communication partners and many other instances determine the configuration needs.
These requirements are defined in *high level policies*. These policies describe in an abstract form the required security and communication parameters. The ESAF can map these *high level policies* internally onto *system capabilities policies* to derive the particular configuration that must be applied to the protocols.
Virtualization offers a compelling solution to solve two problems at once. The security and communication requirements must be formulated independently from a particular protocol, but they must still be expressive enough to state the requirements in necessary depth. The usage of the security protocol must be generic enough to replace the protocol in the system without the need to reconfigure or rebuilt the existing applications.
Exchangeability of the protocols only works if the necessary configuration does not introduce hard dependencies to a specific protocol. The socket interface achieves today a certain level of abstraction for applications. It is not possible to exchange the "old" protocol with a new innovative and unforeseen protocol without breaking the application. This is especially true in cases when, for example, security was formerly provided through SSL, but now IPSec is the only secure communication option. Most often such an exchange is not possible at all.
Our solution is to introduce a similar interface to the standard socket interface but offer generic configuration with *high level policies*.
ESAF not only acts system local, but can also assist the applications with the establishment of secure connections. A security context negotiation is performed during connection setup to determine the requirements of the communication partners. *High level policies* are exchanged and their intersection leads to a list of supported and required protocols for the connection. A choice can be made then, what the "best" protocol for this session will be.
In order to proof the concept we already implemented basic mechanism of ESAF for Linux in C++. The ESAF environment is still under development and misses central functionalities like mechanisms for key exchange or the use of certification authorities. For the present we made the assumption that the entities possess means to authenticate their communication partners.
The next subsections will highlight the individual specialties of the different tasks of the ESAF approach.

4.1 Architecture

The layered architecture was designed to achieve communication virtualization and configuration transparency for the application. Consequently the *ESAF Virtualization Layer* is the core of the framework. This layer is the generic communication interface for the application. It accepts *high level policies* as

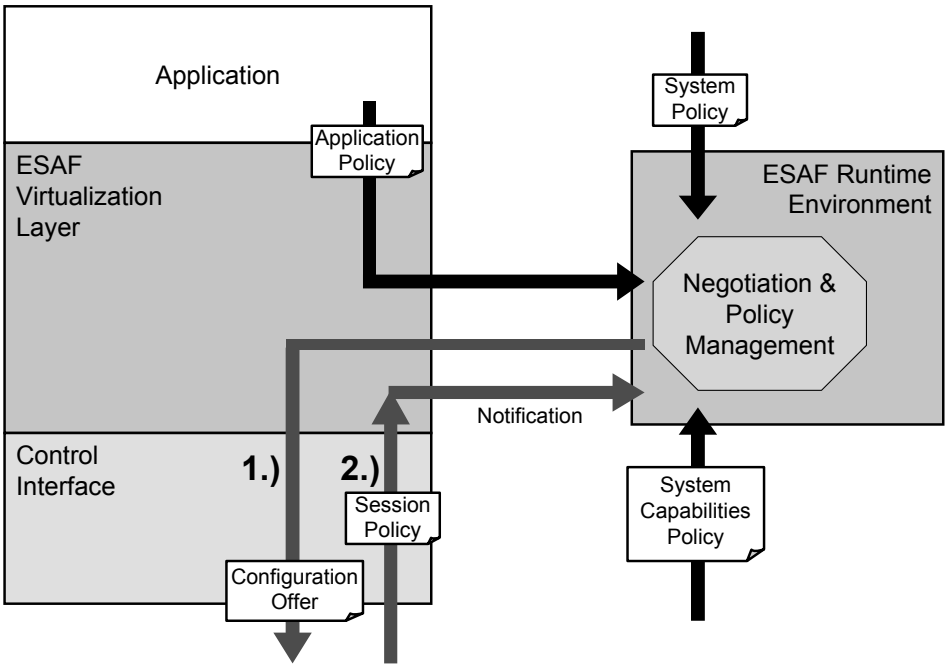


Figure 2: ESAF Policy Processing

configuration requests and chooses autonomously appropriate communication setups. The application utilizes the *Generic Socket Interface* of the ESAF library to carry out the communication then.
The Virtualization Layer uses internally the generic *Protocol Wrapper* interface. This wrapper also comprises a generic interface and takes *system capabilities policies* for configuration. The wrapper allows the ESAF to easily introduce new protocols into the system. The *system capabilities policies* allow to configure the protocols in depth while still being able to provide interchangeability of the particular protocols.
The *ESAF Runtime Environment* is designed as a daemon running constantly in the system. One functionality of the runtime is to make protocol configurations which require root privileges, for example of IPSec. Another aim is to keep the *ESAF Virtualization Layer* comparable lightweight and implement policy related functionalities here. Retrieval of the *high level policies* is such a functionality whereas the daemon can keep track of the currently active security contexts.
The *Control Interface* is part of the application. The application is responsible for accessible ports from outside of the system and runs the control interface there. A remote host, willing to connect, initially negotiates a security context for the communication link, before data exchange can commence. The *Control Interface* allows the negotiation of security contexts during connection setup and the modifications of the context during runtime.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE esaf_high_level_policy SYSTEM "esaf_high_level_policy.dtd">
<esaf_high_level_policy>
  <security_requirements>
    <message_authentication>
      <minimum>5</minimum>
      <ideal>7</ideal>
    </message_authentication>
    <data_integrity>
      <minimum>7</minimum>
      <ideal>10</ideal>
    </data_integrity>
    <confidentiality>
    ...
    </confidentiality>
    <traffic_flow_confidentiality>
```

```
    ...
    </traffic_flow_confidentiality>
    <non-repudiation>
    ...
    </non-repudiation>
    ...
    ...
  </security_requirements>

  <communication_requirements>
    <connection_type>connection-oriented</connection_type>
    <reliability>reliable</reliability>
    <sequencing>yes</sequencing>
    <error_control>yes</error_control>
    <performance>
      <minimum>5</minimum>
      <ideal>10</ideal>
    </performance>
  </communication_requirements>

</esaf_high_level_policy>
```

Listing 1: High Level Policy

4.2 Requirements Description Language - RDL

We defined the ***R**equirements **D**escription **L**anguage RDL* to pass configuration requests along. The public interface of ESAF accepts *High Level Policies* whereas internally a *system capabilities policy* is used to describe the installed protocols.
We decided to use XML as a policy language, because it is easily extensible. Different versions of the ESAF can choose to ignore sections they do not understand. This is of course only possible if the section provides information marked as optional.

4.2.1 High Level Policies

It is important that these policies are truly protocol and configuration independent and describe the full range of requirements in a general manner. For such a policy language it is important to identify a set of language constructs and keywords that are able to express the full range of communication requirements.
The security of a communication link is usually judged based on the degree it provides the following characteristics: authentication, integrity, confidentiality and non-repudiation. We identified two more parameters of high importance for secure communication: reliability and performance.

Security protocols are not equally optimized for all identified parameters. The level of security varies depending on key lengths and utilized encryption algorithms. The performance of the algorithm may also be an important factor, imagine a resource constrained device like a handheld computer. This tradeoff between security and perfromance is also termed Quality of Protection(QoP) [11]. These thoughts led us to the decision to attach a scalar value to each service requirement to express the importance of the parameter on a scale between 0 and 10. The value 0 would mean "no importance" while 10 would give the parameter the highest priority. To differentiate even further we introduced the notion of *minimum* as a knock out barrier and *ideal* as the desired configuration value.

The security requirements are kept apart from the communication requirements in the policy. Inside the *security_requirements* element each parameter is stated with its *minimum* and *ideal* value. This element describes the typical security requirements as stated above. The tag *communication_requirements* encloses parameters like performance or reliability. Listing 1 depicts an example of such a high level policy.

High level policies which reside at the same system can be joined by the ESAF. The application specifies an application dependent high level policy as well as the administrator can specify a system policy. These policies can easily be unified because they refer to the same system capabilities policy. The algorithm is straight forward, all minimum elements of the policy are compared and always the higher value is kept.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE esaf_system_ability SYSTEM "esaf_system_capability.dtd">

<esaf_system_capability>
  <supported_security_protocols>
    <security_protocol id="ipsec">
      ...
    </security_protocol>
    ...
  </supported_security_protocols>

  <supported_communication_protocols>
    <transport_layer>
    ...
    </transport_layer>
    <network_layer>
    ...
    </network_layer>
  </supported_communication_protocols>

</esaf_system_capability>
```

Listing 2: Excerpt of System Capabilities Policy

4.2.2 System Capabilities Policy

High level policies helped the stakeholders express their requirements. Now the question is how these high level policies get mapped onto particular protocols providing the desired properties. Our approach is to use *system capabilities policies*. These policies describe the available communication and configuration means of the system.
The basic elements of this format are the security protocols and the communication protocols. Here are the descriptions what a protocol can do and how it must be configured. If critical bugs in a security protocol are disclosed, the administrator can easily disable the corresponding entries in the *system capabilities policy* or degrade the security level. This will allow that the applications to use more secure protocols for their connections. The user will not even notice the change and the application does not have to bother.

```
...
<supported_security_protocols>
  <security_protocol id="ipsec">
    <supported_security_services>
      <confidentiality>
        <encr_algorithm id="aes128-cbc">
          <key_length>128</key_length>
          <security_level>9</security_level>
          <performance>6</performance>
        </encr_algorithm>
        <encr_algorithm id="aes256-cbc">
          <key_length>256</key_length>
          <security_level>10</security_level>
          <performance>4</performance>
        </encr_algorithm>
        <encr_algorithm id="twofish128-cbc">
        ...
        </encr_algorithm>
        ...
      </confidentiality>
```

```
      <message_authentication>
        ...
      </message_authentication>
      <non-repudiation>
        ...
      </non-repudiation>
      ...

    </supported_security_services>
    <required_communication_protocols>
      <!--protocols, which can be used with this security protocol -->
    </required_communication_protocols>
  </security_protocol>
  ...

</supported_security_protocols>
...
```

Listing 3: Security Services in the System Capabilities Policy

The *system capabilities policy* describes in detail the possible configuration options for each security protocol and a system local security rating. We call policies at this detail *low level policies*. These elements correlate directly with the elements of the high level policy. It is now possible to determine all possible encryption algorithms in the system which can provide a certain security service, for example, confidentiality.
Listing 3 shows an excerpt of the security section of an example *system capabilities policy*. Here are some supported configuration options for IPSec security services defined. This particular part shows some available encryption algorithms. Note how the key-length of 256bits for the AES algorithm increases the security level to 10 but decreases the performance level to 4. If implementations get more efficient or algorithms are considered less secure it is easy to change this policy to reflect the changes. The *security_protocol* tag can contain special information for each algorithm on how to configure the algorithm. In this example it is the *key_length* element.
The system must be aware of the dependencies between the different protocols. Each security protocol contains the section *required_communication_protocols* which determines in what combinations the protocol can be used.

4.3 Communication Interface

The *communication interface* provides abstraction of the actual protocols. Virtualization is accomplished by using *high level policies*. The interface itself must be general enough to allow the exchangeability of the underlying protocols but must must not limit the way a protocol can be used. The level of abstraction of the BSD socket interface[12] has already proved itself. The Socket++ interface[13], we chose to mimic, is an evolution of the BSD socket interface and tries to enhance the ease of use for programmers. We added the method *negotiate_policy* to the interface for configuration by the means of *high level policies*. This method performs internally several steps to establish an agreement about the configuration of the communication link as described in the next section 4.4. After the agreement is reached it establishes the communication with these parameters.

```
class Secure_Connection
{
private:
  void negotiate_policy(const std::string &from, const std::string &to,const std::
    string &policy);
  ...
public:
  inline Secure_Connection(const std::string &from,const std::string &to, const std::
    string &policy) {
    ...
    this->negotiate_policy(from, to, policy);
  }
  ~Secure_Connection();
```

```
    void send(const std::string &data);

    std::string receive();

    void disconnect();

    void renegotiate_policy(std::string* policy);

    ...
};
```

Listing 4: An Excerpt of the Secure Connection Class

The concept of using *high level policies* for configuration allows to extend the functionality of the framework without changing the interface. Applications must not be rebuilt to include these new functionalities. The extensible structure of the XML parameter will allow us to support *low level policies* in the future. These policies contain additional configuration options at the detail level of the *system capabilities policy*. One interface can be used then to provide loose or close control depending on the needs of the application.

4.4 Security Context Negotiation

When a connection has to be established it is necessary to perform a *security context negotiation*. The participants must agree on a set of possible protocols and a selection must then be made which protocols to use. At the moment ESAF supports only end-to-end communication for two participants.
Figure 3 shows the sequence of the negotiation. First a *connection request* must be made in step 1) by A. For this reason the ESAF at host A joins the high level policy of A with the system capabilities policy of system A. It tries to determine a set of protocols and configurations meeting the requirements. Only the entries which possess a security rating of equal or better then the minimum requirement specified by the *high level policy* will be included and form a special policy, the *Configuration Offer*. The generated Configuration Offer is now sent to host B inside the connection request. As an option the high level policy can be included to inform B about the utilized security ratings for host A.

After host B received the request, it starts processing it together with its local policies. First, it must evaluate its own high level policy provided by its server application and join it with its system capabilities policy to get the locally available configuration options. Then, the algorithm starts to determine the adequate configuration taking the minimum and ideal values into account. If the configuration is found the connection is prepared and a *Context Prepared* message is sent to A in step 2), containing the *Session Policy*.

In case host B is not able to find a possible intersection it will send an *Agreement Failed* message back to A and attach its own high level policy and system capabilities policy. In the failure case host A could try to adapt its policies to find at least one possible communication link with host B. This modification should not be done automatically but through human intervention because it could lead to degradation of the security level.

After host A received the *Session Policy*, the connection setup of the protocol can start with the exchanged configuration information as shown in step 3). Furthermore, the application can always perform a runtime modification of the communication setup by renegotiating the parameters, as shown in step 4).
Of course, the ESAF middleware on host A must verify that the selected configuration is consistent with the request it sent in step 1). This must be done to detect manipulation attempts. However, it is a matter of mutual trust that the two hosts do not misuse their various decision options. Authentication of messages is very important for the negotiation process to avoid manipulation attempts of the messages.

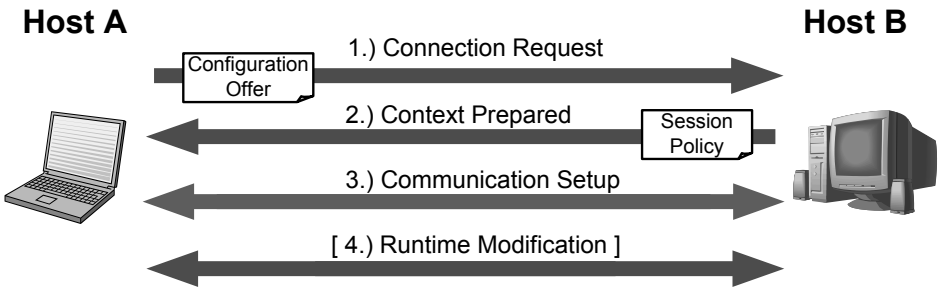


Figure 3: Security Context Negotiation Sequence

5 Evaluation

In this section we briefly evaluate the security aspects of the ESAF system. A thorough analysis and solutions to reduce possible problems are future work.

With respect to the security of the protocols that the system will use, we can state the following. ESAF does not use self-designed new security protocols, but uses well-known and well-evaluated protocols like IPSec or SSL and their implementations in current systems. This is not expected to be a weakness in ESAF.

More important for a future evaluation of ESAF is the analysis of the impact of and threats to ESAF itself. ESAF could be the point of attack. This could be due to weaknesses in the ESAF implementation. The *control interface* could be vulnerable because it accepts connections and could be misused for buffer overflows or DoS attacks. To limit this threat the *control interface* is active in the application and not in the runtime. Configuration weaknesses could be introduced by an attacker when she is able to modify low-level security policies system locally and propagate an insecure protocol as secure. Other possible problems could arise from the interaction of ESAF with security protocols.

We will discuss these points in detail and propose possible solutions in the future.

6 Conclusion

This document describes a framework for the virtualization of secure communication configuration called Extensible Security Adaption Framework. Applications using the framework are unaware of the utilized security mechanisms and the complex configuration thereof. They must only state their communication and security requirements and the ESAF will autonomously select and establish the best matching communication setup. Protocols can easily be introduced into the system or disabled if a critical vulnerability of a certain protocol is discovered. Because the ESAF virtualization hides the protocol stack completely from the application, it does not matter anymore at which layer security functionalities are provided.
Abstraction is reached by specifying two human-readable policy formats. One high level format describes the requirements whereas the other format describes particular protocols and configuration options at the system level. These XML policies are not only used internally but also for the connection setup. The parties wishing to establish a link exchange security policies and leave the connection setup up to the ESAF.
Although the implementation is under progress and the concept advances there are still open issues. First of all a thorough security investigation must be undertaken. Then issues like authentication and key exchange must be supported by this framework.
Our conclusion is that the ESAF approach can provide security virtualization and allows self-configuration. Of course, the benefits of ESAF are currently only available to applications which support the framework. However, the prospect of autonomous and secure self-configuration of communication is tempting.

References

[1] N. Ferguson and B. Schneier, "A cryptographic evaluation of IPsec," tech. rep., 3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA, 2000.

[2] N. L. Nesrine Yahiaoui, Bruno Traverson, "Classification and comparison of adaptable platforms," 2004.

[3] M. Yarvis, P. Reiher, and G. Popek, "A reliability model for distributed adaptation," 2000.

[4] J. Linn, "Generic security service application program interface, version 2." IETF, 1997.

[5] T. O. Group, "Common security: Cdsa and cssm, version 2 (with corrigenda)," 2000.

[6] R. H. K. Morley Mao, "A framework for universal service access using device ensemble."

[7] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based scalable network services," in *Symposium on Operating Systems Principles*, pp. 78–91, 1997.

[8] J. Keeney, "Chisel: A policy-driven, context-aware, dynamic adaptation framework," 2003.

[9] B. Stiller, C. Class, M. Waldvogel, G. Caronni, and D. Bauer, "A flexible middleware for multimedia communication: Design, implementation, and experience," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1614–1631, Sept. 1999.

[10] C. Xu, F. Gong, I. Baldine, L. Han, and X. Qin, "Building security-aware applications on celestial network security management infrastructure.," in *International Conference on Internet Computing*, pp. 219–226, 2000.

[11] C. E. Irvine, T. E. Levin, and T. D. N. et al, "Overview of a high assurance architecture for distributed multilevel security," *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security T1B2 1555 United States Military Academy, West Point, NY, 17–19 June 2002*, 2002.

[12] S. J. Leffler, M. K. JcKusick, M. T. Karels, and J. S. Quarterman, "The design and implementation of 4.3 bsd unix operating system." Addison-Wesley, 1989.

[13] G. Swaminathan, "C++ socket classes (1.12)," 2004.

[14] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," Tech. Rep. HPL-2002-57, HP Laboratories, Palo Alto, March 2002.

[15] M. Yarvis, "Challenges in distributed adaptation," 2000.

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml) 1.0 (third edition)," 2004.

[17] S. Naqvi and M. Riguidel, "Vipsec: Virtualized and pluggable security services infrastructure for adaptive grid computing," 2004.

[18] A. Mukhija and M. Glinz, "Casa – a contract-based adaptive software architecture framework," 2003.

[19] J. Li, M. Yarvis, and P. Reiher, "Securing distributed adaptation," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 38, no. 3, pp. 347–371, 2002.

[20] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet security association and key management protocol (ISAKMP)." Internet Draft (draft-ietf-ipsec-isakmp-08), 1997.

[21] A. Keromytis, "Some ipsec performance indications," 2001.

[22] S. Rao, M. Formanek, and M. Riguidel, "Prospect of new concepts in securing the cyberspace: Virtual paradigms, infospheres and pervasive computing," 2004.

[23] N. Yahiaoui, B. Traverson, and N. Levy, "Classification and comparison of adaptable platforms," 2004.

Transparent Anonymization of IP Based Network Traffic

Lexi Pimenidis and Tobias Kölsch*

RWTH-Aachen University,
Computer Science Department Informatik IV,
Ahornstr. 55, D-52074 Aachen, Germany

{lexi,koelsch}@i4.informatik.rwth-aachen.de

Abstract

This paper presents an approach to enhance the usability of anonymizing networks by creating a virtual anonymous IP-network. An anonymization layer is hidden behind the operating system and transparently reroutes all network traffic. This enables the user to perform all IP based network access without adaption of the used programs. The provided IP can be used to provide services on the network anonymously. The generality of the approach also enables typical IP based services like DNS to be provided to the anonymous hosts.

1 Introduction

The distribution and vast popularity of global networks like the Internet allow end users to communicate world wide. Unfortunately most end users and even professionals do not pay enough attention to secure their systems and network traffic against intrusions, eavesdropping or data analysis. It is difficult to cause awareness to these problem, as the threats are often invisible. As a result sensible data is send carelessly through the network. However, not only the content of a message contains sensible information that has to be protected, e.g. by encryption. The address information that is included in most network transmissions can also contain sensitive information. So sending encrypted data in regular intervals to the web server of a certain political party leaks information of political affiliation.

To counter this leakage of information several anonymization techniques have been proposed, developed, and deployed. E.g. Mixminion [DDM03] anonymizes email traffic on different network layers, freenet [CSWH00] provides an anonymous storage and publication system, JAP [BFK00] and MorphMix [RP02] are anonymizing HTTP-proxies, and Tor [DMS04] provides a Socks proxy and anonymizes TCP/IP streams on the transport layer.

A problem common to all of these techniques is that they are developed for a special purpose. As a result they can not be used by all programs that use the Internet to perform day to day work.

Tor is the one notable exception. The general Socks4a interface it provides, gives all application that support this general proxying protocol the ability to anonymize their traffic. Applications not supporting it can be started using the `torify` command, which redirects calls to the networking functions of the operating system. However, this approach still bears some problems. First, the configuration overhead is significant, as it has to be figured out for every application if it supports the Socks4a protocol and the application has to be configured to use it. This is complex and error prone, especially as the user usually gets no feedback, whether the traffic is really anonymized or not. Second, the `torify` approach does not prevent the torified programs to issue dynamic name service (DNS) requests, that disclose the users target of interest.

Another problem with Tor is that the back channels it provides are not adequate for general usage. The setup of such channels has to be done in the Tor configuration file, and is as such bound to a restart

*Mr Pimenidis and Mr Kölsch are funded by the European Commission's 6th Framework Program. Thanks to Dr. Doğan Kesdoğan and George Danezis for their invaluable help

of the anonymizer. Another deficiency of the present approach is that the anonymous domain names are represented by a cryptographicaly hashed URL that is usually difficult to remember and can not be accessed without using Tor.

The communication suite formerly known as "Tarzan"[FM02] would qualify for a piece of software that actually does provide the envisioned ease of communication. Alas, it has to be compiled by the user, is still in experimental status and primarily supports only FreeBSD. Since we're looking for a solution that can be accomplished with software that is known to work, and does so on a variety of operating systems, we have to choose other components.

Since the configuration needed to communicate anonymously with existing techniques is for experts only, the set of users is small. This reduces the anonymity set provided by a technique. As a result the users are not as well protected as they could be and they are possibly prone to stigmatization for using anonymization techniques.

