EUROPEAN COMMISSION

Community Research

Specific Targeted REsearch Project

PRISM

D3.1.2: Preliminary Data Protection Algorithms Specification and Analysis

Project acronym: PRISM
Project full title: Privacy-Aware secure Monitoring
Contract No.: 215350
Project Document Number: IST-2007-215350-WP3.1-D3.1.2-R1
Project Document Date: February, 15, 2009
Workpackage Contributing to the Project Document: WP3.1
Deliverable Type and Security: Public
Author(s): Simone Teofili, Matteo Pomposini, Giuseppe Bianchi, CNIT
Ivan Gojmerac, Esa Hyytiä, FTW
Brian Trammell, Elisa Boschi, HIT
G. Lioudakis, F. Gogoulos, A. Antonakopoulou, ICCS
D. Kaklamani, I. Venieris, ICCS
Alexander Morlang, Carsten Schmoll, FRAUNHOFER

Abstract:

Goal of this deliverable "Preliminary Data Protection Algorithms Specification and Analysis" is to introduce the data protection concepts and approaches emerging in the PRISM architecture, and provide their preliminary specification and analysis. More specifically, this deliverable describes innovative privacy preservation paradigms tailored to the two tiered nature of the PRISM architecture. This document also describes significant advances in the understanding of the effectiveness of anonymisation mechanisms envisioned for data export purposes, based on the novel idea of data privacy conditioned to the state of the monitoring process. In addition access control paradigms are presented, which are specifically tailored to the PRISM monitoring scenarios and the roles and monitoring purposes emerging therein. The specified approaches shall be used within the PRISM project in a combined manner in order to maximise the protective potential. In addition to the presentation of the individual data protection solutions, this document also describes how these approaches can be deployed over the emerging PRISM architecture as documented in the relevant deliverable D2.2.1.

Keyword list: PRISM, IST-2007-215350, data protection algorithms, anonymisation, per-flow differentiated encryption, state-dependent data protection, access control.

Table of Contents

Abbreviations

AAPI	Anonymisation Application Programming Interface
AES	Advanced Encryption Standard
AMD	Advanced Micro Devices, Inc.
AS	Autonomous System
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
EPAL	Enterprise Privacy Authorization Language
FPGA	Field-Programmable Gate Array
HP	Hewlett Packard
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information eXport
MAC	Media Access Control
MIB	Management Information Base
PC	Personal Computer
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PPC	Privacy-preserving Controller
PRISM	PRivacy-aware Secure Monitoring
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol
WP	Workpackage
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

List of Tables

List of Figures

1 Introduction

1.1 Goal of this Deliverable

This deliverable, "***Preliminary Data Protection Algorithms Specification and Analysis"*** is scheduled at month 11 of the project lifetime. It is produced in the frame of the work package WP3.1, data protection algorithms. Goal of this work-package is to design innovative data protection approaches suitable for traffic monitoring tasks, and assess their suitability for integration in the PRISM architecture emerging in the frame of WP2.2 (and preliminarily documented in D2.2.1). This deliverable reports the specification and analysis of the technical approaches proposed and/or considered during the first year of the project for protecting traffic data (and with it the users' privacy). Moreover, this deliverable briefly discusses how these approaches relate to the PRISM architecture and how they can be integrated with it.

1.2 Deliverable Organization

This deliverable is organized into 4 sections. After this introduction, section 2 discusses how data protection mechanisms are related to the PRISM underlying architectural principles, and, as a consequence, why they have been designed as described next into greater details. Section 3 represents the core contribution of this deliverable. It is organized into three sections each discussing a different data protection approach proposed for the PRISM system, plus a fourth section dedicated to reporting analysis results for existing tools. Section 3.1 presents a brand new paradigm for privacy preservation which naturally emerges in the two tiered PRISM architecture scenario, and which first envisions data protection as a dynamic time varying attribute which can be triggered by monitoring events occurring in the front-end tier. Section 3.2 discusses anonymisation techniques, their risk-utility trade-offs, and their work-in-progress-support within the upcoming standard extensions to the IETF-defined IP Flow Information eXport (IPFIX, c.f. [RFC5101]) protocol. Section 3.3 presents a semantic role/purpose based authorization framework devised to operate on top of the PRISM monitoring architecture composed of two functionally different front-end and back-end tiers (the front-end and the back-end). Section 3.4 provides an analysis of multiple anonymisation tools, devised to understand the issues and the assets underlying their possible integration within the PRISM system, along with a brief review of other tools and the reasons why they have not been ultimately considered for PRISM. Finally, conclusive remarks are provided in section 4.

2 Data Protection Approaches and their Relation with the System Architecture

This section motivates and introduces the data protection approaches envisioned in PRISM. Moreover, it details where and how these mechanisms are meant to be applied within the PRISM system, with respect to the questions (a) at what component in the PRISM system they can be applied and (b) what are the derived implications.

2.1 The PRISM Approach to Data Protection and its Rationale

The effectiveness of the monitoring tools in use today is achieved at the price of **collecting, inspecting and processing data generated by, or related to, the network users, thus infringing their right to privacy**. Anonymisation mechanisms, as indeed discussed later on in this deliverable (see sections 3.2 and 3.4) can alleviate this issue. However, it appears extremely hard (and some published work even says impossible) to find a specific, one-for-all anonymisation approach which can accomplish **both, privacy and utility**. Anonymisation approaches targeted to provide strict privacy guarantees may yield almost useless monitoring data; conversely, monitoring-friendly anonymisation approaches, i.e., devised to retain properties that make data amenable to monitoring processing, are considered to be weak from a security point of view, and easily reversible through statistical analysis methods or approaches based on trace correlation.

We believe that this issue is intrinsic in the traditional *„gather first, process later"* network monitoring paradigm, which envisions data protection as a **static, one-for-all, mechanism applied regardless of the specific intent behind the monitoring process.** Indeed, note that this traditional approach to network monitoring assumes that traffic probes devised to gather data traffic do not add "monitoring intelligence". Their main goal is to capture traffic, and even if further filtering and anonymisation mechanisms are supported, these techniques are statically pre-installed and as such not matched with the monitoring application that will use the captured data.

As extensively discussed in the architecture deliverable D2.2.1, PRISM radically changes this paradigm. In PRISM, the traffic probe device is called "front-end", and it is equipped (along the multiple processing units introduced in D2.2.1, section 4) with pre-installed and eventually hardware accelerated processing and filtering mechanisms which can be programmed to provide a tailored output to a specific monitoring application. This novel architectural vision has two important consequences:

1) The specific type of data delivered as output, and the specific protection mechanisms employed in a given processing unit can be tailored to the specific needs of the considered monitoring application. As such the privacy-versus-utility-trade-off is no more a global one, therefore hardly accomplished, but becomes a trade-off that **can be made specific to every different monitoring application**, which can hence be delivered only the absolutely necessary data, thus technically enforcing the proportionality principle behind privacy preservation. In this new vision, the choice of specific anonymisation mechanisms (sections 3.2 and 3.4) to be deployed can be done to maximize the utility of the specific monitoring application which will make use of such provided data.

2) The possibility to run processing functions over the front-end enables a brand new privacy protection paradigm, which, to the best of our knowledge, has been for the very first time proposed here in PRISM. The idea is to develop data protection mechanisms which **do depend on monitoring state information which can be determined through the operation of the front-end traffic analysis functions**. For instance, assume that the processing logic running on the front-end may discriminate flows whose traffic pattern show a suspicious behaviour, from "well-behaving" ones: In this case, the front-end may deliver all the traffic generated by the suspicious flows to the back-end for further analysis, while it may apply an extreme form of data protection to the well-behaving flows, namely not deliver at all any information about these flows to the back-end. Section 3.1 will discuss a different, and more elaborated, protection form based on this same principle.

In the above introduced PRISM approach, the notion of "which application" (obviously ran by some specific user, as such characterized by a "role") accesses the traffic data, and for "which purpose", becomes of paramount importance. For this reason, a fundamental function of the PRISM system is to enforce a comprehensive authorization framework devised to control not only the type of data which is accessed by a given application ran by a given user's role and for a given purpose, but also to control the specific operations that can be performed on this data and the output produced by such operations. The details concerning the PRISM authorization model are presented in section 3.3, and entail both the control of the authorization permissions for accessing front-end data and associated analysis functions, as well as the typically more elaborated processing of the data collected at the PRISM back-end.

In summary, the data protection and privacy preservation techniques used in PRISM serve multiple goals:

- Protection of the network users' data privacy, specifically matched to the need of the monitoring applications;
- Protection against tapping into the system by unauthorized personnel (be it wiretapping, access to databases, or access to analysis results data)
- Proportionality of collected data based on targeted use
- Efficiency of data export and analysis (a must due to high speed constraints)
- and, last but not least, effectiveness of the finally implemented analysis methods

Towards these goals PRISM devises several (orthogonal) approaches, whose rationale has been provided above. These approaches are briefly described in an overview in the following text and in detail in chapter three. In the final system these techniques shall be applied in conjunction.

2.2 Privacy Preservation conditioned to the Monitoring State

The first dedicated privacy preservation technique explored in PRISM and detailed in section 3.1 “Privacy Preservation through Escrow and Selective Decryption” is the possibility to condition the level of protection to the occurrence of monitoring events detected by traffic analysis functions (well described in D3.2.2) running over the front-end. As stated in [D2.2.1] it might seem sufficient to implement, over the front-end, a mechanism able to isolate the flows of interest for the monitoring applications and to send to the back-end only those isolated flows. However, in practice, the back-end monitoring application might need to analyze all the traffic generated by the suspicious flow, and not only that generated after the flow has been isolated. In other words, the monitoring application might need to go backward in time, but this is clearly impossible as all the traffic prior to the occurrence of the considered events has not been delivered, nor it may be buffered due to the severe memory constraints of the front-end. We propose a solution based on "differentiated" encryption. Each flow information is encrypted with a different key, and the resulting per-flow encrypted data is delivered to the back-end. Per-flow encryption requires a careful management of encryption keys, as flows may not be known beforehand: we have proposed a solution capable both of auto-generate encryption and decryption keys and to deliver those keys to the back-end when a specific flow needs to be analyzed.

With this approach the detection of significant events in the network can be performed directly in the frontend, and traffic flows which contribute to such events can be identified. In the interaction with a flow-based data encryption in the front-end this information can allow dedicated per-flow decryption of only those flows which are deemed an originator of the significant events (e.g. a denial-of-service attack), keeping all other flows encrypted (and un-decryptable for the backend), effectively keeping them privacy-protected.

This approach works well in conjunction with (a) dedicated manual inspection of the algorithms used to identify the significant events, plus (b) well-authorized access to PRISM via the backend, which is going to be explained in the following text.

2.3 PRISM Authorization Control Framework

Another vital building block for the PRISM data protection is limiting access to authorized users and to those datasets which a given user may access. While working with the PRISM system a system user executes a monitoring application in the context of serving a monitoring purpose (e.g., intrusion detection) and in that respect traffic-related data is requested. This access puts the user into a specific role, e.g. requestor for data results of measurement XYZ. The PRISM system now mediates between the data requesting entity (i.e., the user and the application used) and the actual data (network streams or/and stored traces). This process of regulating access to the data with the enforcement of several restrictions that have to do with the so-called “privacy context” is detailed in section 3.3.1 “Authorization and Access Control for the PRISM Frontend and Backend”.

2.4 Secure Data Transmission

Secure encrypted data transmission is an integrated part of the PRISM system. Secure data transmission can be implemented either on the network level, using IPSec, or on the transport level, by using Transport Layer Security (TLS). IPSec is usually deployed for whole networks (subnets), while TLS can be used between any pair of applications, independently of the logical network structure. Due to the distributed and possibly dynamic nature of external monitoring applications (there can be new instances per monitoring task) TLS is

recommended. In PRISM we can apply secure data transmission in three distinct places:

a) between the capture module in the frontend and the processing units of the frontend
b) between the frontend and the backend
c) between the backend and external monitoring applications

The following has to be noted on above data paths:

(a) can be considered optional, because the capture modules and the processing units work in (physically) close cooperation and will therefore also be secured in a physical way, e.g. are positioned in the same locked server rack.
(b) will be deployed within the PRISM system in order to safely transmit the captured and possibly pre-processed network data. A fast cipher suite like, e.g. 256-bit AES, shall be realised, since this data path may contain some high bit rate traffic.
(c) for this data path transport level data protection by encryption is recommended to save the information from malicious wire tapping attacks. Only in cases where the exported information is already heavily anonymised (c.f. 3.2.3 “Anonymised Data Export using IPFIX”) *and* performance limitations exist that would significantly hinder the data analysis processing, this recommendation could be relaxed.

