POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Enhancing Email Forensics: A DKIM archiving and re-verification tool for long-term Signature validation

Supervisor
prof. Andrea Atzeni
prof. Paolo Dal Checco

Candidate
Marco Vitale

Academic Year 2023-2024

To mum and dad

Summary

In the context of modern communications, emails play a fundamental role for a large variety of purposes, communicating with friends and coworkers, exchanging sensitive information between companies, or receiving important communications for users' online accounts.

As with all the other areas in the IT field, emails are the subject of different kinds of attacks; thanks to the possibility of transporting attachments, many attackers infect emails with malware and viruses, masquerading them as harmless files. Other examples include source forgery, also known as email spoofing, or the well-known phishing, aiming to obtain personal information of the targets or possibly fraud them.

In order to limit these kinds of attacks, every email service provider has adopted its own rules that associated with an anti-spam software aims at distinguishing between potentially dangerous emails from healthy ones, leaving the user's inbox as protected as possible. These rules are usually based on authentication protocols, which were born to help the email infrastructure combat cybercrimes and enhance the integrity and trustworthiness of email messages.

The Internet Engineering Task Force (IETF) has developed over the years, three different protocols that nowadays are combined to protect emails under different aspects. Sender Policy Framework, also known as SPF, allows a domain owner to publish into a DNS record in a specified format, the IP addresses of the mail servers authorized to send email on behalf of that domain. DomainKeys Identified Mail, also known as DKIM, aims at providing integrity of the email content and authenticity of the sender's domain, by computing a digital signature and attaching it to the email headers. Finally, Domain-based Message Authentication, Reporting, and Conformance also known as DMARC, allows domain owners to receive reports about their email activity, which is a really helpful feature when contrasting email phishing campaigns and to inform recipients on how their email must be treated when they do not pass DMARC alignment with DKIM and SPF.

When cybercrime occurs, forensic experts have the duty of investigating all the possible traces left by the criminals, which can be found into the victim's inbox, for this reason, it is fundamental the ability of establishing the authenticity of an email message, which can be used as a proof within a legal process. In order to achieve that, the email headers are analyzed and possibly re-verified to ensure the email content has not been modified. Here is the problem discussed by this thesis, as DKIM specifically was designed to be used as soon as the email arrives at the recipient side, and not for verification after long time.

The primary objective of this thesis was initially to study the state of the art of the DKIM protocol, looking for possible products or previous work on the long-term verification of DKIM signatures. After discovering that no open-source solutions had been published about it, it was crucial to look at how this problem was addressed in other contexts.

In the area of digital documents, this is a topic described by the concept of Long Term Validation (LTV) or Long Term Signature (LTS). The primary goal of these standards is the preservation of all the information required to verify the signature in the future, with the signature itself, and to achieve this result, a specific process is followed. After signing the document, a Time Stamp Authority (TSA) provides proof of the signing time by issuing a so-called timestamp token, and to demonstrate the validity of the signer's certificate at that time, a Validation Authority utilizes either OCSP or CRL technologies. This new type of signature can be embedded in various standards, depending on the nature of the file to be signed, such as PAdES, XAdES, and CAdES.

Once understood the technologies and methodologies used, it was time to transport it to the DKIM field. Initially, various solutions came to the surface, specifically those aiming at changing the protocol behavior. However, the adoption level of the DKIM protocol, which by some research appears to be not so high, discouraged these kinds of solutions, given the burden of modifying an already working infrastructure to implement some changes. For all the reasons described so far, an external tool was the perfect choice to limit the interference with the protocol itself but address the issue that has arisen.

The main problem that emerges when an expert wants to re-verify a DKIM signature, is in the availability of the public key needed for the process. In this protocol, when the key couple is created, the private part remains secret and must be installed in the sending SMTP server, while the public part is published inside a DNS TXT record of the domain. This record can be retrieved by the SMTP server of the recipient, by composing a DNS query using the domain and the selector name found in the DKIM header. However, since the DKIM keys must be rotated to avoid a possible compromise, and there is no standard way to complete this process, the domain owner may choose to change the key and keep the same selector or remove the previous key and publish a new one on a fresh new selector.

By creating an archive of DKIM records, it would have been possible to retrieve the requested public key when needed. In order to do so, a Python project was created, with the duty of querying daily a list of pairs domain selector, extracted from the email headers of a bunch of email accounts and writing down the responses to these questions. Additionally, to certify the existence of these records at the time of that request, a Timestamping Authority was introduced, following the same process executed in the field of Long Time Verification which was previously described. Since Timestamping Authorities that follow some specific regulations, provide timestamp tokens with a legal validity upon payment of a fee, which typically depends on the quantity, the user can choose whether to timestamp each domain file separately or timestamp a single zip folder with all the files produced. This is done to reduce the number of timestamps requested. To add to this project a further forensic approach, all the traffic exchanged with the DNS server and with the TSA were captured, providing an additional way to verify the source of this data by simply inspecting the packets.

Since emails are the principal source of DKIM information, noticing a change of selector by a certain domain would have been possible only by inspecting and monitoring some email headers. This is the reason why the "email" module was introduced. By having access to an email inbox via the IMAP protocol, this part of the project had the possibility of downloading the received email and extracting valuable information about DKIM. This inbox was later filled with emails by automating the sending process and creating accounts for the domains that wanted to be monitored. Since not all the domains give the possibility of sending emails directly, different solutions were thought in order to try to receive as many emails as possible. Among them, enabling email notifications for social networks was working for Facebook, while automating the password recovery process via the Playwright's Python library was a more specific and domain-dependent solution that worked for Asos.

Once having created the archive of DKIM records, by slightly modifying the DKIMPY library it was relatively simple to complete the verification process. In particular, the changes made to this library were focused on the possibility of accepting a fixed DKIM record and manipulating the time of verification to simulate the process at the receipt time.

To further enhance the list of monitored domains, a new module was implemented, with the duty of discovering the selector used by specific domains. This service is already offered by some online websites, and after analyzing their functioning and results with a fixed number of domains, it was clear that all of them were using a sort of brute-force approach. By querying the DNS server of the selected domain with a fixed number of selectors, chosen according to their spread, any reply would have been shown to the user. The module was then enriched with the possibility of querying multiple domains at once and generating a list of selectors by providing a list of regexes.

Initially, the tool was developed with the intention of possibly working alone after being installed onto a server. This behavior was achieved by simply scheduling a cronjob to run the Python script at a specific time every day. After having added some active functionalities, like the discovery and the verify, which request the user interaction, a more structured way to use the tool

was needed, besides the possibility of using the command line. For this reason, a web interface was developed, always within the Python project thanks to the Flask framework. At this point, by starting a web server, all the tool's functionalities were available, with a more user-friendly approach. A specific company can then choose to install the tool within its network and make it available for its employees at a specific address.

Finally, at the end of the development process and thanks to temporary licenses, it was possible to compare the solution with two tools developed by Metaspike (Forensic Email Collector and Forensic Email Intelligence), an email forensic software company. Among the similarities, there is the usage of a local archive integrated with live queries when records are missing, DKIM signature verification capable of discriminating between body or header modifications, and the ability to timestamp logs, even if with a different target.

Acknowledgements

This work would not have been possible without the help and guidance provided by my supervisors, Prof. Andrea Atzeni and Prof. Paolo Dal Checco, to whom I am deeply grateful for their time and their always precise and on time advices. Additionally, I would also like to thank Metaspike for providing temporary licenses, which allowed me to enrich my thesis by comparing my work with state-of-the-art software.

I would like to conclude the rest of the acknowledgements in Italian, so they can directly reach the people concerned.

Non posso che iniziare col ringraziare i miei genitori, mamma e papà, per tutti i sacrifici che hanno fatto per me in questi anni per potermi permettere di percorrere questa strada. Grazie per questo e per il continuo supporto e affetto che mi dimostrate da sempre, che mi hanno reso la persona che sono. Spero a mio modo di aver ricambiato almeno in parte a tutto quello che mi avete dato, e spero di avervi reso orgogliosi, questo traguardo è dedicato a voi.

Un grazie va anche a tutta la mia famiglia, che negli anni mi ha sempre incoraggiato e ha creduto in me. Grazie anche a te, Nonna Enza, so che da lassù sarai fiera di me.

Ringrazio la mia metà, Marta, che ormai da 5 anni mi è vicino e sopporta le mie ansie e paranoie, sempre pronta a tirarmi su con una parola di conforto e ad aiutarmi a credere in me stesso. Grazie perchè oltre ad essere sempre stata presente nei momenti più belli, mi hai accompagnato anche nei momenti più bui, questo traguardo è anche merito tuo.

In ultimo, ma non per importanza, ringrazio tutti gli amici che mi sono stati vicini in questo viaggio. A partire dagli amici di casa, sempre presenti al mio ritorno da Torino, per arrivare agli amici conosciuti tra le mura del Politecnico, senza il quale il mio percorso non sarebbe stato lo stesso.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's digital landscape, emails have become an indispensable tool, used massively by a lot of people for myriad purposes, from casual correspondence with friends and classmates to the exchange of critical and sensitive information between companies and co-workers.

That being said, this particular way of communication that guarantees an ease of use, speed and relative anonymity is also favoured by malicious actors to commit crimes, digital related and non-digital related. Among these, email spoofing stands out, where the apparent source of the email is forged, leading the recipient to trust false identities. Email fraud is another prevalent form, in this case the attacker aims at tricking large accounting departments into paying fraudulent invoices. Additionally, email phishing targets different types of victims, seeking to obtain personal information or deploy malware onto their devices.

In response to the increase of threats, the cybersecurity community has developed different protocols and measures to enhance integrity and trustworthiness of email messages. Despite these efforts, the prevalence of email-based attacks remains relatively high, underlying the importance of the topic and the need for innovative security practices. Additionally, relying solely on the authentication result header is problematic, as these headers can be forged or altered by malicious actors.

When such crimes occur, there is the need to investigate, extract, and analyze emails to collect digital evidence usable in a court of justice. This discipline is known as email forensics. The first step of the investigation process is the collection of emails and inboxes. This is a delicate operation typically performed by some specialized tools, as the evidence must not be affected by the acquisition process and must remain preserved.

At this point, various aspects of emails can produce valuable insights; email headers hold details about sender and recipient addresses, the transmission path, which can be verified by inspecting the mail servers involved, send and receive timestamps, and finally a unique email identifier called message-ID. The content of the email is another important aspect, as malware and viruses are always attached and masquerade as harmless files. Even metadata can help in enriching the context by providing information about the sender's IP address and the email client version.

Lastly, email authentication protocols like DKIM, SPF, and DMARC are always checked as additional sources of information. DKIM (DomainKeys Identified Mail) specifically, is a crucial technology that provides authentication of the sender's domain and integrity of the email content. The result of its verification process is embedded into the email header by the receiving email server, playing a significant role for email forensic experts in reconstructing events. However, a major limitation of DKIM is the difficulty of verifying this type of header for email received in the past. This difficulty arises because the protocol was built to be used as soon as the mail is received by the mail server of the recipient. Furthermore, the choice of not using PKI (Public Key Infrastructure) to certify the keys used in the protocol eliminates the possibility of relying on public repositories for key management, such as OCSP (Online Certificate Status Protocol) and CRL (Certificate Revocation List).

1.1 Objectives

The primary objective of this thesis was to develop and test a tool that enables the verification of DKIM for emails received in the past, creating an archive of the keys used by the different domains over time. This solution would result particularly useful for email forensic expert during investigations. In order to achieve that, various challenges were faced, and throughout the development process, additional functionalities were added to address these problems arise.