Our solution to this problem are *Virtual Anonymous Networks* (*VAN*). They introduce an anonymization layer that is hidden behind the operating systems and allow transparent access to the global network through an anonymizing overlay network. The approach provides users with truly anonymous IP addresses, that can be used for sending and receiving IP packages. As a result all their traffic is hidden behind this temporary address. The user side configuration is by this reduced to setting up just one system instead of having to adapt every program.

We built the proposed architecture using publicly available tools and evaluated its functionality to ensure practicability and usability. The resulting network allowed easy, anonymous, TCP/IP based peer-to-peer communication for all applications without additional configuration.

In section 2 we will describe VANs in more details. Some informations about the proof-of-concept network we created to verify our ideas are given in section 3. The results will be discussed in detail in section 4 and an outlook will be given in section 5.

2 Virtual Anonymous Networks

We propose to deploy an IP based overlay network that is able to anonymize data streams, but looks like any other IP based network to the user. Clients can connect to this network, like they dial up to an Internet service provider. But the IP address they get, is not linkable to them. They can then send and receive data anonymously over this virtual network. Additionally it is possible to add typical service functionality to this network, like DNS. The IP communication itself is not restricted by the approach. However, the operators of the network may introduce restrictions, e.g. using a firewall, to control the traffic flow.

The overlay network consists of two additional network layers. An anonymizer on the lower layer hides the host's real IP address against the VAN servers. We chose Tor for this purpose. The second layer is provided by a virtual private network (VPN) that provides the anonymous IP addresses. OpenVPN [ope04] was chosen to provide this for our implementation. Both tools are installed on every host that should get an anonymous address.

Using the anonymization layer, the VPN clients are then able to connect to one of the VPN servers and receive an IP address from the servers private IP range. Consequently the client can communicate anonymously with other clients in the same virtual network. If different virtual networks are interconnected, all clients can communicate with each other, without knowing the other user's identity.

After our approach is set up every client computer will have at least two IP addresses. The first one is that of the real network device. All traffic that uses this address is not automatically protected and will be routed directly to its destination. The second IP is that provided by the VPN server. Messages that are sent or received by the host on this address can not be linked to its real IP address. It now depends upon the routing of the operating system, which traffic will be anonymized and which will be send directly into the Internet. One possible configuration could send local traffic directly, while sending traffic to remote hosts through the VAN.

In the following we describe the used tools and give a more concise description of the implementation of our method.

2.1 The Anonymizing Overlay Network

As the users host should be anonymous towards the VAN server, all traffic to him is proxied through an anonymizer. As pointed out earlier we chose Tor [DMS04]. Tor has the most general approach to anonymize network traffic and it is capable of relaying real time TCP traffic through untrusted networks. Another advantage of Tor is that it can be run on various operating systems, e.g. Windows and Linux. It also does a basic form of network load balancing because the clients choose their paths through the Tor network at random.

Tor in its current form is not designed to cope with global observers or similar threats often considered in actual research [KAP02, Dan03, KP04, MD04]. But it does a good job in hiding information from locally restricted attackers, as network administrators or other local authorities.

The Tor network consists of Tor servers and clients. A client that wants to communicate, builds a tunnel that randomly passes through some of the servers. TCP/IP packages of a client application that are to be sent to a host in the Internet, are relayed through this tunnel. The last Tor server then proxies the connection to the final destination and sends the answers back through the tunnel. The way in which the tunnel is instantiated guarantees that each server only knows his predecessor and his successor. So only the first server knows the client and only the last server knows the final destination. Also the appearance of the sent packages is changed at every hop such that two non neighboring Tor servers cannot link individual packages to each other based on their appearance.

The Tor client provides a Socks4a interface for applications. This protocol is capable of handling unresolved URLs, such that IP addresses do not have to be retrieved before a TCP connection is set up. As presented in Section 1, applications that do not support proxying with Socks4a can be made to compatible using the `torify` command. However explicit DNS resolution undermines the gained anonymity.

2.2 The VPN Overlay Network

A VPN server allows multiple clients to connect. Each client receives an IP from a given range, and all traffic on this virtual overlay network is then either forwarded to the participating clients or masqueraded to outside IPs (masquerading is also known as *network address translation*). After the initialization of the VPN, the users are able to communicate with each other, as if they were on a single LAN sharing IPs from the same range, although they actually are not and their external IPs are from completely different networks.

Under normal conditions each client needs a unique certificate to authenticate himself to the server. Obviously we do not want to give away a client's identity this way, so we configure the server to accept multiple connections with the same certificate and publish a single client certificate that is needed to connect to the server. On the other hand, it is desirable for the servers to have each a unique certificate so no attacker can impersonate them.

Although a single VPN server would be sufficient to protect anonymity, we want multiple VPN servers for scalability reasons. This leads to a routing problem for traffic between different VPNs. A simple solution is to create a fully connected network between the VPN servers. This can be done by interconnecting the VPN servers to each other. Traffic to the other networks is then directly forwarded to the target VPN server, who then forwards the data to its destination. If the amount of VPNs increases some more dynamic routing method for packages between the private networks might be necessary.

We use OpenVPN 2.0 [ope04] as virtual private network solution. This choice was made because it runs on most major platforms, it is easy to set up, and it is easily adapted for our purpose.

To protect the users DNS queries, a DNS server can be installed on the VAN server. We used the popular `bind` [bin] server for this.

2.3 Connecting and Using the Network

In order to use our *virtual anonymous network (VAN)*, the user needs to install a Tor-client and an OpenVPN-client. After he starts the Tor client, he connects the VPN client to a VAN server through the Tor network. The server provides him with an IP address that can be considered anonymous. The VPN client then replaces the default gateway on the user's computer with the VAN server. He also configures the computer to redirect all DNS-queries to the VAN's DNS server. As soon as the second connection

is established, the user's Internet traffic is effectively anonymized without the need to further configure applications.

It is even possible to configure only the gateway of a local area network. This anonymizes the entire networks outgoing traffic without the need to configure each single computer. The gateway is set up as described above. Though this reduces the setup effort, this solution suffers from the typical effect of masquerading: inbound connections on the anonymous IP-address of the gateway can not easily be targeted to the single machines in the network. Additionally, local observers are able to completely compromise the system.

3 Implementation

To build the proposed architecture we used a closed network of nine networked computers. Five of them were running as Tor and VAN servers. The remaining four were used as clients as shown in Figure 1.

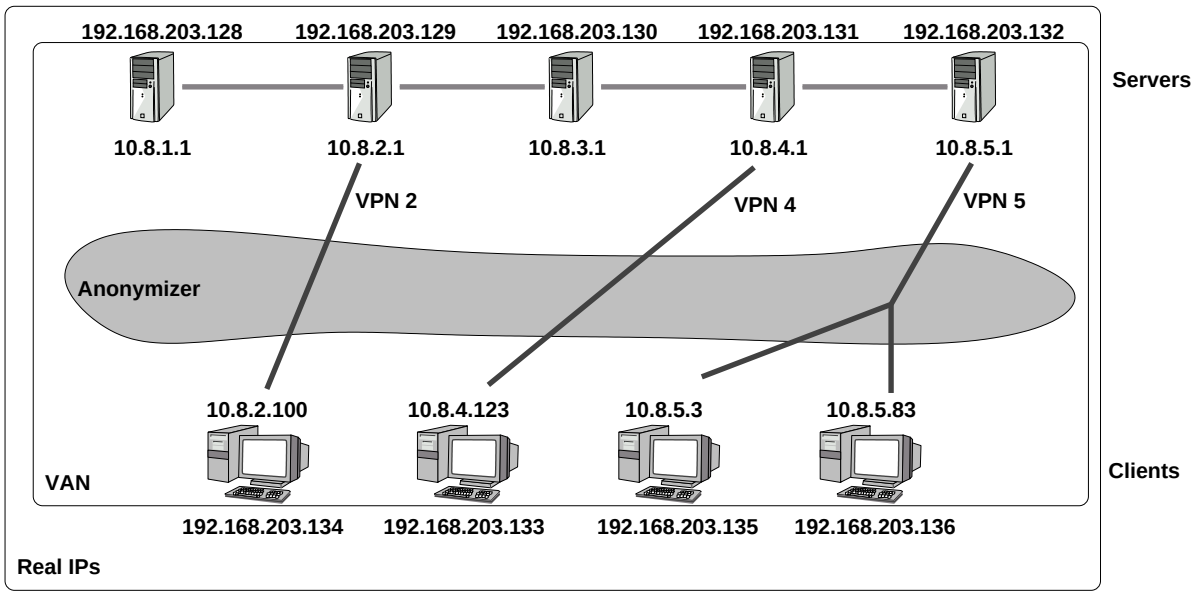


Figure 1: Our experimental network consists of five interconnected VAN servers and four clients. The clients IPs are anonymized by the Tor network and their VAN IPs are used for inter communication. The picture does not show that the VAN servers also act as Tor servers. Also some connections between the VAN servers are left for clarity.

Setting up the Tor nodes in an closed environment requires us to replace the default directory server by one of our Tor nodes. This node also receives the other nodes' fingerprints to classify them as trusted. The nodes interconnect as soon as they are started.

Prior to starting the VPN-servers, it is necessary to create a certificate authority and create keys and certificates for the participants. We created one certificate per VPN-server and one for all clients. After starting the VAN-servers, they need to be interconnected pairwise.

On the four client computers the Tor client needs to be installed and configured to use the local Tor network, instead of the global one. After the Tor client successfully connects, the VPN-client can be started. The client connects to a random VAN-server to avoid linkability with prior IPs and for load balancing purposes. It then redirects non-local traffic to the VAN-server through the Tor network.

For testing purposes we pinged the clients pairwise and transmitted data by TCP and UDP without any problems.

4 Discussion

Once set up, the presented method enables the user to communicate anonymously without any further adjustment of programs. The anonymizing network prevents the VAN-server or an observer from learning the identity of a user through his IP address, so the VAN IP address can not be linked to the individual.

One of the advantages given for free by using OpenVPN is that of being able to resume broken down network connections. This is especially interesting for mobile users.

Also, having regular IPs as addresses, it is possible to assign ordinary fully qualified domain names to anonymized IPs. Such, a name like *medical-forum.net* can be resolved to one of those IPs. The DNS entry has to be changed every time, the service changes it's IP, but there are established solutions to this problem (e.g. *dyndns.org*).

Anonymity in VANs Security properties, like the degree of anonymity, have to be reconsidered in compound architectures because they may be weaker than those of the single parts it consists of.

The anonymity of users in VANs is not better than that of ordinary Tor. Especially as the anonymous IP stays the same over a longer period of time and is the origin of requests from exactly one user, his different communications can be linked to each other. Also, if an attacker finds out a user's anonymous IP, he can attack this user although he does not know his real address. This is impossible against Tor users without VAN. Direct attacks against an anonymous IP include port scanning and fingerprinting. However, the countermeasures known from regular IP networks can be applied here, e.g. fire-walling inbound traffic. If those techniques are correctly applied, active adversaries gain little more information than passive ones.

Problems and Issues In this section we are going to discuss open problems and issues related to the proposed architecture. Where possible we will suggest a solution.

One problem is, how to distribute the OpenVPN certificate for the client computers. This can be solved in different ways: VAN servers can run a HTTP server where they provide access to the needed software and certificates. Another solution is to package it directly with the VAN client software.

Another problem is that a central authority is needed to provide disjunct IP-ranges and certificates to the operators of the VAN-services. This authority might also support the routing between local networks of the VAN, as our approach is only suitable for very small VANs.

Another issue is that the anonymous IPs are not accessible to outside hosts. A solution to this would be to use globally assigned IPs and have the range's owner provide a gateway to the VAN. However it might not be a good idea to make the services available to outsiders, as this reduces their motivation to use the system. As a result they do not increase the anonymity set.

For a large scale system, the choice of the IP-range of the virtual network is also crucial. The private IP-ranges 10.0.0.0/8 and 192.168.0.0/16 are used too often in LANs, such that collisions are likely in a global environment. However mostly the lower values of these ranges are usually used such that sub ranges with higher values might be a fine choice. Also the private range 172.16.0.0/12 is seldomly used and might be a good choice in a global environment. A wide support of IPv6 will help here.

Although it looks convenient to apply DNS-services on anonymous IPs, this allows certain impersonation attacks on anonymous services. If a service recently changed its IP, an attacker can try to gain the old IP to impersonate the earlier owner. The use of SSL secured connections that use a certificate which refers to the DNS entry and is reliably certified by some root CA can solve this problem.

Last but not least, this architecture does not protect the user who identifies himself by his choice of software, the fingerprint of his operating system, or by sending their identity on the application layer. Application layer filters and proxies can provide some help at this stage, but there is no satisfying solution right now.

5 Conclusion and Outlook

We have presented an architecture based on Tor and OpenVPN that enables transparent anonymization of all IP based network traffic and provides users with anonymized IP addresses. This approach reduces the configuration overhead, as it only has to be performed at one single point instead for every single

application that's traffic should be anonymized. This decreased complexity also increases reliability, as the user only has to assure this one system is working correctly.

Since the degree of protection of anonymity networks grows with the number of active users, we consider it important for them to be highly usable and versatile. We showed that current techniques can be extended by existing software in a way that allows most convenient and secure usage of anonymity networks.

Furthermore we discussed arising problems and validated the practicability of our approach by setting up a prototype and evaluating it experimentally.

We look forward to see some of our ideas implemented in a single piece of software, possibly even as kind of a plug-in to Tor or another anonymity system. Alternatively some out of the box package of our prototype could lead to a larger popularity and result in the required anonymity set.

References

[BFK00] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[bin] ISC BIND, Berkeley Internet Name Domain. http://www.isc.org/sw/bind/.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66. Springer-Verlag, LNCS 2009, July 2000.

[Dan03] George Danezis. Statistical disclosure attacks: Traffic confirmation in open environments. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421–426, Athens, May 2003. IFIP TC11, Kluwer.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[FM02] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[KAP02] Dogan Kesdogan, Dakshi Agrawal, and Stefan Penz. Limits of Anonymity in Open Environments. In *Proceedings of Information Hiding, 5th International Workshop*. Springer Verlag, 2002.

[KP04] Dogan Kesdogan and Lexi Pimenidis. The Hitting Set Attack on Anonymity Protocols. In *Proceedings of Information Hiding, 7th International Workshop*. Springer Verlag, 2004.

[MD04] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, LNCS, May 2004.

[ope04] The OpenVPN-project. http://openvpn.sourceforge.net/, 2004.

[RP02] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.

Reducing system call logs with selective auditing

Ulf Larson and Erland Jonsson
Department of Computer Engineering
Chalmers University of Technology.
412 96 Göteborg, Sweden
{ulfla, erland.jonsson}@ce.chalmers.se

Abstract

Event auditing today is a resource consuming process. Rapidly increasing performance of hardware results in event production at a faster rate. Complex software, multiprogramming and extensive connectivity between software components makes it both difficult and resource demanding to discriminate between malicious and benign system events. Thus, an exhaustive auditing approach is not feasible and there is need for a more efficient solution. We propose a method called selective auditing, where only a specific subset of system events are recorded. This will significantly reduce the required amount of auditing and will produce smaller audit logs of higher quality. We illustrate the benefits of the selective auditing method by executing four buffer overflow attacks and show that the logs generated by selective auditing are significantly reduced in size while still giving the same detection rate.

Keywords: Intrusion detection, system calls, auditing, data reduction

1 Introduction

Auditing facilities is a vital part of every operating system. Auditing is used for monitoring system resources during system operation and for finding the correct position to roll a system back to after failure. For intrusion detection purposes, auditing is used as the input upon which the detector bases its decision. An extensive amount of audit log data causes the detection process to progress more slowly. Also, extensive auditing uses up system resources. The performance of today's hardware results in a high-volume event production. Complex software, multiprogramming and extensive connectivity between software components makes it both difficult and resource demanding to discriminate between malicious and benign system events. In this paper we use *selective auditing* for reducing the resources spent on auditing. An audit source that uses selective auditing captures a subset of all system activity. The specific subset is a function of the data used for configuration of the audit source. In this paper, the subset is a function of observed differences between normal and abnormal system operation. Barse and Jonsson [BJ04] developed a framework for extracting such a subset consisting of useful log items, or *manifestations*, from audit logs. Larson et al. [LBJ05] continued the work by creating the METAL tool for automation of the most resource demanding step of the framework.

In this paper we use manifestations extracted by the METAL tool to configure a system call based audit source, *syscalltracker*[1], to use selective auditing. We then run four buffer overflow type attacks to investigate to what extent the selective auditing reduces the sizes of the logs generated. To measure the reduction we compare the size of the logs to the size of logs generated in the case where the subset of system activity equals the set of *all* system activity. The output data from METAL and the input data to the system call auditing tool is not compatible and must thus be parsed. We perform the parsing by building a rule generator that takes as its input data the METAL manifestations and produces as its output rules written in a format accepted by *syscalltracker*.

The outline of the paper is as follows. Section 2 presents related work in the field of manifestation extraction and log reduction. Section 3 describes the manifestation extraction framework, automation

[1] *Syscalltracker* is an open source tool and it can be downloaded from `http://syscalltrack.sourceforge.net`.

of the extraction process and the *syscalltracker* auditing tool. In Section 4 we describe how we generate the data needed to measure log size reduction and how *syscalltracker* rules are generated from extracted manifestations. Section 5 describes our experiments and the method for reduction measurement. Results are presented and discussed in Section 6 and future work in Section 7. Section 8 concludes the paper.

2 Related Work

Data reduction is an important part of intrusion detection. Audit sources of today generates massive amounts of data that needs to be stored and processed, and data reduction methods are used both for reducing the amount of storage space needed and for providing the intrusion detection system with a smaller amount of input data. The majority of reduction methods are used in anomaly detection systems for e.g. reducing classification time, but there also exist methods proposed for misuse detection. In addition to this, we also have the traditional compression methods that does not take into consideration the removal of unnecessary items but only reduces the space used.

Pure compression can be achieved by for example reducing the log files into relational databases. One such tool is SmarterStats [Inc05], that is reported to have a reduction rate of 85%. This will reduce the consumed space, but will not attempt to make any decision regarding the usefulness of the log items.

For anomaly detection purposes, methods such as data filtering and data clustering have been used. Data filtering means that data is removed from the data set with heuristic or *expert* methods. Here, rules are created that separates interesting data from benign. DIDS [SSTG92] and MIDAS [SSHW88] are examples of systems that use heuristic filtering. To overcome the problem with constant updates of the rule set, adaptive filtering can be used. Adaptive filtering can be performed, for example by using neural networks, to adapt to changes in behavior over time. This approach is taken by Debar et al. [DBS92].

Data clustering is used to reduce storage space by either grouping, or *clustering*, events according to a distance measure or storing characteristics of groups of events instead of the events themselves. Distance clustering uses distance measures like the k-nearest neighbor algorithm to calculate the shortest distance from the event to possible candidate clusters. The event can then be represented by a point in the closest cluster, thus reducing the event to a point. Lane and Brodley [LB98] proposes a greedy clustering algorithm together with a similarity measure to reduce storage space for user profiles. Their method operates on sequences of command line entries.

Characteristics storage are used in NIDES [DN85]. Here, descriptive statistics is used to reduce the size of log files. This is done by splitting events into components and for each component calculating a frequency score. An average is taken over the components of the event and events with a low score are considered as unusual. Axelsson [Axe04] use a similar but simpler approach when studying large web server logs.

Even though most work has been done for anomaly detection, data reduction has also been attempted for misuse detection. Kuri et al. [KNMH00] address the performance problem when large amounts of data are to be searched fast. They apply their approach to a misuse detection system by applying a pattern matching technique. In their approach, an input stream consisting of user level command sequences are filtered in order to remove large parts of the stream that do not contain any attacks. This leaves a smaller amount of data to be analyzed by a slower algorithm. Yet another approach is the one taken by Godínez et al. [GHM05]. They use N-Gram models to address the problem on analyzing repetitive spurious patterns. They are using system call sequences as input and their model reduce the log size by tagging the most frequent sequences with labels. When a labeled sequence is encountered, the sequence is replaced by the label, thus reducing the log size.

Of the methods mentioned above, the expert filtering approach is the method that is closest to our work. However, since our expert knowledge is taken as output from an automated tool, the expert does not need to be present to provide the expert knowledge.

3 Manifestation logging and extraction

Our approach to selective auditing depends on that manifestations are available as input. The manifestations we use arise from preprocessing of system call audit log files. We will discuss the method of retrieving, or *extracting*, these manifestations and also briefly discuss the properties of the auditing tool

that is used to record the system calls, but we begin with discussing the foundation for our ideas, the manifestation extraction framework.

3.1 Manifestation extraction framework

Our selective auditing method is based on results from the framework presented by Barse and Jonsson [BJ04]. Their framework consists of eight steps and with the overall purpose to find differences between logs created during normal system operation and abnormal system operation. The logs must be generated in a controlled lab system where few other activities are going on. The auditing is started right before the execution of the normal program behavior or the execution of the abnormal program behavior. The auditing is stopped as soon as possible after the program has finished its actions. The eight steps of the framework are:

1. Identify different parts of the attack, i.e. attack events.
2. Determine **normal events** to which the attack events can be compared.
3. **Classify** the attack events by their usefulness.
4. Extract **event traces** by logging successful attack events, and the corresponding normal attack events.
5. Extract **attack manifestations** by comparing traces.
6. **Classify** the attack manifestations.
7. Create **attack indicators** by using information from the attack manifestations.
8. Define the **log data requirements** of the attack by studying the attack indicators.

For our purposes, we use the framework to extract attack manifestations, classify them and converting them into rules for a system call auditing tool. Throughout this paper, the term *attack manifestations* refer to the events, sequences of events or parts of events that are recorded in a log during an attack but not during normal behavior. For extraction purposes we use the automated Manifestation Extraction Tool for Analysis of Logs (METAL), developed by Larson et al. [LBJ05].

3.2 Automatic extraction of manifestations using METAL

METAL implements step five of the framework, thus extracting the manifestations from the logs. The operation of METAL is shown in Figure 1.

As seen in Figure 1, METAL consist of four modules called *preprocessor, sanitiser, process matcher* and *extractor*. The *preprocessor* prepares input data for analysis by dividing the input logs into separate processes. The process division decision is based on process ID (pid) and process name entries in the file. By using both pid and process name, one file can be created for each process, including forked processes. Processes that are considered as belonging to the logging process itself are removed. The result of the preprocessing is one directory containing the processes extracted from the normal log and one directory containing the processes extracted from the attack log.

The *sanitiser* removes parts of log entries that will disturb matching and comparison by applying a set of rules to each file. The set of rules is created by manually inspecting log files and locating the log items that always change. For example, the return value of the *time* system call always differ and should therefore not be considered when comparing logs.

The *process matcher* compares log files generated by processes active during normal operation to log files generated during an attack to find which processes that are changed by the attack. Each log file in the normal directory is compared to each log file in the attack directory and the degree of difference is recorded. The degree of difference is calculated by comparing system call sequences using the tree based approach as explained in [FHSL96]. First, a profile of the system call sequences for one process is created and then, the sequences from the other process is matched against the profile. This procedure is then repeated by matching all processes from the first log directory with each process in the second log directory. The degree of difference is the number of mismatching sequences divided by the total number of matches made.

