Exchanging threat information between semi-honest parties

Author:
Anton Jongsma
s4230310

Supervisors:
dr. Jaap-Henk Hoepman
Bart Roos MSc
Pieter Rogaar MSc

Master Thesis
Computing Science
Radboud University Nijmegen
June, 2015

Abstract

Sharing information between organizations is important to defend the vital infrastructure against digital threats. However, concerns regarding privacy and confidentiality of information makes organizations reluctant to share such information. In practice the information is only shared with highly trusted partners. Lowering the trust required to share information can encourage new collaborations. In this research we consider methods for exchanging threat information between organizations in a semi-honest adversary model, such that threats can be detected without necessarely sharing details about all known threats. Threat information exchange is characterized and a number of methods are proposed. These methods aim to keep threat information confidential while still remaining useful for detection purposes, anonymize context information of incidents, or anonymously contribute to aggregated statistics related to threat information. We conclude that these methods are promising and can indeed enable new collaborations. However none is without cost in terms of performance or ease of use. Moreover, the methods can only be applied to specific problems within threat information exchange.

Contents

1. Introduction

Today's society depends to a great extent on digital infrastructure. Critical components such as the energy grid, telecommunications, but also governmental agencies, hospitals and factories are often connected to the internet.

This connectedness brings many benefits to society and creates great potential for innovation, but it also brings new risks. Among these risks are criminals who spread malware and (corporate) espionage by so called Advanced Persistent Threats, or APTs. In various ways and with different goals these actors aim to compromise the systems our society relies on.

For individual organizations it is difficult to defend against these new threats because they have only a limited view on the threat landscape and lack concrete information about new threats. Exchanging information about new threats between multiple organizations helps to detect incidents. Such information is often referred to as an Indicator of Compromise (IoC), and can be as simple as an IP address, a URL, a hash of a file or something more sophisticated. Other helpful information that can be exchanged are statistics about IoCs or contextual information of an IoC hit. Together this information is considered threat information. It gives organizations a broader view on the threat landscape.

This exchange of information already happens in different forms. Examples in the private sector are Microsoft Interflow[1], McAfee Threat Intelligence Exchange[2] and AlienVault OTX[3].

In the public sector initiatives such as the *Nationaal Detectie Netwerk*[4] are being developed. Open Source software such as MISP[5] and SoltraEdge[6] is being deployed as well to facilitate the exchange of threat information.

A number of open standards such as STIX[7], TAXII[8] and OpenIOC[9] have been introduced recently. These are gaining support from various vendors. This illustrates the trend of collaboration and automated exchange of threat information in security. Note that automated exchange does not mean everything is done

[1] http://www.microsoft.com/interflow
[2] http://www.mcafee.com/us/products/threat-intelligence-exchange.aspx
[3] https://www.alienvault.com/open-threat-exchange
[4] http://www.rijksoverheid.nl/documenten-en-publicaties/publicaties/2014/03/17/nationaal-detectie-netwerk.html
[5] http://www.misp-project.org/
[6] http://www.soltra.com
[7] https://stix.mitre.org/
[8] https://taxii.mitre.org/
[9] http://www.openioc.org/

automatically without human interference. With automated we mean the data is exchanged regularly using protocols and formats that are machine readable, and not ad hoc.

1.1 Privacy, Confidentiality and Trust

Threat information can be sensitive information. It may contain:

- personally identifiable information such as a name or telephone number,
- proprietary information such as trade secrets, or
- technical information such as network configuration data and passwords.

Concerns over disclosure of such information can make organizations reluctant to share information with arbitrary parties. Another reason against openly publishing threat information is that it might tip off the adversary, who can then circumvent detection. This would make the information essentially useless. The result is that threat information exchange is usually taking place between parties that have a high level of trust in each other. Although this works in small groups, establishing trust between a large number of parties is difficult.

This raises the question if exchanging threat information between parties without such a firm trust relationship is possible. It would allow more organizations to join a group, adding value for the whole group and creating an increased incentive for others to join as well.

1.2 Privacy Enhancing Techniques

A number of academic research fields exist that explore the idea of collaborating parties that have limited trust in each other.

By lowering the level of trust that is needed between two parties to achieve a mutually beneficial goal, it becomes easier for them to start coöperating together. In this thesis, we assume that parties are curious about each others secrets but wil not deviate from the prescribed protocol to learn more about them. This is known as the *semi-honest* adversary model. It is different from the malicious adversary model, where the adversary is allowed to use any efficient means to break the security.

We consider this a suitable adversary model for threat information exchange because the parties need to have at least some trust relationship. In the current model of threat information exchange both parties fully trust each other. With the semi-honest adversary model we take a step down from the fully trusted model towards a semi-trusted model.

Note that if we consider the malicious adversary model, threat information exchange would become impossible. If both parties completely distrust each other, why would they want to exchange threat information?

In the research field of *secure multiparty computation*, two or more parties want to jointly compute a function over their inputs without disclosing their input to the other parties. A more concrete example is the sub field of *private matching*, where set intersection is performed by two parties in such a way that both parties only learn the intersection of the two sets. Another example is *anonymous data aggregation*, where an aggregator aims to compute the sum of the input from a number of clients, without getting to know the individual inputs. These fields all work with some form of limited trust within the group of participants. The methods and techniques presented in these fields will be referred to as Privacy Enhancing Techniques (PETs).

In this research we will discuss the applicability and practicality of this existing research to the problem of exchanging threat information between semi-honest parties.

1.3 Research question

Traditionally, a lot of information exchange is done in a non-automated way, for example by email or phone. Because of the diversity in format, ad-hoc nature and inherently identifiable properties of the medium, like phone numbers and email adresses, this is difficult to do in a confidential or privacy preserving way. Because of this, we limit the scope of this research to automated information exchange.

In a typical threat information exchange we can distinguish two kinds of participants, a central hub and subscribers or spokes. The *hub* distributes threat information, typicaly IoCs, to the spokes. The spokes in return contribute statistics about occurrences of those IoCs in its network (hits) and *hit context*. Useful statistics about the IoCs on an individual or aggregated level give the hub a broader overview of the threat landscape. The *hit context* is data or information that is found surrounding a hit. This information is then used by the hub to refine the threat information. The hit context can be an integral packet dump or only the IP addresses associated with a hit. In Figure 1.1 these information flows are illustrated.

The main research question can be formulated as follows:

Can threat information exchange between semi-honest parties be realized using Privacy Enhancing Techniques?

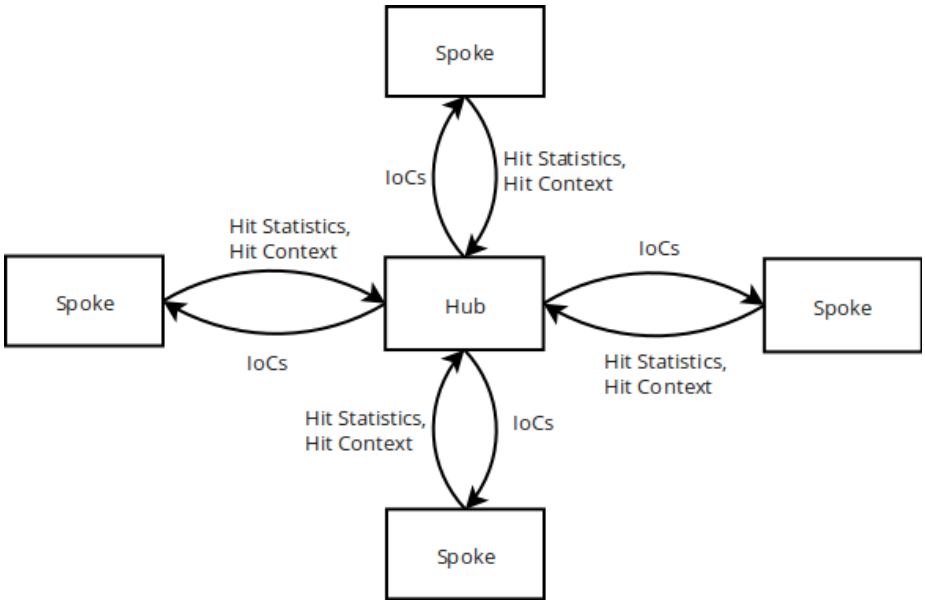

Figure 1.1: A typical threat information exchange setting

We formulate three subquestions to make the research question more concrete.

- Which PETs are available for keeping IoCs confidential?
- Which PETs are available for anonymizing hit context?
- Which PETs are available for anonymous aggregation of statistics?

These three subquestions do not completely cover the main question, most notably non-automated exchange. However, within the automated exchange, the forms of information exchange we consider most important are covered.

1.4 About NCSC-NL

The mission of the National Cyber Security Centre (NCSC) is to help enhance the resillience of the Dutch society in the digital domain. It is a information hub and centre of expertise for cybersecurity. By sharing knowledge and providing insight, it aims to realize a safe, open and stable information society. It is also the Computer Emergency Response Team (CERT) of the Dutch central government and plays a key role in the (international) coordination of major ICT incidents. NCSC-NL falls under the responsibility of the Ministry of Security and Justice of the Netherlands.

Because of its position as an information hub, threat information exchange is a relevant topic for NCSC-NL. Most information exchange currently happens in a non-automated way, but automated methods are being investigated. Due to the number and diversity of the organizations NCSC-NL exchanges information with, confidentiality issues can arise. This thesis investigates technical methods to solve such issues and encourage new collaborations in exchanging information.

1.5 Outline

In this thesis we present a number of methods that aim to enable new collaborations between organizations that currently are reluctant to share information with others. This thesis is structered as follows. In Chapter 2 we elaborate on threat information exchange and present a model of threat information exchange inspired by a number of standards. We identify three major information flows within threat information exchange: indicators, hit context and statistics. In Chapter 3 we investigate these three flows to determine if we need methods to keep (parts of) the information confidential. We also summerize which methods are already used in practice to keep (parts of the) information confidential. In Chapter 4, 5 and 6 we present methods for the three information flows. We have adapted existing methods to fit the threat information exchange setting. In Chapter 7 we draw a conclusion and discuss the constraints of the presented methods.

2. Threat information exchange

In this chapter we will look at the architecture of threat information exchange. We first give an overview based on several standards. Then we give a more abstract model that is more useful for this research. We identify three information flows that are important for threat information exchange between organizations and discuss the types of information that are found within these flows.

2.1 Threat information exchange

The parties in threat information exchange exchange, as the name suggests, information about threats. A threat is usually a very broadly defined concept, which would mean it could be any sort of information.

For example, the ENISA[1] definition includes pretty much anything harmful:

Any circumstance or event with the potential to adversely impact an asset through unauthorized access, destruction, disclosure, modification of data, and/or denial of service.

This is a too broad definition for our research towards confidentiality mechanisms in threat information exchange. In this research we are using a more narrow definition of a threat. Only threats that involve computers and/or computer networks and that can be detected and/or mitigated with network or host based intrusion detection systems (NIDS / HIDS) are considered.

This means threat information must be concrete. Examples of such information are:

- the MD5 hash of a malicious file,
- an IP address of a Command and Control server,
- a packet trace of communication from malware,
- the number of infections for some malware in a particular network.