1.2 Outline

The remaining sections of the document are structured as follows:

- Chapter 2 - Background: this section aims to provide a basic background knowledge of the main topic of this thesis work. It begins with an overview of email architecture to contextualize the problem, then explores the details of the DKIM protocol and finally examines the Timestamping process, which proved to be crucial in our work.
- Chapter 3 - State of the art: in this section the current state-of-the art for DKIM implementation and long term validation is analyzed;
- Chapter 4 - Solution: this chapter describes the various solutions considered and why they were discarded, leading to the design of the final chosen solution.
- Chapter 5 - Conclusion and future development: this final section summarises the result obtained and provides some analysis for future development.

Chapter 2

Background

This section aims to provide comprehensive knowledge to better understand all the topics covered in this thesis. It starts with an overview of email architecture, the foundation upon which all email authentication protocols are built, and then explores DKIM, the protocol of interest in this work. Finally, it discusses timestamping, a helpful technology for certifying the existence of documents at a specific point in time.

2.1 Email architecture

Let's start analysing the Email architecture highlighting first the main actors that interact with it: users. Users are typically people, organization or processes that exchange messages. In Internet Mail there are four types of Users:

- Authors
- Recipients
- Return Handlers
- Mediators

Authors are responsible for creating email messages, writing their contents and selecting one or more Recipient addresses. The Message Handling Service (MHS), which often appears by the user's perspective as single, unified system, despite potentially being operated by different organizations, is responsible for transporting the message from the Author to the Recipients.

Recipients are the actual receivers of email messages. They have the option to reply to the Author, thereby closing the communication loop. Generally, there are three different types of recipients:

1. Primary recipients: Usually listed in the "To" field, these are the main intended recipients of the message.
2. Secondary recipients: Included in the "Cc" (carbon copy) field, who receive the message for their information.
3. Blind Carbon Copy recipients: These recipients receive the message without other recipients knowing, as their addresses are hidden from the other recipients.

The Return Handler, also called "Bounce Handler", is a special Recipient that has the duty of handling the notifications related to the delivery process generated by the MHS, such as failures or completions of delivery.

The Mediator instead receives, aggregates, reformulates and redistributes messages among Authors and Recipients. The common example of this role is a group Mailing List, in this scenario, the Mediator takes the original message from the Author, creates a new message and sends it to all members of the mailing list. Recipients see the message as if it originated from the original Author.[5]

Let's move on now and start discussing the infrastructural actors involved in email architecture:

- Message User Agent (MUA)
- Message Store (MS)
- Mail Submission Agent (MSA)
- Message Transfer Agent (MTA)
- Mail Delivery Agent (MDA)

A Message User Agent (MUA) is typically a computer program used to access and manage a user's email such as Outlook, Thunderbird or webmail services like Gmail. It works on behalf of the users, and it is responsible, from the Author point of view, of the submission of the email into the transfer infrastructure via an MSA. The Recipient MUA on the other hand, processes the received mail.

The Message Store (MS) is a component where email messages are stored and managed. It can be located on the same machine as the MUA or on a remote server. The MS acquires messages from an MDA, and it is accessed by the MUA to retrieve emails using either POP (Post Office Protocol) or IMAP (Internet Message Access Protocol).

The Mail Submission Agent (MSA) plays a critical role in email architecture, it is the first entity to receive the email from the client and has the duty of validating and authenticating the email, ensuring that it adheres to the relevant protocols and standards. Once the email is properly formatted and authenticated, the MSA forwards it to the MTA which then handles the actual delivery of the message to the recipient's mail server.

A Message Transfer Agent (MTA) acts similarly to packet switch or an IP router. Its primary role consists of routing emails between different MTAs, moving the message closer to the Recipients until it reaches the destination MDA. Point-to-point communication between MTAs typically uses SMTP (Simple Mail Transfer Protocol) which provides a basic level of reliability through failure recovery and retransmission mechanisms. Internet Mail MTAs are expected to store messages enabling recovery in case of shutdowns or service interruptions. The routing mechanism of this element relies on DNS MX records, which specifies the MTA through which the queried domain can be reached, allowing any MTA to connect to any other. Given the architectural importance of the MTA, it is typically categorized into three different types:

- Border or boundary MTA: This MTA acts as a gateway between the Internet and the users within an organization. There are two types of boundary MTAs, depending on the message direction (outbound/inbound).
- Intermediate MTA: This MTA is just a pass-through terminal, routing messages from one MTA to another.
- Delivery or Final MTA: This MTA is responsible for the actual delivery of the message to the inbox or to the final MDA.

Finally, the MDA is the component responsible for receiving the email from the MTA and delivering it to the recipient's inbox. This step typically involves interaction between the MDA and the Mail Store (MS) using protocols such as POP or IMAP.

In this architecture, it is generally assumed that authentication schemes are applied at the border MTA level[6]. On the outbound side, the protocol is applied, and on the inbound side, the authentication is verified. In the next section one of these protocols will be deeply analyzed.

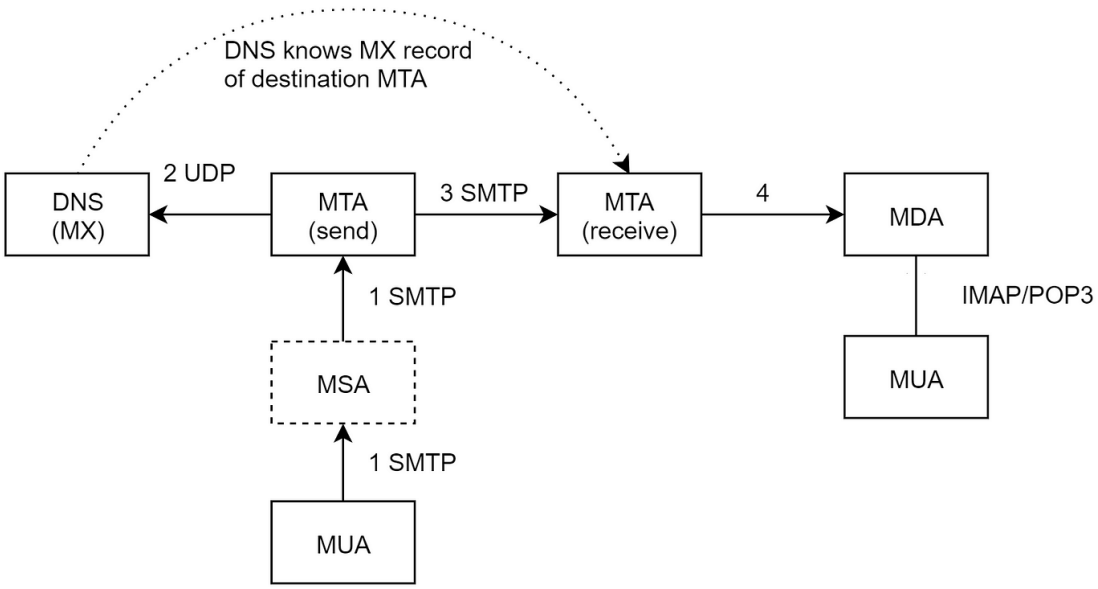

Figure 2.1. Email Architecture scheme (Source: Understanding Email).

2.2 DKIM

Since the Simple Mail Transfer Protocol (SMTP) lacks authentications mechanism, the Internet Engineering Task Force (IETF) has developed three protocols to address this security issue: Sender Policy Framework (SPF), DomainKeys Identified Mail (DKIM) and Domain-based Message Authentication, Reporting, and Conformance (DMARC). These protocols are typically used together to protect email in different ways and address various aspects of security.

After analyzing DKIM in detail, SPF and DMARC will be presented, underlying the link between these three protocols.

2.2.1 Overview

DomainKeys Identified Mail (DKIM) is an email authentication method that relies on digital signature to prevent emails from being forged or tampered with, providing integrity and authenticity protection. This protocol was created from the merger of two earlier protocols:

- Enhanced DomainKeys: Designed by Yahoo to verify the DNS domain of an email sender and the message integrity.
- Identified Internet Mail: Designed by Cisco as a signature-based authentication standard.

The first DKIM draft was published in the 2006. It became a proposed standard in the May 2007 with RFC 4871[7] and was finally established as an Internet Standard in September 2011 with the RFC 6376[8]. Given the fundamental role of cryptographic algorithms in this protocol, two major updates followed:

- RFC 8301[9], issued in January 2018, bans SHA-1 and updates key sizes from 512-2048 bits to 1024-4096 bits.
- RFC 8463[10], issued in September 2018, adds an elliptic curve algorithm to the existing RSA. The added key type, k=ed25519 is adequately strong and features short public keys, more easily publishable in DNS.

Today, major email service providers like Yahoo and Google have adopted DKIM. In February 2024, both companies began requiring bulk senders to authenticate their emails with DKIM to ensure successful delivery to their hosted mailboxes. However, a recent study revealed that only the 28.1% of Alexa Top 1 million domains have enabled DKIM, with 2.9% of those being misconfigured [2]. These statistics highlight the need for a broader adoption of this protocol in order to achieve better security outcomes.

2.2.2 Workflow

To use DKIM, the first step is to create a pair of keys - public and private - that will be used in the authentication process. These keys do not need to be certified by a Certification Authority, as DKIM does not involve x509 certificates. After creating the keys, the public key must be added to a DNS TXT record so that the recipients can retrieve it to verify the signature. If DKIM is enable by the sending email service, a digital signature is generated using the sender's private key on the hash of the email body and some selected headers before the email is sent. This signature is then included in the DKIM Signature header of the email. Upon receiving the email, the recipient's server retrieves the public key and verifies the signature to ensure the email has not been altered during transit.

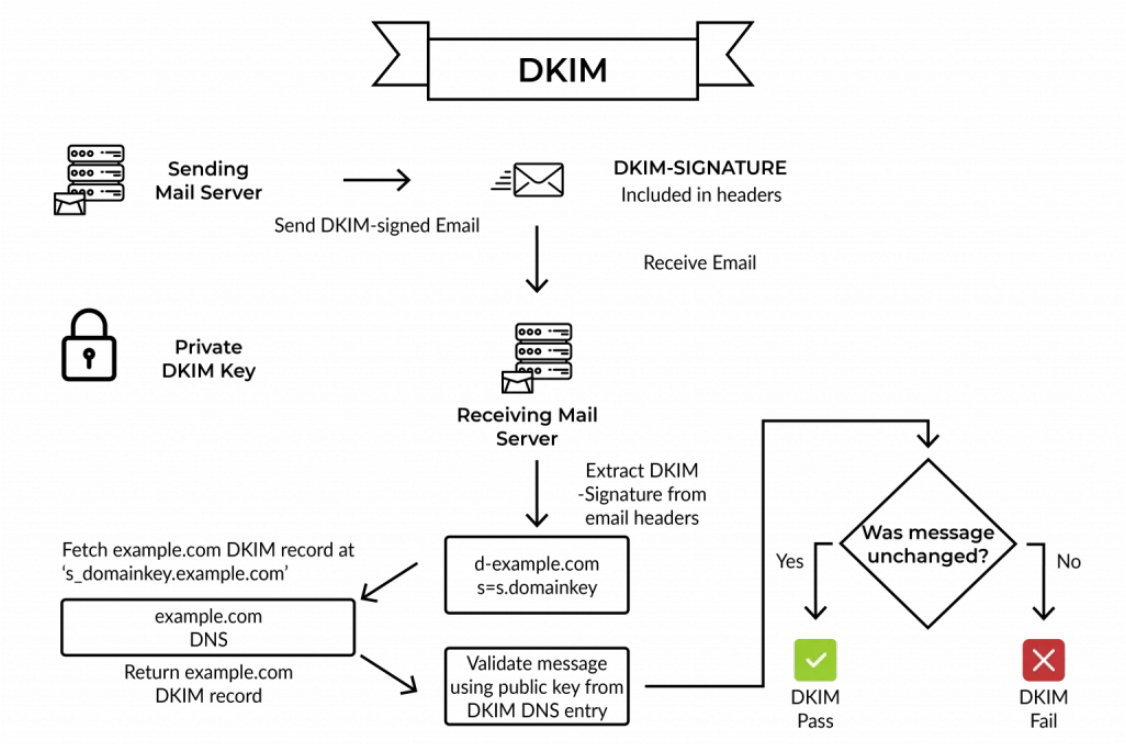

Figure 2.2. DKIM workflow (Source: DomainKeys Identified Mail (DKIM)).