The *extractor* extracts five manifestation types and writes the manifestations into report files, denoted as *Attack reports* in Figure 1. Each comparison between normal and attack activity made by METAL generates n attack reports, where n is the number of processes that were considered to be involved in

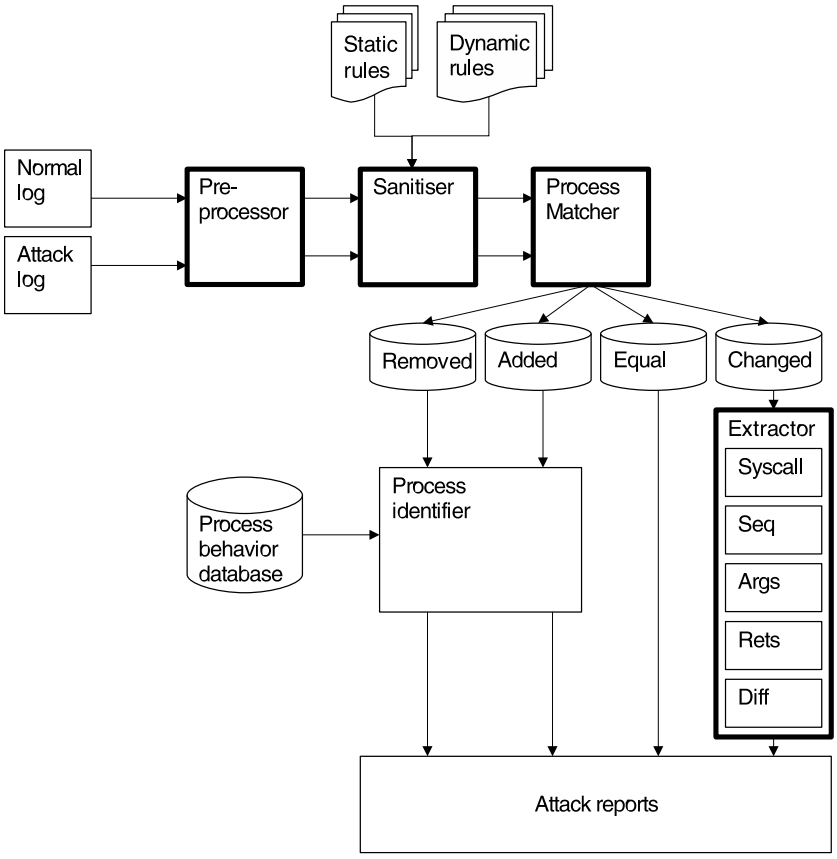


Figure 1: Log comparison in the METAL tool

the attack. The attack report contains the name of the process that caused the manifestations together with the manifestations themselves, grouped by type of manifestation. In Figure 1 the types are denoted as *syscall, seq, args, rets* and *diff*. The types are briefly described as follows: *syscall*, or Unique system calls, denotes the system call names that can be found in one log but not in the other. *seq*, or Unique sequences, denotes sequences of system calls that can be found in one log but not in the other. *args*, or Unique arguments, denotes the system call arguments that can be found in one log but not in the other. Arguments are grouped by system call and argument vector position and compared. *rets*, or Unique return values, denotes the system call return values that can be found in one log but not the other. Return values are as arguments grouped by system call and compared. *diff* shows the differences between files by applying the UNIX *diff* command.

3.3 System call auditing tool

For auditing, we use a publicly available tool called *syscalltracker* [FKM+05]. *syscalltracker* is a tool for capturing system calls and writing them to a file or device. For each specific event that the user want to monitor, a *rule* is created. A rule is made of a *rule id*, which is a number, a *filter expression* that states what conditions that should hold for the system call, and an *action*, that states what should be done when the rule is matched. The filter expression is made up from process field variables such as process id, command issued and effective user id as well as operators like '==' and '&'. Consider below Figure 2 showing the rule for matching execution of a command shell by a program called tcpdump.

The rules are written by the user and stored in a configuration file. This file is loaded into *syscalltracker* during runtime.

```
rule
{
  syscall_name = execve
  rule_name = tcpdump_exec_sh
  filter_expression {COMM == ‘‘tcpdump’’}
  action {
    type LOG
    log_format { %comm issued execve command}
  }
  when = before
}
```

Figure 2: A *syscalltracker* rule for matching execution of a command shell by the tcpdump program

4 Producing data for reduction measurement

Selective auditing means that we can choose what events we want *syscalltracker* to log for a specific attack. In order to instrument *syscalltracker* to log a selected subset of all events we must first have prior knowledge of what events that are interesting and what events that can be discarded. We have chosen to use as input the manifestations extracted by the METAL tool.

In order to generate the manifestations, we first use *syscalltracker* with a configuration file that allow us to capture all events generated by all active processes, as described in [LBJ05]. We call this configuration file `REFERENCE.conf`. We start *syscalltracker*, load `REFERENCE.conf` into *syscalltracker* and capture normal process activity. Normal activity is saved to a file called `REFERENCE_NORMAL.log`. We then start *syscalltracker* again, load `REFERENCE.conf` and capture attack activity. This activity is saved to a file called `REFERENCE_ATTACK.log`.

We then use the METAL tool to generate one set of `Attack reports` for each attack. As input to METAL we use the `REFERENCE_NORMAL.log` and the `REFERENCE_ATTACK.log` for each attack respectively. This procedure is described in Section 3.2 and shown in Figure 1. Figure 3 illustrates how the attack reports are created.

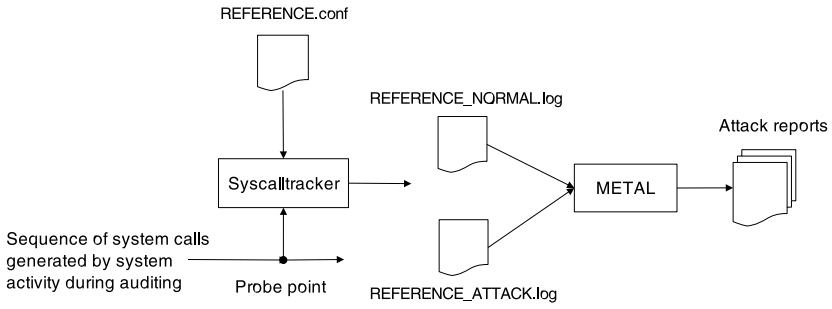


Figure 3: Creating the Attack reports

Each `Attack report` contains a number of manifestations, the number is the same as the number of differences between the `REFERENCE_NORMAL.log` and the `REFERENCE_ATTACK.log`

The next step is to convert the `Attack reports` into a configuration file containing rules that *syscalltracker* can accept as input. Since we want to log events caused by an active attack, we need only consider parts of the `Attack report` file. As seen in Figure 4, each type of manifestation has two parts, of which we will use the part caused by the attack. For future references to this part of the attack report we use the expression *manifestations present in attack but not in normal*.

In Figure 4 the attack part is represented by the *11_execve* system call. Each `Attack report` contain one section for each manifestation type. In Figure 1, these types are referred to as *Syscall, Seq, Args, Rets* and *Diff*. The *Diff* manifestation type can not be used to generate rules and is therefore excluded.

For converting the `Attack report` into a configuration file we have built a tool called the rule generator. The rule generator is used once for each `Attack report` and its internal operation is further described in Section 4.1. The output of the rule generator consists of one rule for each manifestation

```
================================================================================
Unique system calls from [normal] 8972_tcpdump

4_write
================================================================================
Unique system calls from [attack] 8782_tcpdump

11_execve
```

Figure 4: The two parts of the manifestation type syscall

in the `Attack report` file. The collection of rules is stored in a *syscalltracker* compatible configuration file referred to as `ADAPTED.conf`. When we have finished this step, the `ADAPTED.conf` contains *contributions from all tested attacks*. To perform the selective auditing, we now start *syscalltracker*, load the `ADAPTED.conf` file and capture *once again* attack activity. This activity is saved into a file called `ADAPTED.log`. This procedure is repeated once for every attack that we want to include in our experiment.

We now have all the information we need to calculate the ratio between the `REFERENCE_ATTACK.log` and the `ADAPTED.log` files which gives the reduction percentage. The formula for calculating the reduction percentages is given in Section 5.4.

4.1 Internal operation of the rule generator

We use the rule generator to create configuration files for the syscalltracker tool. The rule generator takes as input the METAL generated attack reports and outputs configuration files for the syscalltracker tool. The operation of the rule generator consists of three steps. Figure 5 illustrates the internal operation of the rule generator.

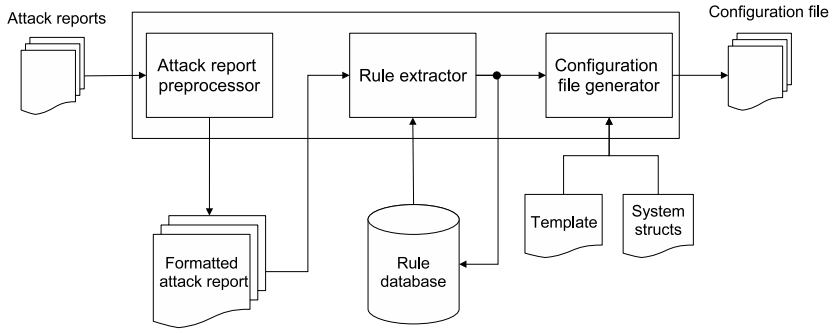


Figure 5: Internal operation of the rule generator

The *Attack report preprocessor* accepts k attack reports, where k is a non-negative integer. From these reports, the manifestations from the *manifestations present in attack but not in normal* parts are extracted. All manifestation types *syscall, seq, args* and *rets* are extracted and written to the `Formatted attack report`. The `Formatted attack report` is then used as input to the *Rule extractor* together with a `rule database`. The `rule database` contains all manifestations extracted during previous use. The *rule extractor* reads the database and the `formatted attack report` and merges them into an updated version of the `rule database`. The updated version is used both as input to the *Configuration file generator* and as the new `rule database` for future use. The configuration file generator also receives data from two files called the `Template` file and the `System structs` file. The `template` file contains static information and instructions regarding the formatting of the configuration file and the `system structs` file contains detailed information about operating system structures (`structs`). The information contained in the `system structs` file is necessary when constructing filter expressions that includes the use of struct fields. The *configuration file generator* creates from its input a parseable configuration file for use as input to *syscalltracker*. As an example of output format from the *configuration file generator*, consider Figure 6. The attack report for the `execve` system call executed by the `tcpdump` process has been transformed into a syscalltracker rule. This rule instruments syscalltracker to log all execve calls if

the calling process is tcpdump.

```
/* A rule to log 11_execve calls. */
rule
{
  syscall_name = execve
  rule_name = rule_number_171
  filter_expression
  { COMM == "tcpdump" }
  action{
          type = LOG
          log_format{syscall: %pid[%comm]: %sid_%sname, euid=%euid, uid=%uid (rule%ruleid)}
  }
  when = before
}
```

Figure 6: A sample rule as generated by the configuration file generator

5 Experiments

The rule generator is implemented in the Python programming language and arguments, i.e. input and output file names, are provided at the command line when the program is invoked. The attacks that we use are four buffer overflow attacks and the target system is a RedHat 6.2 system. The data from the attacks is collected by *syscalltracker*. The log reduction is calculated by comparing the size of the logs generated with selective auditing and the logs generated by logging all activity in the system.

5.1 Choosing attacks

We have chosen to use buffer overflow type attacks in our experiments and the following four attacks were used: The first attack is a remote exploit that targets a vulnerability in the tcpdump program.[2] The second attack is a remote format string stack overwrite vulnerability targeting the wu-ftpd service.[3] The third is a buffer overflow attack against the dip program[4] and the fourth is a buffer overflow attack targeting the xlock program[5]. The decision of using only buffer overflow attacks is based on that we at a later step will be able to investigate and possibly identify general similarities and attempt to make a generalization from the data. Such a generalization would be a first step towards identifying items that are common for this group of attacks. The generalization would also be the first step towards collecting manifestations with intention of finding unknown attacks that are constructed in the same way as the investigated attacks. This would further prove the usefulness of the approach.

5.2 Experiment setup

For the experiments we use the following system setup. One attack system running RedHat Linux 9 with a 2.4.20 kernel and one target system running RedHat Linux 6.2 with a 2.2.19 kernel and with *syscalltracker* installed. The target system also has the vulnerable software packages installed. The attack and target systems are connected via a TCP/IP network. Remote attacks are carried out over the TCP/IP connection and local attacks are carried out by first logging in via ssh from the attack system to the target system and then executing the attack.

5.3 Experiment execution

The experiment is executed according to the method described in Section 4. For each attack, we first execute the attack program. After the attack program has generated the command shell, we issue the

[2]Tcpdump AFS ACL packet buffer overflow vulnerability, Bugtraq ID 1870, CVE-2000-1026.
[3]Wu-ftpd remote format string stack overwrite vulnerability, Bugtraq ID 1387, CVE-2000-0573.
[4]dip buffer overflow vulnerability, Bugtraq ID 86, CVE-1999-0137.
[5]xlockmore user supplied format string vulnerability, Bugtraq ID 1585, CVE-2000-0763.

`whoami` command[6] and finish with the `exit` command.

5.4 Measurement of log size reduction

After collecting the data we measure the log size reduction. The following formula is used to derive the reduction percentage.

$$\text{reduction percentage} = (1 - \frac{size(\text{ADAPTED.log})\ kB}{size(\text{REFERENCE\_ATTACK.log})\ kB})\ x\ 100$$

The result of a calculation using the formula is a decimal number between 0 and 100 representing the log size reduction in percent. The closer to 100 the reduction percentage is, the greater the reduction.

6 Results and discussion

We present the reduction percentage for the four attacks. Note here that the `ADAPTED.log` for each case is generated by using an `ADAPTED.conf` file that contains contributions from *all used attacks*. That is, before starting to generate `ADAPTED.log` files we first collect manifestations from all attacks.

We further discuss the reasons for the different reduction rates. The results are presented in Table 1. Table 1 is organized as follows: The first column contains the name of the attack. The second column contains the size of the `REFERENCE_ATTACK.log`. The third contains the size of the `ADAPTED.log` and the fourth contains the reduction percentage.

Table 1: Measuring reduction of log file size

Attack	*Log file size reduction*		
	Ref. size	*Adp. size*	*Red. %*
tcpdump	1500 kB	1.8 kB	99.9
wu-ftpd	840 kB	35 kB	95.8
dip	1400 kB	160 kB	88.6
xlock	1860 kB	670 kB	64.0

As seen in Table 1 we have a significant reduction in size for both the tcpdump (99.9%) and the wuftpd (95.8%) attacks. We have a good reduction rate for the dip attack (88.6%) and a moderate reduction for the xlock attack (64.0%).

By observing the content of the logs we can find the reasons for the reduction percentage in the various cases. The logs with the highest reduction percentage, i.e. the `tcpdump` and the `wuftpd` logs, contained many matches on unique system calls and few matches on arguments and return values. A unique system call is logged with only the system call name and the return value, thus generating fewer output bytes to the log. Also, these processes generates few read and write calls and the major part of the argument strings are short. The `dip` log contain about as many read and write calls as the `tcpdump` and `wuftpd` logs, but the argument strings are longer. Also, since this attack is carried out over ssh on a local account, long strings are sent over ssh to the attacker, thus creating strings with long arguments in the log. In addition, the padding of '\0' to make strings 4096 bytes long also increases the size of the log. The moderate reduction of the `xlock` log has two reasons. First, due to large differences between the attack and the normal input logs, METAL produces a somewhat erroneous output for this attack, i.e. it mismatches processes. This causes many manifestations to appear that otherwise should not be in the list of manifestations. Second, a high degree of read and write operations are present in the log. Many of these also coincides with the manifestations that erroneously are present in the log.

For addressing the quality of the traces we make the following assumptions: Any resulting trace, or manifestation, is a function of one rule present in the syscalltracker configuration file. These rules are generated by the METAL tool and one extracted manifestation give rise to one rule. The manifestations are observed differences between an attack and an approximation of a corresponding normal behavior. The coverage of the traces is dependent of the extraction method in METAL and using additional

[6] We assume that this is a normal command issued by the attacker to find out whether the attack was successful or not.

extraction methods would possibly yield more rules and thus resulting in a larger log, *possibly* containing more traces. Whether the quality of the traces would be improved by this or not is not yet investigated.

7 Future work

Future work includes developing METAL into a general tool that can extract manifestations from different kinds of attacks. We will also run more attacks, both buffer overflow attacks and other types of attacks. This will enable us to draw conclusions about similarities that may be generalizable for many attack types.

8 Conclusions

We have used an existing method for manifestation extraction and adapted it to the specific case of reducing the size of system call logs. As input we used attack reports containing manifestations. These manifestations were transformed into rules suitable for usage with *syscalltracker*. We used two different rule sets to gather data for comparison. We ran four attacks with both rule sets and collected data with *syscalltracker*. The logs were compared and the results were summarized. From the results we observe a significant decrease in log size. For the three relevant attacks we got an average log reduction of 95 percent. We conclude that while our approach at this stage only finds known attacks, we plan to use the data as input for generalizable with intention of finding previously unknown attack based on generalizable similarities in manifestations.

References

[Axe04] Stefan Axelsson. Visualising intrusions: Watching the webserver. In *Proceedings of the 19th IFIP International Information Security Conference (SEC2004)*, Tolouse, France, 22–27 August 2004. IFIP.

[BJ04] Emilie Lundin Barse and Erland Jonsson. Extracting attack manifestations to determine log data requirements for intrusion detection. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, USA, December 6-10 2004. IEEE Computer Society.

[DBS92] Herve Debar, Monique Becker, and Didier Siboni. A neural network component for an intrusion detection system. In *Proceedings of the 1992 IEEE Computer Sociecty Symposium on Research in Security and Privacy*, pages 240–250, Oakland, CA, USA, May 1992. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA.

[DN85] Dorothy E. Denning and Peter G. Neumann. Requirements and model for IDES—A real-time intrusion detection system. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, 1985.

[FHSL96] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[FKM+05] S Fish, G Keren, Mulix, A Shalem, and E Shemer. System call tracker - design, implementation, goals. `http://linuxclub.il.eu.org/lectures/44`, June 2005.

[GHM05] F Godínez, D Hutter, and Raúl Monroy. Audit file reduction using n-gram models (work in progress). In *Proceedings of the Ninth International Conference on Financial Cryptography and Data Security*, 2005.

[Inc05] SmarterTools Inc. Introducing smarterstats 3.x. `http://www.smartertools.com/Products/SmarterStats/Overview.aspx`, September 2005.

[KNMH00] Josué Kuri, Gonzalo Navarro, Ludovic Mé, and Laurent Heye. A pattern matching based filter for audit reduction and fast detection of potential intrusions. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 17–27, London, UK, 2000. Springer-Verlag.

[LB98] Terran Lane and Carla E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *5th ACM Conference on Computer & Communications Security*, pages 150–158, San Francisco, California, USA, 3–5 November 1998.

[LBJ05] Ulf Larson, Emilie Lundin Barse, and Erland Jonsson. METAL - a tool for extracting attack manifestations. In *Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment workshop (DIMVA)*, Vienna, Austria, July 7-8 2005.

[SSHW88] Michael M. Sebring, Eric Shellhouse, Mary E. Hanna, and R. Alan Whitehurst. Expert systems in intrusion detection: A case study. In *Proceedings of the 11th National Computer Security Conference*, pages 74–81, Baltimore, Maryland, 17–20 October 1988. NIST.

[SSTG92] Steven R Snapp, Stephen E Smaha, Daniel M Teal, and Tim Grance. The DIDS (distributed intrusion detection system) prototype. In *Proceedings of the Summer USENIX Conference*, pages 227–233, San Antonio, Texas, 8–12 June 1992. USENIX Association.

Some security problems raised by open multiapplication smart cards

Serge Chaumette Damien Sauveron[‡]

LaBRI, Laboratoire Bordelais de Recherche en Informatique
UMR 5800 – Universite Bordeaux 1
351 cours de la Liberation, 33405 Talence CEDEX, FRANCE.
{serge.chaumette,damien.sauveron}@labri.fr, http://www.labri.fr/

Abstract

Multiapplication smart cards now make it possible to have several applications sharing the same physical piece of plastic.This raises new security problems by creating additional ways to attack them. This is particularly true with future ***open multiapplication cards*** that allow anybody to load his/her own application. For example an attacker can load her own programs to determine the physical signature of instructions using measurements on side channels (electromagnetic or power signal) and build a database of the whole set of instructions in order to reverse-engineer an application already installed on the card. The vulnerabilities of these cards are the topic of this paper. The attacks are described for open multiapplication cards in general and illustrated by means of code samples for Java Cards.

KEYWORDS: *Smart Cards, Java Cards, Open Multiapplication Cards, Security, Attacks.*

1 Introduction

Even though multiapplication smart cards allow several applications to share the same medium, dynamically loading a new application requires the knowledge of authorization keys. With the open multiapplication smart cards this constraint will not exist anymore.

The aim of this paper is to describe new security problems that arise when dealing with these **open** multiapplication smart cards. First we will present the common way to explore and attack classical smart cards and then we will clearly define what are closed multiapplication smart cards and open multiapplication smart cards. In section 4 we will show the general attacks against these new cards. Then, in the context of the open multiapplication smart cards, we will expose in section 5 some physical characterization methods and we will investigate in section 6 how to use them in order to achieve effective attacks on these cards. Finally, we will present our experiments, the related work and we will conclude. The contributions of this paper are the description of the physical characterization methods to characterize a platform and the explanation of the new threats of these methods in the context of open multiapplication smart cards.

2 Exploring and attacking closed smart cards

Even though there are not many ways to explore a closed card (*i.e.* a card that does not allow codes to be uploaded after it has been issued) before trying to attack it, there are at least two possibilities that remain:

[*] LaBRI (Talence, FRANCE) and LMSI (Limoges, FRANCE).

[†] This work was partly supported by a doctoral grant from the french ministry of research and SERMA Technologies.

- Software approach. Since loading code on a closed card is impossible, the only accessible information are its external interfaces. More precisely, the only visible external interface of a card is its communication port. The communication architecture between a smart card and a reader connected to a host can be seen as a protocol stack, from the physical layer to the application layer (*see* Fig. 1). It is described in the ISO7816-3 [1] and ISO7816-4 [2] standards. One common

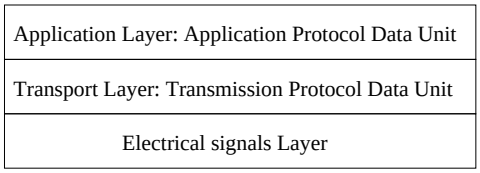


Figure 1: Stack of communication protocols.

possibility to set up an attack at the level of the communication layer is to send all the possible APDU commands to the card, in order to identify all the services available from the outside and to perform fuzzing attacks[1]. But these kinds of attacks are quite inefficient because it is easy to block the unwanted APDUs.

- Hardware approach. Based on the results obtained by Kocher, the main path of attack to explore a card without damaging it, is to observe side channels [3] such as the physical emanations from the chip or the time consumptions [4]. The power analysis (PA) [5, 6] or electromagnetic analysis (EMA) [7, 8] methods make it possible to identify patterns corresponding to operations achieved by the card. These attacks are carried out as a blind man, or a semi-blind man if the specifications of the card operations are known.