2.5 Classical Anonymisation

Within PRISM we can also apply „classical“ data anonymisation techniques, such as one-way mapping of header-fields, removal of fields, (partial) remapping of fields, reduction of accuracy by aggregation and/or sampling of attributes as well as cryptographic hashing or direct encryption of data fields. These algorithms may be applied either on the front-end, when this is computationally possible, or can be applied in the backend before data storage to the database backend, when their need of processing power and additional stateful storage during the anonymisation process itself does not make them viable for front-end support.

The concrete mechanisms to do so have already been documented in the previous deliverable [D3.1.1] in chapter 3. Building on top of that deliverable we document in this follow-up deliverable in section 3.2 “Privacy Preservation through Anonymisation” a broader classification of these techniques based on the cardinality of the set of original and anonymised data sets and their (present or absent) recoverability, i.e. the capability of reversing the anonymising transformation at all, given that one knows all the parameters of the anonymisation process (e.g. the used encryption key). This section further evaluates the impacts of such techniques on the risks for anonymity and the utility of the resulting data sets.

3 Specification of Data Protection and Privacy Preservation Methods

This chapter presents the core contribution of this deliverable. It is organized into three sections each discussing a different data protection approach proposed for the PRISM system, plus a fourth section dedicated to reporting analysis results for existing tools. Section 3.1 presents a brand new paradigm for privacy preservation which naturally emerges in the two tiered PRISM architecture scenario. Section 3.2 discusses anonymisation techniques, and their risk-utility trade-offs. Section 3.3 presents a semantic role/purpose based authorization framework devised to operate on top of the PRISM monitoring architecture. Section 3.4 provides an analysis of anonymisation tools, devised to understand the issues and the assets underlying their possible integration within the PRISM system.

3.1 Privacy Preservation through Escrow and Selective Decryption

As already stated in the Deliverable [D2.2.1] the reduction of the amount of data that a monitoring application has to process in order to focus data processing on really meaningful information is a fundamental requirement in terms of performance and scalability issues. In addition privacy requirements strengthen the need to focus data collection and processing activities on a subset of the observable data, namely that which is strictly necessary to perform a specific monitoring task.

These goals have led to the two-stage architecture described in [D2.2.1]. The first stage, called front-end, is placed as close as possible to the source of the data to collect, filter, and pre-process only the data strictly necessary for performing a specific monitoring task. The second stage, called back-end, is devised to process the subset of pre-filtered data.

Unfortunately it is not always possible to completely eliminate traffic irrelevant for a given measurement task (by reasonable means) within the front-end through isolation, or to significantly reduce the information reported for each flow through extraction. In this case the information passed to the second stage requires additional protection to reduce the risks to privacy. A monitoring task usually needs to analyse all of the information collected for a given flow (or a given set of flows), but the isolation of the flow of interest for a monitoring application is technically possible only after the flow has been partially processed, or compared to information derived from other flows. A simple example of this situation is traffic thresholding based on flow volume percentiles. Here the operation requires access to some information (flow volume distribution) from every flow. The state required to support this operation as part of flow isolation at the front-end is prohibitive, especially on resource-constrained devices. To address this issue, the front-end may selectively or completely protect, through encryption, measured data on a per-flow basis, delivering this encrypted data to the second stage. According to the requirements of the specific measurement task, the second stage is then allowed to selectively decrypt *only the flows for which a detailed analysis is deemed necessary*. This selective encryption is to be enabled by using distinct encryption keys on a per-flow or per-class-of-flows basis, so that the second stage will be able to access only a subset of the data delivered by the front-end. A single encryption key for the whole traffic delivered to the second stage would yield an all-or-nothing approach, which is considered to be unacceptable

This approach requires some method for conveying keys to the second stage, either i) a key repository, that can be selectively accessed only if certain conditions emerge, can be added to the PRISM architecture, or ii) key material can be embedded within the data delivered to the

second stage, so that the key can be reconstructed only if certain conditions emerge. These approaches may be applied in a complementary fashion.

With concrete reference to a simplified anomaly detection scenario, we show how it is possible to realize a per-flow or per-class-of-flows basis selective encryption and a method to embed the decryption key of every flow within the data delivered to the second stage, so that the key can be reconstructed only if certain conditions emerge. Furthermore, an elementary mechanism to isolate in the front-end the flows that need to be decrypted in the back-end to allow further decryption is described in the following paragraphs in order to show how the protection mechanisms described hereafter are complementary. The description of advance and performing "on the fly" flow isolation mechanisms are subject of work packages WP3.2 and WP4.1.

3.1.1 Reference Anomaly Detection Scenario

In this section, we discuss and formalize the simplified anomaly detection approach considered hereafter as a reference monitoring application example. To avoid confusion, we remark that the terminology introduced in this section goes somewhat apart from traditional literature in this area as it necessarily refers to more restrictive concepts. For simplicity, we assume that a probe monitors aggregated traffic delivered over a single network link.

The concept of flow – We say that a packet belongs to a flow if it matches a classification rule that can be operated on a per-packet basis. This is a quite broad definition which formally states as follows: If p is a generic packet travelling over the link, there exists a function $f_i = f(p)$ which allows to compute from each packet p a bit-string f_i (flow label). We say that different packets belong to a same flow f_i if, for all these packets, the function f(p) yields the same flow label fi. We remark that this notion of a flow is quite general, and the related bit-string f_i has (in principle) no limitations in size and form. For instance, a flow can be identified as all the traffic generated by the same IP address (in this case f_i could be conveniently set to the specific IP address itself, IP_i), or as the traditional 5-tuple flow (in this case, f_i could be a string composed of the five fields IP source/destination, port source/destination, and protocol field in the IP header). In another example, we can identify "flow" as the large class of packets whose destination is outside the considered network domain, or all the packets whose size is greater or equal than 500 bytes (in these cases, f_i may be chosen as a suitable artificial identifier, say a number or text). The restrictive assumption considered in this section is that such classification is purely done on a per-packet basis, i.e. given a single packet p it is possible to classify it as belonging to a given flow only on the basis of the information contained in the packet itself, i.e., without looking at prior or subsequent packets[1].

The concept of feature - We also define a second function, $g_j = g(p)$, which is devised to extract a "feature" from the packet p. Again, g_j is an arbitrary bit-string (feature label) in a possibly large set. For instance, g_j can be the specific numeric value contained in the protocol field of the IP header (e.g., g_0=0001=ICMP; g_1=0006=TCP; etc). Also in this case we restrictively assume that the feature can be extracted by only inspecting a single packet, i.e. without analyzing multiple packets in the flow. For convenience of presentation we consider only the case of a single class of features, i.e. only a single function g(p). The generalization to multiple classes of features is, at least in principle, straightforward.

[1] For instance, according to our operative definition of flow as the output of a suitable function computed over single, independent, packets, two different TCP connections originated from the same IP address and reusing the same ephemeral source port (indeed a quite particular case) may not be distinguished (in the absence of supplementary application layer information suitable for classification purposes), as such distinction would require to keep track of SYN/SYNACK/ACK sequence of packets as indication that a new connection is being set-up.

The concept of critical feature - A subset of specific features in the set of all the possible values g_j are defined as "critical" if their occurrence is above a given frequency (as quantified below) which can be considered a symptom for an anomalous operation. For simplicity of presentation, in what follows we non-restrictively assume that our anomaly detection application is designed to look only for a single critical feature. For example, in the case of an application wishing to detect the occurrence of "frequent" ICMP packets (e.g. a ping flood attack), the critical feature will be g_0=0001=ICMP. Clearly, the anomaly detection application is expected to look for a specific critical feature g_0 in any existing flow f_i. For instance, if we define the flow f_i as classified only on the basis of the IP source address IP_i, then the application needs to extract from each packet p the flow-feature pair (f_i, g_j), check whether g_j=g_0 and account this occurrence to flow f_i=IP_i.

Anomaly detection application example and related definition of suspected anomaly: For sake of clarity, we now introduce a precisely defined anomaly detection application (not realistic, but *sufficiently representative*), which will be used to illustrate the proposed privacy-preserving approach. Such an application is described in the following text, with its adaptation to our proposed approach already in mind.

Assume that a flow f_i is classified on the basis of only the IP source address. Assume, non-restrictively, that the critical feature (i.e. the feature under monitoring to detect an anomalous operation) is the occurrence of ICMP packets, i.e., g_i being the IP protocol field, and the critical feature being g_0=0001. For implementation reasons that will be clarified in section 3.1.2.2, we further assume that the time is slotted into constant-sized time windows. Each window lasts W seconds, with W being a relatively small value (say in the order of a few seconds). During each time window W, the anomaly detection application is expected to count the occurrence of ICMP packets generated by every occurring IP source address, i.e. keep track of the occurrence of flow-feature pairs of type (f_i= IP_i, g_0=0001).

We now define a "suspected anomaly" condition for the considered application. This condition is clearly related to the occurrence of "frequent" generation of ICMP packets. Since it is essential for the system operation that such condition is precisely stated in quantitative terms, we assume that the considered application uses the following specific "suspected anomaly" condition: Generation, by a same IP address, of more than N ICMP packets in at least M not necessarily consecutive time windows over a total predefined "monitoring time frame". This is a possibly long time interval, e.g., in the order of hours.

Whenever such a suspected anomaly is detected, a more detailed processing is performed in the back-end on the packets generated by the flow f_i that caused the suspected anomaly o be detected, on order to check whether the suspected anomaly is for real and to react accordingly. In this deliverable we are not concerned about how this second level processing is actually performed, but we assume, in most generality, that this second phase processing requires to analyze (including inspection of supplementary header fields and packet payload) all the packets generated by the flow during the whole "monitoring time frame". For all the packets, we mean that the analysis is not restricted to ICMP packets (namely, those which triggered the suspected anomaly), but includes also any other packets (e.g., TCP and UDP) generated by the flow during the whole monitoring time frame[2].

[2] This highly invasive monitoring operation could be contrasted by arguing that this requirement is excessive for a traffic anomaly detection application (i.e. not following the "proportionality" principle set forth by data protection regulation). However this is not relevant here, especially as long as we will show that our framework is able to cope with such possibly excessive requirements meanwhile preserving the privacy of the legitimate flows

It is obvious that such a simple application poses an unsolvable dichotomy between users' privacy and monitoring application utility. In fact, a priori, any possible IP address may generate an anomaly, meaning that the application has no prior knowledge of which specific IP address to look for, but has to monitor on all the possible IP addresses. Hence, in order to just count the occurrence of ICMP packets, the application must be able to either analyze in clear all the IP addresses or rely on deterministic IP address anonymisation which, as discussed in the introduction, can be considered a weak form of protection in terms of privacy preservation, given the possibility to run a statistical analysis.

But by far the most critical privacy concern is a second one: If an anomaly is suspected, the above defined application requires analyzing header and payload of all the packets generated by the suspicious flow during the whole monitoring time frame. Since a suspected anomaly may emerge only later on in time (and hence a priori no flow can be considered safe), all packets travelling across the considered link must be collected and stored for possibly needed deeper analysis in the second phase processing in the back-end.

3.1.2 Proposed Approach

Our approach is designed to fulfil only the processing goals explicitly required to operate the anomaly detection application. We remark that, for the specific application described in section 3.1.1, even if overly restrictive with respect to "normal" data anonymization, the following requirements hold: i) To prevent from statistical analysis, data protection should be non-deterministic, meaning that there should not be a constant filtering key that allows the extraction of all packets belonging to a same flow; ii) Even if the back-end knows the label for a flow, it should not be possible for it to determine which packets in the trace belong to the considered flow; iii) The flow label should be disclosed only if a suspected anomaly emerges; iv) In this case, all packets belonging to the considered flow and generated during the whole monitoring time frame should be retrieved in clear.

3.1.2.1 Basic System Idea

Figure 1 sketches the basic idea described in [D2.2.1].

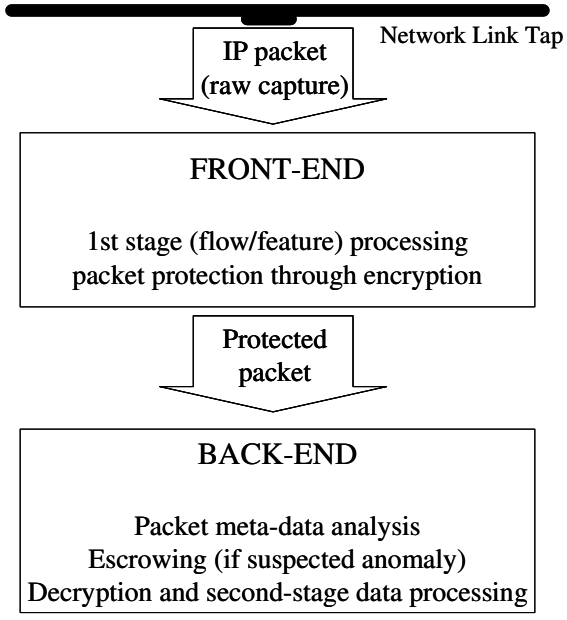

Figure 1 – System Architecture Components and Roles

Here the front-end system captures the traffic data delivered over the network link. It performs a goal-specific first stage processing of the raw (non-protected) data. Goal of this front-end processing is in this example to count the number of occurrences, in a time window W, of the same critical feature g0 for a flow fi. Of course this operation must be performed "on the fly", and on a possibly resource-constrained front-end system, with very limited storage resources. This implies the need to perform this functionality in a very efficient and memory saving manner. Several efficient approaches have been proposed in the literature [EST02] [RAM03] to perform counting operations over a large base of possible flows. (Enhanced) counting Bloom filters [BRO05] [BON06] [FIL08] are considered to be a reasonable solution despite the involved approximations in the counting process.