[1] http://www.enisa.europa.eu/activities/risk-management/current-risk/risk-management-inventory/glossary#G51

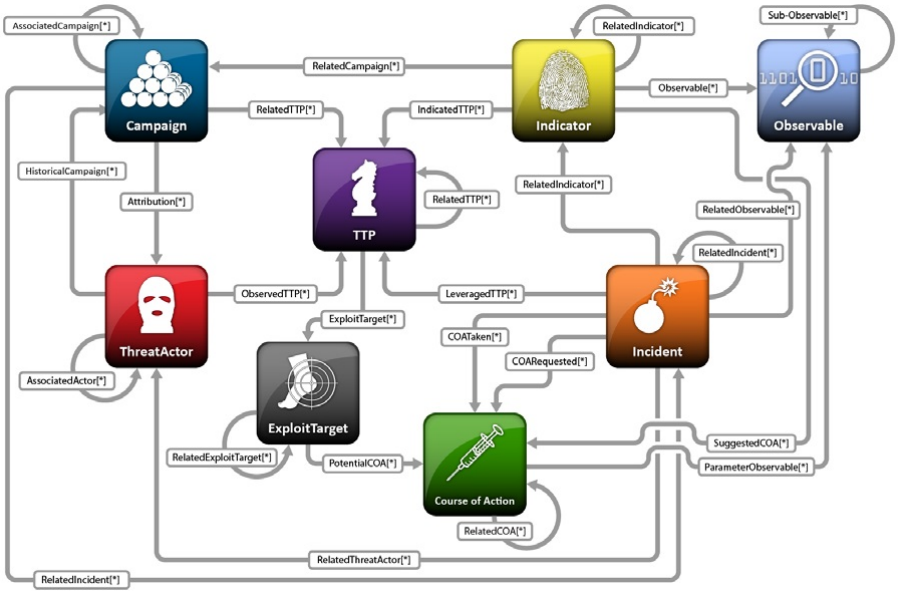

Figure 2.1: Overview of STIX[a]

[a]https://stixproject.github.io/getting-started/whitepaper/

With this limited definition we can use a number of industry standards as reference points for automated threat information exchange. These standards are introduced in more detail in the next section. With these standards we have a more concrete model of threat information exchange. We can investigate and propose technical means for this model to address privacy and confidentiality issues. In this research we use the term privacy when we aim to protect personal data. If we aim to protect other data we use the term confidentiality.

We use a model of threat information exchange inspired by a number of standards that have recently emerged: Structured Threat Information eXpression[2], Cyber Observable eXpression[3] and Trusted Automated eXchange of Indicator Information[4], all developed by MITRE[5].

2.1.1 Structured Threat Information eXpression (STIX)

Structured Threat Information eXpression (STIX) is an XML-based language to describe a threat in a structured way. A number of classes are defined, such

[2]http://stix.mitre.org
[3]http://cybox.mitre.org
[4]http://taxii.mitre.org
[5]http://mitre.org

as actors, incidents, indicators and observables. An overview of the achitecture of STIX is shown in Figure 2.1.

Our research relates to the classes *Incident* and *Observable*. The observables contain concrete information that satisfy our definition of a threat from the previous section. These observables are expressed in the language CyBOX and are also known as Indicators of Compromise (IoCs).

If observables are actually observed by an organization this is usually called an event, hit or incident. Such an incident can be enriched with additional information that is relevant. Examples are the involved host(s), packet trace during the hit and software versions.

Below is an example of STIX describing a small portion of the Poison Ivy malware family[6]. It illustrates the use of indicators, Tactics Techniques & Procedures (TTPs) and a course of action (COA).

```
<stix:Indicator id="fireeye:indicator-0036" xsi:type="indicator:IndicatorType">
  <indicator:Type xsi:type="stixVocabs:IndicatorTypeVocab-1.1">Domain Watchlist</
      indicator:Type>
  <indicator:Observable idref="fireeye:observable-915b"/>
  <indicator:Indicated_TTP>
    <stixCommon:TTP idref="fireeye:ttp-e55c6"/>
  </indicator:Indicated_TTP>
  <indicator:Suggested_COAs>
    <indicator:Suggested_COA>
      <stixCommon:Course_Of_Action idref="fireeye:courseofaction-70b3d"/>
    </indicator:Suggested_COA>
  </indicator:Suggested_COAs>
</stix:Indicator>

<stix:TTP id="fireeye:ttp-e55c6" xsi:type="ttp:TTPType">
  <ttp:Title>PIVY Variant (e74d62dfdc308df3038e61dfc4e4256)</ttp:Title>
  <ttp:Behavior>
    <ttp:Malware>
      <ttp:Malware_Instance>
        <ttp:Name>e74d62dfdc308df3038e61dfc4e4256</ttp:Name>
      </ttp:Malware_Instance>
    </ttp:Malware>
  </ttp:Behavior>
</stix:TTP>

<stix:Courses_Of_Action>
  <stix:Course_Of_Action id="fireeye:courseofaction-70b3d" xsi:type="coa:
      CourseOfActionType">
    <coa:Title>Analyze with FireEye Calamine Toolset</coa:Title>
    <coa:Description>Calamine is a set of free tools to help organizations detect and
        examine Poison Ivy infections on their systems.
    </coa:Description>
  </stix:Course_Of_Action>
</stix:Courses_Of_Action>
```

[6] https://stixproject.github.io/examples/

2.1.2 Cyber Observable eXpression (CyBOX)

Cyber Observable eXpression (CyBOX) is an XML-based language to describe measurable events and stateful properties that can be observed with host- and network-based intrusion detection systems. It is a flexible language that can describe a wide range of properties and events. In this research we call these stateful properties and observable events *information types*. For this research we are only interested in the information types that are useful for network- and host-based intrusion detection.

Examples of information types useful for network-based intrusion detection are:

- an IPv4 address,
- an IPv6 address,
- an IP address + port,
- a URL,
- a domain name,
- an e-mail address,
- raw packet data, or
- an HTTP header.

This list is not nearly exhaustive, but it gives an impression of the sort of information that is useful for threat detection. The information types are not very strictly defined, but rather serve to illustrate the kinds of information that is used.

For host-based intrusion detection common information types are:

- the contents of a file,
- a filename,
- a Windows registery key/value, or
- a Windows service.

For a complete overview of the supported types we refer to the CyBOX documentation[7].

[7] http://cybox.mitre.org/language/version2.1/xsddocs/cybox_default_vocabularies.html

Below is a snippet of CyBOX describing two observables from the Poison Ivy malware family:

```
<cybox:Observable id="fireeye:observable-915b">
  <cybox:Object id="fireeye:object-e911c">
    <cybox:Properties type="FQDN" xsi:type="DomainNameObj:
        DomainNameObjectType">
      <DomainNameObj:Value condition="Equals">microsoftupdate.ns01.biz</
          DomainNameObj:Value>
    </cybox:Properties>
    <cybox:Related_Objects>
      <cybox:Related_Object idref="fireeye:object-1a32">
        <cybox:Relationship xsi:type="cyboxVocabs:ObjectRelationshipVocab-1.0">
            Resolved_To</cybox:Relationship>
      </cybox:Related_Object>
    </cybox:Related_Objects>
  </cybox:Object>
</cybox:Observable>

<cybox:Observable id="fireeye:observable-4354f">
  <cybox:Object id="fireeye:object-1a32e">
    <cybox:Properties category="ipv4-addr" xsi:type="AddressObj:AddressObjectType"
        >
      <AddressObj:Address_Value condition="Equals">180.210.206.240</AddressObj:
          Address_Value>
    </cybox:Properties>
  </cybox:Object>
</cybox:Observable>
```

2.1.3 Trusted Automated eXchange of Indicator Information (TAXII)

Trusted Automated eXchange of Indicator Information (TAXII) is an XML-based standard to exchange threat information between organizations. The standard defines a number of protocols and services that can be used to exchange STIX-formatted information. As the name implies it assumes this information is exchanged between trusted parties. In this research we are looking to lower the required level of trust.

We can distinguish three different roles in a threat information exchange. New threat information orginates from a *source*. A *distributor* collects and optionally filters this information and makes it available to the *consumers*.

In practice, parties often have multiple roles. Common dual roles are *source* and *distributor*, and *source* and *consumer*. TAXII accomodates the most common sharing models for threat information. In this research we assume the *Hub-and-Spoke* model, as this is the most common model used to exchange information between larger organizations. The other models are discussed here because they are also used in practice and technical methods proposed in this research mostly fit those models als well.

Source-subscriber In the source-subscriber model the information flows in only one direction, from the distributor to the consumers. This is a common model for commercial vendors, for example vendors of traditional Anti-Virus products. Other examples are (semi-) open repositories.

Hub-and-spoke In the hub-and-spoke model, the hub collects and distributes the threat information. The hub often analyses, processes and anonymizes the information. This hub can be build around software like MISP or Soltra Edge and is usually run by some authorative entity. Examples are CERTs, Information Sharing and Analysis Centers (ISACs) or commercial companies.

In addition to receiving information, the spokes contribute information back to the hub. Examples are statistics about threats they have seen, and new threats they have detected. This can be ready-to-use IoCs, or "raw" data that can be used to refine IoCs. Figure 1.1 of the previous chapter illustrates this model.

Peer-to-peer In the peer-to-peer model the participants share the information directly with each other as equal peers. There is no single data owner.

Below is an example of a TAXII service discovery request and response[8]. It illustrates how TAXII defines a number of services to exchange STIX formatted messages.

```
POST http://taxiitest.mitre.org/services/discovery/ HTTP/1.1
Host: taxiitest.mitre.org
Proxy-Connection: keep-alive
Content-Length: 97
X-TAXII-Content-Type: urn:taxii.mitre.org:message:xml:1.1
X-TAXII-Accept: urn:taxii.mitre.org:message:xml:1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/39.0.2171.95 Safari/537.36
Content-Type: application/xml
Accept: application/xml
Cache-Control: no-cache
X-TAXII-Services: urn:taxii.mitre.org:services:1.1
X-TAXII-Protocol: urn:taxii.mitre.org:protocol:http:1.0
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

<Discovery_Request xmlns="http://taxii.mitre.org/messages/taxii_xml_binding-1.1"
    message_id="1"/>
```

And the response:

```
HTTP/1.1 200 OK
Date: Fri, 19 Dec 2014 13:22:04 GMT
Server: Apache/2.2.15 (Red Hat)
X-TAXII-Protocol: urn:taxii.mitre.org:protocol:http:1.0
```

[8] https://taxiiproject.github.io/documentation/sample-use/

```
X-TAXII-Content-Type: urn:taxii.mitre.org:message:xml:1.1
X-TAXII-Services: urn:taxii.mitre.org:services:1.1
Content-Type: application/xml
Transfer-Encoding: chunked
Connection: keep-alive
Proxy-Connection: keep-alive

<taxii_11:Discovery_Response xmlns:taxii_11="http://taxii.mitre.org/messages/
      taxii_xml_binding-1.1"
  message_id="1466" in_response_to="1">
  <taxii_11:Service_Instance service_type="INBOX"
    service_version="urn:taxii.mitre.org:services:1.1" available="true">
    <taxii_11:Protocol_Binding>urn:taxii.mitre.org:protocol:http:1.0</taxii_11:
        Protocol_Binding>
    <taxii_11:Address>http://taxiitest.mitre.org/services/inbox/default</taxii_11:
        Address>
    <taxii_11:Message_Binding>urn:taxii.mitre.org:message:xml:1.1</taxii_11:
        Message_Binding>
    <taxii_11:Content_Binding binding_id="urn:stix.mitre.org:xml:1.0"/>
  </taxii_11:Service_Instance>
  <taxii_11:Service_Instance service_type="POLL"
    service_version="urn:taxii.mitre.org:services:1.1" available="true">
    <taxii_11:Protocol_Binding>urn:taxii.mitre.org:protocol:http:1.0</taxii_11:
        Protocol_Binding>
    <taxii_11:Address>http://taxiitest.mitre.org/services/poll</taxii_11:Address>
    <taxii_11:Message_Binding>urn:taxii.mitre.org:message:xml:1.1</taxii_11:
        Message_Binding>
  </taxii_11:Service_Instance>
  <taxii_11:Service_Instance service_type="DISCOVERY"
    service_version="urn:taxii.mitre.org:services:1.1" available="true">
    <taxii_11:Protocol_Binding>urn:taxii.mitre.org:protocol:http:1.0</taxii_11:
        Protocol_Binding>
    <taxii_11:Address>http://taxiitest.mitre.org/services/discovery</taxii_11:Address>
    <taxii_11:Message_Binding>urn:taxii.mitre.org:message:xml:1.1</taxii_11:
        Message_Binding>
  </taxii_11:Service_Instance>
</taxii_11:Discovery_Response>
```

2.2 Information flows

In the previous section we have described a concrete model of threat information exchange based on a number of standards. In this section we identify the three main information flows in threat information exchange, *Indicators of Compromise*, *Hit Context* and *Statistics*. We also describe some additional parts present in a threat information exchange which we need later in this research to investigate privacy and confidentiality issues.

2.2.1 Indicators of Compromise

Indicators of Compromise are used as signatures in signature based intrusion detection. In the model of the previous section these indicators are described in the CyBOX language.