  The non-invasive attacks cited above are fundamental in order to succeed to build the dictionary that we propose in section 6.1. Besides, smart cards are prone to many other invasive (e.g. microprobing) and non-invasive (e.g. glitches[2] on the different pads of the card or fault-injection with laser) attacks [9, 10, 11, 12] which are out of the scope of this paper (but they can be used as described in sections 4.3 and 6.2).

3 Two types of multiapplication smart cards

A multiapplication card is able to embed several applications. Today, these applications do not run simultaneously because common OS of smart cards have a single thread. The two main standards of multiapplication cards are Java Card [13, 14] and MULTOS [15]. Recently, two new technologies appeared with the birth of Smartcard.NET [16, 17] by Hive-Minded and of the MultiApplication BasicCard ZC6.5 [18] by ZeitControl. Windows for Smart Cards by Microsoft can also be cited even though it does not seem to be active any longer. Java Card was the first to appear but all these technologies share many concepts anyway. The architecture of multiapplication smart cards (*see* Fig. 2) consists of:

- an embedded Operating System that supports the loading and the execution of several applications. The OS is a runtime environment with a virtual machine (VM) that provides security features (e.g. a firewall between applications).
- the applications that are interpreted by the VM.

3.1 The closed multiapplication smart cards

Until now no multiapplication technology was completely proven secure by formal methods to reach the high level of security required to allow it to run uncertified applications that may embed malicious code to attack the assets of the platform and those of the other applications. To prevent these problems,

[1] Fuzzing attacks consist in giving all the possible values to the parameters of a service.
[2] A glitch is a sudden change in voltage in an electrical current.

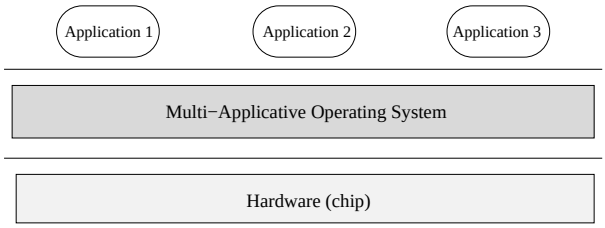


Figure 2: Architecture of a multiapplication smart card.

the smart card issuers began to support standards such as GlobalPlatform [19] which specifies how to securely load, install and manage applications on a card. Indeed this standard is used to easily set up and use cryptographic mechanisms to authorize or prevent the loading of an application on the card. For example the application can be digitally signed off-card by a trusted party, e.g. the card issuer. Once loaded, the card can check the signature and accept or reject the application. The major drawback of this solution is its centralized model because of the trusted third party required to sign the application; it thus decreases flexibility.

For short, if the user does not own the authentication keys, she can only use the card as a closed platform; else, if she owns the keys she will be able to load her code and to apply the internal attacks presented in section 4. She will also be able to use the physical characterization methods shown in section 5 to set up attacks as described in section 6. All these methods are intended to the ITSEF[3] or card manufacturers (who can access the keys) to test the security.

Note that in our paper, GlobalPlatform compliant cards are considered as closed cards since knowledge of keys are needed to load a new application.

3.2 The open multiapplication smart cards

Although Java Card designers first thought it was impossible to embed a verifier due to the resource constraints of the smart cards, on-card verifiers have been developed [20, 21, 22, 23, 24, 25, 26]. Among the different solutions that have been proposed, three of them (*i.e.* the defensive VM, the verifier based on code transformation and the stand-alone verifier) allow to bypass the signature step of the application without jeopardizing the card security and thus allow everybody to freely load his/her own application – *i.e.* anyone can still load a rogue application but it will eventually be rejected by the verifier or blocked by the defensive VM. Note that the verifier based on code transformation [22, 23] requires an off-card program to normalize the application whereas the two others (*i.e.* the defensive VM [24] and the stand-alone verifier [26]) are stand-alone. We only consider as ***open multiapplication cards*** the multiapplication cards based on one of these two stand-alone solutions. Both are equivalent [25] but the defensive VM dynamically checks each executed bytecode whereas the stand-alone verifier statically checks the application once at load time and it is associated with an offensive VM that does not check the bytecode at execution time.

In the past, the FIPR (Foundation for Information Policy Research) thought that the multiapplication cards were a bad idea, except in limited applications such as GSM SIM cards [27]. Nevertheless they are now a reality but they are only available as closed cards. However, even if these open multiapplication smart cards are not yet available on the market, they most probably are the future of smart cards since all the research projects presented above will contribute to make them as secure as required in a near future.

4 Internal attacks

Obviously a common problem of the open multiapplication cards is the possibility to load a malicious application to create internal attacks. Such an application may:

[3]Information Technology Security Evaluation Facility.

- identify available services on the card, collect information and then try to deduce the possible behavior of an official[4] application (since it cannot use unavailable services!);
- directly attack the VM and the firewall.

4.1 Identification of services and collection of information

The preliminary fundamental steps required to attack a card are the identification of the services that it offers and the collection of information regarding its OS and the loaded applications. There are several ways to achieve these operations.

First, to identify the services offered by the card, the documentation may be helpful. If it is not accessible, an attacker can try to load an application on the card to test every service defined in the specifications of the technologies supported by the card. For example, to identify the cryptographic services of a Java Card, she can load an application that uses the `Cipher.getInstance` method in a proper way[5] with all the valid parameters.

Second, if the card supports special mechanisms similar to the GlobalPlatform management functions (e.g. the `GET STATUS` command) the simplest way to collect information is to send the proper APDU commands. Else the attacker can also load an application on the card to achieve this task. For example, to detect all the available applications on a Java Card and to work out if they offer services (through the `Shareable` interface), she can use the `JCSystem.getAppletShareableInterfaceObject` method and fuzz it. Note that this particular method may return false negative results due to access control rules defined by the targeted applications to share their services. If it succeeds and the attacker obtains `Shareable` interfaces, she can then try to apply the illegal reference casting method cited in [28].

4.2 Attacks against the VM and the firewall

There exist many attacks directed against the VM and the firewall but we do not detail them here because they have already been explained in the literature: for example two attacks against the firewall mechanism (AID impersonation and illegal reference casting that provides access to all interface methods of a class) are described in [28]; based on type confusion, attacks against the VM are shown in [29]; there also exist attacks coming from problems in the specifications such as those presented in [30].

A bad specification or a bad implementation can also lead to attacks against the VM. For example the program of listing 5 that forges a `reference` would normally be rejected by an open multiapplication card: at loading time for those based on the stand-alone verifier; at execution time for those based on the defensive VM. But an attacker should always try to load such malicious codes to test if there are implementation problems or not.

The piece of code shown listing 5 (provided in Java Card Virtual Machine bytecode language) forges a `reference` (since the `saload` bytecode reads a `short` at an index in an array and it normally takes two arguments, one of `reference` type – *i.e.* an object, the array to read – and one of `short` type – *i.e.* the index –, but it gets here two arguments of `short` type) and tries to read one `short` of the object by considering it as an array. This function enables (if the call succeeds) to traverse all the allocation table (using all the possible values of `Address`) and to read the contents of all objects as if they were `short` arrays.

Note if the `reference` type is represented as a memory address (*i.e.* the equivalent of a pointer) in the VM implementation this code allows to traverse all the memory.

4.3 Mixed hardware and software attacks

To improve the efficiency of the attacks on the card, it is possible to elaborate simultaneous hardware and software attacks. Using hardware attacks makes it possible to create faults at software level [10, 11, 12] (e.g. to bypass verification). These modifications of the normal behaviour can be exploited by the loaded application to access unauthorized services or information. To perform hardware attacks, it is still possible to use equipments such as a laser, source of heat, etc.

[4] An application installed by some trustworthy organization, e.g. a banking application.

[5] If the card supports the garbage collection mechanism, the method can be called directly anywhere in the code, else, the call should be inserted in the constructor, the application should be installed on the card, it should test if the service is available and it should be deleted to avoid to overflow the heap with multiple instances of the `Cipher` class.

Listing 5: Get short value at the specified address

```
short readShortAtAddress(short Address) {
  sload_1   // Get the Address local variable. We hope this short will be considered as a reference
  sconst_0  // Represent the index 0 in the false array starting at Address
  saload    // Try to get one short and push it on the stack. If this operation succeeds the stack
            // is not well implemented since the saload bytecode should only accept that the first
            // argument is of reference type (and not a short type as Address) and that the
            // object referenced is also a short array
  sreturn
}
```

One problem of these kinds of attacks is the precision required (in space but especially in time) to induce usable faults. To improve it, we need to have a better localization of where the trigger should be placed in the software execution. To achieve this task, we can use the physical characterization methods described in section 5.

For example it is possible to try to bypass the firewall checks with attacks described in section 6.2. A very good paper [31] also illustrates how to achieve mixed hardware and software attacks on the VM of a PC using a lamp as source of heat.

5 Physical characterization methods

Once the services have been identified, an interesting possibility to set up attacks against these services and the applications which use them, consists in observing their physical signatures using measurements on side channels. For example an attacker may load her own programs on an open multiapplication card to determine the physical signature of basic instructions in order to build a database of the whole set of instructions to reverse-engineer (from its execution trace) an official application already installed on the card. In this section we present the applicable physical characterization methods that we identified in the context of open multiapplication smart cards and in the following section we will show how they can be exploited to set up real attacks.

The main physical signals that can be observed come from side channels such as the power consumption, the electromagnetic emissions or the execution time. For instance it is possible to use a cryptographic service and determine the characteristics of the physical signals emitted during the execution of this service (duration in time, location of the best electromagnetic emissions, power consumption, etc.). This characterization can target the use of a whole service or an elementary operation, e.g. the interpretation of a single bytecode or a sequence of bytecodes. Note that in the remainder of this paper, the term "signature" will refer to the physical signature and not to the digital signature discussed section 3.

The difficulty to isolate a pattern from all those composing the trace leads to set up reliable characterization methods to determine the physical signature of the interesting pattern. For example, Fig. 3 shows the difficult task to identify the pattern to observe in the normal execution trace of an applet[6].

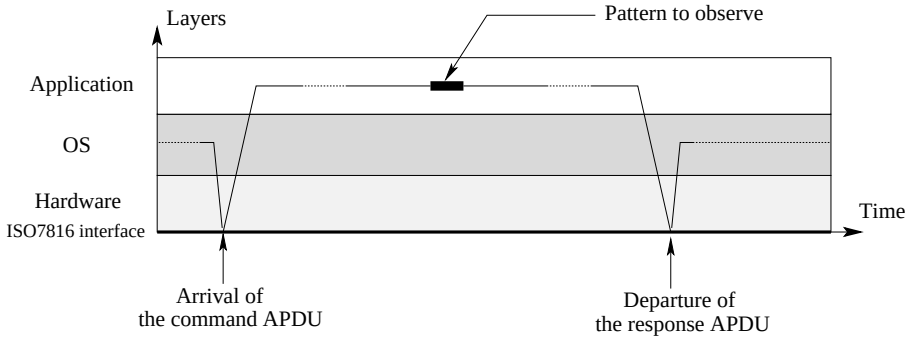


Figure 3: Trace of a normal execution in the layers.

[6] Java Card applications are also called *applets*.

We identified two main ways to easily, efficiently and quickly determine the physical signatures. The first which is also the simplest is achieved by using glitches on the I/O channel of the card, *i.e.* sending output data from the card to the reader. The second consists in causing the pattern to observe to be repeated. In the following sections we present these two approaches and discuss their advantages and drawbacks. We eventually propose a hybrid method that overcomes the problems and takes the best of the two previous methods.

Note that if the physical characterization methods explained below can be applicable on a closed card, it is always in a special context (e.g. the user who wants to attack the card owns the key to load her program). But with open multiapplication smart cards these methods can be applicable without restriction.

5.1 The glitches based method

This method consists in enclosing the targeted pattern with events that are visible from outside the smart card (*i.e.* for the attacker). The only visible event to an external observer that the card can produce is the emission of bytes on the communication channel. Fig. 4 shows that it is easier to locate a pattern in the trace if the execution is glitched using this event.

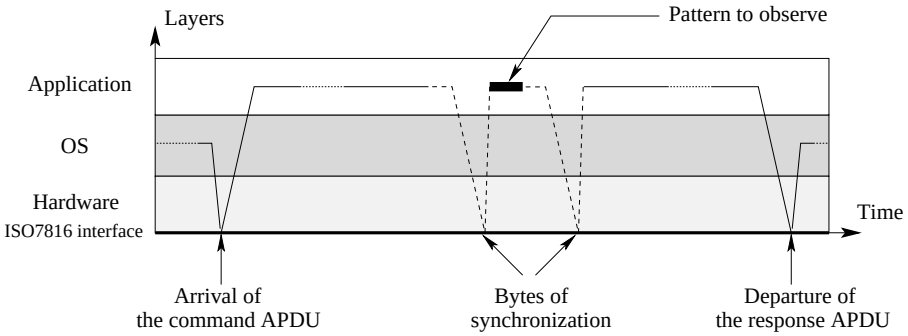


Figure 4: Trace of a glitched execution in the layers.

The code inserted to trigger the glitches causes an overhead and thus the pattern to observe does not appear at the same instant in the first trace and in the second trace (*i.e.* the pattern in the second trace is time shifted).

Another advantage of this approach is that the trace to save is shorter (because the area to observe is smaller – *i.e.* only the part of the signal between the glitches) and thus it is possible to get a better sampling of the signal, still producing the same amount of data.

For more details, the reader is refered to one of our previous papers [32]. It presents the methodology to easily set up test suites with the tools provided by the JCatools suite [33, 34]. The JCatools suite was developed during the Java Card Security project between the LaBRI and the ITSEF of SERMA Technologies. The paper also describes how to modify the CAP[7] file by working at bytecode level to improve the precision of the observation.

5.1.1 Preventing this characterization method from being used

Some solutions to prevent this characterization method from being used can be implemented. For instance, it is possible:

- to introduce a random delay on the I/Os to block these attacks. But it is a bad solution because with many tries it should still be possible to cope with this randomness.
- to store all the data in a buffer until the end of the execution before sending the buffer on the I/O. This would most likely require a second buffer much bigger than what remains acceptable on a card.

[7]The standard binary file format of the Java Card platform.

- to store the data in a buffer until their size reaches a fixed threshold before sending them on the I/O. In this case the hacker may instrument her applet to generate the exact amount of data needed to obtain a response on the I/O and to work out the beginning of the pattern to observe.

- to store the data as previously in a buffer but with a random threshold. A hacker could still bypass this security using a probabilistic technique.

For all these reasons we believe that mixing a random delay and a random threshold storage of the data in a buffer seems to be a solution that may be worth considering.

Note that even if the manufacturers find a better other, effective, countermeasure, it will still be possible to locate the pattern by enclosing it with something that induces high power consumption (e.g. a cryptographic mechanism) to produce an event visible from the outside.

5.2 Repeated pattern based method

Since the manufacturers can add the countermeasures proposed above, we consider another method that still makes it possible to capture the signature of the operations achieved on the card. This solution consists in repeating many times an identical sequence so that it is easier to locate the area to study and then to identify the elementary pattern.

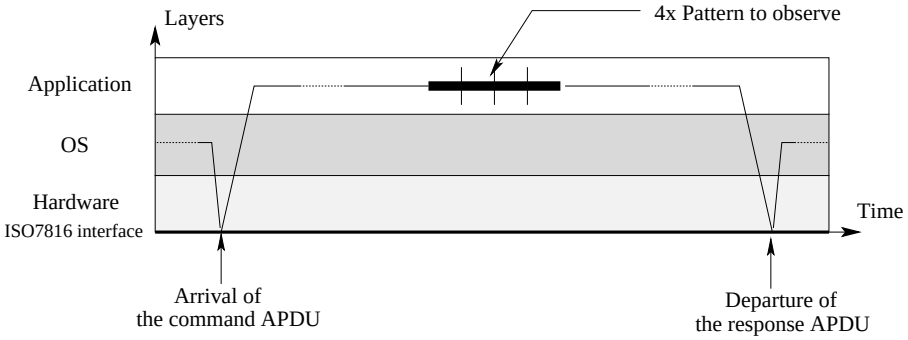


Figure 5: Trace of a multiple patterns execution in the layers.

For instance inserting the code shown listing 6 in a Java Card applet makes it easier to find the `dup` elementary pattern. The physical emanations corresponding to the execution of the above program will

Listing 6: Patterns method

```
...
// assume that there is at least one element on the stack
dup     // pattern to observe
dup     // pattern to observe
dup     // pattern to observe
dup     // pattern to observe
...
```

contain four times the pattern corresponding to the `dup` operation.

5.2.1 Preventing this characterization method from being used

The first solution to prevent such a characterization method from being used is to put constraints on the on-card verifier to check for instance that the loaded code does not contain a sequence of two `dup` bytecodes, *i.e.* `dup dup`. Indeed the converter will produce a `dup2` and not this sequence. But it is not possible to do such verifications for all the possible bytecodes due to the fact that it would require a verifier capable of understanding the semantic of the sequence of bytecodes at application level (*i.e.* what the application wants to do). The verifier can only understand the semantic of the sequence of

bytecodes at VM level (*i.e.* if the chain of bytecodes is valid). A defensive VM can do the same checks but the solution remains the same.

A better solution would be to use hardware countermeasures (e.g. a random internal clock or asynchronous processors [35, 36]) to better jam the physical emanations.

5.3 The hybrid method using both glitches and patterns

This method minimizes the overhead caused by the transitions between the application layer, the OS layer and the chip layer (only two glitches for the sequence) due to the insertion of glitches. It allows to easily locate the sequence of identical patterns in the trace and then it is easy to find the elementary pattern in the located sequence. An example of this hybrid method to observe `sxor` is shown listing 7.

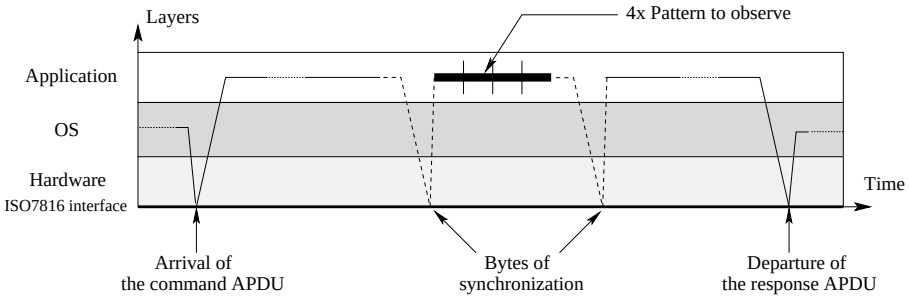


Figure 6: Trace of a glitched multiple patterns execution in the layers.

Listing 7: Sample code used by the hybrid method

```
...
aload_1     //
sconst_0    // Here are
sconst_m1   // several arguments
sconst_m1   // needed to the
sconst_m1   // execution of
sconst_m1   // the patterns
sconst_1    // to observe and
aload_1     // the glitches
sconst_0    //
sconst_1    //
invokevirtual    0x0 0xc // glitch 1
sxor     // pattern to observe
sxor     // pattern to observe
sxor     // pattern to observe
sxor     // pattern to observe
invokevirtual    0x0 0xc // glitch 2
...
```

6 Applications of the physical characterization methods

Based on the physical characterization methods explained above we describe in this section two possible applications:

- the matching attack that allows to reverse-engineer an official applet;
- the physical attack that allows to quickly work out the feasibility of an attack against an official applet.

Note that even though an attacker has access to blank smart cards of the same model as that of the targeted open multiapplication card and she develops and loads her own OS on it, the physical characterization methods would be useless to compare the two implementations since the OS would be different and so would be the physical signatures of all the operations.

6.1 Matching attack

This attack consists in matching the signal identified for the pattern in the malicious applet with the signal of an official applet to reverse-engineer it. The first step required to set up this attack is to build a dictionary matching pattern signals with bytecode instructions. The second step is to identify patterns of the dictionary in the execution trace of the official applet to discover the operations effectively performed. It is a difficult task but there are research projects in progress to automatically recognize patterns in a signal [37]. If this attack succeeds, it jeopardizes the confidentiality of the code of the official applet (*i.e.* the code is known) but it does not imply that an attack will be possible against the execution of this official applet. Nevertheless in the Common Criteria[8] [38] evaluation method, the code of an official applet is often an asset to protect and such an attack is then considered a problem. Moreover, the knowledge of the code enables the use of analysis tools to find breaches of security. Finally, it is also possible to combine the knowledge of the code with the attacks described below.

6.2 Physical attacks

Thanks to the physical characterization methods presented above it is possible to quickly evaluate the feasibility of an attack against an official applet. Using the mixed hardware and software attacks presented in section 4.3, a hacker is in the best experimental position to easily and quickly attack a card implementation (e.g. to try to bypass the access conditions or perturb a cryptographic algorithm) without needing to perform many useless tests, thus saving a lot of time. If her attacks succeed then she can attack an official applet or the platform. For instance if she knows a way to attack a cryptographic algorithm that the official applet uses, she can try to set up the attack against this algorithm by calling it from her own applet and enclosing it by glitches to quickly test if its implementation is secure. It is also possible to add a glitch just before accessing the data of the targeted applet so as to synchronize a physical attack to try to bypass the checks of the firewall thanks to the disturbance of the hardware component.

7 Experiments

A few years ago, Sebastien Garcia[9] carried out experiments based on identical techniques to achieve matching attacks with the power and electromagnetic emanations for the ITSEF of SERMA Technologies. He worked at the assembly language level and he obtained interesting results but with the product that he used it was very hard to get a good dictionary because for some instructions it was even difficult to differentiate the signals. We are in a different situation. A bytecode is in fact a sequence of assembly language instructions that represents a bigger pattern than a single assembler instruction, therefore we believe that it may be possible to effectively build such a dictionary of Java Card bytecodes. Furthermore, since each bytecode corresponds to a really different sequence of assembler instructions, it is most likely that their signature will also be significantly different.

We have carried out some experiments on Java Cards at the ITSEF of SERMA Technologies using the physical characterization methods described in this paper in order to develop a methodology to reverse engineer official applications. We succeeded to build our characterization tests with the JCatools software environment [33, 34] that we have developed and we have encountered no problem to load them on all the Java Cards (since the sequence of operations is legal from the bytecode verification point of view). Nevertheless these tests were performed unsuccessfully on Java Cards of the GemXpresso Pro family already certified at level EAL5+ of the Common Criteria. Based on our expertise, we now are sure this was not a good idea to begin with these products since we were only given two full days to make our experiments. Moreover we wanted to work on the most interesting – but also the most difficult – tests: the physical identification of bytecodes (we had already successfully tested the attacks described in section 6.2 during real evaluations of other products). We focused our efforts on the electromagnetic emanations (since this domain was the most interesting for SERMA Technologies and since most of the

[8]The Common Criteria for Information Technology Security Evaluation, abbreviated "CC", defines a language for defining and evaluating information technology security systems and products. The framework provided by the CC allows a manufacturer to define a set of security functional and assurance requirements for his product. The CC also provides evaluation laboratories with procedures for evaluating the products or systems against the specified requirements.