Each packet is then suitably encrypted (as described next), meta-information is added to each packet, and encrypted packets are delivered to the back-end. Since our goals are on one side the description of the protection algorithms and on the other side the provisioning of a clear example of the interaction of these algorithms with the flow isolation functionalities objectives of WP 4.1 and WP3.2, we do not consider here the whole set of functionalities offered by the back-end described in the [D2.2.1]. The back-end system in this example performs only the following operation: It stores all the encrypted packets in a database. Using the meta-data associated to the packet trace it can further i) detect if a suspected anomaly has occurred, and in such case, ii) extract from the meta-data associated with the packet trace information needed to decrypt the packets belonging to the specific flow for which the suspected anomaly has been detected. In this latter case, packet decryption occurs, but restricted to the packets belonging to the same flow that has caused the suspected anomaly. Note that packets belonging to other flows will remain encrypted and will not be distinguishable in terms of which flow they do belong to. This prevents further statistical traffic analysis on individual "legitimate" flows.

We remark that this operation achieves privacy protection in the back-end system, because the traffic delivered to the back-end is natively encrypted by the front-end before being delivered to the back-end and stored therein. This is a major difference with respect to ordinary approaches for storing data traces, as the encryption is typically performed by the back-end itself and only for secure storage purposes. Moreover, we note that the proposed approach guarantees *selective reversibility*, meaning that decryption is technically made possible if and only if a given flow triggered a suspected anomaly in the front-end.

3.1.2.2 Front-end Operation

The front-end is assumed to operate in real-time on raw (unprotected) captured data packets, meaning that no data packet storage facilities are provided, but a packet is first processed and then delivered to the front-end encryption module and in turn to the backend. Some limited storage space for keeping inter-packet state might be available at the front-end.

We decided to adopt Counting Bloom filters in order to implement the flow isolation functionality in the front-end. In particular they are used to keep track of the number of times a critical feature for a given flow repeats during a time window W. The counting process is reset at each start of a new time window. This implies that the front-end treats each window W as independent, and does not maintain any information spanning multiple windows. The size of the window W is by design "small". This is mainly motivated by practical efficiency reasons. In fact this allows a compact design of the counting structure and reduced risk of counting overflow.

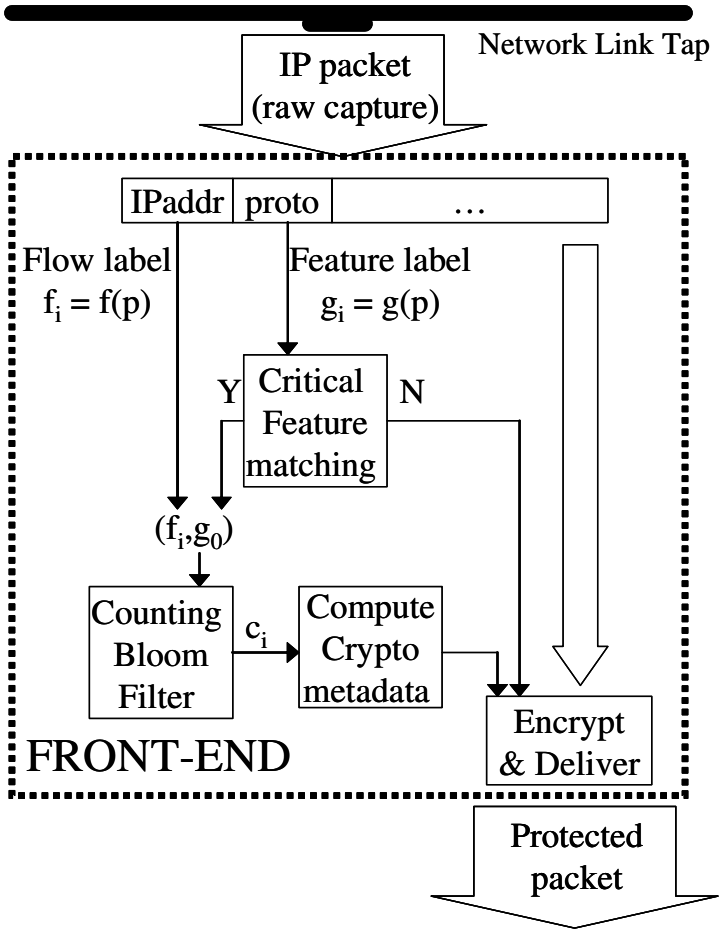

Figure 2 – Front-end Processing/Encryption Flow

As illustrated in Figure 2, the front-end implements the following functionalities:

- flow classification and critical feature(s) detection
- counting (if critical feature detected)
- eventual computation of cryptographic meta-information (depending on the output of the counting filter)
- packet encryption and delivery to the back-end

The pre-filtering performed by the critical feature matching module is not strictly necessary, but it is introduced to avoid that packets with feature which do not require to be controlled by the application are nevertheless passed to the counting bloom filter. As a side advantage, this relieves the counting bloom filter and hence allows obtaining a better performance in terms of memory consumption at the same target false-positive rate.

For the reasons discussed in the next section, after the counting, the cryptographic processing of the packet includes two steps: Preparation of cryptographic material that will be used by the back-end to de-anonymise (decrypt) the data under the occurrence of a suspected anomaly, and the actual data protection carried out through ordinary (randomized) encryption.

3.1.3 Data Protection and Basic Escrow Approach

The key idea of our approach is described in this section while extensions will be addressed in section 3.1.4.

3.1.3.1 The Basic Escrow Mechanism

Let us assume that packets are encrypted with a symmetric approach, and let K_i be the key used to encrypt packets belonging to flow i. We require that decryption of packets of the same flow must be made possible if and only if a "suspected anomaly" is detected by the active algorithms in the front-end.

We recall from the example presented in section 3.1.1 the fact that we assume the following specific condition to occur for triggering a "suspected anomaly" event: More than N ICMP packets belonging to a flow have been received in a time window W, and this must occur for at least M not necessarily consecutive time windows inside the monitoring time frame.

This is accomplished in our proposed approach as follows: We recall that the Counting Bloom Filter (CBF) is reset at each start of a new time window W. Therefore, the CBF is able to monitor only the occurrence of the first condition that characterizes a suspected anomaly, namely the occurrence of more than N ICMP packets in the considered window. If we accept the approximation induced by the counting process implementation as a CBF, this is trivially accomplished by checking if the output c_i of the CBF overflows the threshold N. The index i associated to the counter c_i stresses that the CBF simultaneously counts the occurrence of a critical feature for all the possible flows.

We now check, on a per-packet basis, whether the condition $c_i > N$ holds. If this is the case, we generate escrow meta-data information which is associated to the packet and delivered with it to the back-end (Figure 3). Now, the specific suspected anomaly condition set by the application translates in the fact that the encryption key K_i should be disclosed whenever the overflow of the threshold N occurs at least M times.

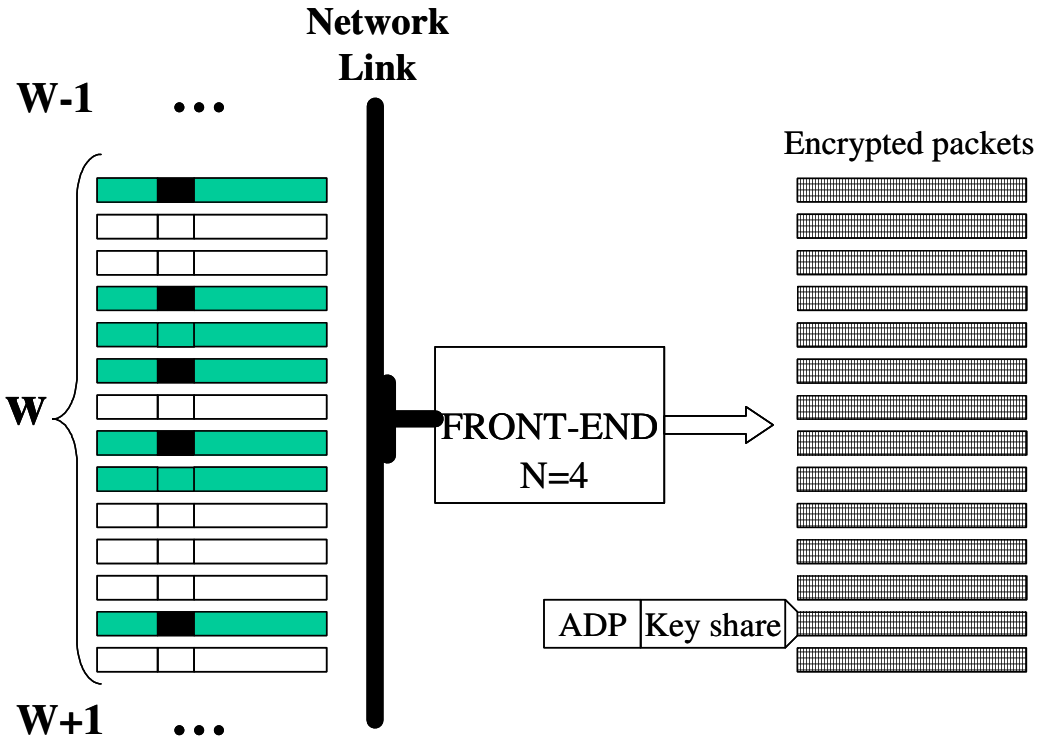

Figure 3 – Escrow material: Example with N=4.[3]

This can be interpreted as a classical secret sharing problem (i.e. a threshold cryptographic scheme), where a secret, in our specific case the per-flow encryption key K_i, should be revealed only if at least M suitably constructed "shares" of the secret are known. The basic original approach proposed by Adi Shamir [SHA79] can be used as starting point for our scope. We recall that Shamir's secret sharing approach consists in defining a polynomial

$$P(x) = a_{M-1}x^{M-1} + a_{M-2}x^{M-2} + \cdots + a_1 x + K_i$$

whose constant addendum K_i is the secret to be shared, i.e. the per flow decryption key, and whose coefficients $a_1, \ldots, a_{M-1}$ are initially known only by the frond-end system (i.e. possibly random values). Since the Shamir's algorithm employs modular arithmetic, the coefficients of the polynomial belong to a finite field of p elements, with p prime. The value of the prime number p has to be set by construction to be larger than both the coefficients of the polynomial and of the value K_i. Note that any value $P(x_i)$ obtained as the result of the evaluation of the polynomial for an arbitrary value x_i also belongs to the same finite field, and as such its size is bounded (this is an useful property in terms of reduced overhead) Now, once M pairs

$$(x_1, P(x_1)), (x_2, P(x_2)), \cdots, (x_M, P(x_M))$$

are disclosed (with x_i being random numbers different from 0), it is trivial to compute the term K_i through polynomial interpolation.

In order to apply the above secret sharing mechanism to our approach we need to address the following two problems. First, we need to construct the polynomial coefficients so that every flow is characterised by a different polynomial, but these coefficients are unpredictable and constant across different time windows W. Secondly, we need to provide the back-end with a

[3] In the example, the 13th packet in a same window consists in the repetition of a critical feature above the threshold N=4. To this packet, meta-information (ADP and key share) is added and exported.

way to distinguish key shares belonging to different flows, without revealing either the specific flow considered or any supplementary information. The first problem is solved by setting:

$$a_j = PRF\left(S^*, T_i, j\right) \quad j \in [1, M-1]$$

Here, S^* is a secret known only by the front-end, and constant for the whole duration of the monitoring time frame (i.e. constant across multiple time windows W). $T_i = (f_i, g_0)$ is a bit-string univocally associated to the flow f_i being considered and the specific feature g_0 target of the monitoring. Finally, j is the coefficient index. These three values are used as input of a robust pseudo-random function [GOL86], from whose output a pseudo-random number lower than the finite field size p is determined – approaches for constructing PRFs based on state of the art hash functions can be found in [KEL07]. Note that the secret S^* is needed otherwise the backend could construct the polynomial coefficients once it estimates which specific flow-feature pair $T_i = (f_i, g_0)$ is under analysis (and hence circumvent the protection mechanism set forth for the key K_i). Note also that the term $T_i = (f_i, g_0)$ is inserted to guarantee that different flow-feature patterns under analysis are provided with different polynomials.