Detection is done by comparing (or matching) a signature to information observed in a network or on a host. There are many types of information that are useful for intrusion detection. We have decribed the common information types for host and network based intrusion detection in Subsection 2.1.2. The IoCs flow from distributors to consumers.

Matching methods

For our research it is important to understand how the IoCs are used in intrusion detection systems. In this section we describe four methods of "detecting" or "matching" that are commonly supported in intrusion detection systems. These matching methods match the information from an IoC to information observed in the network or on the host that is currently being inspected. In this context the IoC is more commonly known as a *signature*. The matching happens at a *detector*.

String-based The information types described in Subsection 2.1.2 can be seen as simple strings or byte arrays. If the signature and the observed information are of the same information type, this allows for simple comparison operations.

The most common operations are:

- **Equals**, a piece of observed information is equal to the signature,
- **StartsWith**, the observed information begins with the signature,
- **Contains**, the signature is somewhere in the piece of observed information, and
- **InclusiveBetween**, the observed information falls within a range of possible values (e.g IP address falls in a range).

Regular Expression-based A regular expression is used to do pattern matching on strings. The pattern can include, amongst other constructs, repetitions. It can also match characters from a given subset. This makes it more powerful than the string operations mentioned previously. Regular expressions can be applied to all the information types, as long as they can be represented as strings. In this case the regular expression is the signature.

Hash based A hash function is a function that maps inputs of arbitrary length to outputs of a fixed length. Hash-based detection methods can be seen as similiar to the *Equals* operation of the string-based detection method. The difference is that the signature is not a string-like object, instead it is the hash of the original string-like object. Commonly used hash functions are *MD5* and *SHA-256*.

For inputs longer than the length of the output of the hash function, e.g. files, using this method results in shorter signatures. This is one of the main reasons this detection method is used in practice. It can also be used as a confidentiality mechanism. This aspect is discussed in Section 3.3.1.

Bloom filters A Bloom filter is a space- and time-efficient data structure to test inclusion in a set of elements. Elements can be added to the set and tested for membership, but not removed. Because of its probabilistic nature, false positives are possible when testing for membership, but not false negatives. A more in depth description can be found in Subsection 4.2.1.

A Bloom filter can be used as a detection method. To do this, create a Bloom filter with some elements as a signature and check current information for membership in the signature. The main reason to use a Bloom filter is the generally smaller memory size it needs to represent a set compared to other data structures. It can also be used as a confidentiality mechanism. This is discussed in Subsection 3.3.2.

2.2.2 Hit context

If a matching engine finds a match between a signature and some current piece of information this is called a *hit*. The context of this hit can be useful for determining wheter the hit is a legitimate occurence of the threat it is trying to detect or a false positive. In case of a false positive the context can contain useful information for refining the signature such that it results in fewer false positives in the future. In case of a true positive the context information can be used to better understand the threat.

Possible relevant context information:

- the time of occurence,
- the hosts involved (IP address and/or MAC address), or
- a packet trace surrounding the hit.

2.2.3 Statistics

In case of the *Source-Subscriber* or *Hub-and-Spoke* models of threat information exchange, it can be useful for the *source* or *hub* (hereafter referred to as the *hub*) to have statistics regarding to the threat information they share. These statistics enable the *hub* to get a broader view of the threat landscape. A concrete example of this is the *Cybersecuritybeeld Nederland* (*Cyber Security Assessment Netherlands*)[9], which explains and interprets trends in cybersecurity on a national level.

Statistics that can be useful include:

- (unique) hits per IoC,
- hits per organization,
- hits per IoC in a group of organizations, or
- occurences per type of malware.

In the threat information exchange model we described in the previous section this flow of information is not explicitly present. Instead, these statististics can be computed from the hit context. By seperating these two flows here, we create the possibility for a spoke to contribute to aggregated statistics without revealing hit context. We consider it a realistic scenario that a spoke is willing to contribute to statistics, but does not want to reveal hit context information.

[9] https://www.ncsc.nl/english/current-topics/cyber-security-assessment-netherlands.html

3. Confidentiality in existing platforms

In the previous chapter we have discussed the information flows within a threat information exchange. We saw that Indicators of Compromise flow from the hub to the detector, and that hit context information and statistics flow from the detector to the hub. In this chapter we will examine which of those information flows may need to be kept confidential by one of the parties. We also describe for each information flow which confidentiality preserving methods and features are already present in existing platforms. In the following three chapters we will describe new ideas for protecting the information in each of those flows.

3.1 Criteria

To determine whether an information flow needs confidentialty we need criteria to classify it as sensitive information. For this classification we take an organizational point of view, as most threat information exchange takes place between organizations. We note that there are three categories of reasons: legal reasons, organizational reasons and technical reasons.

Legal reason Privacy and data protection laws usually put requirements on handlers of certain data. For example, in the case of personal identifiable information in the EU, a processor of that data has under the EU Data Protection Directive (Directive 95/46/EC) an obligation to keep that information secure from any potential abuse. For this thesis we consider IP and MAC addresses to be personal identifiable information. Although that is not always the case, there are enough situations where for example an IP address can linked to a individual person that we consider it always personal identifiable information.

Organizational reason Organizations usually keep proprietary information such as trade secrets and financial data. An organization typically requires that this information is kept confidential.

Technical reason Information that should be kept confidential for technical reasons is for example information about network configuration, software ver-

sions and secret values such as passwords or access tokens. This information gives away clues about the configuration of the network, and by extension, the organization.

To summarize, we have three main categories of information that have confidentiality requirements:

- personal identifiable information,
- proprietary information such as trade secrets, and
- technical information such as network configuration data and passwords.

3.2 Confidentiality levels

To assess how well protected a piece of information is we make a distinction between three levels of confidentality. This distinction is useful when considering schemes with two parties in the semi-honest adversary environment. The first party A is the distributor of the information and party B is the receiver. We assume A wants to keep the information confidential from B, while still enabling him to do something useful with the information. For example network detection.

Not Confidential In a non-confidential setting B can determine the information without significant effort. Protection methods that can be broken by a sufficiently knowledgable and powerful (but computationally bounded) adversary are also considered to be in this category. For example, the hash of a poorly chosen password.

Conditionally Confidential The information can not be determined by B unless a certain (rare) condition is met.

For example, a conditionally confidential IoC is secure against bruteforce attacks, but is still useful for intrusion detection. However, if a hit is detected the confidentiality is broken, hence "conditional" confidentiality. Another example is the hash of a well chosen password. It is not possible to recover the password, unless (a substantial part of) the password becomes known.

Strongly Confidential The information can not be determined by B for any possible execution of the protocol, assuming the hardness of the problem underlying the protocol.

In the example of IoCs this requirement makes the signature useless for the detector. However, if the hub can determine if a hit has occured this requirement

can be met in a useful scenario. In the next chapter we present the a scheme that meets this requirement.

Let us now look at the confidentiality requirements of the flow of information from the hub to the detector (IoCs, Section 3.3) and from the detector back to the hub (statistics, Section 3.5 and hit context, Section 3.4).

3.3 Indicators

In the previous chapter we have given a overview of typical information types found in IoCs for network based and host based intrusion detection systems.

The flow of IoCs is from the hub to the detector. This implies that we only need to consider confidentiality requirements from the hub, and not the detector.

The hub can have confidentiality requirements for the information contained in an IoC. For example, if a particular IoC detects a specific threat, and the actor behind the threat can obtain the IoC, it can alter its threat so that it is no longer detected, making the IoC essentially useless. If the IoC can be kept confidential this problem is reduced or eliminated.

The hub can distribute IoCs based on information from third parties. If the third party requires the information to be confidential (for any reason), this is a motivation for the hub to keep it confidential as well.

As noted in Subsection 2.2.1 there are two confidentiality preserving methods already in use for IoCs: hashes and Bloom filters. The confidentiality preserving proprerties of these methods will be discussed in the next sections.

3.3.1 Hash based

As noted in Subsection 2.2.1 a primary reason to use a hash based detection method is the convenience of shorter signatures. It can also be used as a confidentiality mechanism. For example the network based intrusion detection system Snort gained support for this with the keyword `protected_content`[1] since version 2.9.7[2], released in October 2014. For host based intrusion detection systems this is a common functionality for detecting malicious files.

The security of this method depends on the strength of the particular hash function used and the domain and probability distribution of the values of the input. The most important property the hash function should satisfy is *pre-image resistance*: given only the hash of a value it should be infeasible to recover an input value with the same hash [18].

[1] http://manual.snort.org/node32.html#SECTION00452000000000000000
[2] http://blog.snort.org/2014/10/snort-297-has-been-released.html

The *domain* is the set of all possible inputs. If the domain is sufficiently large, it is computationaly too expensive to try all possible inputs. To be on the safe side, we consider 2^{128} possibilities to be safe, also noted as 128 bit security [7].

The effective size of the domain and distribution can not always be easily determined. If the attacker can make an assumption about the input it can reduce the size of the domain, or search it more efficiently. For example, at first glance IPv6 has 2^{128} possible addresses. However, most of the address space in not yet allocated for use, reducing the effective size of domain. Another example is a 16 character password. At first glance it has 94^{16}, or about 2^{104} possibilities. However, if we can assume it is made up from English dictionary words, there are far less posibilities.

Of some of the information types mentioned in Subsection 2.1.2 we can determine that the domain is (far) smaller than 2^{128}. For IPv4 addresses the domain is (at most) 2^{32} and for a IPv4 + port number it is 2^{48}. Domain names are a bit trickier. If an attacker has a list of domain names an exhaustive search is feasible. In januari 2015 Verisign reported there are 284 million registrered domain names globally[3]. State actors can probably obtain such lists. Also a dictionary attack may reveal partial information, as many domain names contain common words.

Given the confidentiality levels of Section 3.2, we consider this method allows for conditionally confidentiality because the confidentiality of the input of the hash is broken when a hit occurs.

3.3.2 Bloom filters

The security properties of the Bloom filter based detection method are very similar to those of the hash based method. From a attackers perspective, a Bloom filter can be seen as a collection of hashes. To make it hard to determine the elements of a Bloom filter the hash function used should satisfy the "pre-image" resistance property. The considerations about domain size of the input are equally applicable to the elements of the Bloom filter.

Similar to hash based methods, we consider this method to be conditionally secure because the confidentiality of the information that a certain element is in the set is broken when a hit is produced.

[3] https://www.verisigninc.com/assets/domain-name-report-january2015.pdf

3.4 Hit context

The hit context can contain a lot of sensitive information. IP addresses for example are considered personal identifiable information[4]. Packet traces can contain proprietary information and both usually contain (technical) information about the network configuration.

3.4.1 Hosts involved

If a hit occurs and this information is shared with the hub, sharing the internal IP or MAC address of the host as well is useful. This allows the hub to correlate hits on IoCs to hosts, leading to a better understanding of the situation. It also allows the hub to distinguish between a single host being hit a hundred times, indicating a single malware infection, and a hundred hosts being hit a single time, indicating a hundred malware infections.

However, we consider IP or MAC address as personal identifiable information that can also give away clues about the internal network. This makes it sensitive information.

A common practice is to use pseudonyms for the hosts instead. This still permits the correlation between hosts and IoCs, but prevents the sensitive information from being leaked to the hub. To prevent correlation over larger periods of time, the pseudonyms should be refreshed at a certain interval (e.g. a week). These pseudonyms can be generated with a pseudorandom function.

Given the confidentiality levels of Section 3.2 we consider this method to yield strong confidentiality because the real identity is never revealed to the hub, not even in the case of a hit.

3.4.2 Packet traces

Packet traces from the hit and the surrounding packets are likely to contain sensitive information. The packets contain IP addresses, which we consider to be personal identifiable information. The application-specific payload data can also contain virtually any kind of information. This makes a packet trace certainly sensitive information.

Snort has a preprocessor that can filter known sensitive information[5]. This method requires that all sensitive data is known in advance, which is likely not the case.

[4] http://www.europarl.europa.eu/sides/getAllAnswers.do?reference=P-2013-000873&language=EN

[5] http://manual.snort.org/node17.html#SECTION00321700000000000000

Other tools exist that can filter packet traces to various degrees. Examples are TraceWrangler[6], tcprewrite[7] and tcpmkpub[8]. The first is capable of filtering both the header and the payload of a packet, while the others only consider the header information.