[9]IXL laboratory of the University Bordeaux 1.

time the integrated circuit chips of the Java Cards already have very good countermeasures against the power analysis approach) but we have failed to establish a simple dictionary. This is mainly due to the many expensive techniques required by the recent chips to really perform a systematic and complete search (removing the shield against the electromagnetic emanations, etc.). Moreover in our case the measurement of the electromagnetic emanations is specific for a given sample of a specific model of an integrated circuit chip because of the precision required for the antennae. However with a better equipment it may be possible to have the same measurements for all the samples of a chip model. So it should also be noted that if someone succeeded to establish such a dictionary it should be significant only for an individual chip type (perhaps a family of chips that integrate the same technology).

Some attacks presented in this paper or some improved versions are used by the ITSEF of SERMA Technologies to test and attack Java Card platforms, applets and their services for instance within the framework of a Common Criteria evaluation. We succeeded to load applications using a code similar to the one shown listing 5 and to execute them on real "old" Java Cards. Besides, we often used the physical characterization methods to be in the best experimental positions in order to perform attacks against cryptographic algorithm implementations (*see* Section 6.2). Most of the time these attacks are used with the theoretical assumption that the attacker could load her code (cards evaluated are not yet open). Obviously this assumption is taken into account in the quotation of the attacks to know if they exceed the desired evaluation assurance level.

8 Related work

Some similar projects on multiapplication smart cards [39, 40, 41] have been carried out in 1999 by the Gemplus teams. They addressed many issues related to code loading, the virtual machine, object sharing, the information flow between the applets, but they did not identify the attacks based on the physical characterization methods that we have described in section 6. More recent work focused on different topics: the project presented in [42] enables a smart card issuer to verify that a new applet securely interacts with already loaded applets; in [43] a generic security model for operating systems of multiapplication smart cards is presented, that formalizes the main security aspects of secrecy, integrity, secure communication between applications and secure loading of new applications.

Our approach is thus original: we have not found any related work based on the physical methods that we propose to use to characterize a platform. Perhaps they have already been used for power analysis or electromagnetic analysis of closed smart cards. Nevertheless such an approach has never been presented as a killer method to characterize any product because if it were used it could only be in a restricted context (*i.e.* with many assumptions: e.g. "code loading is forbidden on this closed smart card but using it anyway saves a lot of time for the identification of the interesting patterns; if this method were not used it would still be possible to achieve the same goal – although this is not obvious"). The contribution of this paper is to clearly introduce these methods to characterize open multiapplication cards. In this context, these methods really threaten the security because it is not possible to prevent code that uses them from being loaded since this is a valid operation.

9 Conclusion and future work

This paper describes what open multiapplication smart cards are and it surveys the problems that they raise. The majority of the problems and attacks presented in this paper (*i.e.* sections 5 and 6) were unpublished till now and we hope that sharing our experience with others interested in the area will help to secure the future open smart cards. Even if the problems raised and the countermeasures proposed may sometimes seem obvious, we have already used them successfully during a real product evaluation to quickly set up attacks. It should be noted that people at the Smart Card Centre of the Royal Holloway (University of London) seem to begin working in the same direction [44]. In the near future, at the XLIM[10] laboratory of the University of Limoges we will continue our experiments so as to get a bytecode dictionary.

[10]The laboratory will be created in 2006 and will gather in particular people of physics specialized in the antennas – old IRCOM laboratory –, people of mathematics specialized in the cryptographic – old LACO laboratory – and people of computer science specialized in the information security – old LMSI laboratory.

References

[1] International Organization for Standardization: Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols. ISO (1997)

[2] International Organization for Standardization: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. ISO (2005)

[3] Muir, J.A.: Techniques of Side Channel Cryptanalysis. Master's thesis, University of Waterloo, Ontario, Canada (2001) Master of Mathematics in Combinatorics and Optimization.

[4] Kocher, P.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1996) 104–113

[5] Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1999) 388–397

[6] Coron, J.S., Kocher, P., Naccache, D.: Statistics and Secret Leakage. In: Proceedings of Financial Cryptography (FC2000). Volume 1962 of Lecture Notes in Computer Science., Springer-Verlag (2001) 157–173

[7] Quisquater, J.J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. In: Proceedings of E-smart 2001. Volume 2140 of Lecture Notes in Computer Science., Springer-Verlag (2001) 200–210

[8] Gandol, K., Mourtel, C., Olivier, F.: ElectroMagnetic Analysis: Concrete Results. In: Proceedings of CHES'2001. Volume 2162 of Lecture Notes in Computer Science., Springer-Verlag (2001) 251–261

[9] Kömmerling, O., Kuhn, M.G.: Design Principles for Tamper-Resistant Smartcard Processors. In: Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 9–20

[10] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. In: Proceedings of Workshop on Fault Detection and Tolerance in Cryptography, Italy (2004)

[11] Giraud, C., Thiebeauld, H.: A survey on fault attacks. In: Proceedings of CARDIS'04, Smart Card Research and Advanced Applications VI, Toulouse, France, Kluwer academic publisher (2004) 159–176

[12] Skorobogatov, S., Anderson, R.: Optical Fault Induction Attacks. In: Proceedings of Workshop on Cryptographic Hardware and Embedded Systmes (CHES 2002), San Francisco Bay (Redwood City), USA (2002)

[13] Chen, Z.: Java Card™ Technology for Smart Cards: Architecture and Programmer's Guide. The Java™ Series. Addison-Wesley (2000)

[14] Sun microsystems: Java Card™ 2.2.1 Specifications. Sun microsystems (2003)

[15] MULTOS: The MULTOS™ Specification. (`http://www.multos.com/`)

[16] Hive-Minded: Smartcard.NET. (`http://www.hiveminded.com/`)

[17] SmartCards Trends: .NET Brings Web Services to Smart cards. SmartCards Trends **1** (2004) 12

[18] ZeitControl: BasicCard. (`http://www.basiccard.com/`)

[19] GlobalPlatform: GlobalPlatform. (`http://www.globalplatform.org/`)

[20] Rose, E., Rose, K.: Lightweight bytecode verification. In: In Workshop on Fundamental Underpinnings of Java, OOPSLA '98 Workshop., Vancouver, Canada (1998)

[21] Casset, L., Burdy, L., Requet, A.: Formal Development of an embedded verifier for Java Card Byte Code. In: Proceedings of the IEEE International Conference on Dependable Systems & Networks, Washington, D.C., USA (2002)

[22] Leroy, X.: On-Card Bytecode Verification for Java Card. In: Proceedings of the International Conference on Research in Smart Cards, E-Smart 2001, Springer-Verlag (2001) 150–164

[23] Leroy, X.: Bytecode verification on Java smart cards. Software-Practice & Experience **32** (2002) 319–340

[24] Cohen, R.M.: Defensive Java Virtual Machine Version 0.5 alpha. (1997)

[25] Barthe, G., Dufay, G., Jakubiec, L., Melo de Sousa, S.: A Formal Correspondence between Offensive and Defensive JavaCard Virtual Machines. In: Proceedings of VMCAI'02. Volume 2294 of Lecture Notes in Computer Science., Venice, Italy, Springer-Verlag (2002) 32–45

[26] Deville, D., Grimaud, G.: Building an "impossible" verifier on a Java Card. In: 2nd USENIX Workshop on Industrial Experiences with Systems Software, Boston, USA (2002)

[27] Foundation for Information Policy Research: Framework for Smart Card Use in Government – Consultation Response. http://www.cl.cam.ac.uk/users/rja14/cards.html (1999)

[28] Montgomery, M., Krishna, K.: Secure Object Sharing in Java Card. In: Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999)

[29] Witteman, M.: Java Card Security. Information Security Bulletin **8** (2003) 291–298

[30] Betarte, G., Giménez, E., Chetali, B., Loiseaux, C.: FORMAVIE: Formal Modelling and Verification of Java Card 2.1.1 Security Architecture. In: Proceedings of E-Smart 2002, Nice, France (2002) 215–229

[31] Govindavajhala, S., Appel, A.: Using Memory Errors to Attack a Virtual Machine. In: Proceedings of IEEE Symposium on Security and Privacy. (2003)

[32] Chaumette, S., Sauveron, D.: An efficient and simple way to test the security of Java Cards. In: Proceedings of 3rd International Workshop on Security In Information Systems : WOSIS 2005, (Miami, Florida, USA)

[33] Chaumette, S., Hatchondo, I., Sauveron, D.: JCAT: An environment for attack and test on Java Card. In: Proceedings of CCCT'03 and 9th ISAS'03. Volume 1., Orlando, FL, USA (2003) 270–275

[34] Hatchondo, I., Sauveron, D.: The JCatools website. (http://sourceforge.net/projects/jcatools/)

[35] Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G.: Improving Smart Card Security using Self-timed Circuits. In: Proceedings of ASYNC'02. (2002) 211–218

[36] Jacques J.A. Fournier, S.M., Li, H., Mullins, R., Taylor, G.: Security Evaluation of Asynchronous Circuits. In: Proceedings of CHES'2003. Volume 2779 of Lecture Notes in Computer Science. (2003) 137–151

[37] Quisquater, J.J., Samyde, D.: Automatic Code Recognition for smart cards using a Kohonen neural network. In: Proceedings of the 5th Smart Card Research and Advanced Application Conference (CARDIS'02). (2002)

[38] CCIMB: International Common Criteria home page. (http://www.commoncriteriaportal.org/)

[39] Girard, P., Lanet, J.L.: Java Card or How to Cope with the New Security Issues Raised by Open Cards? In: Proceedings of Gemplus Developer Conference, Paris, France (1999)

[40] Girard, P., Lanet, J.L.: New Security Issues raised by Open Cards. In: Information Security Technical Report. Volume 4. (1999) 19–27

[41] Girard, P.: Which security policy for multiapplication smart cards? In: Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 21–28

[42] Bieber, P., Cazin, J., Girard, P., Lanet, J.L., Wiels, V., Zanon, G.: Checking Secure Interactions of Smart Card: Applets Extended Version. Computer Security **10** (2002) 369–398 Special issue on ESORICS 2000.

[43] Schellhorn, G., Reif, W., Schairer, A., Karger, P., Austel, V., Toll, D.: Verification of a Formal Security Model for Multiapplicative Smart Cards. In: Proceedings of ESORICS 2000. Volume 1895 of Lecture Notes in Computer Science. (2000) 17–36

[44] Smart Card Centre of the Royal Holloway (University of London). (http://www.scc.rhul.ac.uk/projects.php)

Use of Rijndael Block Cipher on J2ME Devices for Encryption and Hashing

Serdar S. Erdem[1], Aysel Uyar[1], Hacı H. Kılınç[1], Mustafa Toyran[1,2]

[1] Elektronik Bölümü, Gebze Yüksek Teknoloji Enstitüsü, Turkey
serdem@gyte.edu.tr

[2] UEKAE, TÜBİTAK, Turkey
mtoyran@uekae.tubitak.gov.tr

Abstract

The Rijndael is a block cipher with variable block and key size. The Rijndael with 128 bit block size is adopted as the Advanced Encryption Standard (AES) in 2000 and has become widely used in the bulk data encryption. This work investigates the efficient implementations of the Rijndael for the 32 bit resource limited mobile devices using Java 2 Micro Edition (J2ME). It presents some general implementations of the Rijndael encryption and decryption supporting all possible block sizes and discusses the use of these implementations to construct hash functions. Fast key scheduling is crucial for the performance of the block cipher based hash functions. Thus, an efficient implementation of the Rijndael key scheduling algorithm for the resource limited devices is also presented.

The implementations are tested on the java phone platform and their performances are compared with the performances of the implementations in a public Java crypto library. Also, the performances of the hash functions based on these implementations are evaluated and compared with the performances of the popular hash algorithms.

Keywords: Rijndael cipher; Advanced encryption standard; Block Ciphers; Hash algorithms; Wireless security; J2ME; Java phones.

1 Introduction

Today, there are a wide variety of mobile devices on the market. People use them for communication, entertainment, connecting to internet, m-commerce, online banking and many other ways. The advances in technology improve the storage capabilities and computational power of these devices significantly. Nowadays, for a reasonable price, consumers can buy a mobile phone or PDA having MBytes of memory and a 32 bit processor operating at several hundred MHz. Moreover, many mobile devices have lightweight multithreaded operating systems which can run third party applications developed with Java and even C++. The Java enabled phones are the most common and easily affordable examples of these devices. A small Java virtual machine (KVM) placed in the Java phones runs the applications developed with J2ME (Java 2 Micro Edition). The J2ME platform supports a basic runtime environment with the core Java libraries. Several companies provide free development tools for J2ME. The J2ME platform and its tools allow developers to design numerous applications and utilities for mobile devices.

The mobile devices are widely used in security sensitive applications such as m-commerce and online banking, which require strong cryptography. The cryptographic algorithms and primitives required for these applications can be implemented with J2ME. The Bouncy Castle lightweight API [2] and the IAIK JCE Micro Edition API [8] are the important examples of the cryptographic libraries build for J2ME.

This work presents some efficient implementations of the Rijndael cipher having different features from the common implementations and evaluates their performances in Java phones. The encryption, decryption and key scheduling timings of the implementations are compared with the timings of the corresponding implementations in the Bouncy Castle library. The hash functions construction from the

block ciphers are discussed [12]. The performances of the hash functions constructed from our Rijndael implementations are evaluated and compared with the timings of the popular hash functions.

Our implementations have the following features.

1. The secret key is expanded with an optimized key schedule avoiding integer division and modulo operations, which are expensive in the constrained devices. This key schedule runs three times faster in Java phones than the one implemented in the Bouncy Castle lightweight API.

2. All the key sizes and block sizes permitted by the Rijndael specifications in [3, 7] are supported without a significant sacrifice in the speed. The Rijndael is extendable to different block sizes and key sizes easily though many cryptographic libraries support only the parameters mandated by the Advanced Encryption Standard (AES), which is a subset of the Rijndael.

These features increase the potential uses of our implementations. For example, a fast key schedule is very beneficial for the devices with low dynamic memory such as mobile phones. This is because the expanded key of a block cipher is quite large, thus recomputing its bytes as needed in each encryption and decryption may be preferred to storing it into the precisious dynamic memory.

Also, key scheduling is a computational bottleneck for the hash functions based on the block ciphers. The improved key scheduling presented in this work enables the construction of the fast hash functions based on the Rijndael. The hash functions producing hash values in different sizes can easily be constructed using a general Rijndael implementation supporting different block and key sizes.

The results show that the encryption and the decryption throughput varies between the order of a few Kbit/s and a few Mbit/s depending on the phone used. Also, for the computation of the hash values larger than 200 bits, the hash functions based on the Rijndael are faster than the dedicated hash algorithms including SHA224 and SHA256. SHA224 and SHA256 are the alternatives for the widespread hash algorithm SHA1, for which a collision attack is reported recently [13].

The remainder of this paper is organized as follows. The next section gives mathematical preliminaries needed to understand the Rijndael algorithm. Section 3 gives the details of the Rijndael encryption, decryption and key expansion. Also, it presents a highly optimized key expansion method supporting all the allowed key sizes. Section 4 describes some Rijndael encryption and decryption implementations which can handle all the allowed block sizes beside the 128 bit AES block size. Section 5 has the timing results, comparisons and an evaluation of the hash functions based on the Rijndael.

2 Mathematical Preliminaries

The transformations used in the Rijndael are based on the finite field arithmetic. The data bytes are treated as elements of GF(2^8) in these transformations. The GF(2^8) elements are the polynomials of degree 7 or less, with binary coefficients. A data byte b with the bits $b_7, \ldots, b_1, b_0$ represents the following field element.

$$b = b_7x^7 + \cdots + b_1x + b_0$$

For example, the byte $\texttt{C2} = (1100\,0010)_2$ represents

$$\texttt{C2} = x^7 + x^6 + x$$

The arithmetic in GF(2^8) is similar to the polynomial arithmetic except the following differences.

1. The addition and the subtraction of the binary coefficents are both defined as XOR operation

2. The results of the arithmetic operations are reduced modulo a degree 8 irreducible polynomial. The Rijndael uses the irreducible polynomial $\texttt{11B} = x^8 + x^4 + x^3 + x + 1$.

As a result, addition and subtraction of elements in GF(2^8) are performed XORing their corresponding coefficients. Also, multiplication of elements can be performed by succesive multiplications by $\texttt{02} = x$ and additions. The operation"multiplication by $\texttt{02} = x$ in GF(2^8)" is called *xtime* and can be used to implement some Rijndael transformations. This operation can be performed as follows.

$$\begin{aligned} x \cdot b &= x(b_7x^7 + \cdots + b_1x + b_0) \bmod x^8 + x^4 + x^3 + x + 1 \\ &= (b_6x^7 + \cdots + b_1x^2 + b_0x) - (b_7x^4 + b_7x^3 + b_7x + b_7) \end{aligned} \tag{1}$$

As understood, the multiplication of the element b by $\texttt{02} = x$ can be performed at the byte level by shifting its byte representation one bit left and, if b_7 is nonzero, subtracting (XORing) the result with $\texttt{1B} = x^4 + x^3 + x + 1$.

3 The Rijndael Cipher

The Rijndael [3, 4, 5, 6] is a symmetric block cipher with a variable block size and key size. The Rijndael with 128 bit block size is adopted as the AES in Oct 2000 by NIST. This new standard is a replacement for the obsolete DES (Data Encryption Standard).

The Rijndael algorithm encrypts data in nb word blocks with an nk word key where a word is 32 bits. nb and nk can be 4, 5, 6, 7 or 8 words (128, 160, 192, 224 or 256 bits) independently of each other [3, 7]. The 128 bit encryption is sufficient for most security applications. This corresponds the Rijndael encryption with the parameters $nb = nk = 4$. The larger values for nb and nk enhance the security further. Figure 1 illustrates the Rijndael encryption. The encryption starts with the addition (actually bitwise XOR) of the plaintext block with a subkey (key 0). Then, it continues with the nr rounds of a nonlinear transformation where $nr = \max(nb, nk) + 6$.

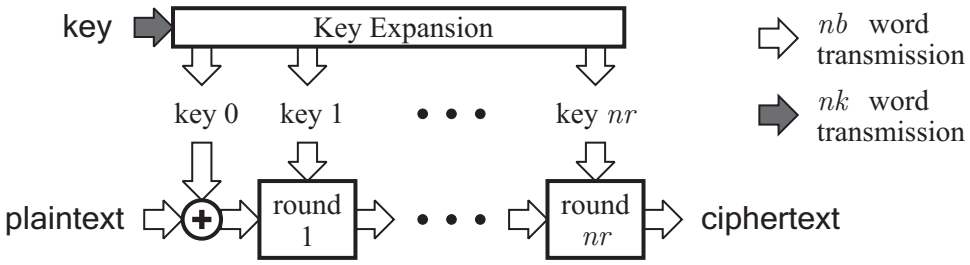


Figure 1: AES encryption and key expansion.

Each round uses a different subkey (key 1, 2, ... , nr) called the round key. Each subkey is nb words. They are all generated by the expansion of the nk word key with a key scheduling algorithm. The key scheduling algorithm needs to work only one time. Once the subkeys are generated, they can be stored in memory and used in the encryption of the multiple data blocks.

The nb word intermediate results between the rounds are called *state*. Let $\{s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j}\}$ denote the bytes of the jth word of a state. The Rijndael algorithm arranges these bytes in a $(4 \times nb)$ state matrix as shown below.

$$state = \begin{bmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,nb-1} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,nb-1} \\ s_{2,0} & s_{2,1} & \cdots & s_{2,nb-1} \\ s_{3,0} & s_{3,1} & \cdots & s_{3,nb-1} \end{bmatrix} \tag{2}$$

The encryption rounds consist of four transformations which process the state matrix. These are

SubByte, ShiftRow, MixColumn, and AddRoundKey

transformations in order. The final round omits the MixColumn transformation.

The decryption is illustrated in Figure 2. It is the inverse of the encryption operation. From the last

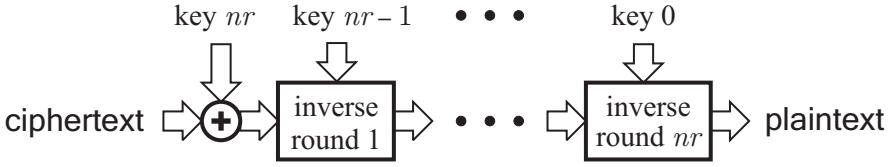


Figure 2: AES decryption.

one to the first one, the inverses of the encryption steps are performed in the reverse order. Naturally, the subkeys are used in the reverse order too. The decryption rounds perform the transformations

InverseShiftRow, InverseSubByte, AddRoundKey, and InverseMixColumn

in order. These are the inverses of the four transformations in the encryption rounds. The final round omits the InverseMixColumn transformation. Also, note that the inverse of AddRoundKey is equivalent to itself. This is because the addition is actually defined as XOR (exclusive or) operation in the Rijndael.

3.1 SubByte and Inverse SubByte Transformations

The SubByte transformation substitutes each byte in the state matrix with another byte using a fixed 256 byte lookup table as shown below.

$$state = \begin{bmatrix} \text{sbox}[s_{0,0}] & \text{sbox}[s_{0,1}] & \dots & \text{sbox}[s_{0,nb-1}] \\ \text{sbox}[s_{1,0}] & \text{sbox}[s_{1,1}] & \dots & \text{sbox}[s_{1,nb-1}] \\ \text{sbox}[s_{2,0}] & \text{sbox}[s_{2,1}] & \dots & \text{sbox}[s_{2,nb-1}] \\ \text{sbox}[s_{3,0}] & \text{sbox}[s_{3,1}] & \dots & \text{sbox}[s_{3,nb-1}] \end{bmatrix}$$

where sbox[ ] denotes the lookup table for the SubByte transformation. The inverse SubByte transformation performs the inverse operation by using another 256 byte lookup table.

The SubByte transformation is actually a nonlinear function over GF(2^8). $\text{sbox}[s_{i,j}] = f \cdot (s_{i,j})^{-1} + g$ where $f = \texttt{1F} = x^4 + x^3 + x^2 + x + 1$ and $g = \texttt{63} = x^6 + x^5 + x + 1$. If $s_{i,j} = 0$, $\text{sbox}[s_{i,j}] = g$.

3.2 ShiftRow and Inverse ShiftRow Transformations

The ShiftRow transformation cyclically shifts left (rotates left) the rows of the $(4 \times nb)$ state matrix in (2). The shift amounts depend on the row number $i = 0, 1, 2, 3$ and the plaintext block size $nb = 4, 5, 6, 7, 8$. Table 1 gives the shift amount for each row.

shift		row# i			
		0	**1**	**2**	**3**
nb	**4,5,6**	0	1	2	3
	7	0	1	2	4
	8	0	1	3	4

Table 1: Cyclic shift amounts for each row in ShiftRow transformation.

After ShiftRow transformation, the new elements of the state matrix, $\hat{s}_{i,j}$, are as follows

$$\begin{bmatrix} \hat{s}_{0,j} \\ \hat{s}_{1,j} \\ \hat{s}_{2,j} \\ \hat{s}_{3,j} \end{bmatrix} = \begin{bmatrix} s_{0,j} \\ s_{1,(j+1 \bmod nb)} \\ s_{2,(j+sh2 \bmod nb)} \\ s_{3,(j+sh3 \bmod nb)} \end{bmatrix}$$

Here, $sh2$ and $sh3$ are the shift amounts for the 2nd and the 3rd rows respectively. $sh2 = 2$ or 3 and $sh3 = 3$ or 4 according the block size nb. The inverse ShiftRow transformation cyclically shifts the rows to the right instead of the left to undo the ShiftRow transformation.