The second problem is more subtle, and is best explained through the following example. Suppose that M=2, i.e. that a key is reconstructed once two key shares are collected for a same flow. Now, suppose that during time window W=1 two key shares are produced, one for flow 1 (share A) and one for flow 2 (share B). Then at time window W=2 another key share is produced for flow 2. Since the back-end has no way to distinguish to which flow a key share belongs to, it would have no other possibility to understand that this new share has to be combined with share B to reconstruct K_2 other than the brute-force method of trying all possible combinations (which grow exponentially with M and with an increased number of flow-feature pairs generating the key shares). An easy way to solve this problem is to add, to the key share, an "anomaly detection preamble" (ADP) which repeats whenever a share belongs to a given flow-feature pair. This preamble can be for instance constructed as

$$ADP_i = H\left(S^{**}, T_i\right)$$

where H is a hash function and S^{**} is a secret known only by the front-end (for improved robustness we assume that S^{**} differs from the previously used secret S^* - this assumption not being necessary for the system functional operation).

To summarise, the backend operates on the data delivered by the front-end as follows:

- Every packet received by the back-end and having no extra associated information is stored in the back-end database or otherwise directly sent directly to the associated analysis application.
- The packets for which an overflow of the anomaly threshold N has occurred in the counting mechanisms performed by the front-end CBF will be transmitted along with meta-data organized in two fields: i) the anomaly detection preamble ADP_i, and ii) the key share pair $(x_i, P(x_i))$.
- By looking at ADP repetitions, the back-end is able to collect multiple shares for the same flow.
- When the number of shares received for a given flow (namely, the same ADP) reaches M, the backend will be able to extract the decryption key K_i.

3.1.3.2 Data Packet Encryption

The protection of packets delivered from the front-end to the back-end occurs through standard symmetric encryption algorithms. To partially prevent from statistical traffic analysis, the only functional requirement posed to the specific encryption algorithm is that the same repeating packet, once encrypted, should yield different cipher text (namely, random encryption algorithms).

Two supplementary issues must be discussed. We recall that the symmetric encryption key K_i is, by construction, different for each flow. Hence, a first problem is how to generate the per-flow key K_i in order to avoid that the front-end is forced to keep an explicit mapping table between a flow f_i and its corresponding encryption key K_i. This is trivially accomplished by generating the key K_i through a pseudo-random function which has, among its inputs, the flow label f_i. For instance, $K_i=PRF(S^+,f_i)$ where S^+ is, as in the above discussed similar cases, a secret known only by the front-end and constant throughout the whole monitoring time frame.

The second problem is more subtle. We recall that the back-end is (by voluntary construction) unable to determine not only which specific flow a received packet belongs to, but it also cannot discriminate whether two or more packets belong to a same flow or not. As a consequence, all the encrypted packets are stored in a same trace. Unfortunately, packets are encrypted with different keys, depending on the flow they do belong to.

Therefore, once a decryption key has been reconstructed, it is necessary to implement a mechanism that allows distinguishing the packets that are properly decrypted, namely those which belong to the flow under decryption, from the ones which do not belong to the flow and hence who's "decryption" would provide a "random" result. One possibility could be the implementation of policies that allow to distinguish valid packets from the analysis of the "decrypted" text, for instance by reclassifying it as belonging to a given flow and understanding whether this flow pattern repeats (as it is the case for properly decrypted packets) or it appears as randomly generated. However, this would require a non trivial definition of policies (we remark that the flow label is not known before decryption and thus cannot be used as a filtering pattern). For instance, a randomly generated 4 bytes pattern is in most cases a valid IP address! We conclude that operating along this way would limit the generality of the proposed mechanism and add unnecessary complexity.

For this purpose, we propose, for the time being (an improved and fundamentally different approach is discussed in the next section 3.1.4) a different mechanism, which bases its operation on the explicit verification of whether a "decrypted" pattern is a valid one or not. For this purpose, we can encrypt, in addition to the packet, also a supplementary field devised to check the integrity of the decrypted information. Specifically, being P the packet, prior to perform encryption, the front-end adds to the packet a supplementary encryption integrity check field $F(P)$, and encrypts the resulting string $(P, F(P))$. F can be an hash function with relatively small digest (using standard CRC fields for this purpose may not be appropriate for some encryption mechanisms, see e.g. a discussion concerning CRC issues emerging with RC4 [BOR01]). The above mentioned mechanisms provide protection also from the occurrence of an accidental collision of the above introduced Anomaly Detection Preambles belonging to packets of different flows (an event which may be considered if, for overhead reduction, small-digest hash functions are considered for the generation of ADPs).

3.1.4 Improving Decryption Efficiency

One significant drawback of the decryption approach presented in section 3.1.3.2 is that it is only possible to assess whether a packet belongs to a given flow only after its full decryption, i.e. by verifying the integrity of the resulting decrypted text. Since this has to be performed over the whole packet trace collected in the database (i.e. for all packets belonging to all possible flows), this approach is practically not feasible.

It would be much preferable to operate in the opposite way, and specifically first assess whether a packet belongs to a given flow, and only later on perform the actual decryption. This can be accomplished by devising approaches inspired to the seminal work from Song, Wagner and Perrig [SON00], dealing with efficient searches on encrypted data.

The idea is to generate ADPs not only for packets whose count has exceeded the threshold N, but for all delivered packets (this increased overhead is compensated by the fact that the encryption integrity check field introduced in section 3.1.3.2 is not necessary anymore). The ADPs should be constructed in a way that permits the backend to determine whether the associated packet belongs to a given flow only if the relevant decryption key has been escrowed. Otherwise, the backend should not be able to gather any information from such ADPs.

This problem is highly related to that considered, and effectively solved, in [SON00], and very similar solutions can be employed also in our case. Assume that, upon reception of an arbitrary k-th packet, the front-end generates (computes) a random number R_k. As an exception of this rule, we assume that the value R_k is not random, but deterministically computed for all packets of a given flow carrying a critical feature and which causes the overflow of the threshold N at the counting Bloom filter in the front-end (e.g. these can be computed according to the same approach described in section 3.1.3.1, namely $R_k=H(S^{**},T_i)$).

Now, we generate the ADP value for the generic k-th packet as:

$$ADP_k = R_k, F_{H_i}(R_k)$$

where F is a keyed secure pseudorandom function as defined in [SON00] and the key H_i is unique for packets of a same flow. As such, the back-end will not be able to extract any information without knowing H_i used as key for the pseudorandom function F. Conversely, by knowing H_i, it is immediate to locate (in linear time, by verifying that the second part of the ADP, namely $F_{Hi}(R_k)$ matches the first part of the ADP, namely R_k) the ADPs belonging to the i-th flow. Now, it is sufficient to use, as function key H_i, the actual decryption key K_i (or, for improved security, a value which can be deterministically derived from K_i, e.g. $H(K_i)$). Note that the escrow mechanism described in section 3.1.3.1 remains valid (ADPs generated in different windows as a result of the threshold N overflow will be equal). Hence, once the backend escrows the K_i value, will be immediately able to collect all the packets of the flow, before decrypting them.

We conclude by noting that the described improvement is still limited to linear search time. An open research challenge is how to suitably construct ADPs to perform such search in sub-linear (logarithmic) time, i.e. comparable with search approaches over clear clear text data, while still relying on non-deterministic (and hence non traceable) ADPs.

3.1.5 Conclusions and Current Research Issues

Although technically founded on existing and well assessed fast data processing algorithms (e.g. counting Bloom filters) as well as cryptographic tools (e.g. secret sharing approaches and search techniques over encrypted data), and albeit preliminary (security and performance assessment is ongoing work as well as algorithms' optimization) our approach is the first, to the best of our knowledge, to suggest a new direction in privacy-preserving network monitoring.

Specifically, we suggest an alternative to the traditional anonymisation approaches where data is first anonymised and then processed by a monitoring application. Rather, we suggest re-thinking traffic monitoring approaches and anonymisation mechanisms as a tightly integrated joint process. Our architectural vision is that of a system composed of two entities: A front-end dedicated to a preliminary (and fast) processing of raw (unencrypted) data, and a back-end devised to provide a more in-depth analysis of the delivered encrypted traffic.

The privacy-utility trade-off is circumvented by using data protection mechanisms in the front-end which allow decryption by the back-end only under specific conditions. In details, we propose to include escrow mechanisms in the anonymisation (data protection) mechanism performed at the front-end, and tightly bind the capability of the back-end to de-anonymise data, to the occurrence of specifically formalized conditions enforced and controlled at the front-end. We remark that our overall position strongly favours privacy (but without penalizing monitoring application's utility). Indeed, our approach allows to design systems where the monitoring application is fed only with the minimal (but adaptive and extensible when really needed, thanks to the escrow mechanisms implemented) amount of information strictly necessary to fulfil the monitoring application's purpose. For instance, no flow information may be provided, and data can be protected to even prevent most forms of (illegitimate or out of the application's purpose) statistical analyses.

We further remark that, though the front-end operates in real-time, the two-party-based data protection mechanism described can be deployed also over systems where the data collector does not operate in real-time. This is for instance representative of the scenario where traces are collected and stored by an operator (hence with less critical privacy concerns), but the analysis of such traces is delegated to external entities. In such a scenario, most of the restrictions considered concerning the front-end processing may be overcome, and more advanced solutions are possible.

Furthermore, it could be argued that, under some circumstances (e.g. because of data retention purposes, or for investigations), access to stored traces should be possible beyond the limits set forth by the anonymisation mechanisms – this might be an additional requirement, one that our approach does not prevent to satisfy. In fact we derived all flow-related keys from a same stable secret (S+, in practice deployed as a sequence of refreshed secrets on a long term basis). Suitably managed access to these properly stored and protected long-term secrets allows decrypting collected data, irrespective of our data protection approach and of the fact that flows are or not suspected of anomalies.

Further technical extensions (object of current research work) appear possible. Escrow mechanisms, for example, can be generalized to permit different levels of de-anonymisation depending on the occurrence of different "suspected anomaly" conditions. While we have tackled an "ON-OFF" case in which the data traffic is either protected or reverted into clear form, a hierarchy of protection levels can be deployed (by using a hierarchy of escrow mechanisms), thus matching more carefully privacy-utility trade-offs which can emerge in real world traffic monitoring applications.

Moreover, to allow flexible exploitation of the proposed approach, it seems important to specify an application programming interface for the front-end which allows to jointly tuning both the front-end data processing mechanisms as well as the front-end cryptographic (escrow) approaches, in order to deploy sophisticated anonymisation mechanisms tightly integrated with the monitoring application operation.

Among the possible extensions for the proposed framework an extension to the proposed escrow mechanisms is presented in the next Section.

3.1.5.1 Stochastic Escrow Mechanism

In general, one can identify two extreme cases of anomaly detection: i) short time-scale phenomena, and ii) long time-scale phenomena. For the former, the window size W is small and also the number of shares M, i.e., the counting threshold, is typically a relatively small number. For the latter case, however, either the window size W is large or the number of events (corresponding to suspected anomaly) that should occur before key is revealed is large, or both. Suppose that the desired counting threshold M is large. Firstly, this implies a considerable transmission overhead as a large number of shares will be transmitted from front-end to back-end during the key escrow process. Moreover, a high-degree-polynomial P(x) means a high computational burden for both, the front-end computing the shares and back-end storing the shares and reconstructing the polynomial.

This problem can be mitigated, e.g., by applying a thinning procedure at the front-end as follows: First we define a thinning probability q. Then, instead of publishing a random share every time an event corresponding to a suspected anomaly is observed, one first throws a coin and publishes a share only with probability of q. This is illustrated in Figure 4. As a consequence, a substantially lower threshold M can be used for "counting" to M', where M' denotes the targeted number of events and M is still the degree of the polynomial. Note that the coin-tossing is a stateless operation (an i.i.d. random variable), but it does introduce some randomness in the process. However, when M' and M are large, then the effect from coin tossing is surprisingly small. Moreover, for example when the disclosure criteria M' is 1000 events, it is probably sufficient if the key is disclosed after 1000 ± 100 events instead, i.e., in practice the criteria for disclosure is rarely some hard exact number but instead some fuzziness can be tolerated.

It turns out that the stochastic version of the share disclosure process constitutes a Markov chain that can be analysed by using the standard techniques. This allows one to assess that the chosen parameters work appropriately and the design satisfies the given requirements. In particular, both the communication and computational costs can be minimised in controlled manner. Moreover, e.g., for distributed operation with a large number of front-end units, it may make sense to allocate different co-operating front-ends with disjoint sets of shares. If the number of shares in a pool is small, there is a danger that a randomly chosen point, xi, has already been published. The Markov model allows one to choose the operating parameters appropriately also in this case.