2.2.3 Signing

Typically, signing will be done by a service agent that is part of the message author's organization or delegated by them. Architecturally speaking, the job is done by an MTA. The signing operations begins creating a hash of the message body, which may be truncated to a given length l as indicated in the DKIM-Signature header. Next, the selected header fields are hashed in the order specified by the h value in the DKIM-Signature header. Depending on the email sender's server software, the user may have the possibility to choose which headers to sign, except for the FROM field, which must always be signed according to the RFC 6376[8]. At this point, the DKIM-Signature

field of the signature being created, with bh set to the just-computed body hash and b set to an empty string, is implicitly included in the second hash. The existence of the DKIM-Signature value in the h field only refers to a preexisting signature, as a single message can be signed multiple times.

This is an example of a DKIM-Signature header:

```
DKIM-Signature: v=1; a=rsa-sha256; d=example.net; s=brisbane;
      c=relaxed/simple; q=dns/txt; i=@eng.example.net;
      t=1117574938; x=1118006938; l=200;
      h=from:to:subject:date;
      z=From:foo@eng.example.net|To:joe@example.com|
       Subject:demo=20run|Date:July=205,=202005=203:44:08=20PM=20-0700;
      bh=MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTI=;
      b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+yuU4zGeeruD00lszZVoG4ZHRNiYzR
```

Figure 2.3. DKIM signature (Source: RFC 6376).

Let's analyze now the tags present:

- v: This tag represents the Version and must be the first tag in the signature. Its default value is 1.
- a: This tag specifies the Algorithm used to generate the signature.
- d: This tag represents the Domain for which the signature is created, also known as Signing Domain Identifier (SDID).
- s: This tag represents the Selector, which is used together with the domain value to identify a specific public key.
- c: This tag contains two names separated by a "/" and informs the verifier about the header and body Canonicalization algorithms used by the signer, respectively.
- q: This tag represents the Query methods used to retrieve the public key. The only currently available value is "dns/txt"
- i: This tag represents the Agent or User Identifier (AUID) on behalf of which the SDID is taking responsibility. The domain part of this address must be the same as or a subdomain of the domain specified in the d tag.
- t: This tag contains a timestamp indicating when the signature was created. It is represented by the number of seconds since 00:00:00 on January 1, 1970, UTC.
- x: This tag contains a timestamp in the same format as t, after which the signature is considered invalid. It specifies the expiration date.
- l: This tag contains the number of octets of the body of the email that were included in the cryptographic hash.
- h: This tag contains a list of the headers signed by the DKIM signature. It may include multiple instances of a header field name, indicating that multiple occurrences of the corresponding header field are included in the header hash.
- z: This tag contains a list of the header present when the email was signed, along with their values.
- bh: This tag contains the base64 hash of the canonicalized body part of the email.
- b: This tag contains the base64 value of the signature.

2.2.4 Verification

To verify the DKIM signature, the receiver's SMTP server must perform a DNS lookup to find the correct public key stored in a DNS TXT record. This query is created using the selector and domain fields from the signature, represented by the s and d tags, respectively, in the following format:

selector._domainkey.*domain*

The ._domainkey part is always fixed. For example, given the previously described signature (Figure 2.3), the query would be:

brisbane._domainkey.*example.net*

Below is an example of a query reply:

```
v=DKIM1; k=rsa; h=sha256;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCZg4sPvXfCCjb7qOf5HjWYfOFYIqltCHjLhw+kL
VrbtS+LV9XLANxqeoh2z7gS+MPpFZ8rtI8HyD5+j7kLGNa3D2XLf9L8Sj3ggaeyyPaOk4hsVILw97b/
10zXDd6Cut+thVgzJLTSyE3mQ2HVPXIkgTPbiGoj51w8uEJEzccNLwIDAQAB
```

Figure 2.4. DKIM query reply.

Now, let's analyze the tags present in the response:

- v: This tag represents the version of the DKIM key record and must be the first. Its default value is DKIM1.
- k: This tag represents the key type chosen.
- h: This tag is optional and contains the acceptable hash algorithms chosen by the signer.
- p: This tag contains the public key data, base64 encoded. An empty value means that the public key has been revoked.

Canonicalization

Some existing mail systems modify emails while in transit, potentially breaking the DKIM signature. To address this issue, canonicalization algorithms are used. These algorithms allow emails that are represented slightly differently but have the same value to be recognized as equal, ensuring they produce the same hash by applying a series of modifications to the text.

Some signers may accept minor modifications to the email, while others may not tolerate any changes. Additionally, some signers may accept modifications to the email header but are unwilling to accept any modifications to the body of the message.

DKIM accommodates these requirements by allowing users to choose between two canonicalization algorithms for both the header and body: a "simple" algorithm, which permits almost no modifications, and a "relaxed" algorithm which is more tolerant of changes.

Typically, if no canonicalization is specified by the signer, the "simple" one is chosen for both the header and body.

Let's analyze in deeper details these canonicalization algorithms:

- Simple: The more restrictive option, chosen by default, does not permit any modifications and is therefore safest.
  - Header: Header fields are not changed in any way. Specifically, header fields names must not be case-folded, and whitespace must not be altered.

  - Body: Ignores all empty lines at the end of the message body, converting any repetition of "CRLF" to a single "CRLF". If there is no body or no trailing CRLF on the message body, a CRLF is added.

- Relaxed: This option tolerates more modifications to the email and is therefore less secure.

  - Header: Converts all header field names (but not the header field values) to lowercase, unfolds all header field continuation lines (combining header value into a single line), converts all sequences of one or more WSP (whitespace, including tabs) characters to a single SP (the basic space), deletes all WSP characters at the end of each unfolded header field value, and removes any WSP characters before and after the colon separating the header field name from the header field value.
  - Body: Ignores all whitespace at the end of lines, reduces all sequences of WSP within a line to a single SP, and ignores all empty lines at the end of the message body.

Authentication-Results

The result of the whole verification process, including other protocols over DKIM such as DMARC and SPF, is typically written in a specific header of the email called "Authentication-Results" and defined by the RFC 8601[11]. End users are not expected to read this header field directly; instead, it is consumed by programs that use this data to render it in a human-readable form. The host performing this verification may apply additional local policies, such as requiring the signature on the message to be executed only by the domain present in the "From" header, making third party signatures invalid even if they verify.

Let's take a look to an example of an Authentication Result header:

```
Authentication-Results: mx.google.com;
dkim=pass header.i=@mp1.tripadvisor.com header.s=scph0523 header.b="Mtae/947";
spf=pass (google.com: domain of
msprvs1=198088bgon0xd=bounces-266693-906@bounce.mp1.tripadvisor.com designates
137.22.227.229 as permitted sender)
smtp.mailfrom="msprvs1=198088Bg0n0XD=bounces-266693-906@bounce.mp1.tripadvisor.
com";
dmarc=pass (p=REJECT sp=REJECT dis=NONE) header.from=mp1.tripadvisor.com
```

Figure 2.5. Authentication-Results header

As seen in the image above, the authentication result status is provided as the value of the dkim tag. Additionally, some information from the signature is reported, such as the "i" tag representing the Agent or User Identifier, the "s" tag representing the selector, and the "b" tag representing a truncated form of the base64-encoded signature.

The dkim result can be set as follows:

- none: The message was not signed.
- pass: The message was signed, and the signature passed the verification tests of the verifier.
- fail: The message was signed but failed the verification tests. This can be due to an alteration of the email or a mismatch of the public key retrieved.
- policy: The message was signed but failed to meet the verifier's requirements due to a specific issue, such as weak algorithm used or too small key length.
- neutral: The message was signed but the signature contained syntax errors or were not able to be processed. For example, some malformed tags could make the signature not fully evaluable.

- temperror: The message could not be verified due to a temporary error, such as the momentary unavailability of the DNS server used to retrieve the public key. A later attempt may produce a final result.
- permerror: The message could not be verified due to some unresolvable error, such as absent header fields.

2.2.5 Offered functionalities

DKIM establishes a basis for verifying the legitimacy of an email, enabling the association of a verifiable identifier with a message. With this identifier in place, recipients can make informed decisions about how to handle the message based on the credibility of the associated identity.

Use with Spam Filtering: DKIM functions as a method of labeling an email rather than directly filtering or identifying spam. Nonetheless, widespread implementation of DKIM can prevent spammers from forging email source addresses, a common tactic they use today.

Anti-Phishing: DKIM also serves as an effective anti-phishing tool. Organizations in frequently phished domains can sign their emails to verify their authenticity. Recipients can interpret the absence of a valid signature on emails from these domains as a likely sign of forgery. When combined with DMARC, which allows domains to publish their authentication practices (including SPF and DKIM), it simplifies the process for recipients to determine whether an email is spam or not.

Improved Deliverability: Emails signed with DKIM are more likely to reach the recipient's inbox because they include cryptographic evidence of authenticity. The result of DKIM authentication is used as a spam filtering criteria by most of email providers, thus enhancing the chance of successful email delivery.

Reputation Management: DKIM signatures contribute to managing a sender's reputation with email providers and spam filters. Although a valid DKIM signature alone does not directly increase or decrease the trust level of a message, it facilitates the use of other mechanisms to evaluate and establish trust[12].

2.2.6 Open problems

This section analyzes the main issues affecting the DKIM protocol.

Visibility of the verification: As discussed in one of the last sections, the result of the verification process is added to the email header, but it's not intended to be read by end users. Some of the major email services such as Gmail and Outlook, display this information only when viewing the original message, not through the standard interface. On the other hand, Thunderbird does not directly support displaying this information but relies on a public extension of the email client that highlights these results. Typically, the user must take additional steps to verify the DKIM signature's validity.

DOS Attacks: The widespread adoption of DKIM will lead to a significant increase in DNS queries to the claimed signing domain. In the event of large-scale forgeries, DNS servers could experience a substantial increase in queries. If the DNS servers containing the TXT record used by verifiers to check the DKIM signature are attacked, the entire verification process is at risk, potentially causing legitimate emails to be marked as spam.

Key related problems: Creating a digital signature in DKIM requires a key pair, and over the years, the usage and length of these keys have become crucial factors. In 2012, Zach Harris, a US-based mathematician, accidentally discovered that Google was using a weak cryptographic key for certifying the emails sent from their corporate domains. The issue was with the DKIM key used for Gmail[13]. For security reasons, the DKIM standard, at that time required keys to be at least 1,024 bits long, but Google was using a 512-bit key, which could be easily cracked with some cloud computing and time. Today, after Since the issuance of RFC 8301 in January 2018, the allowed key sizes range from a minimum of 1024 bits to a maximum of 4096 bits.

Modification of email content: Mailing lists pose a challenge for DKIM because they often modify email content before sending it to subscribers. Common modifications include adding a footer with the name of the mailing list and a link for un-subscription. As previously remarked, altering the email content will cause DKIM to fail. One of the main issues users cite for not using DKIM is because is its sensitivity to "benign" changes to email content, such as line rewrapping and URL expansion. These common operations in email services, sometimes done for usability, can easily result in invalid signatures[14].