3.3 MixColumn and Inverse MixColumn Transformation

The MixColumn transformation operates on the columns of the $(4 \times nb)$ state matrix in (2). The MixColumn transforms each column into a new column by multiplying it with a (4×4) constant matrix in GF(2^8) as shown below.

$$\begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} \tag{3}$$

Above, the bytes in the constant matrix and the state matrix are viewed as GF(2^8) elements and, their addition and multiplication are carried out according to the rules of GF(2^8) arithmetic.

The inverse MixColumn transformation is the following matrix multiplication.

$$\begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = \begin{bmatrix} 0\text{E} & 0\text{B} & 0\text{D} & 09 \\ 09 & 0\text{E} & 0\text{B} & 0\text{D} \\ 0\text{D} & 09 & 0\text{E} & 0\text{B} \\ 0\text{B} & 0\text{D} & 09 & 0\text{E} \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} \tag{4}$$

The equation (4) gives really the inverse transformation because the multiplication of the (4×4) constant byte matrices in (3) and (4) in GF(2^8) yields the identity matrix.

3.4 AddRoundKey Transformation and Key Scheduling

The AddRoundKey transformation XORs the nb word plaintext block with an nb word subkey. The Rijndael uses $nr+1$ subkeys as seen in Figure 1. The required $(nr+1)nb$ word key material is generated from the nk word secret key by a key expansion algorithm. In this section, we first illustrate the key expansion algorithm of the Rijndael. Then, we present an implementation avoiding the modulo and division operations, which are slow on the resource limited devices.

The key expansion algorithm consists of a series of expansion rounds. Each expansion round generates a new nk word key material as output, taking the key material generated by the previous round as input. Figure 3 illustrates the kth key expansion round having the input words W_j and the output words $\hat{W}_j$ for $j = 1, \ldots, nk-1$. The inputs of the inital key expansion round are the words of the secret key. Because

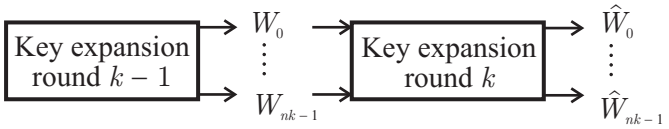


Figure 3: Key expansion round having the input words W_j and the output words $\hat{W}_j$.

we need $(nr+1)nb$ word key material, the round number $k = 1, \ldots, \lceil (nr+1)nb/nk \rceil$.

The input and output words of the kth round are related as follows.

$$\begin{array}{lllll} \hat{W}_0 = W_0 & \text{XOR} & f(W_{nk-1}, k) & & \\ \hat{W}_j = W_j & \text{XOR} & \hat{W}_{j-1} & & \text{for } j = 1, \ldots, nk-1 \end{array}$$

where f is a transformation on the 32 bit words. Let the word w have the bytes b_0, b_1, b_2 and b_3 from least to most significant. f is the following nonlinear transformation.

$$f(w,k) = f\left(\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}, k \right) = \begin{bmatrix} \text{sbox}[b_1] \\ \text{sbox}[b_2] \\ \text{sbox}[b_3] \\ \text{sbox}[b_0] \end{bmatrix} \text{XOR} \begin{bmatrix} \text{Rcon}[k] \\ 00 \\ 00 \\ 00 \end{bmatrix}$$

where Rcon[k] is a table of constants such that Rcon[k] = $02^{k-1} = x^{k-1}$ in GF(2^8). As can be understood, the transformation f rotates a word 8 bits right, performs the SubByte transformation to its bytes and XORs it by an 8-bit constant Rcon[k].

If $nk > 6$, the expansion round becomes

$$\begin{array}{lllll} \hat{W}_0 = W_0 & \text{XOR} & f(W_{nk-1}, k) & & \\ \hat{W}_j = W_j & \text{XOR} & \hat{W}_{j-1} & & \text{for } j = 1, 2, 3 \\ \hat{W}_4 = W_4 & \text{XOR} & g(\hat{W}_3) & & \\ \hat{W}_j = W_j & \text{XOR} & \hat{W}_{j-1} & & \text{for } j = 5, \ldots, nk-1 \end{array}$$

where g is a transformation performing the SubByte transformation to the bytes of its input.

The key expansion algorithm can be found in [3, 7]. Below, we introduce a more efficient implementation avoiding remainder and division operations.

Algorithm 1. Key expansion

INPUT: The screet key words key_j for $j = 0, \ldots, nk-1$,
the expanded key length $Wlen = nb(nr+1)$.

OUTPUT: The expanded key words W_j for $j = 0, \ldots, Wlen-1$.

1. for($j = 0$ to $nk-1$)
2. $W_j = key_j$
3. for($k = 1$ and $j = nk$ in the 1st iteration;
   break if $j \leq Wlen - nk$ in each iteration;
   $k = k+1$ and $j = j+nk$ in the iterations other than the 1st)
4. $W_j = W_{j-nk}$ XOR $f(W_{j-1}, k)$
5. $W_{j+1} = W_{j-nk+1}$ XOR W_j
6. $W_{j+2} = W_{j-nk+2}$ XOR W_{j+1}
7. $W_{j+3} = W_{j-nk+3}$ XOR W_{j+2}
8. if($nk = 4$) continue
9. if($nk > 6$) $tmp = g(W_{j+3})$
10. else $tmp = W_{j+3}$
11. $W_{j+4} = W_{j-nk+4}$ XOR tmp
12. if($nk = 5$) continue
13. $W_{j+5} = W_{j-nk+5}$ XOR W_{j+4}
14. if($nk = 6$) continue
15. $W_{j+6} = W_{j-nk+6}$ XOR W_{j+5}
16. if($nk = 7$) continue
17. $W_{j+7} = W_{j-nk+7}$ XOR W_{j+6}
18. if($j < Wlen$)
19. $W_j = W_{j-nk}$ XOR $f(W_{j-1}, k)$
20. $j = j+1$
21. while($j < Wlen$)
22. $W_j = W_{j-nk}$ XOR W_{j-1}
23. $j = j+1$

Here, the first nk words of the expanded key are set to the secret key. The remaining $Wlen - nk$ words are generated by a series of iterations. Each iteration expands the key nk words. Let q be the number of the iterations. Then, we have

$$Wlen - nk = q * nk + r$$

for some $r < nk$. If $r = 0$, the iteration stops. If not, r more words are generated. Also, we use only the transformation f but not g in this last expansion, because of the fact that $r < 4$ for all possible values of nb when $nk > 6$.

4 A General Implementation for 32 Bit Processors

In this section, we present general Rijndael encryption and decryption algorithms for 32 bit processors, which support variable block sizes. In this work, we consider only 32 bit platforms because the common J2ME devices like the java enabled phones have 32 bit processors. Moreover, the KVM, java virtual machine for J2ME, is designed for 16 and 32 bit processors.

Our implementations are based on the three basic methods mentioned in the AES proposal [3].

1. The first method uses two 256 byte lookup tables for the SubByte and the Inverse SubByte transformations. We will call this method RJ.

2. The second one uses two additional 1 Kbyte lookup tables to perform the substitution layer and the MixColumn layer efficiently. We will call this method RJ+.

3. The third one uses eight additional 1 Kbyte lookup tables to perform the substitution layer and the MixColumn layer efficiently. We will call this method RJ++.

Let w be a 32 bit word with the bytes b_0, b_1, b_2 and b_3 from least to most significant. For 32 bit platforms, the following byte extraction and word construction operations are helpful.

$$b_0 = \text{byte0}(w), \quad b_1 = \text{byte1}(w), \quad b_2 = \text{byte2}(w), \quad b_3 = \text{byte3}(w)$$

$$w = \{b_0, b_1, b_2, b_3\}.$$

These operations can easily be implemented in the programming languages such as C, Java and assembly with some shift and AND operations. Also, some Rijndael transformations perform rotations on the 32 bit operands. In our notation, ROR8(w), ROR16(w) and ROR24(w) denote the rotation of w 8, 16 and 24 bits right respectively.

In our implementation design, we use an $(nb + sh3 - 1)$ word array to store the state matrix where $sh3$ is the shift performed by the ShiftRow transformation in the third row of the state matrix. Let $s_0, s_1, \ldots, s_{nb+sh3-1}$ denote this array. $s_{nb}, s_{nb+1}, \ldots, s_{nb+sh3-1}$ are the redundant words used to ease the ShiftRow transformation. On the other hand, for $j < nb$,

$$s_j = \{s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j}\}$$

in encryption and

$$s_j = \{s_{0,nb-1-j}, s_{1,nb-1-j}, s_{2,nb-1-j}, s_{3,nb-1-j}\}$$

in decryption where $s_{i,j}$ are the byte elements of the $(4 \times nb)$ state matrix in (2). Then, s_j for $j < nb$ contains the jth column of the state matrix in encryption, while it contains the $(nb - 1 - j)$th column of the state matrix in decryption. That is we store the columns in reverse order during decryption. The practical significance of storing the state matrix columns in reverse order during decryption is that the inverse ShiftRow transformation becomes equivalent to the ShiftRow transformation.

4.1 RJ

The method RJ uses a 256 byte lookup table for the SubByte and another 256 byte lookup table for the inverse SubByte transformation. Let us call these tables sbox[·] and isbox[·] respectively. The SubByte and ShiftRow transformations can be performed for all possible values of the block size nb as follows.

Algorithm 2. SubByte and ShiftRow transformations for the method RJ.
INPUT: The words $s_0, \ldots, s_{nb-1}$ storing the state matrix such that $s_j = \{s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j}\}$
OUTPUT: The words $s_0, \ldots, s_{nb-1}$ storing the transformed state matrix.
REDUNDANT: The words $s_{nb}, s_{nb+1}, \ldots, s_{nb+sh3-1}$.

1. for($j = 0$ to $sh3 - 1$)
2. $\quad s_{j+nb} = s_j$
3. for($j = 0$ to $nb - 1$)
4. $\quad b_0 = \text{byte0}(s_j)$, $b_1 = \text{byte1}(s_{j+1})$, $b_2 = \text{byte2}(s_{j+sh2})$, $b_3 = \text{byte3}(s_{j+sh3})$
5. $\quad s_j = \{\text{sbox}[b_0], \text{sbox}[b_1], \text{sbox}[b_2], \text{sbox}[b_3]\}$

Here, the words $s_0, s_1, \ldots, s_{sh3-1}$ are first copied into the redundant words. Then, the substitutions and the left shifts are performed. Because the redundant words $s_{nb}, s_{nb+1}, \ldots, s_{nb+sh3-1}$ are set to $s_0, s_1, \ldots, s_{sh3-1}$, the left shifts perform the cyclic left shifts of the state matrix rows effectively.

The inverse SubByte and inverse ShiftRow transformations can be performed for all possible values of the block size nb as follows.

Algorithm 3. Inverse SubByte and inverse ShiftRow transformations for the method RJ.

INPUT: The words s_j storing the state matrix such that
$s_j = \{s_{0,nb-1-j},\ s_{1,nb-1-j},\ s_{2,nb-1-j},\ s_{3,nb-1-j}\}$.

OUTPUT: The words $s_0, \ldots, s_{nb-1}$ storing the transformed state matrix.

REDUNDANT: The words $s_{nb}, s_{nb+1}, \ldots, s_{nb+sh3-1}$.

1. for $j = 0$ to $sh3 - 1$
2. $\quad s_{j+nb} = s_j$
3. for $j = 0$ to $nb - 1$
4. $\quad b_0 = \text{byte0}(s_j),\ b_1 = \text{byte1}(s_{j+1}),\ b_2 = \text{byte2}(s_{j+sh2}),\ b_3 = \text{byte3}(s_{j+sh3})$
5. $\quad s_j = \{\text{isbox}[b_0],\ \text{isbox}[b_1],\ \text{isbox}[b_2],\ \text{isbox}[b_3]\}$

Here, s_j for $j < nb$ contains the $(nb - 1 - j)$th column of the state matrix. That is the state matrix columns are stored in reverse order. Thus, the bytes of the columns are left shifted as in Algorithm 2 and the both algorithms are exactly the same except the lookup tables used.

The MixColumn and the inverse MixColumn transformations are applied the columns of the state matrix individually as seen from (3) and (4). Thus, the words $s_0, \ldots, s_{nb-1}$ containing these columns are processed independently. The MixColumn layer involves multiplications by small GF(2^8) elements and thus can be performed efficiently by using the xtime operation described in (1). The implementation examples of the MixColumn layer can be found in [1, 3, 6].

4.2 RJ+ and RJ++

The methods RJ+ and RJ++ perform the last round, which omits the MixColumn layer, in the same way as the method RJ. However, these three methods differ in the other rounds.

In encryption and decryption, the method RJ++ uses eight additional lookup tables transforming a byte into a 32 bit word besides the lookup tables used in the method RJ.

For an arbitrary input byte b, the additional tables used in the encryption give the 32 bit words

$$\begin{aligned}
\text{T0}[b] &= \{\texttt{02}\,\text{sbox}[b], \texttt{01}\,\text{sbox}[b], \texttt{01}\,\text{sbox}[b], \texttt{03}\,\text{sbox}[b]\}\\
\text{T1}[b] &= \{\texttt{03}\,\text{sbox}[b], \texttt{02}\,\text{sbox}[b], \texttt{01}\,\text{sbox}[b], \texttt{01}\,\text{sbox}[b]\}\\
\text{T2}[b] &= \{\texttt{01}\,\text{sbox}[b], \texttt{03}\,\text{sbox}[b], \texttt{02}\,\text{sbox}[b], \texttt{01}\,\text{sbox}[b]\}\\
\text{T3}[b] &= \{\texttt{01}\,\text{sbox}[b], \texttt{01}\,\text{sbox}[b], \texttt{03}\,\text{sbox}[b], \texttt{02}\,\text{sbox}[b]\}
\end{aligned}$$

while the additional tables used in the decryption give the 32 bit words

$$\begin{aligned}
\text{invT0}[b] &= \{\texttt{0E}\,\text{sbox}[b], \texttt{09}\,\text{sbox}[b], \texttt{0D}\,\text{sbox}[b], \texttt{0B}\,\text{sbox}[b]\}\\
\text{invT1}[b] &= \{\texttt{0B}\,\text{sbox}[b], \texttt{0E}\,\text{sbox}[b], \texttt{09}\,\text{sbox}[b], \texttt{0D}\,\text{sbox}[b]\}\\
\text{invT2}[b] &= \{\texttt{0D}\,\text{sbox}[b], \texttt{0B}\,\text{sbox}[b], \texttt{0E}\,\text{sbox}[b], \texttt{09}\,\text{sbox}[b]\}\\
\text{invT3}[b] &= \{\texttt{09}\,\text{sbox}[b], \texttt{0D}\,\text{sbox}[b], \texttt{0B}\,\text{sbox}[b], \texttt{0E}\,\text{sbox}[b]\}
\end{aligned}$$

By the help of these tables, we can perform not only the substitutions in the substitution layer but also GF(2^8) multiplications in the MixColumn layer. This can be seen from the equations (3) and (4). As a result, we can combine the substitution and the MixColumn layers.

The following algorithms implement the method RJ++ for all possible values of the block size nb.

Algorithm 4. SubByte, ShiftRow and MixColumn transformations for the method RJ++.

INPUT: The words $s_0, \ldots, s_{nb-1}$ storing the state matrix such that $s_j = \{s_{0,j},\ s_{1,j},\ s_{2,j},\ s_{3,j}\}$

OUTPUT: The words $s_0, \ldots, s_{nb-1}$ storing the transformed state matrix.

REDUNDANT: The words $s_{nb}, s_{nb+1}, \ldots, s_{nb+sh3-1}$.

1. for($j = 0$ to $sh3 - 1$)
2. $\quad s_{j+nb} = s_j$
3. for($j = 0$ to $nb - 1$)

4. $b_0 = \text{byte0}(s_j)$, $b_1 = \text{byte1}(s_{j+1})$, $b_2 = \text{byte2}(s_{j+sh2})$, $b_3 = \text{byte3}(s_{j+sh3})$
5. $s_j = \text{T0}[b_0]$ XOR $\text{T1}[b_1]$ XOR $\text{T2}[b_2]$ XOR $\text{T3}[b_3]$

Algorithm 5. Inverse SubByte, inverse ShiftRow and inverse MixColumn transformations for the method RJ++.

INPUT: The words s_j storing the state matrix such that
$s_j = \{s_{0,nb-1-j}, s_{1,nb-1-j}, s_{2,nb-1-j}, s_{3,nb-1-j}\}$.

OUTPUT: The words $s_0, \ldots, s_{nb-1}$ storing the transformed state matrix.

REDUNDANT: The words $s_{nb}, s_{nb+1}, \ldots, s_{nb+sh3-1}$.

1. for $j = 0$ to $sh3 - 1$
2. $s_{j+nb} = s_j$
3. for $j = 0$ to $nb - 1$
4. $b_0 = \text{byte0}(s_j)$, $b_1 = \text{byte1}(s_{j+1})$, $b_2 = \text{byte2}(s_{j+sh2})$, $b_3 = \text{byte3}(s_{j+sh3})$
5. $s_j = \text{invT0}[b_0]$ XOR $\text{invT1}[b_1]$ XOR $\text{invT2}[b_2]$ XOR $\text{invT3}[b_3]\}$

The AddRoundKey is performed after Algorithm 3 in the encryption and Algorithm 5 in the decryption to complete the round. However, this causes a problem since the AddRoundKey must come before the inverse MixColumn in the decryption. As a result, the subkey used in the round misses the inverse MixColumn transformation in the decryption. To fix this problem, inverse MixColumn transformation must be applied to the words of the expanded key before the bulk data decryption [3].

The RJ+ is the same as the RJ++ but uses only one pair of additional lookup tables. Let it use the pair T0 and invT0. It obtains the results of the other tables from T0 and invT0 by rotations as shown below.

$$\begin{array}{ll} \text{T1}[b] = \text{ROR24}(\text{T0}[b]) & \text{invT1}[b] = \text{ROR24}(\text{invT0}[b]) \\ \text{T2}[b] = \text{ROR16}(\text{T0}[b]) & \text{invT2}[b] = \text{ROR16}(\text{invT0}[b]) \\ \text{T3}[b] = \text{ROR8}(\text{T0}[b]) & \text{invT3}[b] = \text{ROR8}(\text{invT0}[b]) \end{array}$$

5 Performance Evaluation

In this section, we present the timing measurements for our implementations and the corresponding implementations in the Bouncy Castle library. Our experiments with Java phones like Sony Ericsson T610, T630, K700 and Nokia 6600 show that the performance of the java phones varies widely. For example, the encryption throughput of the proposed implementations in K700 is above 1 Mbit/s, while it is just 5 Kbit/s in T610 phones. This is a very amazing result if we think that these phones generally use 32 bit ARM processors running at clock frequencies from a few 10 Mhz to a few 100 Mhz. These wide performance differences may be explained with the significant efficiency gaps in the Java Virtual Machine implementations on the phones.

Here, we present the results of the tests carried out in Sony Ericsson K700. The java applications testing our implementations and the Bouncy Castle implementations are developed with the J2ME wireless toolkit 2.2.0 (Sun Microsystems, 2005).

We also evaluate the performance of the hash functions based on the Rijndael and compare them with the timings of the popular hash algorithms available in the Bouncy Castle library.

5.1 Encryption, Decryption and Key Scheduling Timings

The Bouncy Castle library implements the Rijndael Encryption/Decryption methods RJ, RJ+ and RJ++ mentioned previously. However, these implementations work only for the block and key sizes specified by the AES. That is the Rijndael implementations in the Bouncy Castle encrypt the data in 128 bit blocks with a 128, 192 or 256 bit key ($nb = 4$ and $nk = 4, 6, 8$).

Table 2 gives the performance comparison between the Bouncy Castle AES implementations and our Rijndael implementations. For this comparison, the block size is set to 128 bits ($nb = 4$) as mandated by the AES. Also, the key size is set to 128 bits ($nk = 4$) too. We also test the efficient method given in [1] for the resource limited devices. This method does not use any additional lookup table like the

method RJ. We call this method RJrow since it stores the rows of the state matrix in 32 bit words. This method is easy to implement only for 128 bit encryption.

128 bit key 128 bit blocks		Encryption (msec)	Decryption (msec)	128 bit key 128 bit blocks		Encryption (msec)	Decryption (msec)
Bouncy Castle AES	RJ	0.48	0.58	General Rijndael	RJ	0.46	0.62
	RJ+	0.37	0.35		RJ+	0.30	0.33
	RJ++	0.086	0.070		RJ++	0.092	0.089
					RJrow	0.25	0.36

Table 2: The Performances of the Bouncy Castle code and our code for 128 bit encryption.

Except the method RJ+, the encryption and the decryption timings of the Bouncy Castle implementations are a little bit faster. This is because our implementation is general and support all possible block sizes and key sizes. The method RJ++ shows the best performance thanks to its 8 Kbyte additional lookup tables. Even though RJrow uses no additional lookup tables its performance is as good as RJ+, which uses 2 Kbyte additional lookup tables. The poorest performance belongs to RJ, which use no additional lookup table like RJrow.

Table 3 gives the performance comparison between the Bouncy Castle AES key schedule and our key schedule algorithm. For this comparison, the block size is set to 128 bits ($nb = 4$). As seen, our key expansion algorithm is three times faster.

128 bit blocks	key size	Key Schedule (msec)	128 bit blocks	key size	Key Schedule (msec)
Bouncy Castle AES	128 bit	0.35	General Rijndael	128 bit	0.125
	192 bit	0.40		192 bit	0.130
	256 bit	0.50		256 bit	0.160

Table 3: Key schedule timings for different key sizes.

Table 4 shows the throughput of our RJ++ implementation where the block size and the key size are equal to each other. Note that because the round number is given by ($nr = \max(nb, nk)$), as the block and key sizes get larger, the throughput gets lower.

RJ++	$nk = nb = 4$ (128 bit)	$nk = nb = 5$ (160 bit)	$nk = nb = 6$ (192 bit)	$nk = nb = 7$ (224 bit)	$nk = nb = 8$ (256 bit)
Encryption (Mbit/sec)	1.39	1.26	1.11	1.06	1.02
Decryption (Mbit/sec)	1.44	1.13	1.11	1.06	1.02

Table 4: The throughput of our RJ++ implementation for different block and key sizes.

5.2 Hash Function Performance

Let E(k,d) denote the encryption of the data d with a block cipher using the key k and M_i denote the data blocks. We can construct the following four secure hash functions [12, 11].

- $H_i = \mathrm{E}(H_{i-1}, M_i) \oplus M_i$
- $H_i = \mathrm{E}(H_{i-1}, (H_{i-1} \oplus M_i)) \oplus (H_{i-1} \oplus M_i)$
- $H_i = \mathrm{E}(H_{i-1}, M_i) \oplus (H_{i-1} \oplus M_i)$
- $H_i = \mathrm{E}(H_{i-1}, (H_{i-1} \oplus M_i)) \oplus M_i$

where H_i is the hash after the message block M_i, H_0 is a predetermined initial value and $\oplus$ denote the XOR operation.