We cannot determine which level of confidentiality these methods provide, given the descriptions in Section 3.2. It is not clear under which circumstances which information is disclosed. If sensitive information is removed from the traces we can consider it strongly confidential. However, if the information is replaced (e.g. pseudonymization of IP addresses) careful analysis can probably still reveal information.

3.5 Statistics

The statistics that are mentioned in the previous chapter can certainly be considered sensitive information. Although they do not contain personal data, they might give clues about network configuration. For example, if a organisation has a hit on malware written specifically for certain software, this indicates they use that specific software.

We are not aware of any standards or software products that facilitate automated statistics gathering and have any privacy/confidentialty preserving properties.

To be privacy preserving, it is necessary to define statistics that reveal none or only little information about the underlying data. For example, percentages of infected systems instead of the total number, which gives information about the size of network, or generalizing metrics like the OS from “Microsoft Windows XP build 2600” to “Microsoft Windows”.

[6] https://www.tracewrangler.com/
[7] http://tcpreplay.synfin.net/wiki/tcprewrite
[8] http://www.icir.org/enterprise-tracing/tcpmkpub.html

4. Confidentiality of Indicators

In the previous chapter we have discussed two methods that are already used in practice to keep IoCs confidential: hashes and Bloom filters. Both methods satisfy the "Conditionally Confidential" property and can be used to replace the "Equals" and "StartsWith" functions of the string based matching method. However, both are inefficient for achieving the "Contains" functions because it is not known in advance where the signature is in the string.

In this section we describe the Adapted Rabin-Karp method that can perform the "Contains" function in an efficient way in the "Conditionally Confidential" setting.

We also present the Public-key Encrypted Bloom Filter method that can perform the string based "Equals" method in a "Strongly Confidential" setting.

4.1 Adapted Rabin-Karp pattern matching

In this section we present an efficient method to perform pattern matching with multiple patterns in a "Conditionally Confidential" setting. This method uses Rabin fingerprints and has been proposed in slightly different forms in multiple papers [20][2]. The main improvement of this scheme over a plain Bloom filter or hash function as discussed in Section 3.3.2 is efficiency. With this scheme an intrusion detection system can efficiently check for signatures in data where the position of the hits is not known.

We first discuss the Rabin fingerprint method and Rabin-Karp pattern matching. We then adapt the Rabin-Karp method to be able to match arbitrary length patterns. The last step is to split the algorithm in two parts to make it applicable to a threat information exchange environment and to gain "Conditionally Confidential" security. The first part is executed by the hub and the second by the detector.

4.1.1 Rabin-fingerprint

A Rabin fingerprint [17] is a function that behaves similarly to a hash function. It produces a fixed length fingerprint of a string of a (larger) fixed length. To do this it uses polynomials over a finite field. The fingerprint F for string $s = s_0, \ldots s_{n-1}$, where $s_i \in \mathbb{F}_{2^8}$, which can be interpreted as a single byte, is defined as:

$$F(s) = s_0 \cdot x^{n-1} + s_1 \cdot x^{n-2} + \cdots + s_{n-1} \cdot x^1 + s_n \bmod p(x),$$

where x is a prime and $p(x)$ is a irreducible polynomial of degree k. The degree k determines the length of the fingerprint, and has influence on the overall security of the method.

The property that makes the Rabin fingerprint interesting for pattern matching is that we can efficiently compute $f_1 = F(s_1, \ldots, s_{i+1})$ given the fingerprint $f_0 = F(s_0, \ldots, s_i)$ and s_0 and s_{i+1}:

$$f_{i+1} = f_i \cdot x + s_{i+1} - s_0 \cdot x^{n-1} \bmod p(x).$$

With this property we can efficiently compute the fingerprint of a sliding window in a large string. Since there are only 256 possible values for s_0, the value of $s_0 \cdot x^{n-1}$ can be precomputed and stored in a table.

4.1.2 Rabin-Karp pattern matching

To construct a useful matching method we need to be able to match multiple pattern strings of arbitrary length. The Rabin-Karp[11] algorithm can match multiple patterns of the same length. We use this algorithm as the basis for the final matching method.

First the set H of all Rabin fingerprints of the patterns is computed. Then the Rabin fingerprint f is computed over the input. The length of the sliding window is set to the length of the patterns. At each step of the sliding window a check is done whether the current fingerprint is in P. If this is the case a match is found. In Algorithm 1 this method is presented in pseudocode. Note that the pseudocode implementation stops after the first match. In practice the matching engine should probably continue, in order to find all occurrences.

To be able to match patterns of different length we slightly modify the Rabin-Karp algorithm. We do this by splitting up the patterns in smaller fragments. We set the length of the sliding window m to the length of the smallest pattern. All patterns larger than the sliding window are fragmented into patterns of length m. If the pattern is not a muliple of m the last two fragments overlap, as illustrated in Figure 4.1. The matching algorithm should look for fragments and

Algorithm 1 Rabin Karp String Matching

```
1: procedure RabinKarp(S[1...n], P, m)   ▷ Match patterns P of length m with string S
2:     H ← ∅                                          ▷ Initialize H as empty set
3:     for all x ∈ P do
4:         H ← H ∪ {fp(x)}                            ▷ Add fingerprint of x to H
5:     end for
6:     f ← fp(S[1...m])                ▷ f is the fingerprint of the sliding window
7:     for i ← 0 to n − m + 1 do
8:         if f ∈ H and S[i...i + m − 1] ∈ P then
9:             return i
10:        end if
11:        f ← update_fp(f, S[i], S[i + m])           ▷ Advance sliding window
12:    end for
13:    return notfound
14: end procedure
```

keep track of consecutive matches. To check if the consecutive matches indeed form one of the original pattern, the algorithm should check each possible sub pattern against the original set of patterns. Algorithm 2 details this.

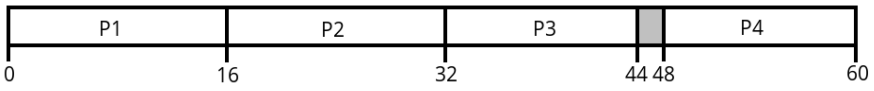

Figure 4.1: Fragmentation of a 60 byte pattern into 4 fragments of 16 bytes

To satisfy the "Conditionally Confidential" property we split the function in two parts. The first part, *Setup*, is executed by the hub and creates the patterns for the detector. The *Match* function is executed by the detector and takes as input the patterns generated by the hub and the string to be matched.

The *Setup* function computes the set H of fingerprints of the fragments, as well as a set P' of hashes of the patterns to replace the H in the original Rabin-Karp. We use *fp* to denote the Rabin fingerprint and *hash* to denote a different cryptographically secure hash. In Algorithm 3 this is implemented in pseudocode.

Algorithm 2 Adapted Rabin Karp String Matching

```
1: procedure RabinKarp(S[1...n], P, m)▷ Match patterns P with string S with fragment size m
2:     H ← ∅                                                    ▷ Initialize H as empty set
3:     for all x ∈ P do
4:         for i ← m to |x| increment i ← i + m do
5:             H ← H ∪ {fp(x[(i − m) + 1...i])}                 ▷ Add fragment to H
6:         end for
7:         H ← H ∪ {fp(x[(|x| − m) + 1...|x|])}                 ▷ Add last fragment to H
8:     end for
9:     F ← ∅                                                    ▷ Set of matching fragments
10:    f ← fp(S[1...m])                          ▷ f is the fingerprint of the sliding window
11:    for i ← 0 to n − m + 1 do
12:        if F ≠ ∅ and F[|F|] + (m − 1) < i then   ▷ F[|F|] denotes the last element of F
13:                                  ▷ Check if any substring of partial matches is a real match
14:            for j ← 1 to |F| do
15:                for k ← j to |F| do
16:                    if S[F[j]...F[k] + (m − 1)] ∈ P then
17:                        return F[j], F[k] + (m − 1)
18:                    end if
19:                end for
20:            end for
21:            F ← ∅                                            ▷ Clear matching fragments
22:        end if
23:        if f ∈ H then
24:            F ← F ∪ {i}                                      ▷ Add matching fragment to F
25:        end if
26:        f ← update_fp(f, S[i], S[i + m])                     ▷ Advance sliding window
27:    end for
28:    return notfound
29: end procedure
```

Algorithm 3 Split Matching

```
1:  procedure SETUP(P, m)  ▷ Setup fragments for patterns P with fragment length m
2:      P' ← ∅                                          ▷ Initialize P' as empty set
3:      H ← ∅                                           ▷ Initialize H as empty set
4:      for all x ∈ P do
5:          for i ← m to |x| increment i ← i + m do
6:              H ← H ∪ {fp(x[(i − m) + 1 ... i])}      ▷ Add fragment to H
7:          end for
8:          H ← H ∪ {fp(x[(|x| − m) + 1 ... |x|])}      ▷ Add last fragment to H
9:          P' ← P' ∪ {hash(x)}                         ▷ Add hash of complete pattern to P'
10:     end for
11:     return P', H
12: end procedure
13:
14: procedure MATCH(S[1 ... n], P', H, m)▷ Match (hashed) patterns P' with string S using fragments H of length m
15:     F ← ∅                                           ▷ List of matching fragments
16:     f ← fp(S[1 ... m])                              ▷ f is the fingerprint of the sliding window
17:     for i ← 0 to n − m + 1 do
18:         if F ≠ ∅ and F[|F|] + (m − 1) < i then
19:                 ▷ Check if any substring of partial matches is a real match
20:             for j ← 1 to |F| do
21:                 for k ← j to |F| do
22:                     if hash(S[F[j] ... F[k] + (m − 1)]) ∈ P' then
23:                         return F[j], F[k] + (m − 1)
24:                     end if
25:                 end for
26:             end for
27:             F ← ∅                                   ▷ Clear matching fragments
28:         end if
29:         if f ∈ H then
30:             F ← F ∪ {i}                             ▷ Add matching fragment to F
31:         end if
32:         f ← update_fp(f, S[i], S[i + m])            ▷ Advance sliding window
33:     end for
34:     return notfound
35: end procedure
```

4.1.3 Correctness

Compared to the original Rabin-Karp pattern matching algorithm we have added two parts. First we added fragmentation of patterns to facilitate matching patterns of different lengths. This adds the possibility of false positives. A

string can contain fragments of a pattern without containing the whole pattern. In case of a single matching fragment, false positives are discarded. A second possibility is a false positive in series of consecutive matching fragments. To find all true matches, and only those, all possible consecutive fragments are checked to be an actual pattern (see Algorithm 3 line 20-27). Due to the nature of the Rabin fingerprint it is also possible a fragment matches a string other than its original pattern. These hash collisions are also discarded.

Secondly we added hashes instead of the original patterns. When a cryptographically secure hash function is used, such as *SHA-256*, this results in a negligibly small chance of hash collisions, and thus of false positives.

Because we handle all deviations of the original algorithm, we assert the adapted algorithm is correct.

4.1.4 Performance

We will now look at the runtime performance of both the *Setup* and *Match* functions.

The *Setup* function is executed by the hub to generate the fragments. A single run of the algorithm has a runtime of $\mathcal{O}(n+m)$ where n is the number of patterns and m the sum of the lengths of all patterns. How many times the function is executed depends on how often the set of patterns changes. We estimate this to be somewhere between every hour and every day. Given that hashes can be computed very quickly we consider this practical.

The overall performance of the *Match* algorithm depends on the likelihood of matching a fragment. In the worst case scenario the matching fragments cover the full string, resulting in a runtime of $\mathcal{O}(l^2)$ where l is the length of the string. In the best case scenario there are no matching fragments, resulting in a runtime of $\mathcal{O}(l)$.

We estimate that in a real world scenario the runtime will be close to $\mathcal{O}(l)$ because hits are rare, if the patterns and fragments are chosen properly. Fragments that are abundant should be avoided (e.g, the fragment `GET / HTTP/1.1\r\n` would match quite often when network traffic is considered).

To further improve the performance of the algorithm we propose to use a Bloom filter for the set of fragments H. By using the fingerprint f as the hash for determining the indices, half the work is already done for the member test of a Bloom filter. To obtain k hashes for the index lookups, one could split f if it is large enough or compute multiple fingerprints $f_0 \dots f_{k-1}$ with different irreducible polynomials.

With a Bloom filter it is possible to have false positives for the membership test $f \in H$. Fortunately, the algorithm deals with those because it checks all possible substrings of fragments.