2.2.7 Verification tools

Various online tools can help users who would like to enable DKIM for their domain. These tools offer assistance at various stages of the enabling process:

Creation of the DNS TXT record: By providing a domain name, a selector name, and the key size, these tools can generate the appropriate DNS TXT record to be inserted into the DNS server. In addition, they also generate the private key, associated with the public key contained in the record, to be saved locally on the SMTP server. Given the importance of DKIM in authenticating your domain, it is highly discouraged to use third-party services to generate the private key, which can easily be generated locally using various libraries.

Verification of the DNS TXT record: Other tools can verify that DKIM is set up correctly, by querying the entered domain-selector pair, and analyzing the resulting DNS record, checking that it is correctly formatted and compliant with standards.

Testing DKIM functionality: Some tools instead allow a real end-to-end verification of DKIM functionality, providing an address to send an email to which will then be analyzed to understand if the DKIM signature is executed correctly.

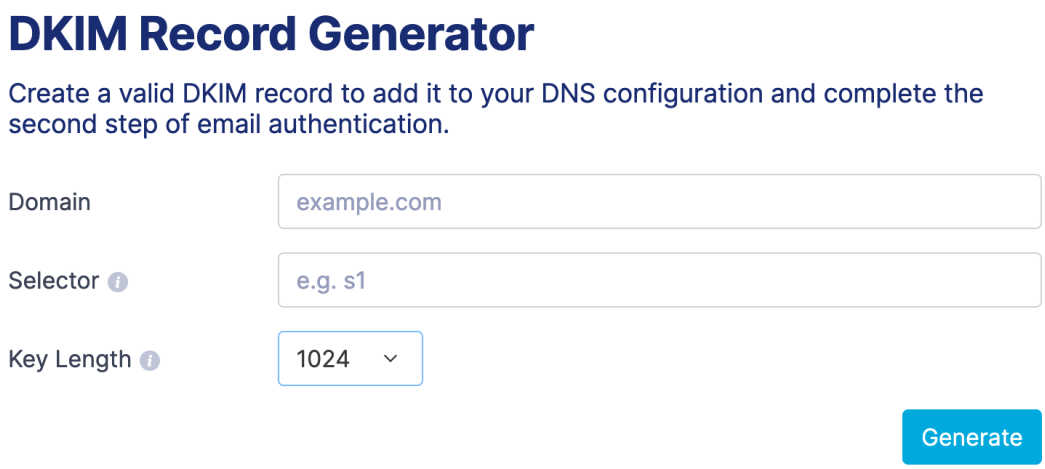

Figure 2.6. DKIM generator (Source: EasyDMARC).

In addition to these tools available online, there is a specific Python library, called "dkimpy"[15], which offers some interesting features to manage emails containing DKIM signatures. Furthermore, the library provides three default scripts, to generate the key pair and DNS record, to create the DKIM signature and header given a message in RFC 5322[16] format, and finally to verify the DKIM signature given an email in eml or txt format.

2.2.8 Testing environment

To better comprehend the scenario and the difficulty that comes with the adoption of the DKIM protocol a testing environment has been set up.

The initial idea was to set up two local email domains on the same Virtual Machine, publishing their DKIM record on the DNS server, which would have been the VM itself, and then exchange the email signed. The problem that came up with this setup was that emails were not being signed by the mail server and looking at the documentation of the mail server itself the reason behind that behavior seemed to be that the email not leaving the server, mail from local to local, are never signed.

The next step was trying to make the mail leave the server, evaluating the possibility of using a public domain and no longer a local domain in a way that the emails could have been sent to any existing mail server. The problem here arose with the registration of a public domain with a free service, given that:

- There does not seem to be a service that actually gives a public domain for free, the main ones found give a sub-domain and still at a cost, albeit a very low one.
- The secondary problem concerns the management of the DNS associated with the domain, some of these services allow the creation of the domain without, however, having access to the DNS records.
- A further problem is the fact that in order to be able to send emails from the VM server, a public IP must be associated with the MX record so that emails can be received, which would require a fixed public IP address and internal network rules to make the mail server accessible from the public network.

The solution found instead was to set up another mail server, on a different machine on the same private network. The following schema will resume the configuration:

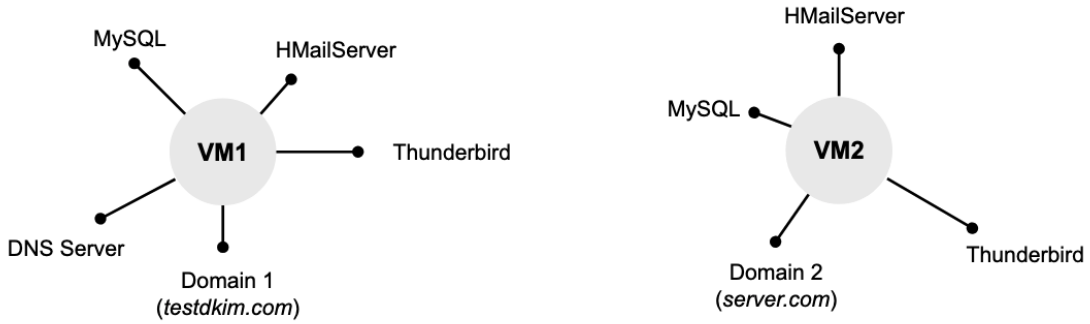

Figure 2.7. VM setup

As can be seen, there are multiple programs and services that needs to run in order to have a working environment. HMailServer is a free, open-source, e-mail server for Microsoft Windows. This tool requires the installation of MySQL as its own database server. The email domains are created directly inside the HMailServer user interface which allows even for the enable of DKIM. Lastly, Thunderbird is used as an email user agent.

From a network perspective instead, the two VMs have a different IP address in the same subnet, but the first VM (VM1), acts as the DNS server of the private network, so is configured as the DNS server of VM2.

The full documentation on how to set up the environment is available in the appendix A.

The following image 2.8 is an example of an email received with a DKIM signature inside the environment:

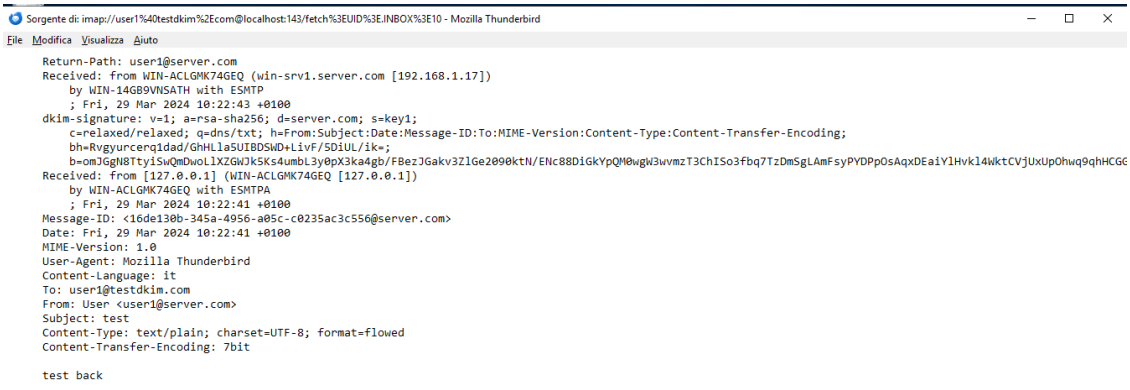

Figure 2.8. Signed Email

2.2.9 Connection with SPF and DMARC

The Sender Policy Framework (SPF) is a proposed internet standard, lastly defined in the RFC 7208 [17], that allows a recipient's mail server to verify that a message comes from a mail server authorized to send emails on behalf of the sender's domain. It is particularly useful in contrast to spoofing attacks, moreover, emails with no SPF info are more likely to be marked as spam by the main mail service providers.

To achieve this, the domain owner must first collect the IP addresses of all the email servers used to send mail. It is important to notice that if third-party services are used for email marketing or other purposes, even these domain names need to be registered.

At this point, the SPF DNS TXT record can be created, and there must be one record only for each domain. It is composed of a line of plain text that includes a list of tags, also known as mechanisms, and values which are typically IP addresses and domain names.

These are the tags:

- v: Mandatory tag, that must be written as first and identify the version of SPF. Its value must be "spf1".
- ipv4 or ipv6: Identify the mail servers authorized to send on behalf of the domain by their IP address or address range.
- a: Identify the mail servers authorized to send on behalf of the domain by their domain name.
- mx: Identify the mail servers authorized to send on behalf of the domain by their MX records.
- include: Mechanism used to recursively check the SPF record of the domain inserted as a value. This is useful when allowing third-party services to send emails.
- all: Default mechanism that applies to anything that has not been caught by other SPF tags. It must be written as the last mechanism and together with a qualifier.

These are qualifiers, used to define some sending limitations:

- +: Also known as pass, is traduced in authorized to send.
- -: Also known as fail, identify a domain not authorized to send and reject the message.
- ~: Also known as soft fail, accepts the message but tags it as SPF failed. This is particularly useful when the domain owner is not sure to have written down in the record all the domains and IP addresses and wants to avoid getting marked as spam legitimate emails.

- ?: Also known as neutral, accept the message but the host authorization is not specified.

It is important to not use the "+" qualifier in front of the "all" tag as this would open the door for all unauthorized senders, making SPF useless. This is an example of a correct SPF record that authorizes two specific IP addresses, includes Google and Yahoo as mail services, and soft fail any other mail server that is not listed:

v=spf1 ip4:192.168.0.1 ip4:203.0.113.5 include:_spf.google.com include:spf.mail.yahoo.com ~all

Once the email is received, the recipient mail server must perform a DNS query to retrieve the SPF record, and then use it to check against the "HELO" and "MAIL FROM" message identities. Specifically, the HELO is the address used by the sending server to handshake with the receiving server, while the MAIL FROM specifies the return path, which might not be the same as the From.

The connection between SPF and DKIM is provided by DMARC, fully described by the latest RFC 7489 [18], which is built on top of these two protocols, allowing a domain owner to specify how the recipient's mail server should handle messages that do not pass the so-called DMARC alignment. Specifically, after the verification of SPF and DKIM, by evaluating the DMARC policy these messages can be quarantined, rejected, or simply logged by the protocol. Another key strength of DMARC is its ability to provide reports about email activity on the domain, enabling domain owners to detect and mitigate phishing and spoofing activities. Since all of these settings must be set inside a DNS TXT record, let's analyze the different tags:

- v: Mandatory tag that describes the protocol's version, by default set to DMARC1
- pct: Represent the percentage of messages that are subject to filtering, its value must be an integer between 0 and 100. This is typically useful when first configuring DMARC so that users can start with a low percentage and then ramp up when the receiving reports seem to be okay.
- ruf: Must be followed by the URI, typically of the postmaster, that will receive forensic reports which are copies of email messages that have failed authentication.
- rua: Must be followed by the URI, typically of the postmaster, that will receive aggregate reports, which are XML docs that give statistical insights about emails that claim to have originated from that domain.
- p: Which describes the requested mail receiver policy when DMARC alignment fails. "None" does not take any action, "quarantine" flags the email as suspicious or places it into the spam folder, and "reject" rejects the message during the SMPT transaction.
- sp: Which describes the requested mail receiver policy for subdomains, again between none, quarantine, or reject.
- adkim: Is used to specify the alignment mode for DKIM, between strict or relaxed. The strict DMARC alignment requires an exact match between the domain taken from the "d=" tag of the signature and the domain present in the From field. The relaxed alignment instead, has a larger range of acceptance, as even subdomains are considered valid. For example, if "d=test.com" and the From address is "xyz@commercial.test.com" the relaxed test would pass while the strict test would fail.
- aspf: Is used to specify the alignment mode for SPF, between strict or relaxed. In this case, the alignment requires not only the "Mail From" header to be SPF verified but also the "From". Again, the difference between strict and relaxed resides in an exact match between the fully qualified domain names in the case of strict.