Note that a key expansion and an encryption is required to hash each message block in the above hash functions. Table 5 gives the total time of these computations for different block and key sizes.

RJ++	$nk = nb = 5$ (160 bit)	$nk = nb = 6$ (192 bit)	$nk = nb = 7$ (224 bit)	$nk = nb = 8$ (256 bit)
Encryption (msec)	0.127	0.174	0.212	0.250
Key Schedule (msec)	0.160	0.185	0.255	0.300
Total (msec)	0.287	0.359	0.467	0.550

Table 5: The total time elapsed for encryption and key scheduling with the method RJ++.

The four hash functions constructed from a block cipher process the bits as many as the block size in the time elapsed for the key expansion and the encryption. Thus, the throughput can be computed as

block size / (key expansion + encryption time)

If we use the RJ++ method, which is the fastest method, in the encryption, we can generate more efficient hash functions. Then, the main memory cost of the Rijndael based hash will be the large 8 Kbyte lookup tables used by the RJ++ method. Compared to these large lookup tables, the key scheduling requires an insignificant amount of memory. Thus, the memory requirements and cache usage of the Rijndael based hash will be similar to those of the encryption with the RJ++ method.

Table 6 gives the throughput of the hash functions based on our Rijndael implementations together with the timings of the hash algorithms implemented in the Bouncy Castle Library. Note that the size of the hash value equals to the size of the block and key sizes. However, we can obtain the 512 bit hash value using the 256 bit Rijndael encryption with 256 bit key using the MDC2 scheme [12]. The MDC2 scheme applies two key expansions and two encryptions for each 256 bit data. Thus, a 512 bit hash function based on the Rijndael has half the rate of the 256 bit one.

Table 6 shows that the Rijndael based hash functions producing more than two hundered bit hash value is faster than the dedicated hash functions.

Hash value in bits	Throughput(Mbit/sec)		
	Rijndael based hash	Dedicated hash algorithms	
160 bit	0.56	0.44 0.72	RIPEMD160 SHA1
192 bit	0.54	1.01	Tiger
224 bit	0.48	0.28	SHA224
256 bit	0.47	0.28	SHA256
512 bit	0.23	0.12	Whirlpool

Table 6: The performance comparison of the Rijndael based and the dedicated hash algorithms.

6 Conclusion and Discussion

Many Java enabled mobile devices have become quite capable and run various applications. The Rijndael block cipher is an important cryptographic algorithm and can be used to encrypt the data and to produce hash values in these devices. The encryption and decryption can be performed efficiently using a few Kbytes additional lookup tables. These additional lookup tables are easily affordable since even a simple java phone has tens of Mbytes of program memory.

The Rijndael can be used to construct fast hash functions with a fast key schedule. To support various hash sizes, we need a general Rijndael implementation supporting different block and key sizes. In this work, we present such a Rijndael implementation. Instead of using several hash functions to produce the hash values in the various sizes, using only a general implementation of the Rijndael for encryption and hashing is an attractive solution.

However, the hash functions based on the block ciphers may show weaknesses which are not relevant to the encryption. Thus, the security of the hash functions based on the Rijndael may need to be investigated.

References

[1] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti,and S. Marchesin "Efficient Software Implementation of AES on 32-Bit Platforms", *Cryptographic Hardware and Embedded Systems- CHES 2002*, pp. 159-171, B.S. Kaliski Jr., Ç. K. Koç, C. Paar.

[2] The Bouncy Castle Lightweight API Release 1.5, http://www.bouncycastle.org/download/

[3] J. Daemen, V. Rijmen "AES Proposal: Rijndael" 1999.

[4] J. Daemen, V. Rijmen. "The design of Rijndael", Springer-Verlag, 2002.

[5] J. Daemen, V. Rijmen, P.S.L.M. Baretto "Rijndael: Beyond the AES" *3rd Czech and Slovak Cryptography Workshop*, Dec 2002 Prague, Czech Republic.

[6] B. Gladman. Implementations of AES (Rijndael) in C/C++ and Assembler. http://fp.gladman.plus.com.cryptography__technology.rijndael

[7] B. Gladman "A Specification for Rijndael, the AES Algorithm", Sep 12 2003. http://fp.gladman.plus.com.cryptography__technology. rijndael.aes.spec.311.pdf

[8] IAIK JCE and iSaSiLk APIs, http://jce.iaik.tugraz.at/download

[9] "J2ME application-layer end-to-end security for m-commerce", *Journal of Network and Computer Applications 27*, pp. 13-32, 2004.

[10] NIST FIPS 197, "Advanced encryption standard (AES)", Nov. 26, 2001.

[11] B. Preneel, "Analysis and Design of Cryptographic Hash Functions", Ph.D. dissertation, Katholieke Universiteit Leuven, Jan. 1993.

[12] B. Schneier, "Applied Cryptography", John Wiley & Sons, 1996

[13] X. Wang, Y. L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1", *In proc. 25th Annual International Cryptology Conference (Crypto'05)*, August, 14-18 2005.

Forensic Geolocation of Internet Addresses using Network Measurements

Espen A. Fossen[1,*] André Årnes[2,*]

[1]Department of Telematics,
Norwegian University of Science and Technology,
O.S. Bragstads plass 2B, N-7491 Trondheim, Norway, espenaf@junta.no.

[2]Centre for Quantifiable Quality of Service in Communication Systems,[†]
Norwegian University of Science and Technology,
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway, andrearn@q2s.ntnu.no.

Abstract

This paper presents a method for estimating the geographical location of Internet hosts, providing Internet investigators with important evidence. A prototype application called GeoLocate has been developed for use in digital forensics investigations. GeoLocate provides real life results using a network delay measurement method called Constraint-Based Geolocation (CBG), using public network Looking Glass (LG) services as landmarks.It is shown that GeoLocate, using a small number of landmarks, can estimate the location of an Internet address within the timeframe of 1–2 minutes. The results depend on the "quality" of the landmarks used, making it possible to acquire better results if the correct landmarks are used.

Keywords: Computer Crime, Delay Measurements, Geolocation, Digital Forensic Science.

1 Introduction

In computer crime investigations, one often needs to find the topological or geographical location of Internet addresses. However, it is often difficult to accurately determine the geographical location of an IP or DNS address, and there are many uncertainties. A method for determining geographical location information for Internet addresses would be an important asset to law enforcement, advertisers, and other fields. This paper presents as prototype application called GeoLocate for acquiring geographical location of Internet addresses. The paper is based on work conducted in [1] and [2].

In contrast to the Public Switched Telephone Network (PSTN) system, the Internet is connectionless. Connectionless data transfer means that data is sent as packets and may be routed differently each time, making it a dynamic transport system that will work even if some of the routes are unavailable.

For example, only one link is needed between the machines and two routers to be able to transfer packets from host X to Y in figure 1. If only the links A1, C2 and B3 was working, a packet in a connectionless network would get through. PSTN systems on the other hand use the same route each time, making it more vulnerable to uavailable routes.

The strong hierarchical design of the PSTN systems is usually linked to geographical locations, making it easier to find the geographical location and owner of a telephone address. The Internet on the other hand was not designed with the same geographical topology in mind.

*This research has been performed in cooperation with the High Tech Crime Division of the Norwegian National Criminal Investigation Service. Both authors are associated with the High Tech Crime Division.

[†]The "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" is appointed by The Research Council of Norway, and funded by the Research Council, NTNU and UNINETT.

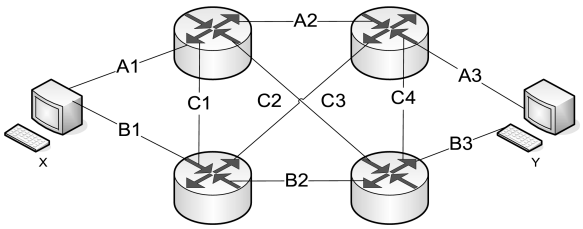


Figure 1: Possible topology of a connectionless network.

1.1 Tracing Internet Addresses

Digital forensic investigators often need to trace Internet addresses. There are two main forms of address tracing on the Internet, either active or passive tracing. Active tracing means probing or connecting to a given address and check for response time, network path or querying for services. This might mean using simple network auditing tools such as `ping` and `traceroute`, but also more complex tools such as `nmap`[1] and `amap`[2]. Active tools have in common that they usually create socket connections with the target address, meaning that it may be detected by the owner of the host.

The tools `traceroute` and `ping` can be used to acquire location information. They both send out Internet Control Message Protocol (ICMP) packets with it's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE. Traceroute, however gives us more helpful information, namely the path of a packet from the sender to the target host. Table 1 shows a traceroute from *zulu01.item.ntnu.no* to *uk.ny1.ny.geant.net*. Each intermediary hosts Round Trip Time (RTT) to and from the sender host is recorded and presented as a path with increasing RTT.

Hop	Host address	IP address	RTT [ms]	Δ [ms]
1	el-gsw.ntnu.no	(129.241.208.1)	0.211	0.211
2	ntnu-gsw.ntnu.no	(129.241.76.29)	0.242	0.031
3	trd-gw.uninett.no	(158.38.0.221)	0.243	0.001
4	oslo-gw1.uninett.no	(128.39.46.1)	7.988	7.745
5	no-gw.nordu.net	(193.10.68.101)	8.074	0.086
6	se-kth.nordu.net	(193.10.68.29)	15.470	7.396
7	nordunet.se1.se.geant.net	(62.40.103.117)	15.685	0.215
8	se.uk1.uk.geant.net	(62.40.96.126)	50.570	34.885
9	uk.ny1.ny.geant.net	(62.40.96.169)	119.277	68.707

Table 1: Traceroute from zulu01.item.ntnu.no to uk.ny1.ny.geant.net.

The packet going to *uk.ny1.ny.geant.net* first takes the path through several hosts in the *ntnu.no* domain, before it is routed to a gateway router in Trondheim, Norway. Hop 3 shows that the packet is routed through *trd-gw.uninett.no*, and then directly to *oslo-gw1.uninett.no*. While the RTT to the routers located in the *ntnu.no* domain and the gateway in Trondheim is very small (below 0.3 ms), the RTT to the gateway router in Oslo, Norway is close to 8 ms.

This network delay between the two hosts can be used to measure geographical distances. The link between the two gateway hosts in Trondheim and Oslo is probably connected via an optical fibre running alongside a road or railway track[3]. It should then be possible to calculate the cable length using the relation in equation 2, given the velocity of the signal v and the transmission time t.

The signalling speed in an optical fibre is $v = 1.962 \times 10^8$m/s [18]. The time t from Trondheim to Oslo is half the RTT to *oslo-gw1.uninett.no* and subtracting the one way delay to *trd-gw.uninett.no*. The transmission delay given by equation 1 used in relation 2 then gives us a distance of $d = 755.0$ km.

[1] `http://www.insecure.org/nmap/`

[2] `http://www.thc.org/thc-amap/`

[3] According to UNINETT, the Norwegian National Research and Education Network (NREN), the link is rented from Bane Tele. Bane Tele's fibre optics cables follow the railway track owned by The Norwegian State Railway (NSB).

$$t = \frac{\text{RTT hop 4} - \text{RTT hop 3}}{2} \tag{1}$$

$$d = v \times t \tag{2}$$

In comparison, the railway track[4] running from Trondheim to Oslo is 553.00 km. This makes for a 1.365 ratio between the real cable distance and the estimated distance. Finding the exact cable distance will often be very difficult for an investigator, as detailed information about the infrastructure of the optic fibre network is needed. In many cases it may be difficult enough to find any information about the links the packets use, as DNS names not always describe their geographical location. Research networks tend to provide more geographical information in DNS names than commercial networks, but investigators will have to work with both types of networks.

It is easier to estimate the cable distance using geographical coordinates, because finding two points on a map is alot simpler than finding the correct road or railway track distance between them. The geographical coordinates for Trondheim are latitude = 63.4200° N, longitude = 10.3900° E, and for Oslo latitude = 59.9100° N and longitude = 10.7900° E. Taking into account that the earth is not a perfect sphere, but rather an ellipsoidal the Haversine formula [17] is used to calculate the distance between the two points, giving us a distance of $d_{to} = 391.6$ km.

The estimated cable distance compared to the railway track distance is off by a ratio of 1.365, and the estimated cable to flight distance is off by 1.928. Both these results are poor compared to the actual distances, but measuring the distance over an ocean may provide better results. At hop 9 in table 1 the packet goes from the Cambridge in UK, to New York, USA. The one way delay comes to 34.41 ms, and by subtracting 0.200 ms for fibre optical signals boosters, the estimated delay comes to $t = 34.21$ ms. The results is then calculated to $d_{cn} = 5919$ km. This gives us a flight-cable distance ratio of 1.121, and the offset is most likly a result of the ocean floor not being completely flat, making the cable distance somewhat longer than a straight line. The cable length will probably not be the only source of error; it is highly likely that a large part of the error comes from processing time and packet queuing.

Even with this margin the results are much better then the one calculated for a land based connection between Trondheim and Oslo. It seems much more accurate to calculate the distance of cables running across the ocean floor than across land. Still there is the problem of knowing where the routers are physically located, resulting in time consuming detective work.

Another aspect of measurement based localization is direction. Suppose the destination is unknown, and the DNS names does not give us a hint in where it may be located. In which direction is the packet going, North? West? South? If the calculation had revealed that the packet had travelled the 391.6 km corresponding to the distance from Trondheim to Oslo, the location could be somewhere else, figure 2 shows where the host may be located in relation to Trondheim. Without knowing the correct direction, the packet could have ended up in Bergen, Norway, or maybe somewhere in Sweden. Without explicit knowledge about the network topology or direction it is impossible to know.

Passive tracing on the other hand uses third party databases or other resources not directly connected to the Internet address. The best example of passive tracing is to conduct whois [8] information searches for Internet addresses. This can be done for almost all Internet addresses. Information sources such as whois are open to anyone with the right software and a connection to the Internet. There are a number of public information sources that can be utilized by investigators, such as DNS lookup information, map services and Border Gateway Protocol (BGP) information. A type of information source that is available to anyone are Looking Glass (LG) servers.

1.2 Looking Glass

LG is a set of tools with a web based user-interface, providing different types of network auditing tools such as `traceroute`, `ping` and BGP routing information. LG servers makes it possible to run these network auditing tools remotely, and catch the feedback with a web browser, as the output is sent as HyperText Markup Language (HTML) or plain text via the HyperText Transfer Protocol (HTTP).

There are many different implementations of LG; the source code for one of them can be found at [15]. LG servers are usually altered to fit each network operators needs, but they almost always use

[4] `http://www.jernbaneverket.no/jernbanenettet/jernbane_nettet_i_tall/`

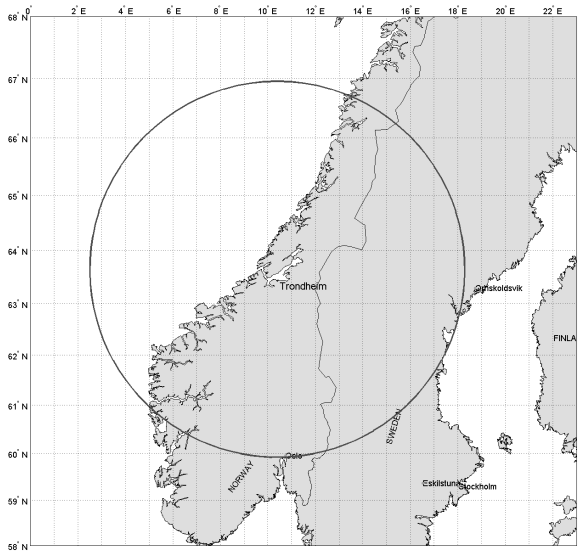


Figure 2: Circle with radius of 391.6 km centered in Trondheim, Norway.

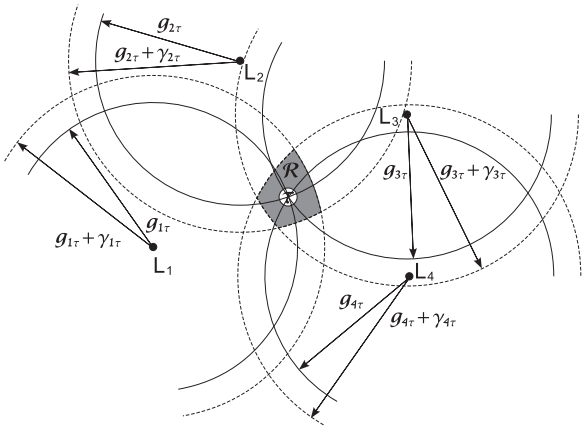


Figure 3: Multilateration with geographic distance constraints.

the same set of HTTP commands. Network operators and universities around the world set up web servers with LG, making them available to the public domain. [12] and [11] contains fairly updated lists of sites offering reverse traceroute and LG servers.

1.3 Related Work

Using DNS information to provide geographical location information was first proposed in [7]. This approach however requires a change in the DNS records, and such changes will not come over night. If geographical location information is added into the DNS records, it does not mean that the information is accurate or correct, as it is provided by the registering part. Beside DNS records there are tools such as eTracing [2], NetGeo [9] and IP2ALL [10] that can be used to acquire DNS and IP whois information. DNS and IP whois information can be used to acquire geographical location of Internet addresses, but this information may be unreliable and might not be accurate.

Padmanabhan and Subramanian investigated three other methods for acquiring geographical location of hosts [5]. The first one is called GeoCluster, the second GeoTrack and the third GeoPing. GeoCluster uses partly BGP routing information and partly information submitted by users, where the submitted information was gathered anonymously from 3 online services. The second method (GeoTrack) looks at traceroute information from several probes to the target host, using the DNS names of the target host or nearby hosts to infer the geographic location. The third method (GeoPing) uses network delay or one way delay from multiple landmarks to a target host. The location is then estimated based upon the assumption that the delay of a probe is fixed. By measuring the fixed delay from several landmarks, the position of target host can be approximated. The two first methods relays entirely or partly on information submitted by users, making it somewhat unreliable, while GeoPing uses only no submitted information. However GeoPing leads to a discrete space of answers, i.e. the number of answers is equal to number of reference hosts. Ziviani, Rezende and Duarte have later improved GeoPing [3]. The improvement reduces the number of landmarks needed, thus reducing number of distance vectors needed to be found. This approach does improve the efficiency of the method, but the fundamental problem of a discrete space of answers is still present. Gueye et al. has proposed a new method for geographical location of Internet hosts called Constraint-Based Geolocation (CBG) [4].

In [4] CBG is only tested using data from network traffic measurement projects such as the NLANR Active Measurement Project [13] and RIPE Test Traffic Measurements [14]. This data will only provide an implication on how well the method performs, and can not provide any real life results. Getting real

life results would require a large number of landmarks, distributed both geographically and in different types of network, like commercial backbones, university networks, research networks etc. By using public information sources with network auditing tools such `traceroute`, `ping` or LG, it is possible to acquire a large set of landmarks in different networks. Both the CBG method and methods for using and acquiring network delay measurements from public information sources has been implemented in the GeoLocate application.

2 Geolocation of Internet Addresses

The CBG methods uses distances from multiple known landmarks to estimate the position of an unknown host. This is conceptually similar to triangulation, but triangulation strictly means using angles from three known points to find the location of a fourth point. As CBG uses distances from multiple points of reference, instead of angles from three points, it has been called multilateration in [4]. Following the notation of [4], figure 3 shows how the multilateration is conducted using the geographical distance constraints from 4 landmarks.

However, before multilaterating the location of a target τ, it is necessary to find the distance constraint $\hat{g}_{i\tau}$ for each landmark L_i, and this is done by measuring the distance between landmark L_i to all other landmarks L_j, where $i \neq j$. Each distance is found by equation 2, where delay d is found by measuring the one way delay from L_i to landmark L_j using a LG server. By using the signalling speed in fibre, which is about 2/3 the speed of light in vacuum, or 1.962×10^8m/s, it is possible to find the distance between each landmark.

For calculating the distance constraint it would seem logical to use the lowest measured delay t to get the correct answer. As shown in [4] CBG is however much less dependent on the actual delay measurement, making it less affected by packet queuing and processing delay. Using the one way delay for calculating the distance between two hosts could possibly results in a underestimated geographical distance constraint as shown in figure 4(c). It is therefore necessary to increase the value of the delay by a factor of 2 to get more valid results, even if the result might be overestimated as in figure 4(a). Gueye, Ziviani, Crovella, and Fdida do not point out this fact directly in [4], but uses the full RTT between a landmark and the target as delay.

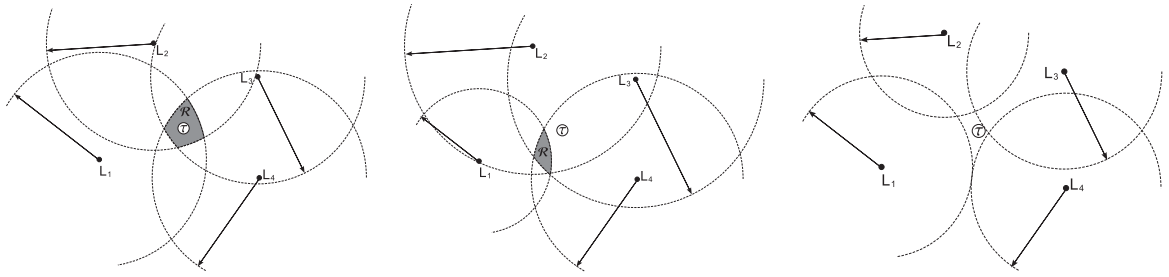
When the distances between all landmarks have been measured, a distance constraint is calculated. A more detailed explanation of this can be found in [1] and [4].

The next step in finding the position of a target host is to calculate the geographical distance constraints $\hat{g}_{i\tau}$ between the target τ and each landmark L_i. As shown in figure 3, the geographical distance constraint $\hat{g}_{i\tau} = g_{i\tau} + \gamma_{i\tau}$ between each landmark g_i and target τ is given by two factors, the real geographical distance $g_{i\tau}$ and an additive distance distortion $\gamma_{i\tau}$. While the real geographical distance $g_{i\tau}$ is fixed, the additive is affected by many factors, for example routing policies, queuing delays and cable lengths. An overestimate of the geographical distance constraint as shown in figure 4(a) is therefore necessary to obtain some results. It is however important to reduce the additive distance as much as possible, and CBG is able to do this by optimizing itself as described [4].

If the geographical distance constraint $\hat{g}_{i\tau}$ however is underestimated as shown in 4(c), it means that the landmark is not located where it was thought to be. Testing for underestimates will discover if the position of a landmarks is correct, making it possible to eliminate landmarks, or try to find their correct position. Landmarks with incorrect position can also produce mismatched results, as shown in figure 4(b). This error is harder to discover, but if the same landmark is tested on a target located in the opposite direction the results will probably be underestimated, and then we know that is is misplaced.

2.1 Confidence Region

The grey area created by the additive distance distortion $\gamma_{i\tau}$ in figure 3 is known as the confidence region $\mathcal{R}$. The confidence region gives us an estimate on how good the results are, and the higher the value, the lower the confidence in the result. This estimate is important for any location-aware application, as the results may be too uncertain to be used in some cases. As an example, a result may cause a location-aware application to show advertisement in the wrong language if the confidence region spans over several countries.