4.1.5 Security

We now investigate if the plaintext patterns can be recovered given the set of hashes of the patterns P' and the set of fragments H. We discuss both preimage resistance and brute force attacks.

We assume the hash function used to obtain the hashes in P' is cryptographically secure, thus it is preimage resistant. The patterns should be at least 16 bytes long to provide 128 bit security, assuming the patterns are not predictable.

The preimage resistance of the fragments in H depends on the Bloom filter. Given only the Bloom filter it is not possible to recover the elements.

To prevent brute force attacks on the fragments the length of the fragments should be at least 16 bytes to provide 128 bit security, under the assumption that the pattern is not predictable. This means we cannot match patterns smaller than 16 bytes. If the pattern is somehow predictable, for example if the attacker knows it only consists of printable ASCII characters, the window should be larger. How much larger depends on the predictability of the pattern. In case of printable ASCII characters, there are only 95 possible characters per byte instead of 256, resulting in 6.6 bits entropy per byte. The fragment size should then be at least 20 bytes to provide 128 bit security. Packets that contain a lot of predictable content (e.g. XML documents) may well require a length of hundreds of bytes.

Another consideration is the fragmentation of the pattern. If the whole pattern contains enough entropy to be secure, but the entropy is not distributed evenly in the pattern, some fragments of the pattern might be easily predictable. In a worst case screnario this can lead to a recovery of the whole pattern. Because it is possible for the fragments to overlap at arbitrary places, there is room for mitigation strategies. In conclusion, the pattern and fragments should be carefully chosen to prevent bruteforce attacks on the individual fragments. Intimate knowledge is necessary to make a correct estimate of the required pattern length for a given signature.

4.2 Public-key Encrypted Bloom Filters

In this section we describe a scheme [13] by Kerschbaum which can be used to satisfy the "Strongly Confidential" property described in the previous chapter. The goal is to perform a set-intersection such that the *hub* only learns the intersection, and the *detector* learns nothing about the intersection. Both should learn nothing about each others set.

The scheme uses three building blocks to accomplish its goal: Bloom filters, Goldwasser-Micali encryption and the method of Sander, Young and Yung for

computing the AND function. This scheme assumes a semi-honest adversary. We will first sketch a rough outline of the scheme, then discuss the individual building blocks and finally discuss the scheme in more detail.

The *hub* has a set of indicators and the *detector* has a set of observed elements of the same type. The hub then creates a Bloom filter of its set and uses Goldwasser-Micali encryption to encrypt it. This encrypted Bloom filter is transmitted to the detector together with the public key.

The detector also creates a Bloom filter of its observed elements and encrypts it with the public key. It then computes the AND function of both encrypted Bloom filters. This is then transmitted to the hub. The hub then decrypts the intermediate result and obtains the set-intersection.

We will now describe the building blocks in more detail.

4.2.1 Bloom filter

A Bloom filter [1] is a probabilistic data structure that allows one to efficiently test an element for membership of a set and add an element to a set. It is space and time efficient and has a chance of false positives, but not false negatives.

An empty Bloom filter b is a array of m bits set to 0 and k hash functions with $0 < k < m$. The hash functions f_i are independent and have a range of 0 to $m - 1$:

$$f_i : \{0,1\}^* \to \{0, \ldots, m-1\}$$

To add an element x to the Bloom filter the element is hashed with all hash functions f_i and the corresponding k bits at indices $l_i = f_i(x)$ are set to 1.

To check if an element is in the Bloom filter the element is hashed with all hash functions f_i and if all corresponding k bits at indices l_i are 1 then the element is reported to be in the set.

This test can yield a false positive, but not a false negative. The chance of false positives increases with the number of elements in the set, but decreases with the size of the Bloom filter. To compute the required size of a Bloom filter to achieve a given false positive rate we can use the formula:

$$m = -\frac{n \log(p)}{\log(2)},$$

where m is the size of the Bloom filter, p is the desired false positive rate and n the number of elements.

4.2.2 Goldwasser-Micali encryption

The encryption used is Goldwasser-Micali (GM) [10], which is a homomorphic randomized public-key encryption scheme. In GM only one bit at a time can be encrypted.

GM uses quadratic residuosity to encode a bit. A quadratic residue $r \in \mathbb{Z}_n$ is a number such that there exists a number s: $r = s^2 \bmod n$. A number that isnt a quadratic residue is called a quadratic non-residue. In GM a 0 is encoded as a quadratic residue and a 1 is encoded as a quadratic non-residue. The public key is n, v where $n = pq$ where p and q are large primes. v is a quadratic non-residue modulo p and a quadratic non-residue modulo q.

To encrypt a 0, one chooses a random $r \in \mathbb{Z}_n$ and computes $r^2 \bmod n$. To encrypt a 1, one chooses a random $r \in \mathbb{Z}_n$ and computes $vr^2 \bmod n$. To decrypt a value one needs to check if the value is a quadratic residue. Differentiating between a quadratic residue (0) and a quadratic non-residue (1) implies knowledge of the factorization of n [10] [13].

Because of the random r, two ciphertexts with the same plaintext are not distinguishable without knowing the private key, which makes the scheme semantically secure in the IND-CPA setting.

GM is homomorphic because the multiplication of two ciphertexts is equivalent to the encryption of the `XOR` of their plaintexts:

$$E(x) \cdot E(y) = E(x \oplus y).$$

This follows from the definition of 1 and 0 as given earlier. We note three different cases:

$$\begin{array}{lcllcll} E(0) \cdot E(0) & = & r_1^2 \cdot r_2^2 \bmod n & = & (r_1 \cdot r_2)^2 \bmod n & = & E(0) \\ E(0) \cdot E(1) & = & r_1^2 \cdot v \cdot r_2^2 \bmod n & = & v \cdot (r_1 \cdot r_2)^2 \bmod n & = & E(1) \\ E(1) \cdot E(1) & = & v \cdot r_1^2 \cdot v \cdot r_2^2 \bmod n & = & (v \cdot r_1 \cdot r_2)^2 \bmod n & = & E(0) \end{array}$$

Encrypting a Bloom filter is done by encrypting each individual bit of the Bloom filter using the public key. Note that the resulting ciphertext is substantially larger than the plaintext Bloom filter. If we assume p and q to be 1024 bits prime numbers, n is 2048 bits in size. Because each bit of the Bloom filter is encrypted individually, the resulting ciphertext is 2048 times the size of the plaintext Bloom filter.

4.2.3 Sander, Young and Yung method

Using the technique from Sander, Young and Young (SYY) [19] it is possible to do a single AND operation on two (single bit) ciphertexts. First the ciphertext

(a single bit) $\sigma = E(x)$ is expanded to the expanded ciphertext $\boldsymbol{\sigma}$, which is a vector of length u. This is done by repeating the following operation u times $(0 \leq i < u)$.

1. Flip a coin u times: $r_i \in \{0,1\}^u$
2. Compute ciphertext σ_i according to the random coin and set

$$\sigma_i \leftarrow E(e_i) = \begin{cases} E(x) \cdot E(1) = E(x \oplus 1) & \text{if } r_i = 0 \\ E(0) & \text{if } r_i = 1 \end{cases}$$

If $x = 1$ then $x \oplus 1 = 0$ and $e_i = 0$, so $\sigma_i \in \{E(0)\}$. If $x = 0$ then $x \oplus 1 = 1$ and $e_i = 1$, so the result is randomly distributed: $\sigma_i \in \{E(0), E(1)\}$.

To compute the logical AND of two GM encrypted bits we use the expanded ciphertext $\boldsymbol{\sigma}$ for $E(x)$ and the expanded ciphertext $\boldsymbol{\rho}$ for $E(y)$. The encrypted AND $\boldsymbol{\tau}$ is computed by pairwise multiplication of the elements of $\boldsymbol{\sigma}$ and $\boldsymbol{\rho}$, thus:

$$\begin{aligned} \tau_i &= \sigma_i \cdot \rho_i \\ &= E(e_i) \cdot E(d_i) \\ &= E(e_i \oplus d_i) \end{aligned}$$

The encrypted elements of $\boldsymbol{\tau}$ can be decrypted by the private key holder to determine the result of the AND function. We note two possible outcomes:

$$D(\tau) \leftarrow \begin{cases} D(\tau_i) \in \{0,1\} & \text{if } x \wedge y = 0 \\ D(\tau_i) \in \{0\} & \text{if } x \wedge y = 1 \end{cases}$$

If all decrypted bits are 0, the result of the AND function is 1, otherwise the result is 0. There is a probability of 2^{-u} that a false positive occurs, where $E(x \oplus y) = 0$ is falsely decrypted as a 1.

4.2.4 Final scheme

We assume two parties, the *hub* and the *detector*, and assume the semi-honest adversary.

The *hub* generates a public key n, v and private-key p, q and constructs a Bloom filter b_h containing elements that need to be detected by the *detector*. The size of the Bloom filter should be chosen large enough to accomodate the (likely larger) set of the *detector*.

The Bloom filter is encrypted: $EB_h = E(b_h)$ by encrypting each bit of the Bloom filter with the public key. The public key n, v and EB_h are then sent to the *detector*.

The *detector* constructs a Bloom filter b_d with all elements it has observed in some period of time, and encrypts it with the public key to obtain EB_d.

The *detector* then computes the AND of EB_h and EB_d by applying the SYY method to each pair of bits of the encrypted Bloom filters. This yields the combined Bloom filter EB_c, which contains the encrypted set intersection of b_h and b_d, this is then sent to the *hub*.

The *hub* decrypts all bits of EB_c to obtain b_c. If all bits of b_c are 0, the intersection of b_h and b_d is empty and no hit is registered. If b_c is non-zero all elements of b_h should be tested against b_c to check which element(s) both Bloom filters have in common, and thus caused a hit.

4.2.5 Security

In this scheme the *hub* should only learn the intersection of b_h and b_d and nothing else. The *detector* should learn nothing about b_h. The scheme presented is only secure in the semi-honest attacker model. In the malicious model the *hub* could send a Bloom filter with all bits set to 1 to obtain b_d from the *detector*. In the semi-honest model the *hub* is not allowed to do this.

However, under certain circumstances the *hub* can deduce some more information from b_c than the intersection. If the domain of the elements is small (e.g. IPv4 addresses) the server can deduce some elements that were definitly not observed by the detector. For each element e in the domain compute the indices $l_i (0 \leq i < k)$. If any of the indices l_i is 1 in b_h, but 0 in b_c the element was not in b_d. As as result the hub can determine of some elements the detector has not seen them. Of how many elements the hub can determine this depends on the size of the Bloom filter and the number of elements in b_h.

4.2.6 Practicality

To investigate the practicality of this scheme we will estimate the amount of data that needs to be transferred under some assumptions for the false positive rate and the number of elements seen by the detector.

We start by estimating the number of expected false positives. The false positive rate is influenced by two factors, the false positive rate p of the Bloom filter, and the size of the vector $\boldsymbol{\sigma}$, u, of the SYY method. As we are comparing two Bloom filters, we use n for the number of elements in the Bloom filter of the detector, and use m the number of elements in the Bloom filter of the hub.

We estimate the p and u parameters for a range of desired false positives per run and number of elements in the Bloom filter. The range we have chosen for the desired false positive rate is $\frac{1}{10}$ upto $\frac{1}{10000}$, meaning we expect one false positive per 10 runs of the protocol upto one false positive per 10000 runs.