This is an example of DMARC record:

v=DMARC1; p=none; rua=mailto:ar@test.it; ruf=mailto:af@test.it; pct=100; adkim=s; aspf=s

2.3 Timestamping

One of the most important aspects related to the analysis of digital data during an investigation is certainly the time reference, which can indicate a precise time of existence of this data. It is obviously necessary that this reference be reliable and as precise as possible, in order to avoid the case where external malicious actors can take advantage of this information or even falsify it. Examples of the application of a time stamp concern the certification of the presence, or even absence, of a certain content within a website during a forensic copy [19], or even the verification that a digital signature has occurred on a document before the corresponding certificate has been revoked. This allows a revoked public key certificate to be used for verifying signatures created prior to the time of revocation ensuring the integrity and authenticity of the document at the time it was signed.

Trusted timestamping is a mechanism used to certify the existence of electronic information before a specific point in time. The key term here is "before", as it indicates that while we cannot determine the exact creation time of the document, we can affirm its existence prior to a given timestamp. This is typically inferred from the metadata associated with the document.

According to RFC 3161[20], which establishes a global standard for the timestamp protocol, this certification is generally performed by a trusted third-party service known as a Time Stamping Authority (TSA), even if an organization might require a TSA for internal time-stamping purposes. To maintain a trusted time reference, the TSA often uses a reliable source, such as a Network Time Protocol (NTP) server, which it queries regularly to keep its time updated and accurate.

Let's examine the general workflow of the timestamping process:

As a first step, the user wishing to timestamp a digital data must create a hash of it, usually algorithms such as SHA-256 OR SHA-512 are used for this operation. Then via an HTTP/HTTPS request, the newly produced hash, the name of the algorithm used to produce it and optionally other additional information is sent to the relevant Timestamping authority. The TSA will then produce a timestamp response, including details of the request received and inserting the so-called Timestamp Token (TST), created by digitally signing with its private key a combination of the received hash and the current timestamp. This TST serves as proof of existence of a certain type of data at a precise instant of time, conferring integrity and authenticity.

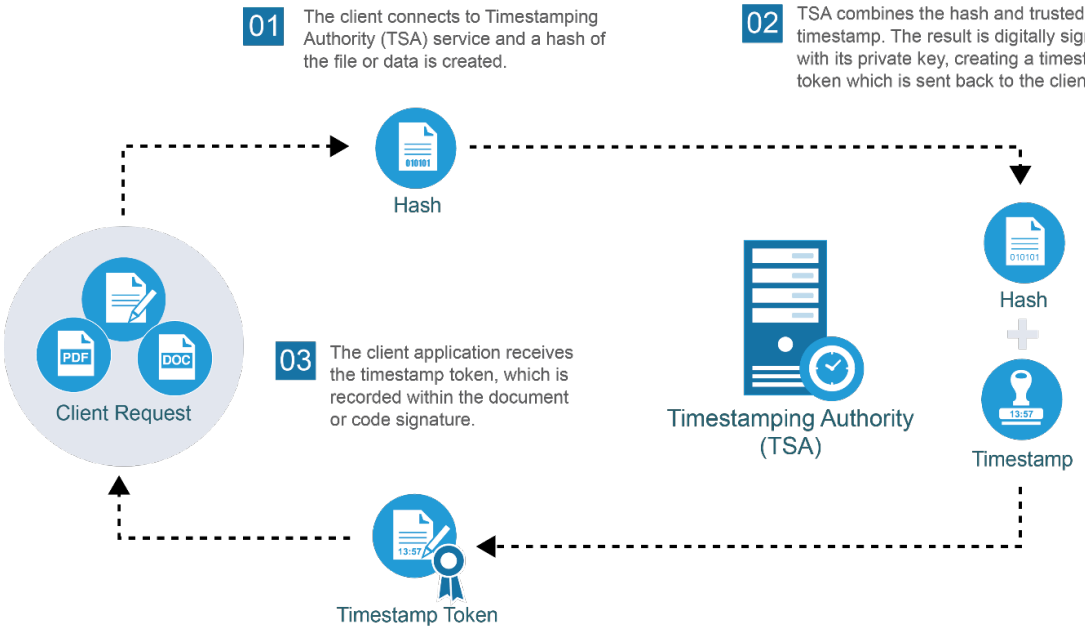

Figure 2.9. Timestamping workflow (Source: What is Timestamping?).

In the following paragraphs, the format of the timestamp request and response as defined in RFC 3161 will be analyzed in detail.

2.3.1 Timestamp request

A timestamp request is composed of several critical fields:

```
TimeStampReq ::= SEQUENCE  {
   version                      INTEGER  { v1(1) },
   messageImprint               MessageImprint,
     --a hash algorithm OID and the hash value of the data to be time-stamped
   reqPolicy             TSAPolicyId              OPTIONAL,
   nonce                 INTEGER                  OPTIONAL,
   certReq               BOOLEAN                  DEFAULT FALSE,
   extensions            [0] IMPLICIT Extensions  OPTIONAL  }

MessageImprint ::= SEQUENCE  {
        hashAlgorithm                AlgorithmIdentifier,
        hashedMessage                OCTET STRING  }
```

Figure 2.10. Timestamp request (Source: RFC 3161).

- version: This field specifies the version of the timestamp request, which is currently v1.
- messageImprint: This field includes a sequence of two objects (Figure 2.10). The first object represents the hash algorithm used, which should be a known, one way, collision-resistant algorithm. If the TSA does not recognize such algorithm or classify it as weak or compromised, it should refuse to reply with a valid timestamp token and should inform the requester in the response. The second object is the hash of the data to be timestamped, represented as an octet string. Its length must correspond to the hash value length of the specified algorithm.
- reqPolicy: This optional field specifies the policy under which the timestamp is requested. The timestamping authority might need to perform different validation, verification, or cryptographic operations based on the specified policy to ensure compliance with its requirements and objectives.
- nonce: This optional field contains a large random number, that if inserted is generated by the client and must be returned by the TSA in the response, otherwise, the response should be considered invalid. Typically, this is a 64-bit integer, which helps in preventing replay attacks
- certReq: This field indicates whether the client requests the inclusion of the TSA's public key certificate in the response. Typically, this is referenced by the ESSCertID or ESSCertIDv2 identifier within a Certificate attribute in the certificates field from the SignedData structure (Figure 2.12). This certificate is essential for the client because it is used in the process to verify the authenticity and integrity of the timestamp response.
- extensions: This field allows for the inclusion of additional information in the request. It provides a flexible mechanism for future enhancements and extensions.

A timestamp request can be easily created using the OpenSSL's "ts" command[21], which is specifically designed for timestamping. The command is as follows:

openssl ts -query -data <input_file_path>-sha256 -cert -out ts_request.tsq

In this command, "-sha256" specifies the hash algorithm used to create the hash, while the "-cert" flag requests the TSA's certificate.

The generated timestamp request can be sent via HTTP or HTTPS to the timestamping authority with "curl", and the response saved in a file named "timestamp_response.tsr", thanks to the following command:

curl -H "Content-Type: application/timestamp-query" -data-binary '@ts_request.tsq' <TSA URL>>ts_response.tsr

2.3.2 Timestamp response

A timestamp response is more complex compared to a request:

```
TimeStampResp ::= SEQUENCE  {
      status                  PKIStatusInfo,
      timeStampToken          TimeStampToken     OPTIONAL  }

PKIStatusInfo ::= SEQUENCE {
      status        PKIStatus,
      statusString  PKIFreeText     OPTIONAL,
      failInfo      PKIFailureInfo  OPTIONAL  }
```

Figure 2.11. Timestamp response (Source: RFC 3161).

Let's start with the status field of TimeStampResp, which is composed of three different pieces of information, as shown in the above Figure 2.11.

The status of PKIStatusInfo must be one of the following:

- granted (0): This indicates that the TimeStampToken has been successfully released.
- grantedWithMods (1): This indicates that the TimeStampToken has been released but with modifications.
- rejection (2): This indicates that the timestamp request has been rejected. Further information about the rejection will be provided by the failInfo field, such as unsupported algorithm, policy, or extension, incorrect data format, or other reasons.
- waiting (3): This status indicates that the request is waiting for further processing. It is a transitional status and does not conclude the request processing.
- revocationWarning (4): This is a warning indicating that the revocation of the TimeStampToken is imminent.
- revocationNotification (5): This indicates that the TimeStampToken has been revoked. This notification informs the requester that the previously issued token is no longer valid.

When the status is granted (0) or grantedWithMods (1), a TimeStampToken must be present in the response. On the other hand, when the status is any value other than zero or one, a TimeStampToken must not be present.

When the TimeStampToken is not present, the failInfo field indicates the reason why the timestamp request was rejected. Possible reasons include:

- Unsupported algorithm, policy, or extension.
- Incorrect data format submitted.
- Additional information required but not understood.
- System failure.

The statusString field of PKIStatusInfo may be used to include additional reason text, such as "messageImprint field is not correctly formatted." This field provides human-readable information to help diagnose the reason for the status provided.

The TimeStampToken (Figure 2.11) is defined in the SignedData format, encapsulated inside a ContentInfo as specified in Cryptographic Message Syntax (CMS) defined in RFC 5652[22]. As shown in the image below, there is a certificate field where the certificates of the TSA (Timestamping Authority) can be included if requested.

```
TimeStampToken ::= ContentInfo
     -- contentType is id-signedData (CMS)
     -- content is SignedData (CMS)

ContentInfo:: = SEQUENCE {
        contentType ContentType,
        content [0] EXPLICIT ANY DEFINED BY contentType }

SignedData ::= SEQUENCE {
        version CMSVersion,
        digestAlgorithms DigestAlgorithmIdentifiers,
        encapContentInfo EncapsulatedContentInfo,
        certificates [0] IMPLICIT CertificateSet OPTIONAL,
        crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,
        signerInfos SignerInfos }

EncapsulatedContentInfo ::= SEQUENCE {
        eContentType ContentType,
        eContent [0] EXPLICIT OCTET STRING OPTIONAL }
```

Figure 2.12. Timestamp Token (Sources: RFC 3161 and RFC 5652).

The eContent field of the EncapsulatedContentInfo within the SignedData structure contains an octet string that reflects the DER-encoded value of the TSTInfo. This TSTInfo includes all the information related to the provided timestamp. Below is a list of the detailed contents:

```
TSTInfo ::= SEQUENCE  {
   version                      INTEGER  { v1(1) },
   policy                       TSAPolicyId,
   messageImprint               MessageImprint,
     -- MUST have the same value as the similar field in TimeStampReq
   serialNumber                 INTEGER,
    -- Time-Stamping users MUST be ready to accommodate integers up to 160
bits.
   genTime                      GeneralizedTime,
   accuracy                     Accuracy                 OPTIONAL,
   ordering                     BOOLEAN             DEFAULT FALSE,
   nonce                        INTEGER                  OPTIONAL,
     -- MUST be present if the similar field was present
     -- in TimeStampReq.  In that case it MUST have the same value.
   tsa                          [0] GeneralName          OPTIONAL,
   extensions                   [1] IMPLICIT Extensions   OPTIONAL  }
```

Figure 2.13. Timestamp Token Info (Source: RFC 3161).

- version: This field specifies the version of the timestamp request, which is currently v1.
- policy: This field indicates the TSA's policy under which the response was produced. If present in the Timestamp Request, it must have the same value, to ensure consistency between response and request.