(a) Overestimated distance constraints

(b) Mismatch

(c) Underestimated distance constraints

Figure 4: Effects of over and underestimation of geographic distance constraints.

For forensic applications, the results will always be interesting, even if the confidence region is large, as it may provide valuable forensic information about network topology and the target host. The confidence region may actually work better in forensic applications than in many other location-aware applications.

2.2 Landmarks

A landmark can be a wide range of devices, primarily a computer or router with known location capable of running `ping`, `traceroute` or other network delay measurements tools. For benchmarking, it is often best to use network measurement data from projects such as the RIPE Test Traffic Measurements or NLANR AMP. Both these projects provide network traffic monitoring data from scientific research networks, but geolocating a target host located in a commercial or home network will provide different results. The estimate does not need to be worse, the quality of the result depends on a wide range of factors, such as quality of transmission links, amount of traffic, packet queuing and routing policies. Research networks usually route traffic on the Best-effort policy, but commercial networks may implement other kind of routing policies.

A real world application such as GeoLocate benefits from having landmarks in many different networks, since a target host may be located both in commercial and research networks. Research networks typically have very high capacity links, but they may not always own their own cables. A large set of landmarks distributed geographically and placed in different networks may provide a better real life result, but getting access to this kind of landmarks may be difficult. This may be solved by using public information sources like web servers providing public LG servers or scripts that provide remote access to either `ping` or `traceroute` applications. The best source of such remote access is LG servers, and the thing is that almost all research or commercial networks have such applications publicly available. Our problem with this solution is to get the correct geographical position of the landmark. There are several approaches to achieve this. For example, the network operator's website might give some clues, IP whois information, DNS names and DNS lookup information can be used. Users registering a domain might provide the DNS registry with falsified information, but it will be highly unlikely that a major network operator or backbone provider will provide false information on their web site or in the IP whois or DNS registry.

3 GeoLocate

GeoLocate is a prototype application for assessing the accuracy of the CBG method with the use of public information sources. Using LG servers as landmarks solves a lot of problems with deploying such an application, but it also poses some new questions. GeoLocate has so far only been used as a prototype application, a conceptual model of GeoLocate is shown in figure 6.

A set of 50 landmarks was selected from various LG lists for use in the testing. These landmarks were scattered across Europe in both research and commercial networks. During testing, several of the

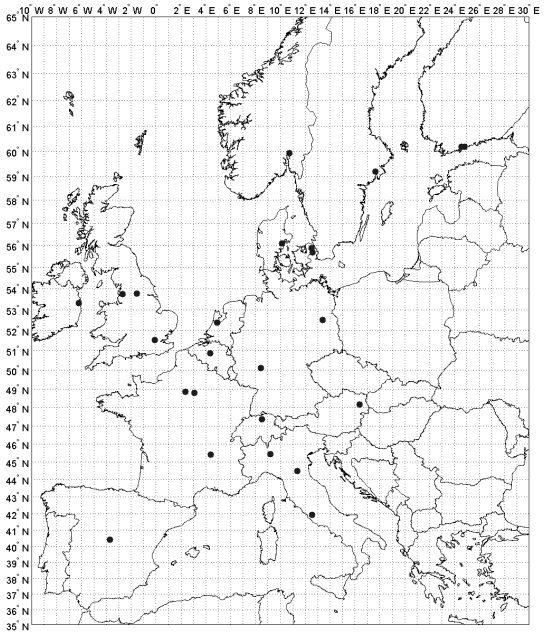


Figure 5: Geographical location of landmarks used by GeoLocate.

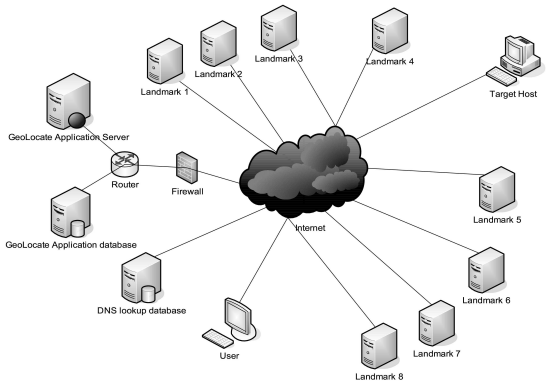


Figure 6: Conceptual model of GeoLocate application for location of Internet addresses.

Host	Lat.	Long.	Est. Lat.	Est. Long.	Δ[km]	$\mathcal{R}$[km^2]
tt01.ripe.net	52.3558° N	4.9510° E	52.4101° N	4.6481° E	21.5	13993.0
tt102.ripe.net	53.5508° N	10.0473° E	54.3444° N	8.9467° E	114.2	23010.0
tt109.ripe.net	63.8246° N	20.2673° E	59.0487° N	16.8649° E	562.4	1120300.0
tt126.ripe.net	47.4401° N	9.7563° E	48.9161° N	8.7529° E	180.4	211440.0

Table 2: GeoLocate results compared to actual location.

initial landmarks had to been removed. The biggest problem was not being able to find the correct geographical location of the landmarks. Figure 5 shows the location of the remaining 27 landmarks that was used for obtaining the experimental results.

3.1 Experimental Results

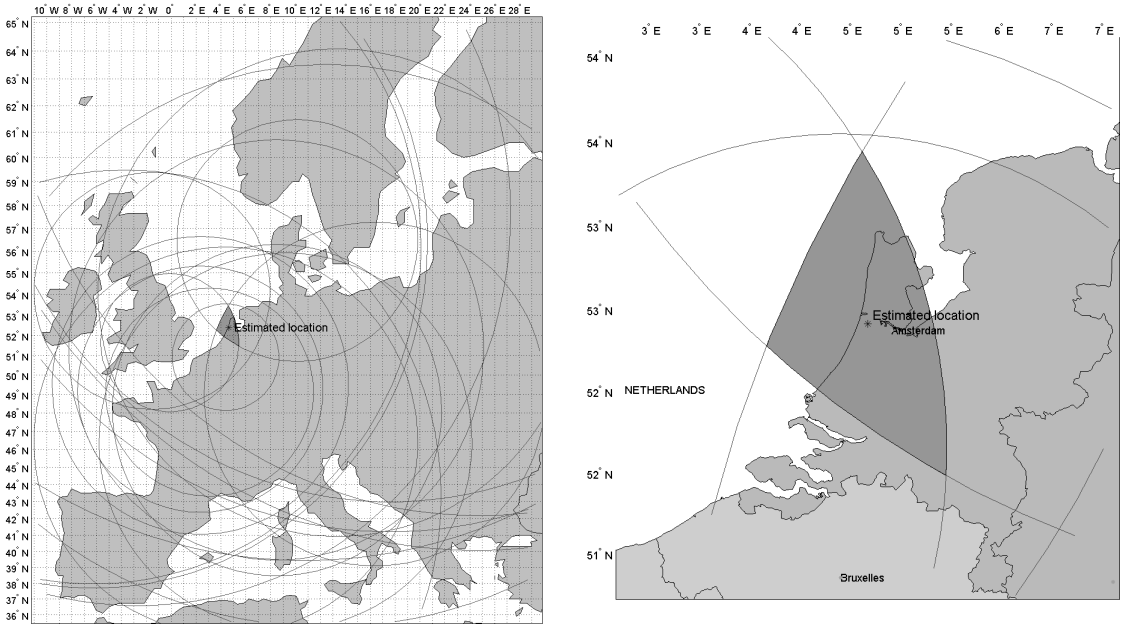
Four tests were conducted. Table 2 shows the hostname, actual position, estimated position and confidence region for each test. Each host is part of the RIPE TTM [14], and are located in different geographical areas. Each host has a Global Positioning System (GPS) card and a high speed connection to the Internet.

3.1.1 Location 1: tt01.ripe.net

The first test was run against the host "tt01.ripe.net", this test server is located in Amsterdam, Netherlands. Figure 7(a) shows a map of Europe with all the closed circles generated by the geographical distance constraints from each landmark to the target. The grey area or confidence region $\mathcal{R}$ looks small compared to a map of Europe, but it actually covers 13993.0 km^2. The smaller the region $\mathcal{R}$ is, the greater the confidence of the results.

Figure 7(b) takes a closer look at the actual region $\mathcal{R}$. The centroid of the region $\mathcal{R}$ is the estimated position of the address, and compared to the actual location of the address the difference is 21.5 km. While the position is somewhat off track, it is still within the estimated region $\mathcal{R}$.

Since the estimated position is 21.5 km from the target and the confidence region has an area of 13993.0 km^2 the results may be considered good compared to other location methods. The confidence region is within the same country, and the estimated point is very close to Amsterdam. Advertisement applications would look upon this result as reasonable, and provide the user with an advertisement from Netherlands or the Amsterdam area, but that all depends on the advertisement service itself. For a forensic application the results may confirm or disconfirm whois information acquired about the address. E.g. the whois information may claim that the address is registered to a company or individual located in South America. If whois information confirms the location of the target, further investigation on the location may not be needed. Should the geolocation find the address in another part of the world, it might be necessary to investigate further. For example, further investigations may discover that the address has been hijacked or that it is being rerouted to another source.



(a) Overview

(b) Closer view

Figure 7: Geolocating "tt01.ripe.net" with known location in Amsterdam, Netherlands.

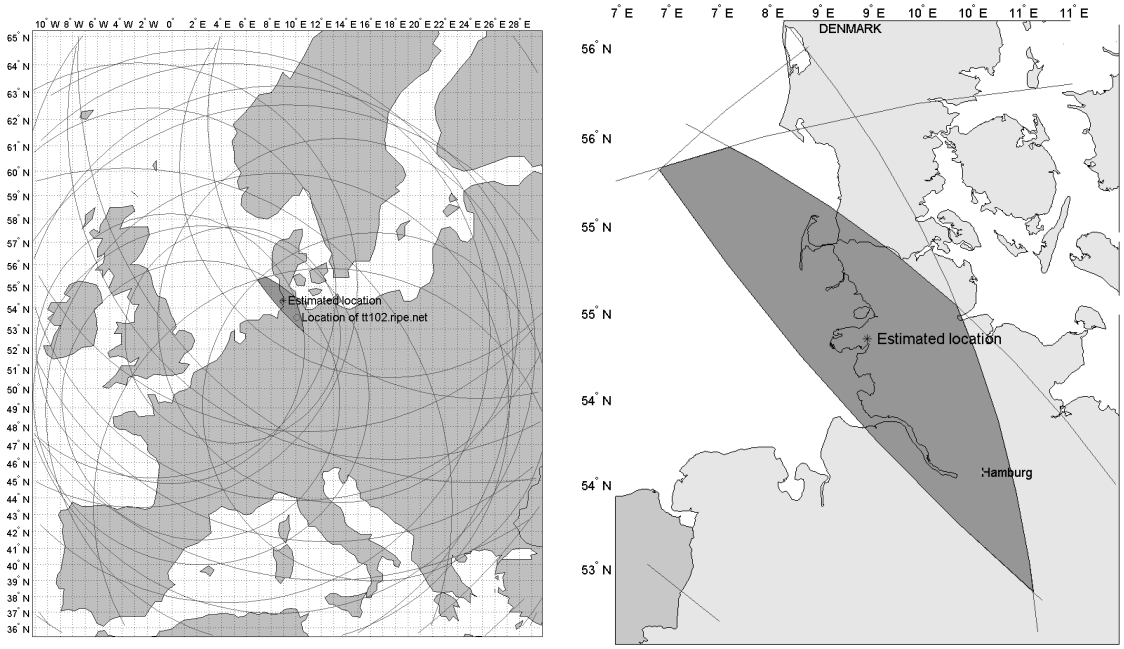
3.1.2 Location 2: tt102.ripe.net

The second test was run against the host "tt102.ripe.net". This test server is located in Hamburg, Germany. This results is less accurate, as the estimated position is 114.2 km from the actual position, but the confidence region is only 23010.0 km^2. Compared to the deviation between estimated and true positions in test 1, this position estimate is less accurate. This shows that the actual position does not need to be very accurate, but it will be located within the region $\mathcal{R}$. Indicating the importance of a reasonable sized uncertainty region, both for a forensics application and an advertisement application.

This time the region $\mathcal{R}$ also spans over two countries. This might make it difficult for an advertisement application to get usefull results, but for a forensic application where the user is provided with the visual projection it should not pose a major problem.

3.1.3 Location 3: tt109.ripe.net

The third test was run against the host "tt109.ripe.net". This test server is located in Umeå, Sweden. Figure 9 shows a visualization of the multilateration. Compared to previous results this geolocation is



(a) Overview

(b) Closer view

Figure 8: Geolocating "tt01.ripe.net" with known location in Hamburg, Germany.

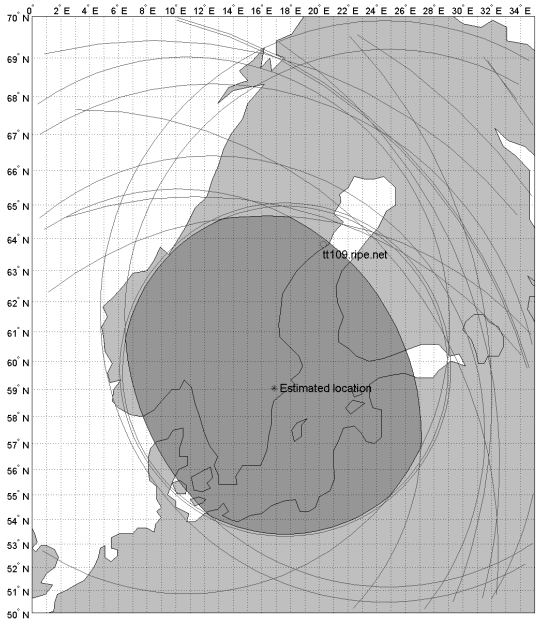


Figure 9: Geolocating RIPE TTM host "tt109.ripe.net", with known location in Umeå, Sweden.

not very good. With a confidence region of $\mathcal{R} = 1120300.0$ km^2 this result is inaccurate, and as shown in the figure 9 the estimated position is nowhere near the actual position of the target. Figure 5 may provide some answers to the bad target estimate. None of the landmarks used are geographically located north for the target being located, resulting in some very bad estimation of the location. The good thing is that many of the landmarks located to the south of the actual position have circle edges located very close. Providing one or two landmarks to the north of the text box server would drastically decrease the confidence region and improve the position estimate.

GeoLocate will have problems with geolocating at the edge of a geographical region. Targets located close to coasts or geographical regions may be harder to geolocate, making it necessary to use landmarks from several geographical regions.

3.1.4 Location 4: tt126.ripe.net

The fourth test was run against the host "tt126.ripe.net". This test server is located in Schwarzach, Austria. Figure 10(a) shows that the region $\mathcal{R}$ for this geolocation is much larger than for the first two tests. The confidence region $\mathcal{R} = 211440.0$ km^2 and the estimated location is 180.4 km from the actual position. This result could be considered not to good, and by looking at figure 10(b) it shows that the region $\mathcal{R}$ actually spans over six countries: France, Germany, Switzerland, Austria, Belgium and Luxembourg. For an advertisement application this confidence region is very large, and the results will probably be of little practical use. For a forensic application this result might still provide some valuable information.

The large confidence region of this geolocation may have several causes. None of the landmarks are located very close to the target. This will directly affect the results, but it will probably not be the only reason. Almost all landmarks are routed through the GÉANT research network and onto Austrias own research network called ACOnet. Most traffic coming into Austria is routed through Vienna[5], introducing a very large detour for some of the landmarks with best connectivity. As shown in figure 10(b) Vienna is located far away, actually it is located about 500 km from Schwarzach.

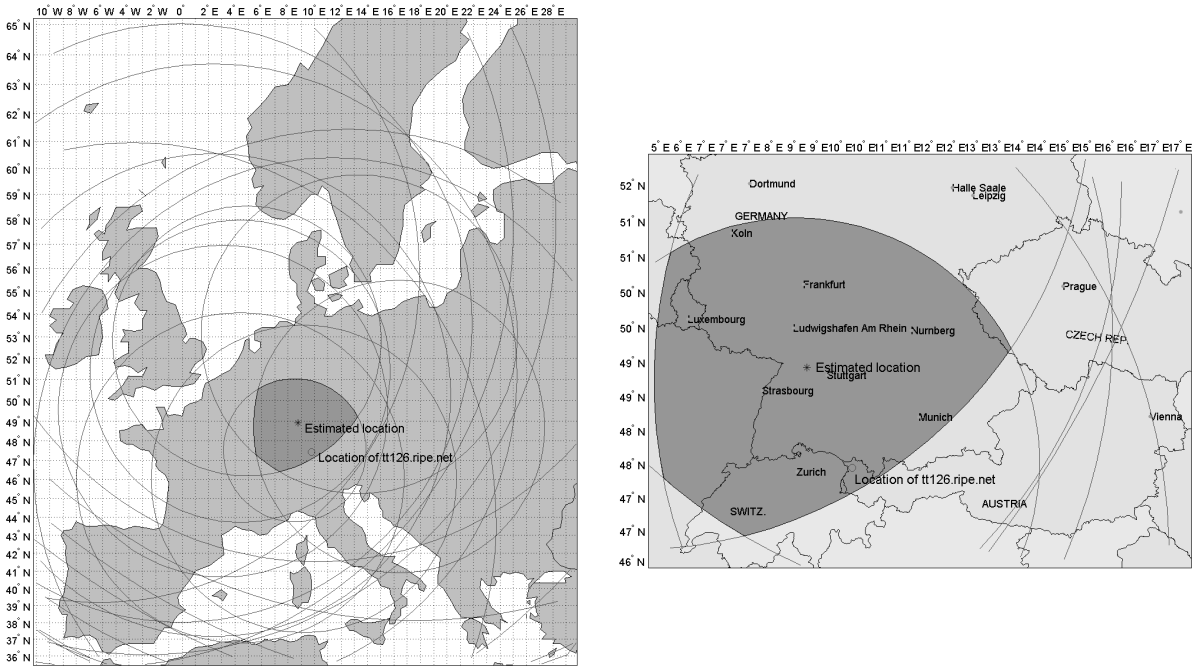
4 Discussion

As indicated by the results obtained, there are usually 3–5 landmarks that actually provide the regions where the targets are located. All other landmarks do not help with the geolocating directly, they only help to improve the distance constraint for the landmarks that actually find the region. To reduce the number of landmarks used, it is therefore necessary to find the landmarks most suited to be used, but this can not easily be done without conducting a measurement from all landmarks to the target.

Dr Ren Wei [6] proposed a framework called distributed agent-based network forensic system. This proposal uses a server and client based system, where all the agents collect different types of information, which is transported into a centralized forensic server and database. Such an architecture could also be used in GeoLocate, but it would introduce the same problems with acquiring landmarks as has previously been experienced using public information sources. The agents would need to be installed by users, preferably investigators or other personnel interested in using the application. The results would have to be available for everyone wanting to deploy agents. This will introduce a hierarchical trust relationship between the people deploying the agents and the ones keeping the server. Server operators having all access, and the agents only partial access.

Another solution would be to use a Peer-to-peer (P2P) based agent architecture instead. This removes the client/server scheme and introduces a "web of trust" hierarchy, as used by Pretty Good Privacy (PGP) [16]. Such a P2P based agent may solve problems with deploying the application in a wide scale, as all agents will have the same access. The performance of GeoLocate depends heavily on the number of Landmarks used and how the network delay measurement between the landmarks and target is done. If all landmarks perform measurements at the same time, GeoLocate will usually take 1–2 minutes to run, as some LG sites use a long time to respond. To reduce the amount of network traffic to the target simultaneously, GeoLocate landmarks can be configured to probe sequentially. This dicreases the performance, but reduces the chance of detection. The tradeoff between speed and stealth will have to be assessed by an investigator.

[5] ACOnet topology: `http://www.aco.net/aconetus.htm`



(a) Overview

(b) Closer view

Figure 10: Geolocating "tt01.ripe.net" with known location in Schwarzach, Austria.

5 Conclusions

The GeoLocate application shows that it is possible to use location-aware and content delivery based measurement methods for forensic investigations. GeoLocate also shows that public information sources can be used as landmarks to acquire real life results for network measurement methods. The results acquired by GeoLocate depends heavily on the number and "quality" of the landmarks used. It is therefore possible to achieve better results with GeoLocate than what has been presented here. However it is clear that further development is needed, and acquiring landmarks is one of the biggest challenges of such tools. We therefore propose to use an agent based P2P application, making it easier to acquire better landmarks and to provide better location estimates.

6 Acknowledgements

The authors would like to extend their gratitude to Professor Svein J. Knapskog at the Department of Telematics for his helpful feedback during this research. The authors would also like to thank the High Tech Crime Divistion at the Norwegian National Criminal Investigation Service for their involvement in the project.

References

[1] Espen A. Fossen. Principles of Internet Investigations: Basic Reconnaissance, Geopositioning and Public Information Sources, MSc-thesis, Department of Telematics Trondheim, Norway. June 2005. `http://www.junta.no/forensics/`

[2] Espen A. Fossen. Automatic tracing of Internet addresses. Minor thesis, Department of Telematics, Trondheim, Norway. November 2004. `http://www.junta.no/forensics/`

[3] Artur Ziviani, Serge Fdida, José Ferreira de Rezende, and Otto Carlos Muniz Bandeira Duarte. Toward a measurement-based geographic location service. In *Proc. of the Passive and Active Measurement Workshop - PAM'2004*, Lecture Notes in Computer Science (LNCS) 3015, pages 43–52, Antibes Juan-les-Pins, France, April 2004.

[4] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based geolocation of internet hosts. In *Proc. of ACM/SIGCOMM Internet Measurement Conference - IMC 2004*, pages 288–293, Taormina, Sicily, Italy, October 2004.

[5] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for Internet hosts. *Proceedings of SIGCOMM'2001*, page 13, 2001.

[6] Dr Ren Wei. A framework of distributed agent-based network forensics system. *Proceedings of DFRWS'2004*, August 2004.

[7] C. Davis, P. Vixie, T. Goowin, and I. Dickinson. A means for expressing location information in the domain name system. Internet Engineering Task Force: RFC 1876, January 1996.

[8] L. Daigle. Whois protocol specification. Internet Engineering Task Force: RFC 3912, September 2004.

[9] D. Moore, R. Periakaruppan, J. Donohoe, and K. Claffy. Where in the world is netgeo.caida.org? In Proc. of the INET'2000, Yokohama, Japan, July 2000.

[10] University of Illinois at Urbana-Champaign. IP Address to Latitude/Longitude. `http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll/`

[11] CAIDA. Reverse traceroute and looking glass servers in the world. `http://www.caida.org/analysis/routing/reversetrace`.

[12] `http://www.tracereoute.org`. Traceroute.org. Traceroute and looking glass resources.

[13] NLANR Active Measurement Project. `http://amp.nlanr.net/active/`.

[14] RIPE Test Traffic Measurements. `http://www.ripe.net/ttm/`.

[15] Cistron. Looking glass source code. `ftp://ftp.cistron.nl/pub/people/miquels/net/`.

[16] P. Zimmerman. *Pretty Good Privacy User's Guide, Volume I and II*, 14 june 1993 revised edition. Distributed with the PGP software.

[17] R. W. Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2):159, 1984.

[18] John E. Midwinter Optical Fibres for Transmission John Wiley & Sons, NY, 1979.