As we have seen in Section 4.2.1, the size of a Bloom filter can be determined from the desired false positive rate and the number of elements inserted. The complete scheme also incorporates the SYY method to accomplish the intersection of two Bloom filters. This method has the possibility that a 0 is wrongly decrypted as a 1 with probability 2^{-u}. If this happens for all k hash functions an element is wrongly assumed to be in the Bloom filter. When both false positive rates are combined we can estimate the expected number of false positives per run f with

$$f = m \cdot (k \cdot 2^{-u} + p),$$

where m is the size of the smallest Bloom filter. k can be determined with

$$k = \frac{s}{n} \log(2).$$

s can be determined with

$$s = -\frac{n \log(p)}{\log(2)}.$$

We distribute the false positive rate equally between p and u. This distribution might be tweaked to optimize the final size of the ciphertext, but we omit this for simplicity. We determine p with

$$p = \frac{\frac{1}{2}f}{m},$$

and we determine u with

$$u = -\log\left(\frac{\frac{1}{2}f}{k \cdot m}\right).$$

Substituting k gives us

$$u = -\log\left(\frac{\frac{1}{2}f}{\frac{s \cdot m}{n}\log(2)}\right).$$

Substituting s gives us

$$u = -\log\left(-\frac{\frac{1}{2}f}{m\ \log(p)}\right).$$

Assuming $m = 100$ we get the table for p and u.

f \ n	10^2	10^3	10^4	10^5
10^{-1}	$5 * 10^{-4}$, 10	$5 * 10^{-4}$, 12	$5 * 10^{-4}$, 14	$5 * 10^{-4}$, 17
10^{-2}	$5 * 10^{-5}$, 12	$5 * 10^{-5}$, 15	$5 * 10^{-5}$, 17	$5 * 10^{-5}$, 19
10^{-3}	$5 * 10^{-6}$, 15	$5 * 10^{-6}$, 17	$5 * 10^{-6}$, 19	$5 * 10^{-6}$, 22
10^{-4}	$5 * 10^{-7}$, 17	$5 * 10^{-7}$, 19	$5 * 10^{-7}$, 22	$5 * 10^{-7}$, 24

The largest amount of data that needs to be transferred is the expanded ciphertext that is the result of the AND operation of both Bloom filters. The size of this ciphertext can be determined with

$$z = -\frac{n\ \log(p)}{\log(2)} \cdot x \cdot u,$$

where x is the GM ciphertext size of one encrypted bit. We choose the size of the GM modulus to be 2048 bits, providing 103 bit security [8]. We consider this to be a good tradeoff between ciphertext size and security. As a result the size of x is also 2048 bits. Given the previous table, this results in the following table.

f \ n	10^2	10^3	10^4	10^5
10^{-1}	2.8 MB	33.6 MB	393 MB	4.76 GB
10^{-2}	4.4 MB	54.8 MB	621 MB	6.96 GB
10^{-3}	6.6 MB	76.6 MB	856 MB	9.92 GB
10^{-4}	9.0 MB	101 MB	1178 MB	12.86 GB

The computational overhead is linear with the data transferred. To obtain an estimation of how fast this is we have done a benchmark using *libgm*[1], which is a thin wrapper around the *GNU Multiple Precision Arithmetic Library*[2]. The code can be found online[3]. We found that a recent Intel Xeon E5-2430L processor takes around 1.5 millisecond per decryption.

This results in the following table.

f \ n	10^2	10^3	10^4	10^5
10^{-1}	16 seconds	3 min 17 sec	38 min 22 sec	7 hrs 46 min
10^{-2}	26 seconds	5 min 21 sec	1 hrs 0 min	11 hrs 18 min
10^{-3}	40 seconds	7 min 29 sec	1 hrs 23 min	16 hrs 8 min
10^{-4}	53 seconds	9 min 57 sec	1 hrs 55 min	20 hrs 55 min

Note that this computation can be done in parallel with low overhead and implementation effort.

[1] https://github.com/areteix/libgm
[2] https://gmplib.org/
[3] https://github.com/felrood/gmspeedtest

To conclude, if this method is practical depends on the circumstances. To illustrate, if we want to check 10k domain names per day with a false positive roughly once every 2 months this amounts to at least 621 MB of traffic per day and one hour of CPU time for decryption.

4.2.7 Other considerations

The scheme we have presented is patented in the US [12], Europe and China. Although the original paper by Kerschbaum does not describe the use of two Bloom filters, the patent does, as Figure 4.2 shows.

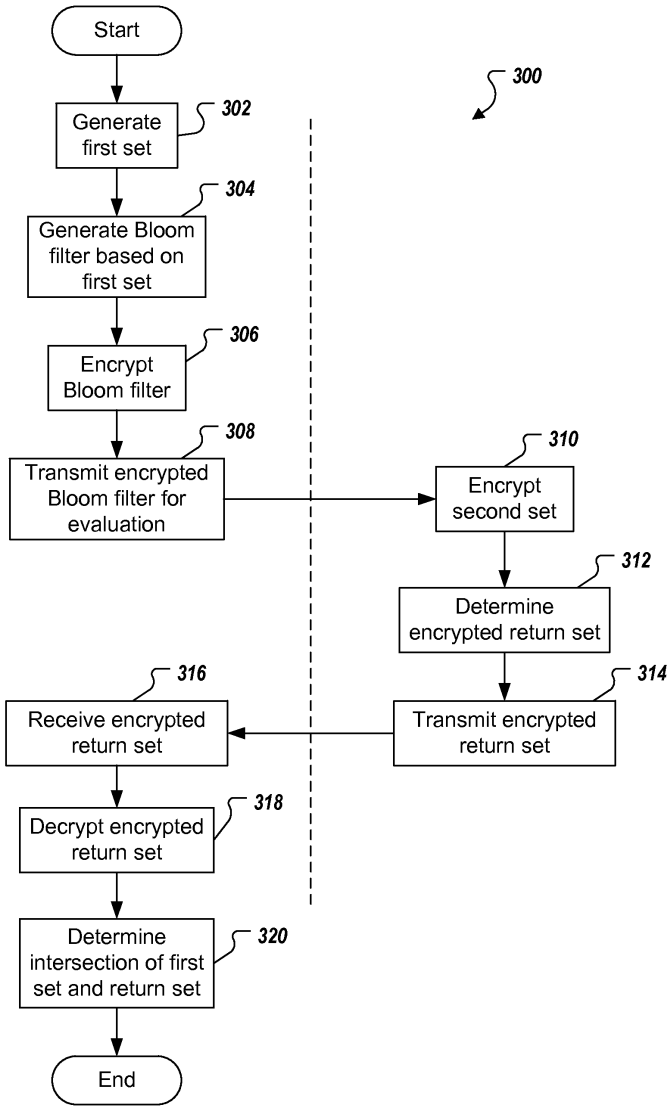

Figure 4.2: Patent [12] showing the use of two encrypted Bloom filters

5. Confidentiality of Hit context

In Section 3.4 we have briefly described some existing tools and methods used to provide privacy and confidentiality in hit context. Here we focus on anonymizing packet traces because a packet trace is the most common form of hit context.

A packet trace surrounding a hit is useful for further investigation into the hit. However, as we have argued in the previous chapter, a packet trace is considered privacy sensitive. In this chapter we present methods to remove the privacy sensitive information from a packet trace in such a way that the remaining content is still useful for investigation.

5.1 Anonymization API

Koukis et. al. describe an Anonymization Application Programming Interface (AAPI) [14]. They describe three goals of network trace anonymization:

- protect the privacy of monitored users,
- hide information about the internal network, and
- be as realistic as possible.

What these goals precisely mean is different for each organization. The authors propose a framework which provides a flexible way to express an anonymization policy. The anonymization functions can be applied to any field in a packet up to the application level.

The main concept of the AAPI is that a series of functions is applied to a traffic stream. There are three categories of functions.

The first category are the filter functions. Currently the berkely packet filter (BPF) and a string search function are supported. With a filter, a series of functions (e.g. anonymization) is applied on only a part of the stream. The second category are the anonymization functions that alter the content of a packet. The last category are the application level functions `COOK` and `UNCOOK` that perform TCP stream reassembly and splitting the stream back into the

original form. This is needed to do anonymization on the application level of TCP streams.

The anonymization functions that are currently supported are: hashing a field, mapping to sequential values or some distribution, random value, replacing with a constant, prefix-preserving for IP addresses, regular expression-based substitution and removing fields. The protocols currently supported are: IP, TCP, UDP, ICMP, HTTP and FTP. Due to the design of the AAPI it is easy to extend the implementation to support new protocols, filters, anonymization functions and input sources.

5.1.1 Usefulness

The authors of AAPI have attempted to measure the usefulness of anonymized traffic for intrusion detection. The policy implemented with AAPI was: *"prefix-preserving anonymization of IP address, set the TTL and IP identification number to constants, removal of the HTTP payload - but not of HTTP headers"*. The anonymized traffic was then passed to Snort. When rules are used that only need packet headers and no content, the anonymized traffic generates the same number of hits as the original non-anonymized traffic. When using rules that also need the packet contents, the anonymized traffic generates 572 hits, versus 1892 for the original traffic. This result illustrates that anonymization and usefulness are often conflicting goals.

5.1.2 Practicality

The flexibility of the AAPI is both a strength and a weakness. Because each organization has a different interpretation of privacy, confidentiality and usable packet traces, a generic method allows individual organizations to apply their own interpretations.

On the other hand, the method does not provide any guidelines or insight into how different anonymization methods affect usability and privacy. This means organizations need to have a good understanding of the anonymization methods. Lack of understanding can lead to deanonymization or reduced usability for particular purposes.

6. Confidentiality of Statistics

As we argued in Subsection 2.2.3, we want to allow a spoke/detector to contribute to aggregated statistics without revealing individual hits. This problem is similar to a problem in the field of smart metering: collecting aggregated power consumption without revealing individual power consumption.

Power consumption is considered privacy sensitive information. However, power companies are interested in power consumption of customers because it allows them to do, amongst other things, power leakage detection and fraud detection. With the aggregated power consumption of a group of customers (typically a street or neighbourhood) leakage and fraud detection is still possible. This gave rise to a number of methods to allow a central entity to calculate the aggregated power consumption of customers without knowing the power consumption of individual customers. We describe three of these methods. We have changed the parameters and names to reflect a threat information exchange environment instead of smart metering.

The schemes need to achieve (at least) two properties we consider important:

- **Authenticity**: only authenticated clients can participate,
- **Unlinkability**: the aggregator cannot link a client to a individual value.

We require authenticity to make sure only clients that are within a certain group can contribute to the aggregate value of that group. The unlinkability is needed to provide anonymity to the clients.

6.1 Using simple additive secret sharing

This scheme is proposed by Kursawe et. al. [15]. The goal is to let the aggregator learn the sum of the hit counts of the clients, without revealing individual hit counts.

Let each client C_i have a hit count $c_i \in \mathbb{Z}$ and a random mask $x_i \in \mathbb{Z}$. For now we assume that the sum of all masks is 0:

$$\sum_{i=1}^{n} x_i = 0$$

The client computes $z_i = c_i + x_i$ and sends it to the hub. The hub then computes the sum:

$$\begin{aligned}\sum_{i=1}^{n} z_i &= \sum_{i=1}^{n} c_i + x_i \\ &= \sum_{i=1}^{n} c_i.\end{aligned}$$

To make sure the sum of all masks is zero, the authors propose to use leader election between the clients. The leader collects the masks and calculates his mask such that the sum is zero. Another possibility is to make the sum of all random masks known to the aggregator. This also requires communication between the clients to coordinate the calculation of the sum of the mask.

6.1.1 Practicality

This scheme is quite simple and almost trivially correct. However, it is not a complete solution to the problem. The most important issue is the mask calculation. Either the sum of the masks must be known to the aggregator, or the clients must ensure it is 0. This requires communication between the clients, either directly or via the aggregator. This preserves the privacy of the clients because the aggregator never learns the individual hit counts and never learns the individual masks.

This means the security of protocol completely depends on the security of the leader-election protocol. In the semi-honest setting we could assume leader-election is done honestly, as the clients are required to execute the protocol faithfully. The leader can then calculate the sum of the masks and send it to the aggregator.

6.2 Using homomorphic encryption

Garcia et. al. [9] propose a scheme which uses homomorphic encryption to provide privacy friendly aggregation. The Paillier [16] homomorphic encryption is used because it a good fit, but the scheme does not depend on it.

The protocol uses a homomorphic encryption scheme that satisfies the property that the product of two encrypted values equals the encrypted sum of both values. Let $\{a\}_{pk}$ denote the encryption of a against public key pk. Then

$$\{a_1\}_{pk} \cdot \{a_2\}_{pk} = \{a_1 + a_2\}_{pk}.$$

The goal is to learn the aggregated value of all hit counts, without revealing information about individual hit counts.

Assume an aggregator A is connected with m clients $C_1 \dots C_m$. All the clients have a certificate containing their (Paillier) public key pk_i and have knowledge of the corresponding private key sk_i. In the original protocol this certificate is signed by a third party. We omit this requirement for now for simplicity. The aggregator initiates the protocol by sending the certificates of all clients participating to every client, so every client has the certificates of all other clients. If there are less than m_{min} participants, the client aborts the protocol. This is to prevent information leakage when there are few clients (e.g. two).