- messageImprint: This field must have the same value as the corresponding field in the Timestamp request. Since it represents the hash of the data to be timestamped, a different value would result in a modification of the data between the request and the response.
- serialNumber: This field is a unique integer assigned by the TSA to each TimeStampToken. This is done to permit the identification and the reference of each token uniquely.
- genTime: This field represents the time at which the TSA created the timestamp token expressed in Coordinated Universal Time (UTC). As a direct consequence this corresponds to the precise record of when the data was timestamped.
- accuracy: This value can be added or subtracted from the genTime, to obtain respectively an upper and a lower limit of the time at which the timestamp token was created. This is particularly useful to reduce the possible time range of creation.
- ordering: If the ordering field is present and set to true, it indicates that every timestamp token from the same TSA can be sequentially ordered based on the genTime field, regardless of the accuracy. This ensures a reliable chronological order of issued timestamps.
- nonce: The nonce field must be present if it was included in the Timestamp request. In such cases, it must equal the value provided in the TimeStampReq structure.
- tsa: The purpose of the tsa field is to provide a hint for identifying the name of the TSA. If present, it must correspond to one of the subject names included in the certificate used to verify the token. However, the actual identification of the entity that signed the response will always be through the certificate identifier (ESSCertID or ESSCertIDv2) within a SigningCertificate attribute, which is part of the signerInfo. This field helps in verifying the authenticity and integrity of the timestamp token by linking it to the TSA's certificate.

2.3.3 Verification of the timestamp

Validating a trusted timestamp involves several key steps:

1. First, the hash of the file being verified needs to be compared with the messageImprint field of the timestamp token.This ensures that the timestamp response corresponds to the file for which the timestamp is being checked.
2. If the hashes match, then the timestamp request and response must be cross validated. For example, if the timestamp request included a nonce, the same nonce should appear in the corresponding timestamp response. The cryptographic hash algorithm specified in the timestamp request should also match the algorithm in the messageImprint of the timestamp response, and so on.
3. At this point, the timestamp response must be validated itself. This involves verifying the digital signature on the signed timestamp data and checking the certificate chain to ensure it can be traced back to a trusted root or top-level certificate authority.

It is possible to verify the timestamp response using OpenSSL with the following command:

openssl ts -verify -in ts_response.tsr -queryfile ts_request.tsq -CAfile <certificate_chain_file>

The CAfile parameter is optional but useful for providing multiple certificates in PEM format to facilitate certificate chain verification.With this parameter is often included the certificate of the TSA's certification authority (CA).

In addition to OpenSSL, there are various online tools available for verifying timestamp tokens. For these tools to work the user needs to upload the timestamp response, the timestamp request, and possibly the original file to ensure a complete verification process. These services provide a convenient and reliable way to ensure the authenticity and integrity of the timestamped data.

Chapter 3

State of the art

In the early stages of this thesis, substantial research was conducted on the present implementation of the DKIM protocol, as well as any known studies about the verification of past DKIM signatures. Since no earlier open-source research on the last topic presented was discovered, the focus shifted towards the broader theme of validating historical documents.

3.1 DKIM Deployment

DKIM, along with SPF and DMARC, is a protocol which provide a way for users to authenticate the sender's identity, designed to prevent email spoofing, a common technique used in phishing attacks. Even though these protocols became internet standards, these attacks remain common and relatively simple to carry out. This inconsistency may be due to the adoption rate of these protocols which has been the subject of recent studies.

In 2015, research by the University of Michigan and the University of Illinois [1] found that just 47% of the Alexa Top Million mail servers used SPF policies and a very low 1% utilized DMARC policies. Because of the nature of the DKIM protocol, direct measurement of its adoption was not possible; however, it was reported that in April 2015, 83.0% of the emails received by Gmail contained a DKIM signature as can be seen in the table below 3.1.

Authentication Method	Nov. 2013	Apr. 2015	Change
DKIM & SPF	74.66%	81.01%	+6.31%
DKIM only	2.25%	1.98%	−0.27%
SPF only	14.44%	11.41%	−2.99%
No authentication	8.65%	5.60%	−3.00%

Figure 3.1. Gmail Incoming Mail Authentication. (Source:[1])

A comparable study conducted in 2022 by Tsinghua University students [2] focused more specifically on DKIM deployment. Their investigation included data from two sources: DKIM records parsed from Passive DNS datasets provided by important Chinese security organizations (Qi-Anxin and 360), as well as DKIM signatures retrieved from email headers provided by Coremail, China's largest email provider. When evaluating the adoption rate of the Alexa top 1 million domains with an MX record, they discovered that DKIM adoption (37%) was between that of DMARC (15.1%) and SPF (69.8%). They also offered statistical data on adoption rates across various top-level domains (TLDs), noting that .edu had the highest adoption rate, while at country level .ru domains had the highest number of DKIM-supporting domains, while Australia showed the highest percentage (58.6%). All this information can be found in the three figures below 3.2 3.3 3.4.

	# All Domains (%)	# MX Domains (%)
Alexa List	1,000,000 (100.0%)	748,993 (100.0%)
w/ SPF	541,008 (54.1%)	522,696 (69.8%)
w/ DKIM	280,786 (28.1%)	276,827 (37.0%)
w/ DMARC	118,468 (11.9%)	112,798 (15.1%)

Figure 3.2. Adoption Rate among Alexa Top 1 Million Domains. (Source:[2])

ccTLD	Country	MX Domains	w/ DKIM (%)
.ru	Russia	34,754	12,107 (34.8%)
.de	Germany	25,105	5,744 (22.9%)
.jp	Japan	17,740	2,467 (13.9%)
.uk	United Kingdom	15,496	7,058 (45.6%)
.br	Brazil	13,990	6,737 (48.2%)
.fr	France	11,012	4,141 (37.6%)
.au	Australia	7,452	4,363 (58.6%)
.cn	China	5,439	422 (7.8%)

Figure 3.3. DKIM Adoption Rate among Multiple ccTLDs. (Source:[2])

gTLD	MX Domains	w/ DKIM (%)
.com	371,040	143,156 (38.6%)
.org	33,271	13,787 (41.4%)
.net	33,101	9,926 (30.0%)
.info	5,531	1,443 (26.1%)
.co	3,559	1,453 (40.8%)
.edu	3,062	2,183 (71.3%)
.biz	1,955	534 (27.3%)
.gov	810	431 (53.1%)

Figure 3.4. DKIM Adoption Rate among Multiple gTLDs. (Source:[2])

Despite the fact that these email security protocols have become more widely adopted over time, mismanagement of DKIM remains a major issue within the email industry. This mishandling is caused by several problems, including:

- Invalid DKIM records: Some records have invalid tags, not defined by the standard, or no p field which identify a missing public key. Moreover, some domains have multiple records for the same selector.
- Long lifetime keys: Some DKIM keys are used for more than five years, although they should be cycled on a regular basis to reduce the danger of compromise.
- Shared keys: Some email service providers give users DKIM keys and selectors that are shared by several other users, increasing the risk of compromise due to the continuous usage.
- Weak keys: Despite the recommendation for 2048-bit keys, some domains continue to utilize DKIM keys with 1024 bits or less, which are considered weak.

Previous research [14] have investigated this absence of widespread adoption, interviewing users that revealed numerous crucial insights:

- Sensitivity to Benign Changes: Users believe DKIM is extremely sensitive to tiny, non-malicious changes in email content, such as line rewrapping, which might invalidate signatures.
- Critical Mass and Incentives: Some believe that not enough entities have adopted DKIM to make it desirable for others to implement. A critical mass is required to accelerate adoption. The benefit of publishing these DNS records, such as improved reputation, is seen as too vague, especially for domains that do not host email services. However, if more domains publish SPF, DKIM, or DMARC records, the overall advantage grows, since a greater number of incoming emails can be authenticated.
- DNS Record Management: Some users noted that certain services do not have direct control over their DNS records. Implementing SPF, DKIM, or DMARC requires additional cooperation with their DNS providers, which can increase overhead and complexity.

- Concerns About System Disruption: Because of the unfamiliarity with these protocols, there is concern that the currently working email system may be disrupted. While this worry is particularly relevant for DMARC, it also has an impact on DKIM adoption.

3.2 Long Term Verification

Since no previous work has been conducted explicitly on verifying old DKIM signatures, let's look at how documents can remain verifiable for many years after their initial signing.

Verifying the authenticity of a digital signature usually involves several steps that rely on cryptographic techniques and standards. Here is an overview of the whole procedure:

- The process begins by obtaining the signed document and its associated digital signature.
- The signer's digital certificate, which relies on Public Key Infrastructure (PKI), must be obtained. PKI is a framework that manages digital certificates and public-private key pairs. Certification Authorities (CAs), trusted entities that issue these certificates, play an important part in this process.
- The certificate's validity is verified by following the certificate chain up to a trusted root CA. The root CA's certificate must not be present in the chain because it is inherently trusted by the verifier. This involves:
  - Verifying Digital Signatures: The digital signature of each certificate in the chain is validated.
  - Checking for Revocation: Certificate Revocation Lists (CRLs) or the Online Certificate Status Protocol (OCSP) are used to determine whether any certificate in the chain has been revoked. CRLs are lists of revoked certificates, containing the reason and date of revocation, signed by the issuing CA. The certificate usually includes information on where to obtain the appropriate CRL. OCSP involves querying an OCSP server to obtain the certificate's current status.
- Once the certificate is validated, the digital signature on the document can be verified with the signer's public key retrieved from the digital certificate.

However, since many business applications require their signed documents to be verifiable months or even years after signing, the question arises: Will this digital signature always be verifiable in the future?

There are several factors that may prevent the completion of the verification process, and the majority of them are related to the used certificates:

- Due to a loss or job change, the signer's certificate may be revoked.
- Since certificates have limited lifetime, the signer's certificate may be expired.
- The same expiration may concern the issuing CA's certificate, even though they usually have a longer lifetime.
- The used cryptographic algorithms may become weak, making the signature vulnerable to compromission.

These challenges have been researched since the early 1990s, evolving from early cryptographic studies that led to online methods incorporating time stamps and the basic notary protocol, commonly known as digital notarization. This foundational work, initiated by Stuart Haber and W. Scott Stornetta, established a method for linking digital signatures with time stamps to provide proof of document integrity and chronology.

While traditional time stamps offer a way to verify when a document was signed, notary protocols provide a direct (one-way) link between the signature and the confirmation, thereby ensuring proof that the signature was created before the notary issued the confirmation.

Building on this basic technique, Arne Ansper, Ahto Buldas, Meelis Roos, and Jan Willemson's paper "Efficient Long-Term Validation of Digital Signatures" [23] proposed many improvements to solve the issues associated with long-term verification. Their research optimized the computational and storage needs for retaining signature validity over long periods of time, as well as proposed methods for incremental validation.