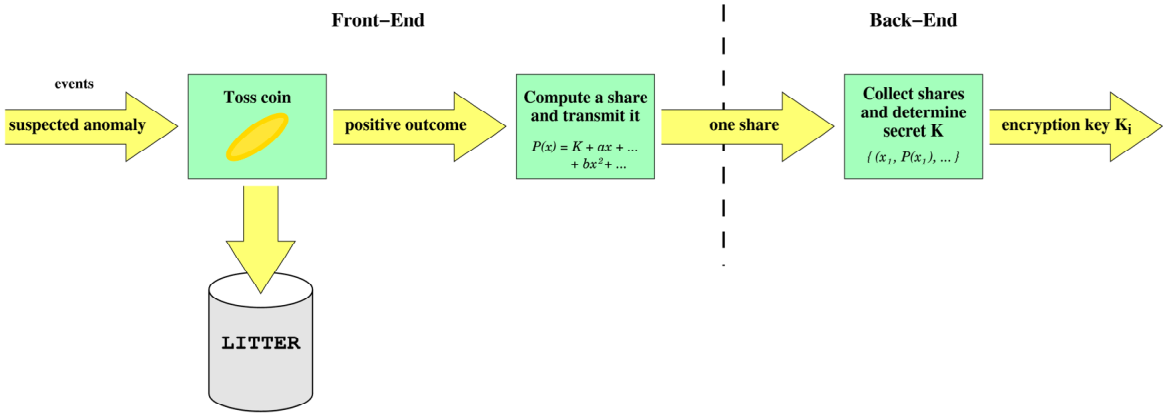

Figure 4 – Stochastic Version of the Key Escrow Process.

3.2 Privacy Preservation through Anonymisation

It is important to note that while anonymisation often entails the application of certain functions and techniques designed solely to obscure real identifiable entities in a collection of network traffic data, any analytic function which serves to reduce the amount of data (such as filtering, sampling, or aggregation) may have an anonymising effect on the data.

3.2.1 Classification of Anonymisation Techniques

Every anonymisation technique conceptually maps the *real* space of identifiers or values into a separate, *anonymised* space, according to some function. These functions may be broadly classified according to two properties of the relationship between the real and anonymised data spaces: *recoverability* and *countability*. This classification is intended to represent anonymisation techniques not by the operations performed, but by the properties of the anonymised data.

An anonymisation technique is said to be recoverable when the anonymising function is invertible or can otherwise be reversed, allowing a real to be recovered from a given anonymised identifier. Note that recoverable techniques need be theoretically recoverable only; an example would be data encryption with a symmetric encryption algorithm where the key is not made available after the encryption operation.

Countability compares the dimension of the anonymised space ***n*** to the dimension of the real space ***m***, and denotes how the count of unique value is preserved by the anonymisation function. If $m>n$, such that the anonymised space is smaller than the real space, then the function is said to *generalise* input, mapping more than one real value to each anonymous value (e.g., as with aggregation). By definition, generalisation is not recoverable. If $m=n$, such that the count of unique values is preserved across the anonymisation function, then the function is said to be a *direct substitution* function. If the $m<n$, such that each real value maps to a set of anonymised values, then the function is said to be a *set substitution* function. Note that with set substitution functions, the sets of values are not necessarily disjoint. Either direct or set substitution functions are said to be *one-way* if there exists no method for recovering the real data point from an anonymised one.

This classification is summarised in the table 1 below. Note that this classification, and indeed most current work in anonymisation techniques, considers each field of the data to be anonymised independently.

	Recoverable	Not Recoverable
m>n		Generalization (e.g. aggregation)
m=n	Direct substitution	One-way direct substitution
m<n	Set substitution	One-way set substitution

Table 1: Classification of Anonymisation Techniques with Respect to Recoverability

Within this classification, removal (or “black-marker” anonymisation) and aggregation both remove some information in a non-recoverable way that does not maintain unique counts.

Certain IP address anonymisation techniques also generalise, e.g. by masking the host portion and reporting only the network address. IP address anonymisation in general, however, is accomplished through direct substitution techniques, because for many analyses running on anonymised data, while the identities of each host involved are not important, the ability to distinguish hosts from each other is. Prefix-preserving anonymisation techniques are a form of direct substitution subject to the additional restriction that relationships among host addresses are preserved across the anonymisation function.

Timestamps are commonly processed both by generalization, which reduces of precision of timestamps to prevent sequence fingerprinting and information leakage attacks, and direct substitution, which adds a constant random offset to the timestamps to foil traffic injection fingerprinting attacks. Time series aggregation techniques which replace packets or flows with "binned" time series also have a generalising effect on timestamps.

Cryptographic approaches to reversible anonymisation are generally of the set substitution variety, because each value in the original data has multiple possible values in the anonymised data.

3.2.2 Evaluating Risk and Utility Impacts of Anonymisation Techniques

Network data anonymisation techniques are, of course, designed and intended to protect privacy-sensitive information and end-user identities. Much attention has been paid recently to breaking this protection, and recovering real information from the anonymised data. Attacks against anonymisation tend to use fields other than the anonymised ones, and to exploit inherent structure in network traffic data to infer information about anonymised entities. For example, [Rib08] describes a method for reversing prefix-preserving anonymisation by fingerprinting known hosts and exploiting the structure of the prefix tree; [Kou06] uses characteristic object sizes to identify Web servers; and [Cou07] constructs behavioural profiles to identify Internet hosts and thereby end-users.

The effectiveness of anonymisation on a data set can be assessed in terms of the *disclosure risk*, or the probability of the successful reversal of an anonymised data set to identify anonymised entities. Note that since attacks generally exploit the structure of the entire data set, calculating disclosure risk requires an accounting not only of the strength of the anonymisation used, but also the other data available within the set and the relationships within. Single-field disclosure risk can be approximated by assuming worst-case attacker knowledge and applying per-field estimated metrics. For example, IP address disclosure risk can be calculated using conditional entropy as in [Bez07] and [Bur08][4], which measures the number of binary questions that must be answered correctly to determine the real address given the anonymised address.

Disclosure risk is, however, not a sufficient metric for determining the effectiveness of an anonymisation operation. Consider the case of not exporting any data at all; from a disclosure risk standpoint this is completely effective, as an attacker cannot infer knowledge from data she has no access to; however, this "anonymisation" is plainly useless. The impact of an anonymisation operation on the *utility* of the anonymised data for a give analysis task must also be taken into account. The aim of designing and configuring an anonymisation technique to provide a given data set for a given purpose can then be formalized to an optimization

[4] "The Risk-Utility Tradeoff for IP Address Truncation" is a PRISM project dissemination product; one project partner is a co-author. PRISM project partners intend to continue work in the generalization and wider application of the techniques described therein.

problem, maximizing utility while minimizing risk, constraining risk lower than a given maximum acceptable threshold, and constraining utility higher than a given minimum.

Determining risk-utility trade-off for anonymisation is very much an open area of research, and must be separately evaluated for each data set and stated purpose. Due to the dependence of both the risk and utility functions on the data set as a whole, useful metrics cannot be presented in a generalized way on a per-anonymisation-technique basis. [Bur08] presents an initial foray into formalizing this trade-off for a simplified anonymisation technique (IP address truncation, a simple generalization function replacing the low *n* bits of an IP address with zero) and a simple analysis task (Kalman-filter-based traffic count anomaly detection), and shows that the risk-utility trade-off is useful in making choices with respect not only to choosing anonymisation technique parameters (in this case, the truncation length) but also to choosing a front-end data reduction and analysis technique. We expect these tools to be useful in separating front-end and back-end analysis functions within in the PRISM system.

3.2.3 Anonymised Data Export using IPFIX

IPFIX [RFC5105], as introduced in Section 7 of [D2.2.1], is a flexible, standardized flow export protocol selected by the PRISM project for export between the Anonymisation Support for IPFIX [IPFIX-ANON][5] outlines a set of common techniques for the anonymisation of single Information Elements within an IPFIX Flow Record, classified according to the scheme outlined in section 3.2.5 above and by the type of Information Element to be anonymised: IP Addresses, transport port numbers, timestamps, counters, and so on. It also adds Options-based anonymisation metadata export support to IPFIX, and presents general guidelines for the implementation of anonymised data export using IPFIX. Within the context of the PRISM project, anonymised data export using IPFIX is intended primarily for the export of data from the backend to an external monitoring application[6] as opposed to providing protection between back-end and front-end components of the PRISM architecture, as the intra-PRISM communication may require additional metadata (e.g., escrowed or partial decryption keys) not suitable for true anonymised export.

3.3 Data Protection through Authorization and Access Control

3.3.1 Authorization and Access Control for the PRISM Frontend and Backend

Authorisation and access control constitute very important aspects of PRISM, since on the one hand the fundamental question that the PRISM framework has to answer during its operation is whether access to data should be granted or not, while on the other hand this operation requires the collaboration between distinct social and computation entities, thus creating authorisation chains. Therefore, the problem of how to best specify and implement authorisation and access control in such an environment constitutes a challenging research topic for PRISM. This section presents the preliminary results of this work-in-progress.

[5] Note that this individual Internet-Draft is a work in progress, and is in part a dissemination product of the PRISM project. It is our plan that continued project work on anonymised data export, including the anonymisation methods supported by the tools examined within this deliverable, will be standardised via this Internet-Draft during the lifecycle of the PRISM project.
[6] see section 7.3 of [D2.2.1].

From an authorisation and access control point of view, the PRISM operation can be roughly summarized as follows: an actor (i.e., a system user) executes a monitoring application in the context of serving a monitoring purpose (e.g., intrusion detection) and in that respect traffic-related data are requested. Between the data requesting entity (i.e., the actor and application used) and the actual data (network streams or/and stored traces), the PRISM system mediates, regulating access to the data with the enforcement of several restrictions that have to do with the so-called "privacy context". As has been frequently documented, this PRISM mediator consists of two components, notably the Front-End (FE) and the Back-End (BE). In this context, there are two scenarios regarding data requests:

- *Online data request*: the data requester asks for data that are retrieved in real-time by the network link.
- *Offline data request*: the data requester asks for data that have been previously stored to the BE Data Repository.

What essentially differentiates the two scenarios is that while in the latter case it is only the BE that participates, in the case of an online data request both the FE and the BE must cooperate in order for the request to be served. In that respect, access control in offline data request refers to regulating data disclosure from the BE to the data requesting entity, while regarding online data request, the enforcement of access control includes additionally a first stage where traffic data are communicated from the FE to the BE in a way that should be controlled. That is, in the case of online data request, two decisions must be taken and enforced, concerning respectively the selective provision of data from the FE to the BE and from the BE to the requesting entity.
Taking additionally into consideration the fact that the FE is natively characterized by limited storage and computational capacity that settle the evaluation of access policies rather impossible, the enforcement of access control in the online scenario becomes very challenging. In fact, from an access control point of view, the offline scenario constitutes a sub-case of the online one and, therefore, in the following will be on the case where both the BE and the FE participate.

As has already been documented, e.g., in [D2.2.1], PRISM follows an approach for access control that is based on policies set forth by the Privacy-Preserving Controller (PPC); these policies are evaluated and enforced by the system in order to take access decisions. The fundamental components that comprise the policy framework are the following:

- A set of personal data types (*DT*)
- A set of purposes (*Pu*)
- A set of roles (*R*)
- A set of actors (*A*)

It is noted that these sets are –more or less– present and similar to all the proposed frameworks for privacy-aware access control.
The set of personal data types (*DT*) is characterized by three relationships, reflecting respectively the inheritance of characteristics, the detail level of the same concept (e.g., location) and the inclusion of a data type to another one. All the three relations define partial orders of the personal data types. More specifically:

- We say that a data type dt_i *inherits from* dt_j (denoted $dt_i \blacktriangleleft dt_j$) when the characteristics of dt_j are inherited by dt_i, referring to any rules that regulate the collection and processing of data.
- We say that a data type dt_i *is less detailed than* dt_j (denoted $dt_i < dt_j$) when the data type dt_j describes the same concept as dt_i, but with greater accuracy.
- We say that a data type dt_i *is contained to type* dt_j (denoted $dt_i \angle dt_j$) when the data type dt_i constitutes part of the type dj_i.