The client has a hit count $a_i \in \mathbb{Z}_n$ where n is a large integer. The client divides its hit count a_i into m random shares, $a_{i,1} \dots a_{i,m}$ such that the sum of the shares equals a_i. He then encrypts $m-1$ shares with the corresponding public key of a client. The client keeps one share for himself. This results in $m-1$ ciphertexts $y_1 \dots y_{i-1} \dots y_{i+1} \dots y_m$ which are sent to the aggregator. The aggregator then (for each client i) multiplies all $m-1$ ciphertexts intended for client $\mathcal{C}_i$. Because of the homomorphic property of the scheme, this equals the sum of all shares for $\mathcal{C}_i$. The aggregator then sends this value to the corresponding client, which decrypts it and adds the last share, and sends the result back to the aggregator. The aggregator can than sum the resulting values of all clients to obtain the aggregated hit count b.

A more formal definition of the protocol:

$\mathcal{A} \to \mathcal{C}_i$: $cert_1, \ldots, cert_m$

$\mathcal{C}_i$: If $m < m_{min}$ then abort

$\mathcal{C}_i$: pick random $a_{i,1}, \ldots, a_{i,m}$ such that $a_i = \sum_{j=1}^{m} a_{i,j} \bmod n$

calculate $y_{i,j} = \{a_{i,j}\}_{pk_j}$ for $j = 1, \ldots, i-1, i+1, \ldots, m$

$\mathcal{C}_i \to \mathcal{A}$: $y_{i,1}, \ldots, y_{i,i-1}, y_{i,i+1}, \ldots, y_{i,m}$

$\mathcal{A}$: $p_i = \prod_{j=1, j\neq i}^{m} y_{j,i} = \{ \sum_{j=1, j\neq i}^{m} a_{j,i}\}_{pk_i}$ (due to homomorphic property)

$\mathcal{A} \to \mathcal{C}_i$: p_i

$\mathcal{C}_i$: $q_i = \{p_i\}_{sk_i}$ (decryption with sk_i)

$r_i = q_i + a_{i,i} = \sum_{j=1, j\neq i}^{m} a_{j,i} + a_{i,i} = \sum_{j=1}^{m} a_{j,i} \bmod n$

$\mathcal{C}_i \to \mathcal{A}$: r_i

$\mathcal{A}$: Calculate $b = \sum_{i=1}^{m} r_i = \sum_{i=1}^{m} \sum_{j=1}^{m} a_{j,i} \bmod n$

6.2.1 Security

The security of this scheme is based on a number of assumptions. The original paper uses the malicious attacker model, with the aggregator as attacker. The certificates from the clients are assumed to be signed by a certificate authority independent from the aggregator. More precisely, the attacker is assumed to be not capable of obtaining large numbers of valid certificates. If the attacker can obtain valid certificates it can do a sybil attack, where it "simulates" a number of clients in a pool with only one legitimate client. This is an unpractical assumption in a threat information exchange environment, because (contrary to the energy sector) such a trusted third party is not naturally available. We can remove this assumption by using the semi-honest adversary model instead of the malicious model. The aggregator is then obliged to distribute the certificates of the clients honestly. Another possibility is that the clients exchange certificates a priori via an out-of-band channel. The scheme is proven IND-CPA secure because the aggregator cannot derive any information from the encrypted shares to which it has access. We also assume the communication channel between the clients and the aggregator to be secure, e.g. via TLS. This requirement is not present in the original scheme, but is necessary because of the semi-honest setting.

6.2.2 Performance

We take a look at both the computational and communication requirements of the scheme for both the client and aggregator.

The client needs to do $m - 1$ encryptions and one decryption, resulting in a $\mathcal{O}(m)$ computational requirement. A single encryption or decryption in the Paillier system is essentialy a modular exponentiation. The communication is also $\mathcal{O}(m)$ because it receives $m - 1$ certificates and sends $m - 1$ encrypted shares.

The aggregator needs to compute the $\{\cdot\}$ function $\mathcal{O}(m^2)$ times, which is a single modular multiplication in the Paillier system. The communication is also $\mathcal{O}(m^2)$ as it collects $m - 1$ shares of all m clients.

6.2.3 Practicality

The aggregator has a computational and communication load of $\mathcal{O}(m^2)$, which limits the size of the group. How much this limits the size depends on a number of parameters. The most important parameters are how often the aggregation is performed and how many metrics (hit counts) are aggregated.

To investigate if this is a problem we estimate the computational and communication load for the aggregator with multiple parameters. The two parameters we consider are the number of participants and the number of metrics that are aggregated.

To estimate the communication overhead we estimate the size of the ciphertexts that need to be transferred by the aggregator. This can be estimated by

$$s = q \cdot m^2 \cdot 2 \cdot n,$$

where q is the number of metrics and m the number of clients.

We choose the length of n to be 2048 bit, providing 103 bit security [7]. We consider this to be a good tradeoff between ciphertext size and security. The size of the ciphertext is 4096 bit, because as [16] notes, the ciphertext is two times the size of the plaintext.

Note that using elliptic curve Paillier [8] can reduce this size substantially, however we have not investigated this any further.

With these assumptions we obtain the following table for the communication overhead for the aggregator

m \ q	10	10^2	10^3	10^4
10	512 KB	5.12 MB	51.2 MB	512 MB
10^2	51.2 MB	512 MB	5.12 GB	51.2 GB
10^3	5.12 GB	51.2 GB	512 GB	5.12 TB
10^4	512 GB	5.12 TB	51.2 TB	512 TB

On the x-axis we see the number of metrics q, the y-axis are the number of participants m. As we expected this is linear in the number of metrics and exponential in the number of participants, and therefor reaches large size quite fast.

Fortunately, we can limit the exponential growth by forming subgroups. As long as the subgroups are large enough to provide some anonymity (e.g. ten companies of roughly the same size/industry) this does not affect the security of the scheme.

To estimate the computational overhead of the aggregator we estimate the time needed to compute all the homomorphic additions. For a single run this can be estimated by

$$t = z \cdot m \cdot (m - 1),$$

where z is the time needed for a single homomorphic addition, which is essentially a modular multiplication. To obtain an estimation of how fast this is we have done a benchmark using *libpaillier*[1], which is a thin wrapper around the *GNU Multiple Precision Arithmetic Library*[2]. The code can be found online[3]. We found that a recent Intel Xeon E5-2430L processor can do around a 100 homomorphic additions per millisecond.

This results in the following table

m \ q	10	10^2	10^3	10^4
10	9 ms	90 ms	900 ms	9 s
10^2	99 ms	9.9 s	1 min 39 s	16 min 13 sec
10^3	1 min 39 s	16 min 39 s	2 h 46 min	27 h 47 min
10^4	2 h 46 min	27 h 46 min	11 d	115 d

Again, due to the exponential nature the computation time rapidly grows with the number of clients. As with the communicational overhead, this can be mitigated by creating subgroups. Also note that it possible to do the calculations in parallel.

[1] http://acsc.cs.utexas.edu/libpaillier/
[2] https://gmplib.org/
[3] https://github.com/felrood/paillierspeedtest

6.3 Anonymous Credentials

The scheme proposed by Cheung et. al. [5] take a different approach than the previous two. Instead of focusing on hiding the hit count number itself, this scheme targets the delivery of the number. This is done by providing an anonymous authenticated channel. The individual hit counts are then submitted to the aggregator such that they are not attributable to a individual client.

The paper implicitly assumes the communication channel is anonymous. We will assume the communication between clients an the aggregator is done via Tor[4] or a similar anonymization network. Although this does not provide anonymity against a global eavesdropper, we consider it sufficient in this setting because the client only needs to be anonymous to the server.

The orginal scheme proposed by Cheung et. al. is not directly usable in a threat information exchange setting. Instead we present a scheme with a similar structure that is more suitable for this setting.

The goal of the scheme is for an authenticated client to deliver a (authenticated) value to the aggregator in such a way that the aggregator cannot attribute the value to that client. We assume there is a single aggregator A, which collects multiple aggregates g in multiple timeframes t. We also assume the semi-honest adversary model.

6.3.1 RSA public-key cryptography

The blind signature scheme is built on the RSA public-key encryption scheme, which we will explain first.

We assume Bob wants to send a message to Alice. We assume Bob already has the public key of Alice, denoted as (n, e). The modulus n is a product of two large (e.g. 1024 bit) primes p and q. The private key d is derived from p and q such that:

$$d \equiv d \cdot e \bmod \varphi(n),$$

where $\varphi(n) = (p-1)(q-1)$. This can be done with the extended Euclidian algorithm. By choosing d, e, n this way the following congruence holds:

$$m^{ed} \bmod n \equiv m \bmod n.$$

To send a message m, Bob encrypts it with the public key of Alice:

$$c = m^e \bmod n.$$

[4] https://www.torproject.org/

Alice can recover the message by decrypting with her private-key:

$$m = c^d \bmod n.$$

Alice can prove a message came from her by signing it with her private key:

$$s = m^d \bmod n.$$

Bob can verify the signature s by checking

$$m \stackrel{?}{=} s^e \bmod n.$$

6.3.2 Blind signatures

The main building block of this scheme are blind signatures [3]. With a blind signature the signer can create a valid signature for a message he has not seen.

There are three parties involved in a blind signature scheme, the client C, the signer S and the verifier V.

Client C has a message m for which it wants a valid signature from signer S. The signer has an RSA keypair (n, e, d) of which (n, e) is known to all parties.

To obtain a signature C first "blinds" the message m with a random value r (where r is relatively prime to n) such that:

$$m' = mr^e \bmod n$$

and sends m' to V. S creates a signature s' for m' by signing it with its private key:

$$s' = (m')^d \bmod n$$

and sends s' back to C. To obtain the signature s, C "unblinds" s' to obtain:

$$s = s' \cdot r^{-1} \bmod n.$$

This is equivalant to

$$\begin{aligned} s &\equiv s' \cdot r^{-1} \bmod n \\ &\equiv (m')^d \cdot r^{-1} \bmod n \\ &\equiv m^d r^{ed} r^{-1} \bmod n \\ &\equiv m^d r r^{-1} \bmod n \\ &\equiv m^d \bmod n. \end{aligned}$$

A validator V can then verify that s is a valid signature for m by checking:

$$m \stackrel{?}{=} s^e \bmod n$$

In our scheme the signer and validator are actually the same party.

6.3.3 Simple scheme

The first scheme is simplified to illustrate the delivery method. This scheme does not consider multiple aggregates, nor timeframes. We assume the aggregator A has a single RSA keypair (n, e, d) of which client C knows n, e a priori.

Authentication phase In the authentication phase the client connects with the aggregator in a non-anonymous way. It authenticates itself to the aggregator and collects a blind signature for the value v the client wants to submit.

The authentication step can be achieved with any form of authentication, for example with a shared secret or with certificates.

Anonymous phase In the second phase client C connects again with aggregator A, but now using a anonymization service such as Tor. It sends v and its unblinded signature s to the aggregator which accepts (and uses) the value if the signature is valid.

6.3.4 Expanded scheme

We adapt the previous scheme to accommodate the aggregation of multiple aggregate groups over many timeframes. For this we assume the aggregator A has two categories of key pairs, one for each aggregate group and one for each timeframe. As a result, each message is signed twice, once with the applicable group key and once with a timeframe key. We use a hash function H to obtain the value $x = H(v, g, t)$ which is used instead of v. We use x' to denote the blinded version of x.

Authentication phase

C		A	
$Auth(C, g, t)$	$\rightarrow$		Authenticate C to A
	$\leftarrow$	OK	Accept if C can authenticate
x'	$\rightarrow$		C sends blinded message x'
	$\leftarrow$	s'_g, s'_t	A sends signatures over x'

Anonymous phase

C		A	
v, g, t, s_g, s_t	$\rightarrow$		
	$\leftarrow$	OK	Accept if s_g and s_t are valid signatures for $H(v, g, t)$

To obtain the aggregate the aggregator sums values v for each group and timeframe.