The concept of Long Term Validation (LTV) or Long Term Signature (LTS) expands the original notary protocol's objective by preserving all important information required to verify the signature with the signature itself, ensuring that the verification procedure can be executed in the future. However, simply retaining this information is not enough. It is necessary to demonstrate that the signer's certificate was valid at the time of signing, meaning it was neither revoked nor expired. To achieve this, the process involves several key steps as illustrated in the image below:

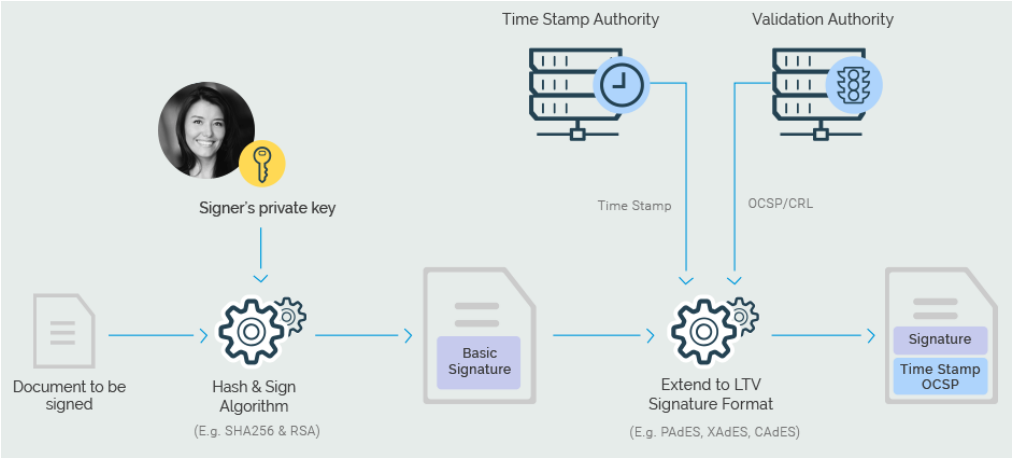

Figure 3.5. LTS Process (Source: LTS).

- Initially, a document is signed using the signer's private key through a hash and sign algorithm, such as SHA256 & RSA, which produces a basic digital signature.
- A Time Stamp Authority (TSA) provides proof of the signing time by issuing a trusted timestamp, which has higher validity than relying exclusively on the signer's claimed time.
- To demonstrate that the signer's certificate was not revoked at the time of signing, a Validation Authority utilize either OCSP or CRL technology.

This method converts the basic signature into a Long-Term Validation (LTV) signature format that follows standards such as PAdES, XAdES, and CAdES.

Regarding the long-term preservation of digital signatures, standards provided by ETSI and ISO play a crucial role. These standards cover the preservation of signed PDFs (ISO and ETSI) and other document types (ETSI). ETSI's Long-Term Preservation (LTP) framework consists of a series of standards and technical specifications that guide developers on how to format digital signatures and enhance them for successful long-term preservation. These standards ensure that all necessary information is retained to maintain the validity of the signature over time. Some of these standards will be discussed in the next subsection.

According to a study conducted in this paper [24] on existing EU suppliers of long-term preservation services for digital signatures, PAdES is the most widely utilized format. This popularity is primarily caused by the widespread use of PDF files in digital signatures. Additionally, the CAdES and XAdES formats, which embed digital signatures within generic files or XML documents, respectively, are used in more than 75% of the examined solutions, demonstrating their popularity and widespread use.

3.2.1 PAdES for Long Term Validation

The ETSI TS 102 778-4 V1.1.2 [3] is responsible for profiling the electronic signature formats specified in ISO 32000-1, ensuring the Long Term Validation (LTV) of PDF signatures. This involves detailing how to embed validation information in a PDF and securing it with timestamps, enabling the verification of a PDF signature long after it was initially signed. While PAdES stands for PDF Advanced Electronic Signature, PAdES-LTV is the enhanced standard for long-term validation.

As outlined previously, verifying a digital signature requires data like CA certificates, Certificate Revocation Lists (CRLs), or Online Certificate Status Protocol (OCSP) responses, typically provided by an online service. If a signature needs to be verifiable long after its creation, such as after the signing certificate has expired, the original validation data might become inaccessible, or there may be uncertainty regarding the validation data used during the initial verification.

The profile previously mentioned, uses a data structure known as the Document Security Store (DSS) to store the necessary validation data for signature verification, optionally including Validation Related Information (VRI) to link this data to a specific signature. The structures of DSS and VRI are reported in the image below 3.6.

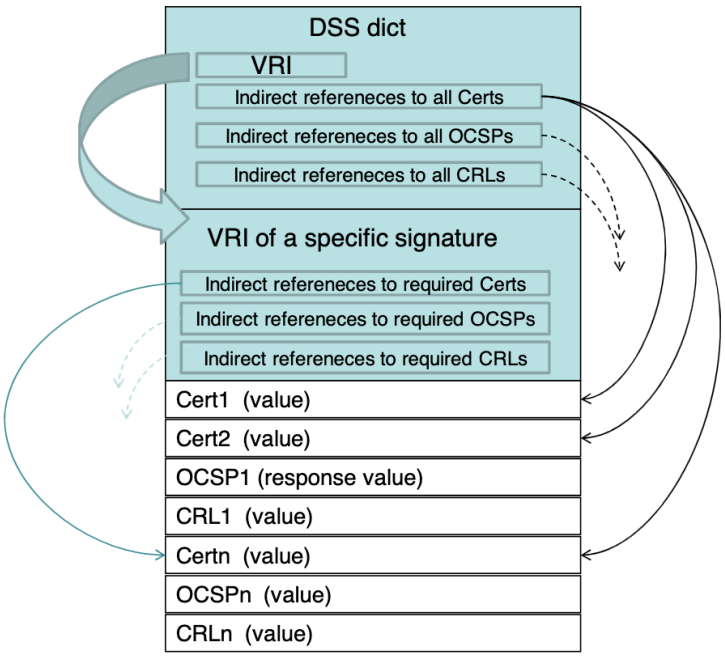

Figure 3.6. DSS and VRI structures (Source:[3]).

To mitigate issues like inability to connect to remote validation servers, especially when offline, the validation-related information must be separate from the signature itself. This allows adding validation information to an already signed PDF at any later time. A new "security store" is defined within the PDF to house these components. This separation also optimizes storage by allowing common components (e.g., certificates, revocation lists) to be stored once and referenced as needed across multiple signatures, thereby reducing the PDF's size.

In the DSS dictionary, the Certs, OCSPs, and CRLs arrays refer to certificates, OCSP responses, and certificate revocation lists used for validating any document signatures. Meanwhile, the VRI dictionary contains Signature VRI dictionaries for each specific signature's validation information. If there is only one signature in the document, this might seem redundant. Each VRI dictionary entry key is the base-16-encoded (uppercase) SHA1 digest of the relevant signature. Below is an example of the actual DSS and VRI structure 3.7.

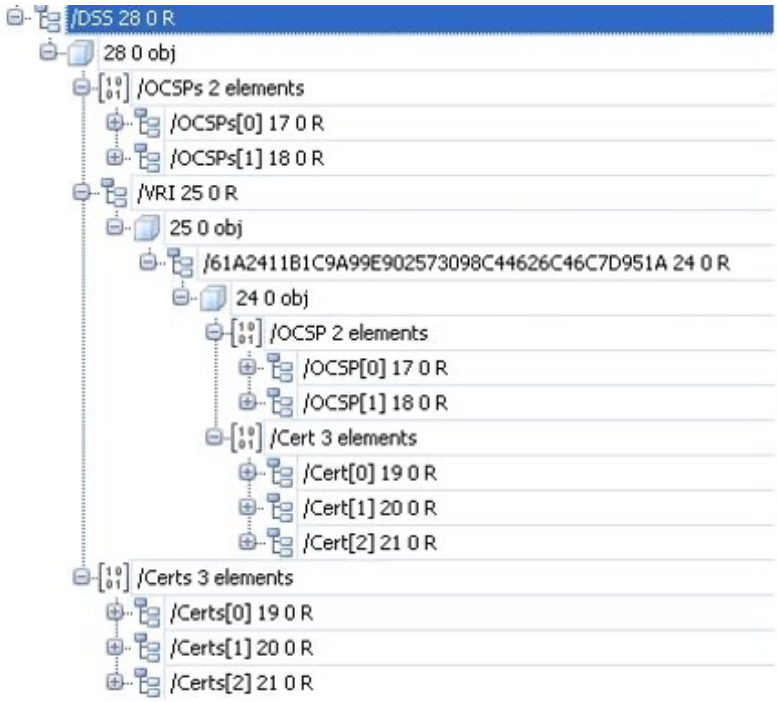

Figure 3.7. Real structure of DSS and VRI (Source: StackOverflow).

To extend the document's protection over time, a Document Timestamp is applied. This timestamp also secures the DSS by linking it to the relevant document. The collection of the DSS and the initial verification of the signature occur before the time indicated in the first Document Timestamp, which can then serve as the assumed signing time during re-verification using the protected validation data from the DSS. Timestamps are signed, so their own validation data can expire. The DSS may also include signature validation data related to the signature timestamp.

To further extend the document's protection beyond the last Document Timestamp, additional DSS information can be added to validate the previous timestamp, along with a new Document Timestamp. Below is a comparison between a document with a single LTV and one with repeated LTV:

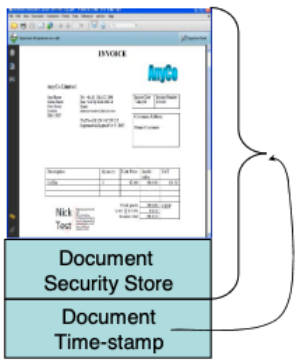

Figure 3.8. Single LTV. (Source:[3])

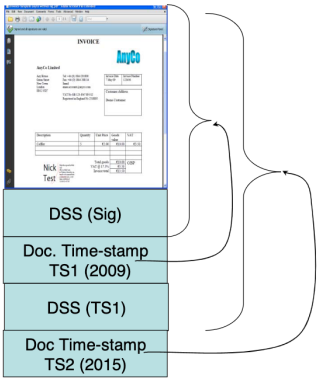

Figure 3.9. Repeated LTV. (Source:[3])

Validation of such documents should follow this process:

1. Validate the "latest" Document Timestamp with current validation data.

2. Validate the "inner" Document Timestamp at its time using the validation data present in the previous DSS.

3. Validate the signature and its Timestamp at the latest innermost LTV Document Timestamp using the DSS-stored validation data.

3.2.2 CAdES for Long Term Validation

As described in the last paragraph, PAdES LTV is used for PDF documents but is sometimes necessary to transition to CAdES due to the nature of the file to be signed, different regulatory environments or specific organizational needs. CAdES is designed for CMS-based (Cryptographic Message Syntax) signatures and to ensure long-term validation (LTV) it offers the CAdES-X Long profile, a reliable framework for maintaining the signature validity over time.

CAdES-X Long improves the CAdES-C format by adding the certificate-values and revocation-values attributes. The certificate-values attribute envelopes the complete certificate chain required for verifying signatures, while the revocation-values attribute holds the CRLs and/or OCSP responses necessary for validating signatures. This guarantees a dependable archive of certificate and revocation data essential for validating a CAdES-C signature, thereby preventing the loss of this information. Using CAdES-X Type 1 or Type 2, one can attach a timestamp either to the entire CAdES-C content or specifically to the certification path and revocation information references.

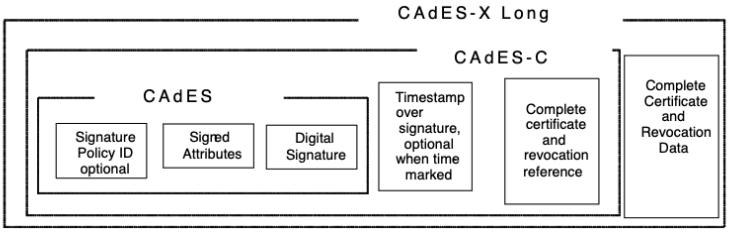

Figure 3.10. CAdES-X Long (Source:[4]).

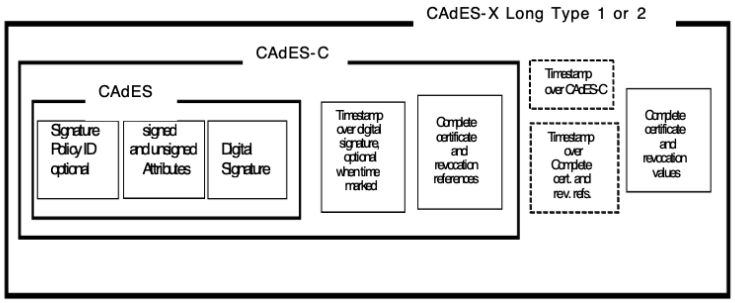

Figure 3.11. CAdES-X Long Type 1 or 2 (Source:[4]).