The latter two relationships imply the former one as follows:

- $dt_i < dt_j \Rightarrow dt_i \blacktriangleleft dt_j$, for positive dt_j authorisations (*positive authorization* for IPSourceAddr, implies positive authorization for IPSourceAddr1stOctet)
- $dt_i < dt_j \Rightarrow dt_j \blacktriangleleft dt_i$, for negative dt_i authorisations (*negative authorization* for IPSourceAddr1stOctet, implies negative authorization for IPSourceAddr)
- $dt_i \angle dt_j \Rightarrow dt_i \blacktriangleleft dt_j$, for positive dt_j authorisations (*positive authorization* for IPv4ProtocolHeader, implies positive authorization for IPv4Flags)
- $dt_i \angle dt_j \Rightarrow dt_j \blacktriangleleft di_j$, for negative dt_i authorisations (*negative authorization* for IPv4 Flags, implies negative authorization for IPv4ProtocolHeader)

In essence, the first two relations create OR-tree hierarchies of the personal data types, while the third relation defines an AND-tree hierarchy.
The set of purposes (*Pu*) is characterized by two relations, defining OR-tree and AND-tree hierarchies, respectively. Inheritance of characteristics of a purpose pu_i from a purpose pu_j is denoted by $pu_i \blacktriangleleft pu_j$, while the AND-decomposition of a purpose pu_j (i.e., to sub-purposes that all should be satisfied) is denoted by $pu_i < pu_j$, meaning that the purpose pu_j contains the pu_i.
Similarly, two relations are defined on the set of roles (*R*):

- We say that a role r_i *inherits from* r_j (denoted $r_i \blacktriangleleft r_j$) when the characteristics of r_j are inherited by r_i, referring to any role attributes and authorisations.
- We say that a role r_i *is part of* r_j (denoted $r_i \angle r_j$) when the role r_i constitutes part of the role r_j.

The former defines an OR-tree hierarchy while the latter essentially specifies an AND-tree hierarchy and reflects the explicit membership of some roles to another, implying the need for interaction between the member-roles in order for an action to be performed.
The actors – users of the system are assigned with roles; this creates the set:

- Role assignments (*RA*): $RA \subseteq A \times R$; it reflects a many-to-many relation, assigning roles to actors.

Additionally, purposes are assigned to roles as permissions, since, intuitively, not all types of roles are permitted to act (i.e., execute a monitoring function offered by an application) with respect to serving all possible purposes. For instance, the responsibilities of a *network security administrator* are clearly different from the tasks of an *accounting manager*. This creates a set:

- Purpose-to-role associations (*PR*): $PR \subseteq P \times R$; it reflects a many-to-many relation, assigning to roles the right to act for specific purposes.

So, actors are assigned roles and roles are empowered with the right to act for fulfilling a monitoring-related purpose. What is now left is the association of personal data types with

{purpose, role} associations, i.e., elements of the *PR*. In that respect, we say that a tuple of personal data $\langle dt_1, dt_2, \ldots, dt_m \rangle$ *is permitted for a purpose-to-role association* pr_{zy}, when all the personal data types comprising the tuple are necessary to any actor a_x assigned with the role r_y, in order to fulfil the purpose pu_z, where $pr_{zy} = \langle pu_z, r_y \rangle$. That is, the following set is defined:

- Permitted data types (*PDT*): $PDT \subseteq DT^n \times PR$.

All the above reflect access control with respect to the interaction between the BE and the entity that requests for data. On the other hand, regarding the controlled access of the BE to data provided by the FE, the situation is more complicated, since the data types defined above as *permitted* may not be known to the FE. For instance, an actor of "*accounting manager*" role may need data related to a "*billing*" purpose, such as the "*total duration of outgoing VoIP calls for a period of two months*". Such data types are clearly not known to the FE and therefore, rules defined for such data types cannot be used by the FE in order to grant access to the BE and –consequently– to the requesting entity.
The different types of data that the PRISM system deals with are categorized either as *raw data types* or as *derived data types*. The former are the ones that are explicitly contained in the monitoring flows (e.g., protocol header fields) or any other types of data that can be provided directly by the FE. On the other hand, as *derived* are characterized these data types that constitute the product of some processing functions that extends the FE capabilities. These data types are generated by the BE and/or the monitoring application.
Access control at the level of the FE can be expressed as follows: *the provision to the BE of the raw data types that are absolutely necessary in order for the BE to be able to produce the derived data types that the requesting entity is authorised to receive, based on the "privacy context"*[7,8]. In the following, it will be described how access control rules for the FE can be specified.
The BE is empowered with a rich library of software tools, referred to as Embedded Processing Components (EPC) [D2.2.1]. The EPC incorporate (among other tools) functionalities for the transformation of a set of data types to another, including simple and complex transformations, aggregation, anonymisation, etc. These functionalities constitute in essence a set of Data Transformation Functions (*DTF*), where each function $f_i \in DTF$ is of the form f_i: $DT^n \rightarrow DT^m$. In order for the BE to produce the derived data types that will be delivered to the monitoring application instead of disseminating the actual raw monitoring data, it makes use of these functionalities. Intuitively, the BE needs to be provided by the FE with the set of data that comprise the necessary input to the corresponding functions.
That is, if for a given role r_i and a given purpose pu_j the permitted data types are pdt_{ij}, we define the *raw permitted data types* as a set pdt_{ij}^*, such that:

- For every personal data type $dt_x \in pdt_{ij}$, there is a transformation path comprised of functions $\in DTF$ that produces the type dt_x from data types that are contained in pdt_{ij}^*.
- The pdt_{ij}^* contains no data types that are not used in the transformation paths.

Therefore, the issue of controlling access at the FE level is equivalent to the problem of calculating the set pdt_{ij}^*. All the necessary information for this purpose is contained in the PRISM Ontology, which provides a convenient and efficient way for such calculations,

[7] Naturally, both the requesting entity and the BE itself must be authenticated.
[8] It is noted here that raw data types may be finally delivered to the monitoring application as well; however this case is considered trivial.

following semantic pointers (OWL object properties)[9]. For the calculation of the set pdt_{ij}^{*}, the algorithm *RawPDT* shown in Figure 5 is used.

```
RawPDT (node)
{
for each (DataTransformation that contains the node data type)
{
check for MatchingDataType through the transformsDataFromType relation;
       for each (MatchingDataType)
       {
              if (MatchingDataType!=RawDataType, that is data
              recognizable by the FE)
              RawPDT (MatchingDataType);
              else
              save and add (RawDataType to the PDT* set);
}
}
return (PDT* set);
}
```

Figure 5 – Calculation of the set pdt_{ij} *(RawPDT) Algorithm*

The above algorithm is based on the DataTransformations Class of the PRISM Ontology and works by taking into consideration the transformsDataFromType relation of each instance, to form a chain backward search procedure. The procedure is finalized when all raw permitted data types are accumulated and attached to a (PDT, PDT*) association.

In contrast to the BE which is not only fully aware of the PRISM Ontology and the rules it describes, but also it has the processing capacity for performing ontological reasoning and evaluating the corresponding rules, the situation regarding the FE is different. The FE neither is aware of the ontology nor has the computational resources to do any reasoning (e.g., calculate the set pdt_{ij}^{*} for a given role r_i and a given purpose pu_j). Moreover, the always "online" nature of the data requests that constitute the subject of FE does not permit the execution of such functionalities. Therefore, some other entity must generate the set of the data types that the FE must provide the BE.
An obvious and apparently convenient alternative could be the execution of the corresponding processing by the BE. However, this approach has a significant disadvantage: it implies that the FE fully trusts the BE. Additionally, despite the computational power of the BE, the high data rates settle any processing saving not only important but critical. Therefore, another alternative should be considered, ideally doing the underlying calculi offline. In that respect, a mechanism grounded on an X.509 infrastructure [ITU-T2005] has been considered in the context of PRISM. This mechanism is presented in the following.
A fundamental foundational principle of the model is that it categorizes the reasoning functions related with access control to *static reasoning* and *dynamic reasoning*. The term *static reasoning* refers to reasoning that is based on rules that can be *a priori* evaluated. And actually, all the afore-described access control principles can be evaluated before the submission of a data request. In fact, from the PRISM Ontology it is possible to specify all the purpose-to-role associations (*PR*), permitted data types (*PDT*) and raw permitted data types (PDT^{*}), with the execution of *RolePurpAlg* algorithm (Figure 6).

[9] The performance assessment of these means constitutes part of the current work for the workpackage WP 2.3; the assessment's results will be described in the deliverable D2.3.1.

```
RolePurpAlg (node)
{
if (node is AllRoles)
{
       for each (rule that contains the node's role)
       do {
       save PDT set (node's Role, matching rule's Service, matching rule's DT);
       mark as visited;
       }
}
else
{
for each (node at equal level)
do {
       for each (rule that contains the node's role)
       do {
       save PDT set (node's Role, matching rule's Service, matching rule's PDT);
       }
if (node has inheritsFromRole relation with its ancestor)
{
              for each (saved set in the node's first level ancestor)
              do {
              save PDT set (node's Role, ancestor's Service, ancestor's PDT);
              }
       mark as visited;
}
       else if (node has containsRole relation with its ancestor)
       {
       save PDT set (node's Role, Σ (ancestor's Service),
       Σ (ancestor's PDT));
       mark as visited;
       }
}
}
if (all nodes are marked as visited)
return (all calculated PDT sets);
else
RolePurpAlg (first node of the next level of descendant nodes);

}
```

Figure 6 – Purpose-to-role (RolePurpAlg) Algorithm

The RolePurpAlg method of the above figure represents an algorithmic approach of determining all possible purpose-to-role associations and biding them with permitted data types, through the visualization of the PRISM Ontology and its transformation into a graph. More specifically, the presented algorithm uses a Breadth First Search (BFS) graph traversal scheme to match each Role in the Roles graph with specific Services and eventually with permitted data types. Moreover, the algorithm takes into consideration the different types of inheritance, being introduced by different types of relations among the nodes of the graph, by thoroughly examining each relation.

On the other hand, the term *dynamic reasoning* refers to access control aspects that can only be evaluated in real-time, during the execution of a data request. These aspects include for instance temporal and spatial access parameters, history-based mutual exclusions between data types and other constraints, as well as the execution of complementary to data access

actions, sometimes met at the literature as "obligations". In essence, these constraints impose further restrictions to the set of data that will be finally delivered to the requesting entity, having as starting point the corresponding data set (a subset of *PDT*) that constitutes the product of the static reasoning. Their detailed specification will be the main research topic of the rest of the project with respect to access control.
While for the dynamic reasoning the only candidate component of the PRISM architecture is the BE, the most advantageous entity for doing the static part is the PPC. That is, from a policy-based systems viewpoint [RFC3198], the PRISM system can be seen as a two-stage decision and enforcement framework. The Policy Decision Point (PDP) in the PRISM policy framework consists of two subcomponents: the Static Policy Decision Point (S-PDP) and the Dynamic Policy Decision Point (D-PDP), with the former residing at the PPC and the latter at the BE. On the other hand, the Policy Enforcement Point (PEP) is split between the FE and the BE. The FE enforces the policies that refer to FE ← BE access control (implementing the concept of the raw permitted data types PDT^*) and the BE enforces the provisions of BE ← requesting entity (realizing the concept of the permitted data types *PDT* and the additional dynamic provisions).
As has already been described [D2.2.1], the PPC constitutes the entity that defines and maintains the PRISM Ontology, as well as the Source of Authority (SoA) of the PRISM authorisation infrastructure, thus holding a vital role in the authorisation schema. As the SoA, the PPC constitutes the entity that is trusted by a privilege verifier as the one with the ultimate responsibility for the assignment and certification of a set of attributes.
In offline mode, the PPC issues the following types of X.509 Attribute Certificates (ACs) with respect to authorisation:

- Role assignment certificates *CertRA* which certify that the holder of such a certificate has been assigned the corresponding role(s). That is, the *CertRA*s constitute the implementation of the *RA* relation.
- Purpose-to-role associations certificates *CertPR* which certify the static permissions that are assigned to a role with respect to a purpose, i.e., it reflect the afore-described *PR* relation. However, this type of certificates can be considered here as redundant since operationally it is overridden by the following two certificates. We plan though to exploit this type in order to enable the definition of conditional roles, and therefore we mentioned it here.
- Two essential types of certificates are the ones certifying the permitted data types and the raw permitted data types, denoted by *CertPDT* and $CertPDT^*$, respectively. That is, these two types of certificates integrate and certify for a given role r_i and a given purpose pu_j the data types that should be delivered from the BE to the requesting entity and from the FE to the BE, respectively, in order for the monitoring task to be fulfilled.

With the use of the certificates' types described above, the authorisation procedure of the PRISM framework can be outlined as follows:
Assume that Alice, an actor a_i, in the context of a monitoring application's execution seeks to make use of the data access and permissions associated with a held role r_j, with respect to monitoring purpose pu_k. In that respect, Alice instructs the monitoring application to submit the corresponding online data request to the BE. In order for the BE to provide the requested data types, the following should meet:

- The actor a_i (Alice) is an authenticated system user.
- The role r_j is assigned to the actor a_i
- The role r_j is enabled for executing the purpose pu_k.
- All the data types comprising the request are permitted types for the Purpose-to-role association pr_{jk}.