Note that this scheme does not guarantee that a client cannot submit its value multiple times, as this is not required in the semi-honest adversary model. This protection can be added by using $x = H(r, v, g, t)$ where r is a nonce. The aggregator can then check if a value is submitted multiple times by checking for duplicate nonces. The nonce r should be sufficiently large and randomly chosen by the clients. This makes the scheme robust against malicious clients, but not against a malicious aggregator.

We can make this scheme more robust against a malicious aggregator by assuming the public keys used by the aggregator are indeed public, such that the aggregator cannot distribute a different key to each client. This prevents in essence a sybil attack, where the aggregator places each client in a unique group. To strongly verify there are multiple client in a group this is not enough, but it allows for detection of such an attack if clients corroberate.

6.3.5 Security

To guarantee the submitted value is not attributable to a specific client three conditions need to be satisfied. Obvious but important, there need to be at least two members in each aggregate group, preferably more. With more clients in a group, it becomes more difficult for the aggregator to link a value to a particular client.

Second, all clients need to finish the authentication phase before any client starts the anonymous phase. If this condtion is not met, the aggregator can rule out the clients that have not yet finished the authentication phase if a client performs the anonymous phase. This can lead to deanonymization of clients.

The last condition is that any information about the value submitted by the client is not known to the aggregator. If the data is cross-referenced with other data (which is allowed in the semi-honest adversary environment) it is possible to deanonymise individual clients under certain circumstances. For example, if the aggregator knows only one client in a group has more than 10 computers, and someone submits the value 12 for unique hits for a particular piece of malware, the aggregator can deduce it was that client. As another example, if a client in a group reports it has seen malware for a particular old type

of mainframe, and there is one bank in the group and all other members are relatively young companies, the aggregator can deduce it probably was the bank who owns a mainframe. Even if one such example does not uniquely identify a single member, combining multiple of such examples might.

To conclude, the groups should be chosen such that the organizations within one group are as similar as possible with respect to the metrics that are aggregated and as large as possible.

6.3.6 Practicality

The scheme we have described should be executed for each value that is to be submitted. If this can be done in practice depends on a number of parameters. The most influencial parameters are the number of clients c, the number of aggregate groups g and interval t.

The number of RSA keypairs the aggregator needs to generate is equal to the number of intervals. For each interval one new keypair is needed.

In each timeframe the aggregator needs to do one authentication and sign two messages per client/group combination, thus the computational overhead is $\mathcal{O}(cg)$.

A client needs to blind one message and unblind two signatures per group in a single timeframe. This results in a $\mathcal{O}(g)$ computational overhead.

The communication overhead of the aggregator is $\mathcal{O}(cg)$ for each timeframe. For the client the communication overhead is $\mathcal{O}(g)$ per timeframe.

6.3.7 Discussion

Both an advantage and disadvantage of this scheme is that the agregator learns the individual values. This means it can perform more types of analysis on the data than just the sum, for example compute the standard deviation or build a histogram. This is also a disadvantage because the clients have no hard guarantees what happens with the data they have submitted. It also gives the aggregator more possibilties for deanonymizing the clients, as more information is available for correlation.

6.3.8 Blind signatures versus group signatures

We have chosen blind signatures instead of group signatures [4] for this scheme because they have better anonymity properties. Specifically, a group signature scheme requires a group manager. Such a party is not naturally found in a threat information exchange setting. Moreover, the group manager has the possibility

to revoke the anonymity of a signature. We consider this a undesirable property in our scheme.

7. Conclusion

In this research we have presented a number of methods that can be used to provide confidentiality of data for certain problems in threat information exchange. All of these have methods have limitations and conditions that need to be satisfied in order to function and be secure.

In the first chapter we have formulated three subquestions:

Which PETs are available for keeping IoC's confidential?

We have described two methods, Adapted Rabin-Karp and Public-key Encrypted Bloom Filters, that each achieve confidentiality in a different setting. In case of Adapted Rabin-Karp it is possible to keep signatures confidential until a hit occurs with little overhead. As a drawback, only string based matching can be performed. With Public-key Encrypted Bloom Filters it is possible for two parties to compare two sets of items such that only one party learns the intersection of the set. However, the overhead of this method is substantial, which limits the applicability.

Which PETs are available for anonymizing hit context?

We have considered anonymizing packet traces and presented the Anonymization API. The flexibility of this method allows each organization to implement a fine-grained policy for anonymization. However, due to this flexibilty it remains unclear what impact different policies have on the anonymity and usefulness for threat information exchange.

Which PETs are available for anonymous aggregation of statistics?

In the field of smart metering we have found a problem that looks very similiar to this. We have presented two methods that each accomplish aggregation of statistics in a different way. The Homomorphic encryption scheme produces only the aggregated sum of all statistics, while the Anonymous Credentials scheme is flexible enough to allow any kind of statistics on the data. The cost of the latter scheme is reduced anonymity, the former has a substantial overhead.

Our main research question was:

Can threat information exchange between semi-honest parties be realized using Privacy Enhancing Techniques?

We can conclude that Privacy Enhancing Techniques can indeed be used to realize threat information exchange. However, there are few generic solutions and there is always a cost in terms of performance or reduced functionality.

Furthermore, the semi-honest adversary model has shown to be useful to model threat information exchange, as it is more close to reality than the malicious model or the trusted model. In conclusion we think the methods presented in this research are valuable for new collaborations in threat information exchange that otherwise would not have been possible.

7.1 Future work

In this research we have shown that Privacy Enhancing Technologies can be applied to confidentiality problems found in threat information exchange. However, further research is needed to demonstrate the applicability in practice. To investigate if the methods presented in this research are indeed viable in the real world, they should be implemented and tested with real world data. In particular, it is not directly obvious how the methods presented fit into the STIX, TAXII and CyBOX standards. These standards are all extendable, but we are unsure if methods such as the Public-key Encrypted Bloom Filter can be fitted into them. Due to the amount of work involved this goes beyond the scope of this research.

Looking at the STIX model, we have only considered the Observables as indicators of compromise. Further research can investigate if other classes of the STIX model can be kept confidential, and if that is desirable.

In case of the hit context, further research is needed to reveal what parts of packet traces are most valuable for analysis, and what the privacy impact is. A novel approach would be to use secure multiparty computation schemes to perform the analysis of a packet trace. This would mean the actual packet trace is never in its entirety revealed to anyone outside the detector. In essence, it applies the semi-honest attacker model and PETs to packet trace analysis. Due to the lack of existing research in this area this is beyond the scope of this thesis.

Defend et. al. have shown in [6] that protocols for privacy-preserving smart metering are indeed viable in the real world. The researchers found that the computational overhead of their aggregation scheme is to a problem on the smart meters deployed in the field. The most difficult problem they encountered was not of a technical nature but organizational: how to explain their method to management and convince them of its necessity. However, their research is based on a different method than we have presented. Therefore further research is needed to confirm our methods are also viable in the setting of threat information exchange.

For the Adapted Rabin-Karp scheme we have asserted that it is not possible to recover the elements given a Bloom filter. Although we have not found any literature contradicting this assertion, a more in-depth study of using Bloom

filters in combination with Rabin fingerprints is neccesary to confirm this is indeed preimage resistant.

7.2 Discussion

We are confident the methods we have presented can be useful in practice. However, we have only shown on paper they are feasible and not implemented them. Real world experience is missing, making it hard to draw any hard conclusions.

Another point of concern are the many assumptions and limitations of the methods we have presented. For example, the Adapted Rabin-Karp string matching method is only capable of matching patterns in a string. IoCs are much more flexible in the matching method they allow. The difficulty of recovering the patterns depends in part on how the pattern is fragmented. Not all patterns can be held confidential, and which can and cannot be held confidential is not easily determined. Such considerations reduce the usability of the method.

We estimate the Public-Key Encrypted Bloom Filter and Adapted Rabin-Karp method cover a significant part of the IoCs used in practice. However, we have not presented any method for the other types of IoCs possible in CyBOX.

In this research we have assumed the *hub-and-spoke* model for threat information exchange. The methods presented also apply to the *source-subscriber* and *peer-to-peer* models, except for a few cases. Public-key encrypted Bloom filters don't make much sense in a peer-to-peer setting, as the result of the method only becomes know to one party. However, technically it is possible to use the method. The methods that provide confidentiality of statistics also don't make much sens in the peer-to-peer model. Statistics are aggregated by a central party which is not naturally present in a peer-to-peer model.

Bibliography

[1] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] Sang Kil Cha, Iulian Moraru, Jiyong Jang, John Truelove, David Brumley, and David G Andersen. Splitscreen: Enabling efficient, distributed malware detection. *Communications and Networks, Journal of*, 13(2):187–200, 2011.

[3] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[4] David Chaum and Eugène Van Heyst. Group signatures. In *Advances in CryptologyâĂŤEUROCRYPTâĂŹ91*, pages 257–265. Springer, 1991.

[5] Jeanno Chin Long Cheung, Tat Wing Chim, Siu-Ming Yiu, Victor OK Li, and Lucas Chi Kwong Hui. Credential-based privacy-preserving power request scheme for smart grid network. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE, 2011.

[6] Benessa Defend and Klaus Kursawe. Implementation of privacy-friendly aggregation for the smart grid. In *Proceedings of the first ACM workshop on Smart energy grid security*, pages 65–74. ACM, 2013.

[7] II Ecrypt. Ecrypt ii yearly report on algorithms and keysizes (2011-2012). *Available on http://www. ecrypt. eu. org*, page 5, 2012.

[8] Steven D Galbraith. Elliptic curve paillier schemes. *Journal of Cryptology*, 15(2):129–138, 2002.

[9] Flavio D Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *Security and Trust Management*, pages 226–238. Springer, 2011.

[10] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[11] Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[12] F. Kerschbaum. Public-key encrypted bloom filters with applications to private set intersection, September 3 2013. US Patent 8,526,603.

[13] Florian Kerschbaum. Public-key encrypted bloom filters with applications to supply chain integrity. In *Data and Applications Security and Privacy XXV*, pages 60–75. Springer, 2011.

[14] Dimitris Koukis, Spyros Antonatos, Demetres Antoniades, Evangelos P Markatos, and Panagiotis Trimintzios. A generic anonymization framework for network traffic. In *Communications, 2006. ICC'06. IEEE International Conference on*, volume 5, pages 2302–2309. IEEE, 2006.

[15] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *Privacy Enhancing Technologies*, pages 175–191. Springer, 2011.

[16] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology EUROCRYPT'99*, pages 223–238. Springer, 1999.

[17] Michael O Rabin et al. *Fingerprinting by random polynomials.* Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[18] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption*, pages 371–388. Springer, 2004.

[19] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for nc 1. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 554–566. IEEE, 1999.

[20] Xiaokui Shu and Danfeng Daphne Yao. Data leak detection as a service. In *Security and Privacy in Communication Networks*, pages 222–240. Springer, 2013.

Glossary

ASCII American Standard Code for Information Interchange. 31

FTP File Transfer Protocol. 41

HTTP Hypertext Transfer Protocol. 41

ICMP Internet Control Message Protocol. 41

IND-CPA indistingishable under chosen plaintext attack. 33, 45

IP Internet Protocol. 4, 6, 19, 23, 41

IPv4 Internet Protocol version 4. 22

IPv6 Internet Protocol version 6. 22

MAC Media Access Control address. 19, 23

MISP Malware Information Sharing Platform & Threat Sharing. 14

RSA Rivest-Shamir-Adleman cryptosystem. 48, 49, 52

TCP Transmission Control Protocol. 40, 41

TLS Transport Layer Security. 45

TTL Time to live. 41

UDP User Datagram Protocol. 41

URL uniform resource locator. 4

XML Extensible Markup Language. 10, 12, 13, 31

Acronyms

AAPI Anonymization Application Programming Interface. 40

BPF berkely packet filter. 40

CERT Computer Emergency Response Team. 7, 14
COA course of action. 11
CyBOX Cyber Observable eXpression. 12

GM Goldwasser-Micali. 33

IoC Indicator of Compromise. 4, 6
ISAC Information Sharing and Analysis Center. 14

NCSC National Cyber Security Centre. 7

PET Privacy Enhancing Technique. 6

STIX Structured Threat Information eXpression. 10
SYY Sander, Young and Young. 33

TAXII Trusted Automated eXchange of Indicator Information. 13
TTPs Tactics Techniques & Procedures. 11