In order for Alice to prove the validity of the above, she will submit to the BE apart from the cryptographic tokens related with authentication the following certificates:

- The $CertRA_{ij}$ certifying that Alice is assigned with the r_j role.
- The $CertPDT_{kj}$ that certifies the set of data types that are the absolutely necessary for the monitoring application in order to fulfil the purpose pu_k, acting on behalf of an actor assigned with the role r_j.

With these two certificates, the BE is able not only to authenticate Alice, but also to authorise her with respect to the underlying access rights, without needing to perform any ontological reasoning yet.
Supposing a successful authorisation, the BE will delegate the request for data to the FE. However, the $CertPDT_{kj}$ certificate is rather useless in the context of the FE $\leftrightarrow$ BE interaction. This is because what the FE needs in order to validate the authorisation is the set of raw permitted types, i.e., the types that is in principle able to provide. Therefore, Alice should additionally submit the $CertPDT_{kj}^{*}$ certificate, in order to be evaluated by the FE.
It is important to stress here that the FE enforces the semantic policies described by the PRISM Ontology, without doing any reasoning. In fact, any reasoning has been done offline by the PPC and the FE participation in the procedure is limited to the validation of X.509 certificates and actually the cryptographic confirmation that the data request it has receive is valid, meaning that:

- The actor a_i is an authenticated system user.
- The BE is an authenticated system component.
- The role r_j is assigned to the actor a_i.
- The role r_j is associated the purpose pu_k.

The raw personal data types specified by the $CertPDT_{kj}^{*}$ certificate are permitted for the pr_{jk} association.
Figure 7 depicts by an example the structure of an implemented Attribute Certificate *CertRA*.

```
Certificate:
  Data:

    Version: 2
    Serial Number: 1234266363562
    Signature Algorithm: SHA512withRSA
    Issuer: CN=PPC
    Holder: CN=Anna Antonakopoulou
    Validity:
      Not Before: Tue Feb 10 13:46:03 EET 2009
      Not After: Tue Nov 10 13:46:53 EET 2009
     Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (512 bit)
               Modulus (512 bit):
              1e1a91078b1eddea1989134badc9c89c2207819d7a9
              67874c73354667fcbe57598b34d6771b2108559762a
              9138c784f7488a605f92c53454afae8475063c707d
    Attribute: PrivacyAuthorityLawfulInterceptor
    Extensions:
      X509v2AttCert Basic Constraints:
        CA: FALSE
```

Figure 7 – Role Assignment Certificate (CertRA) Example

3.4 Evaluation of Anonymisation Tools

3.4.1 Anonymisation tools Analysis

A significant number of tools exist which in theory are applicable as anonymisation tools for PRISM. However our requirements on such tools limit the selection considerably: For our purposes, we can only recommend a tool which has support for multiple header fields, a wide variety of anonymisation primitives already implemented, and a flexible command syntax to specify the anonymisation algorithms to be applied. In addition, we prefer tools that are (a) remotely configurable and (b) represent a flexibly extensible framework rather than a monolithic standalone tool. In theory one can combine several tools in order to achieve the desired result. This, however, tends to lead to unnecessary overhead and dependencies. Therefore we aim to start with the most suitable tool/framework, one which is capable of fulfilling almost all or even all requirements of the PRISM system, and then extend this solution with new functionality which might be needed for specifically selected analysis scenarios we chose to realise within PRISM.

In addition to (a) + (b) (see above) the following requirements should be met by an anonymisation tool or framework for PRISM:

- high anonymisation processing speed
- be able to work in online (life) as well as offline (on traces) fashion
- be able to process traces which do not fit into a computer's main memory
- be able to work on trace files >4GB
- flexible configuration, e.g. via command line, config file (e.g. XML spec file), and remote configuration protocol (e.g. XML-RPC, SOAP, or NetConf)

- supports anonymisation of all known header fields of most well-known protocols (complete IP, TCP, and UDP support is a must)
- various *IP address* anonymisation methods including, renumbering (mapping), partial randomization, prefix-preserving mapping, and hashing (replacement by a hash value)
- flexibly (i.e. also in combination) supports anonymisation primitives (*for header fields in general*) such as
    - delete/nullify field, truncate packet data, and cut off payload
    - add random offset of defined maximal size to field value
    - replace fields by remapped value based on this field
    - replace field by hash value of this field
    - optional: replace fields by a hash value computed across multiple input fields

In this Section we give a brief survey on several publicly available anonymisation tools with the PRISM architecture in mind. The final analysis and decisions will be documented in the follow-up deliverable D313.

3.4.1.1 Anontool

Anontool [ANONTOOL] was developed by the Institute of Computer Science (ICS) of the Foundation for Research and Technology – Hellas (FORTH). It supports anonymisation of NetFlow v5 and v9 and IPFIX traces and is suited as a general tool for anonymising tcpdump traces. It is an open source implementation of the Anonymisation API (AAPI, c.f. [ANONTOOL]), with some example applications. Anontool supports anonymisation of a wide range of header fields; nearly all fields of standardised layer 2, 3, and 4 protocols are covered, plus some layer-7 fields.
Reading traffic directly from the interface is possible, so real-time (online) packet data anonymisation is possible. Dynamic reconfiguration of the anonymisation would need additional implementation work on the tool and its libraries.
All available Anontool applications can work either on input files or network interfaces and apply one ore more of the following predefined anonymisation functions. Note that not all functions can be applied with all applications.

- Anonymise IP addresses, either using prefix-preserving mapping or replacing with zero
- Anonymise TCP ports, either using mapping or replacing with zero
- Anonymise Ethernet addresses by setting them to zero
- Anonymise TCP/UDP payload
- Delete TCP and IP options
- Set TCP flags to zero
- Anonymisation of arbitrary NetFlow fields by any available anonymisation function.

To gain the full anonymisation flexibility of the AAPI, which is available as C-library, a separate tool needs to be developed. For details on what header fields can be anonymised by which functions, textual documentation is provided with the distribution.

3.4.1.2 Ruler

Ruler [RULER] was developed by The Computer Science department of the Faculty of Sciences at the Vrije Universiteit, Amsterdam, The Netherlands. Ruler is a pattern matching and rewriting language made for the anonymisation of network packets. Rules, consisting of a matching pattern and a rewrite pattern, control the handling and rewriting of the packet contents (header fields).

Possible fields and anonymisation algorithms to apply are nearly unlimited due to high flexibility in the use of regular expressions, pattern matching, and so on. However, this may lead to poor performance in case the filters are not crafted carefully.
Reading traffic directly from the network interface is possible and done by ruler. There are different standard templates for various input formats, ranging from pcap over MAPI to log files. Configuration is done by using the ruler language and the command line. Dynamic reconfiguration while capturing for an ongoing task seems not to be supported by ruler out of the box; some implementation effort would be needed to add this functionality.

3.4.1.3 Flaim

Flaim [FLAIM] has been developed by the LAIM Working Group at the NCSA. It is a generic anonymisation framework, which can be used to anonymise network traffic. Configuration is done in XML, with support for validation of these policies. Modules for different data sources are communicating via an API with Flaim. This allows new data sources (sales records, student data, licence plate scans, whatever) to be added later on.
FLAIM supports the nearly all layer 2, 3, and 4 fields and lots of options within a header you can find in network traffic; missing fields should be easy to add. There are standard modules for pcap, NetFlow and other, not network traffic related input sources. Flaim does not support direct packet capture, but it does accept standard in as source for packet data, so it should be possible to pipe data to it via tcpdump. In this case however modules which don't support streaming will reject the input[10]. Flaim can be started via the command line; configuration files will be specified there. The actual configuration of anonymisation tasks is done via policy files written in XML.

3.4.1.4 Pktanon

Pktanon [PKTANON] has been written by the institute of telematics at the University of Karlsruhe. Pktanon has been written for network trace anonymisation, based on profiles. It has anonymisation primitives for most layer 2, 3, and 4 attributes, but it does not cover any layer 7 information. Live packet capture can only be done via piping packet data from tcpdump; usually this tool reads data from a file. Configuration is done via an XML file. Pktanon follows the concept of 1:1 mapping from field to anon algorithm quite strict, so it is questionable whether the idea of hashes over many fields is implementable without high overhead.

3.4.1.5 Other Tools

There are several other anonymisation tools, which were analysed but did not make it into the "final round" due to different shortcomings:

Crypto-PAn [CRYPTOPAN], is a tool aimed at anonymising the IP addresses in prefix-preserving way by using cryptographic techniques (Pan stands for Prefix-preserving Anonymisation). Since we need much more functionalities than just (cryptographic) anonymisation of the IP addresses, this tool cannot be the overall solution for PRISM, even though it does a good job with the IP addresses.

[10] This drawback is not specific to Flaim; any module which needs to analyse input data with several passes will not work on life traffic data.

Tcpdpriv [TCPDPRIV] is one of the older anonymisation programs. Compared to newer programs it is quite limited in functionality and extensibility. The latest release is from 2005.

Ipsumdump [IPSUMDUMP] uses click [CLICK] for packet processing and a tcpdpriv derived click module for anonymisation, thus inheriting many limitations from tcpdpriv.

Tcpurify [TCPURIFY] is another rather old standalone tool. It is not as powerful or flexible as the more modern frameworks.

Tcpmkpub [Tcpmkpub] is a policy based anonymisation tool, which fulfils a lot of our requirements. Unfortunately, policies are defined at compile time which limits flexibility significantly.

Canine [Canine] is the predecessor of Flaim. This tool is deprecated and tnot developed anymore.

Scrub [SCRUB] is a useful small anonymisation tool, but as a typical standalone tool, it is too limited for the flexibility needed within PRISM.

Flowmon [FLOWMON] probe is a small hardware device. This approach is very useful on its own but at this stage limits our options for integration with the PRISM. However, we acknowledge that hardware based anonymisation techniques itself can be part of the future PRISM compatible solutions.

Tcpdump Anonymizer [TCPDUMP ANON] unfortunately is documented only in the source code, and thus it is hard to extend and/or deploy it.

3.4.2 Complexity and Performance

Still upcoming is a real-life evaluation of the final top three anonymisation tools with regards to functionality, as well as, performance. For example, according to [FOU07], there can be very significant performance differences, which depend strongly on the actual anonymisation primitives applied. Results of the planned detailed benchmarking will be documented in Deliverable D2.3.1 "Preliminary Performance and Regulatory Assessment" of Workpackage 2.3.

4 Conclusions

This deliverable "Preliminary Data Protection Algorithms Specification and Analysis" introduces the data protection concepts and approaches to be applied in PRISM, tailored to the two tiered nature of the PRISM architecture. Access control paradigms are presented, which are specifically tailored to the PRISM monitoring scenarios. These approaches will be used within PRISM in a combined manner in order to maximise the protective potential.
This deliverable reports the specification and analysis of the technical approaches proposed and/or considered during the first year of the project for protecting traffic data and with it the users' privacy.

The data protection and privacy preservation techniques used in PRISM serve multiple goals:

- Protection of the network users' data privacy, specifically matched to the need of the monitoring applications;
- Protection against tapping into the system by unauthorized personnel (be it wiretapping, access to databases, or access to analysis results data)
- Proportionality of collected data based on targeted use
- Efficiency of data export and analysis (a must due to high speed constraints)
- and, last but not least, effectiveness of the finally implemented analysis methods

Towards these goals this document specifies several approaches:

- a brand new paradigm for privacy preservation which envisions data protection as a dynamic time varying attribute which can be triggered by monitoring events occurring in the front-end tier,
- a classification of anonymisation techniques, following their risk-utility trade-offs,
- a semantic role/purpose based authorization framework composed of two functionally different front-end and back-end tiers,
- and an analysis of multiple anonymisation tools, showing their specific capabilities, pros and cons.

It is devised to implement in PRISM a combination of the above approaches in order to combine their privacy protecting capabilities and still maintain the flexibility needed for effective traffic analysis.

5 References

[AMD] A. M. Devices. AMD Secure Virtual Machine Architecture Reference Manual. AMD, 2005

[ANONTOOL] Anontool, http://www.ics.forth.gr/dcs/Activities/Projects/anontool.html

[BON06] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, "An Improved Construction for Counting Bloom Filters", Algorithms – ESA 2006, 14th Annual European Symposium, pp. 684–695, 2006.

[BOR01] N. Borisov, I. Goldberg, D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11", ACM SIGMOBILE, 7th international conference on mobile computing and networking, Mobicom 2001, Rome, Italy, July 2001.

[BRO05] A. Broder, M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey", Internet Mathematics, Volume 1, Issue 4, pp. 485–509, 2005.

[BUR08] M. Burkhart, D. Brauckhoff, M. May, E. Boschi. The Risk-Utility Tradeoff for IP Address Truncation. Proceedings of the First ACM Workshop on Network Data Anonymisation (NDA 08), Alexandria, Virginia, October 2008.

[CANINE] CANINE: Converter and ANonymizer for Investigating Netflow Events, http://security.ncsa.uiuc.edu/distribution/CanineDownLoad.html

[CLICK] "The Click modular router", Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297. see also http://read.cs.ucla.edu/click/

[COU07] S Coull, C Wright, F Monrose, M Collins, M Reiter, "Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces", in proceedings of the Network and Distributed System Security Symposium (2007)

[CRYPTOPAN] Crypto-Pan – Cryptography-based Prefix-preserving Anonymization, http://www.cc.gatech.edu/computing/Networking/projects/cryptopan/

[D2.2.1] PRISM Deliverable "D2.2.1: High level System Architecture Specification", 2008, the PRISM project consortium

[D3.1.1] PRISM. Deliverable "D31.1: State of the art on data protection algorithms for monitoring systems", 2008, the PRISM project consortium

[EST02] C. Estan, G. Varghese, "New directions in traffic measurements and accounting", ACM SIGCOMM 2002, Pittsburgh, USA, Aug. 2002.

[FAB06] Fabian Haibl and Falko Dressler, Anonymisation of Measurement and Monitoring Data: Requirements and Solutions, in "Praxis der Informationsverarbeitung und Kommunikation" collection #29, issue #4, pages 208–213, ISSN 0930-5157, october-december 2006

[FLAIM] FLAIM (Framework for Log Anonymisation and Information Management), http://flaim.ncsa.uiuc.edu/

[FLOWMON] FlowMon Probe, http://www.cesnet.cz/doc/techzpravy/2006/flowmon-probe/

[FOU07] Michael Foukarakis, Demetres Antoniades, Spiros Antonatos and Evangelos P. Markatos. "Flexible and High-Performance Anonymization of NetFlow Records using Anontool", in Proceedings of the Third International Workshop on the Value of Security through Collaboration (SECOVAL), September 2007, Nice, France.

[GOL86] O. Goldreich, S. Goldwasser, S. Micali, "How to Construct Random Functions", Journal of the ACM, vol.33, no.4, p.792-807, 1986.

[IPFIX-ANON] E. Boschi, B. Trammell. IP Flow Anonymisation Support. IETF IPFIX individual submission work in progress, draft-boschi-ipfix-anon-02.txt, January 2009.

[IPSUMDUMP] IPSumDump tool, http://www.cs.ucla.edu/~kohler/ipsumdump/

[ITU-T2005] International Telecommunication Union (ITU) – Telecommunication Standardization Sector, "Information technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks", ITU-T Recommendation X.509, August 2005.

[KEL07] S. Kelly, S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", IETF Request for Comments, RFC 4868, May 2007.

[KOU06] D. Koukis, S. Antonatos, K. G. Anagnostakis. On the privacy risks of publishing anonymized IP network traces. Communications and Multimedia Security, vol. 4237 of Lecture Notes in Computer Science, pp. 22-32. Springer, 2006.

[PKTANON] PktAnon - packet trace anonymisation, http://www.tm.uka.de/software/pktanon/

[PRIME] FP6 IST PRIME (Privacy and Identity Management for Europe), https://www.prime-project.eu/.

[RAM03] S. Ramabhadran, G. Varghese, "Efficient Implementation of a Statistics Counter Architecture", ACM SIGMETRICS 2003, San Diego, USA, June 2003.

[RFC3198] A. Westerinen, J. Schnizlein, J. Strassner, et al, RFC 3198 – „Terminology for Policy-Based Management", November 2001

[RFC5101] B. Claise, S. Bryant, S. Leinen, T. Dietz, B.Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. IETF Request for Comments 5101, January 2008.

[RIB08] B. Ribeiro, W. Chen, G. Miklau, D. Towsley. Analyzing privacy in enterprise packet trace anonymisation. Proceedings of the Fifteenth Annual Network and Distributed Network and Distributed Systems Security Symposium (NDSS 08), February 2008.

[RULER] Ruler, https://gforge.cs.vu.nl/gf/project/ruler/

[SCRUB] http://scrub-netflows.sourceforge.net/ and http://scrub-tcpdump.sourceforge.net/index.php

[SEE07] Vidar Evenrud Seeberg, Slobodan Petrovic, "A New Classification Scheme for Anonymisation of Real Data Used in IDS Benchmarking"

[SHA79] A. Shamir, "How to share a secret", Communications of the ACM, volume 22, issue 11, pp. 612-613, 1979.

[SLA04] Adam Slagell, Jun Wang, William Yurcik, "Network Log Anonymisation: Application of Crypto-PAn to Cisco Netflows"

[SON00] Song, D., Wagner, D., and Perrig, A., "Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy", Oakland, CA. IEEE Computer Society Press, Los Alamitos, CA. 44–55

[TAPAS] 5th FWP TAPAS (Trusted and QoS-Aware Provision of Application Services), home page: http://www.newcastle.research.ec.org/tapas/

[TCPDPRIV] TCPDPriv Tool, http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html

[TCPDUMP ANON] Anonymizer Tool, http://sourceforge.net/projects/anonymizer

[TCPMKPUB] TCPmkpub Tool, http://www.icir.org/enterprise-tracing/tcpmkpub.html

[TCPURIFY] TCPurify Tool, http://irg.cs.ohiou.edu/~eblanton/tcpurify/

Appendix A – Glossary

DATA CATEGORIES

Data in the PRISM context is classified into five different categories:

PACKET DATA

- any field from layer 2,3,4 data or payload of packets seen on the wire e.g. EtherType, IP addresses, TCP sequence number, Total Length, etc.
- for such fields the defined names from wireshark Display Filters are useful, e.g. eth.type, ip.src, ip.dst, tcp.seq, ip.len, http.last_modified etc. See also in the wireshark Display Filter Reference

MEASUREMENT DATA

- based on the results of the measurement but not on bits from inside the packet data
- timestamps, link load (if taken from the device, e.g. from the Router MIB via SNMP), CPU load

TASK META-DATA

- any 'setting' used for performing the measurement and information about the measurement setup
- e.g. link capacity, meas. task initiator, router type, start time of task, monitored AS, filter expression
- may include also the anonymisation algorithms and parameters used (but it might be not allowed to disclose all of them!)

DERIVED DATA

- based on statistical analysis of the measurement data and/or packet data ; e.g. flow bandwidth, mean packet size, inter-packet delay distribution, TCP connects per second...

EXTERNAL DATA

- required by monitoring applications to perform analysis and evaluation of the measurement data and packet data ; e.g. IP address -> AS number mapping, IDS signatures

ANONYMISATION OPTIONS:

We can differentiate two basic functional approaches to changing information:

ONE-WAY FUNCTION:

- The term one-way function refers to a mapping which is hard or impossible to reverse.
- Typical usages include hashes (integrity, authentication) and anonymisation.

TWO-WAY FUNCTION (CIPHER):

- The term two-way cipher is a reversible function f from plaintext M to cipher text C parameterised with one or multiple keys.

- A strong requirement for ciphers is that it is computationally infeasible (apart from brute-force attack) to determine a plaintext message M from cipher text C=f(M) without the knowledge of the relevant key material.

DATA FORMATS

- Data can appear in **plaintext**, **encrypted** and/or **anonymised**. In principle, encryption and anonymisation can be applied to all of the above data categories, while not all possible combinations are useful in practice.

PLAINTEXT

- Data in ASCII or binary form.

ENCRYPTED

- data encrypted by a one-way function or a cipher

ANONYMISED

- data where private information about users is reduced
- during the anonymisation process, private information can either be altered or deleted
- the data format is expected to be semantically the same as the original data used as input (e.g. for packet data the tcpdump format can be kept during the anonymisation process)
- anonymised data does not include the description of the applied anonymisation (would be task meta-data)

Appendix B – Anonymisation Tools Features and Overview

The following table shows an overview of common (free) tools used in the research community for performing traffic trace anonymisation. In general one can state that tools for this purpose have become quite versatile and mature. However most of them still lack professional documentation and support and are provided in a as-is manner. From the state of the art with respect to anonymisation tools one can see that at first many highly specialised single-purpose tools were developed and later the a trend towards frameworks and policy languages for the specification of the concrete anonymisation emerged. Creation and validation of such policies is up to the user. These are the characteristics that were evaluated during the analysis of the tools:

PREFIX PRESERVING

- If a tool supports prefix preserving anonymisation for IP addresses this means that it is capable of separately anonymising the network part and the host part of an IP address, thus preserving vital traffic characteristics. This is supported by almost any tool. However as there are different approaches to doing prefix preserving anonymisation one should check for a used tool which algorithm exactly is used.

STATELESS PROCESSING (CONSISTENT ANONYMISATION)

- The ability to perform stateless anonymisation denotes the following: the result of anonymising a packet X does not depend on any specifics of packets seen before. This characteristic allows in effect performing the anonymisation process in parallel in a distributed manner and afterwards getting a consistent result when concatenating the anonymised traces again. This is not a feature which most anonymisation tools provide but it is becoming more and more available in the more recent tools as it is seen as am important feature.

VULNERABILITIES

- Depending on the concrete anonymisation algorithm and the configuration parameters in use the resulting output of anonymisation can be vulnerable to reverse engineering attacks. Some of these attacks are generic, other can be tool-specific. Some links to those are listed in the table.

INPUT

- pcap is the de facto standard for input files to anonymisation tools. Some tools support flow reports (netflow v5,v7, or v9) instead of packet traces as input. Rarely some proprietary input format is observed.

ONLINE PROCESSING

- Most pcap based tools can also do live capture from an interface, but for some configurations of these tools there are restrictions to live capture in case that the used algorithm requires multiple passes over the packet data.

OUTPUT

- The output format is usually equal to the input format (pcap or netflow). Sometimes additional statistics are written as ASCII text output to the console too. Output is per default written into a file, no matter if the input was life traffic or a file too.

REVERSE MAPPING

- This denotes the ability of a tool to explicitly output the mapping which was used for anonymisation to a data file. Such mapping needs to be kept strictly secret but it is useful to revert the processing in case that e.g. an attacker needs to be identified.

Technical Tool Features

Tool	Speed	Ttrace > RAM	Trace > 4GB	Config via CLI	Config via XML	Config via RPC	Usable via API	Extendable via API	Source Quality	Last Update	Remarks
anontool	ca. 6-50x faster then flaim (they claim)	yes		Yes (example app)	Yes (example app)	No, but building a client should be feasible	Yes, AAPI	Extending the API for is possible	API does not look so bad	Version numbers are for wimps, we call everything „latest“ probably 10-2008	Framework with example app. Should not be too difficult to extend.
cpan-dts				Yes	No	No					Just crypto, nothing else
CryptoPan											Just crypto, nothing else
Flaim	Look at a anontool			Yes	Yes	no	No	Yes		05.12.2006	CPI & XML together
pktanon				No	Yes	No		Yes, kind of		26.06.2008	Good documentation
ruler				parts	Nope	Never ever	Not really	It is a language, can do all kind of matching		25.06.07-22.11.2007	It's a whole language

Supported Header Fields

Tool	src_ip	dst_ip	Ttl	Tos	Id	ip_proto	version	Options	Length	src_port	dst_port	SEQUENCE NUMBER	ACK_NUMBER	UDP_DATAGRAM_LENGTH	ICMP type	ICMP code	Div l7	Remarks
anontool	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Many more
cpan-dts	Y	Y																
CryptoPan	Y	Y																
Flaim-pcap	Y	Y	Y	N	Y	N	N	Y		Y	Y	Y	Y		Y	Y	N	Many more
Flaim-netflow	Y	Y	-	N	-	Y	N	-	-	Y	Y	-	-	-	-	-	-	Many more
Pktanon	Y	Y	Y	Y	Y	N	N	Y	N	Y	Y	Y	Y	N	Y	Y	N	
ruler	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	As a pattern matching and rewriting language, it is virtually unrestricted

Supported Anonymisation Techniques

Tool	delete/ nullify	Random offset	Replace by hash	Hash over many fields	Remarks
Anontool	Y	N	Y	unsure	Missing features should be easy to implement
Crypto-PAn	N	N	Y	N	
CryptoPan	N	N	Y	N	
Flaim	Y	N	Y	unsure	Missing features should be easy to implement
Pktanon	Y	N (should be easy to implement)	Y	N	1:1 mapping of field to anontype, so hash over many fields looks unlikely
Ruler	Y	maybe	Y	maybe	Have to understand the language first

Documentation

Tool	Manpage	Commandline help	Web page	PDFs, book like manuals, admin guides, etc	Papers
Anontool	Only a readme	Example apps yes	Y	None	Yes, some
Flaim	Small manpage as html	Y	Y	User guide, module guide, looks usable	
Ruler	Y	Y	Y	Y	Y