By using the components discussed so far, all the necessary data needed for the signature validation is retained, even after long time after its creation. Certificates and revocation data are essential for confirming the signer's identity and its validity status at the moment of signing, therefore they are embedded within the document. Furthermore, timestamps demonstrate that the signature and the related validation data existed at a certain point in time, preventing backdating and other forms of tampering. All of these attributes are considered critical, particularly for regulatory compliance.

In addition to CAdES-X Long, another key profile is CAdES LT, which is detailed in ETSI TS 101 733 V2.2.1 (2013-04)[4]. CAdES LT is intended to ensure that all required validation data

(certificates, CRLs, and OCSP answers) are incorporated in the signature at the time of signing. It can be built on top of any format among CAdES-T, CAdES-C, CAdES-X Long, CAdES-X Long Type 1 or 2 or a CAdES-A. This profile is essential for situations where long-term availability and integrity of the validation data are required, ensuring that the signature can be verified even if the original validation data sources are no longer accessible. CAdES LT provides a solid foundation for long-term validation, similar to PAdES LTV, by maintaining all necessary data within the signature itself.

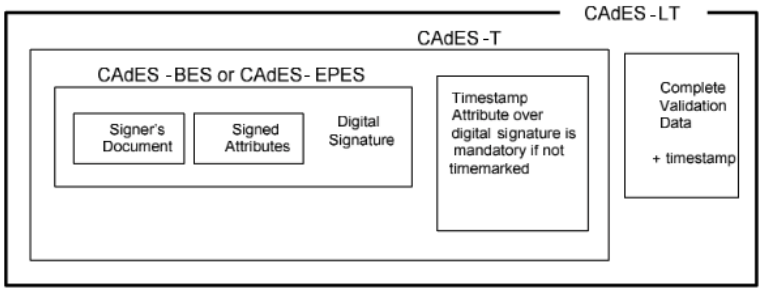

Figure 3.12. CAdES-LT (Source:[4]).

As cryptographic standards evolve and previous algorithms become obsolete, the basic CAdES LT and CAdES-X Long profiles may no longer be sufficient for indefinite long-term validation. This is where CAdES-A (Archive) comes in, offering an improved solution for long-term validation. CAdES-A expands on these profiles by periodically placing archival timestamps to the document. These timestamps include all past validation data, including older timestamps, resulting in a continuously updated chain of trust inside the validation framework. A new archival timestamp is added only when the certificate of the previous archival timestamp is about to expire. This additional timestamp from a trusted third party confirms the content's authenticity at the moment of updating. All validation components required to validate the new TSA answer ('archival timestamp') are also incorporated into the signature.

CAdES-A, introducing new archival timestamps, ensures that the document is verifiable even when cryptographic technologies improves and previous algorithms become obsolete, despite the fact that the archived signature grows in size over time. This proactive approach to preserving the document's integrity overcomes any vulnerabilities that could arise from relying solely on outdated cryptographic methods. As a result, CAdES-A adds an extra layer of security and assurance, making it an excellent solution for situations requiring extensive use of long-term validation.

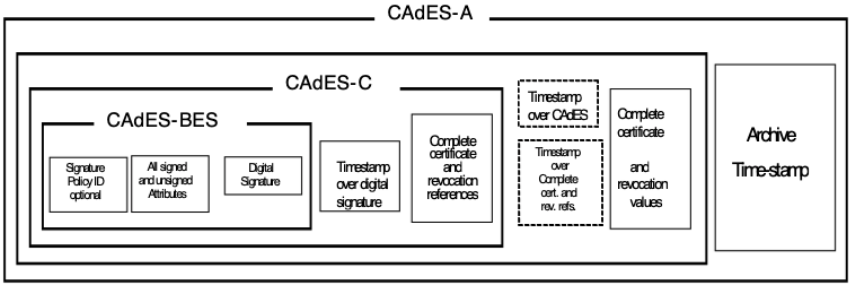

Figure 3.13. CAdES-A (Source:[4]).

To verify the archival signature, one must first validate the original signature. Then, the verifier should check that each subsequent timestamp appropriately validates the prior one. Finally, the last timestamp is validated 'at the current moment,' which may imply contacting online revocation sources.

Chapter 4

Solution

Given the state of the art of the long-term verification environment, and considering the final goal of this thesis, which is to enable the verification of DKIM for emails received in the past, different solutions were initially analyzed. In the following sections, each solution will be discussed along with its pros and cons, even though the work proceeded with the full development of only one of them. In the end, thanks to a temporary license, it was possible to test and compare the solution developed with two Metaspike tools, both of which operate in the computer forensic field and offer similar functionalities.

4.1 DKIM & CAdES

This initial solution aimed to incorporate the DKIM signature into a format like CAdES. This format, in particular, permits encapsulating any kind of document as raw data inside it, and it could fit the requirement of enveloping an .eml file, representing an email. Thanks to the various formats described in the previous chapter, it could also be possible to extend this signature for a long time, permitting validation at any moment after the reception of the email.

This was the possible scenario imagined for a mix of the two protocols:

1. The email is composed and sent to the sender's SMTP server.
2. The sender's SMTP server signs the email and puts the .eml file and the signature into the CAdES format adding the DKIM record correlated to the private key used as "validation data".
3. The resulting file is sent as an attachment to the email itself.
4. The receiver's SMTP server can read the CAdES format and verify if the signature is valid.
5. Depending on the result of the verification, the email is junked or received in the inbox.

Since DKIM is not using PKI for its keys, there is no certificate nor revocation data that could be extracted from this architecture, so the only "verification data" that can permit the future verification of the signature is the DKIM record itself.

4.1.1 Advantages

One of the main advantages of this solution is the increased security and integrity it offers to email communications. Thanks to the fact that the CAdES format accommodates both the DKIM signature and the email itself, these two objects become even more connected, making it possible to detect if a malicious user has modified one of them. In this way, the user sending an

e-mail can achieve a higher level of security, as there is the certainty that the alteration of the e-mail would be recognized.

Permitting emails to be validated long after they have been received is particularly valuable for legal and compliance purposes, where it is essential to prove the authenticity of communications over extended periods.

Finally, by using already existing standards and technologies, the transition to this standard could be faster. The CAdES format is already a widely recognized and used standard for advanced electronic signatures. For this reason, many companies and organizations may already have the infrastructure and software to use it, and thus the effort required for the transition would be considerably reduced. On the other hand, companies that were already using CAdES but not DKIM could be enticed to adopt this protocol, further enhancing the security of their communications with minimal effort.

4.1.2 Potential problems

The first major stumbling block in adopting this solution concerns the added complexity in the management of e-mails. Whereas the standard DKIM signature was simply inserted within the source code of the e-mail, the CAdES format produces an actual file that needs to be sent to the recipient as an attachment. This would certainly affect the processing work of the sending and receiving servers, which could lead to longer times required for the transmission of the e-mail and a consequent higher load on the servers themselves, especially during periods of higher utilization. In addition, this solution would require changes to the current email infrastructure, if servers do not support the CAdES format natively and require further adaptations, which may be unfeasible for companies due to compatibility issues or lack of sufficient resources.

As emerged from the interviews carried out in the previously mentioned paper on the use of DKIM[14], one of the main reasons that hold back the possible adoption of the protocol is precisely the need to modify one's currently functioning infrastructure to adapt to the use of this new technology. This could therefore represent an additional obstacle and could decrease the already low use of the protocol.

Another potential problem concerns the increased weight of the e-mail due to the CAdES attachment present. Given the large number of emails that a company sends and receives daily, this could result in storage problems, running out of space sooner and thus leading to an economic cost to increase it.

4.2 DKIM & TSA

The second solution came up thinking about a lighter environment, removing the usage of a fixed format like CAdES but trying to keep the functional pieces needed for future verification. This time the idea was to incorporate a Timestamp Authority inside the DKIM signature, permitting in some way users to extend the validity of the signature as wanted. The scenario imagined was something like this:

- The email is composed and sent to the sender's SMTP server.
- The sender SMTP server computes the DKIM signature and sends the email.
- The receiver's SMTP server queries the DNS server to obtain the public key needed to verify the signature and then will Timestamp this query reply together with the email content.
- The signature is verified, and the timestamp response, together with the query reply is conserved as attachments of the email.
- Depending on the result of the verification, the email is junked or received in the inbox.

The email content and the query reply may be combined together to send a single timestamp request in this way:

- The email content, including headers and body, is hashed to create a digest. It's important to notice that this hash is not the same contained in the DKIM signature since it covers the whole mail received at recipient side.
- The DNS query reply is also hashed to create another digest.
- These two digests are concatenated into a single hash.

Given this architecture, when a user wants to verify the signature in a future time, it has all the information needed, the email signed, the public key contained in the DNS record, and a timestamp response that certifies the public key at the time of usage correlated to the email received.

4.2.1 Advantages

As mentioned in the introduction this solution weighs lighter than the previous one. Here the only behavior changed is the one on the receiver side, which will always query the DNS server to obtain the public key, but then the timestamp is introduced to permit a kind of crystallization of the important data.

Additionally, given the process of timestamping described in the background chapter, the email receives another digital signature which is placed by the Timestamp authority chosen. In this case, an additional layer of security is inserted, protecting the integrity of the email message itself, ensuring that both the content and signatures are immutable and verifiable over time.

Lastly, thanks to the timestamp, both senders and recipients have a verifiable proof of the email's origin and receipt. This makes much harder denying having sent or having received an email, which is a feature particularly beneficial in legal and compliance contexts.

4.2.2 Potential problems

Even if less than before, another change in the email infrastructure is needed at this time. As described for the previous solutions, these kinds of changes are not always well seen as they introduce the uncertainty of disrupting working and functional services. The major problem here comes from the choice of the timestamp server.

Nowadays given all the regulations in terms of digital or electronic signature, there is a big difference between one timestamp server and the other. Depending by the chosen server the timestamp released can have legal values or not and most of the time the service of timestamping does not come for free. Typically, those kinds of services sell timestamps as tokens, having a certain price for a fixed number of timestamps. Moreover, since this solution requires each received mail with DKIM signature to be timestamped singularly, the number of timestamps requested daily would be extremely high.

By a statistic conducted by the Radicati Group, a company that provides quantitative and qualitative information on a worldwide basis, the average number of emails received by a single user in a day is around 80[25]. On the other hand, confronting various timestamping services the average price for a timestamp token is expected to be from 0,20 € to 0,50 €, depending primarily on the amount of token bought. In the worst-case scenario of 100 mail received and 0,50 €, for each timestamp this would result in 50 €, daily for each worker. Considering the number of employees of each company this solution would practically not stand as too cost consuming.

The worst scenario described here is a free protocol like DKIM which in order to adopt this new update and enhance its reliability for future verification, will now require a sort of fee to be paid to a timestamp service. Obviously, this would decrease significantly the already low adoption of the protocol, possibly creating more problems than the pros provided.

Ultimately, relying on a TSA introduces a dependency that can become a single point of failure. If the TSA is unavailable or compromised, it could disrupt the email verification process. This entity must also be highly secure and trustworthy given that any compromise in its security could undermine the entire system's trust.

4.3 DKIM Logger

The primary challenge with the two previously described solutions lies in the necessity to modify existing, functional infrastructure to accommodate potential updates. In this way, changing the protocol and integrating any other technologies could discourage the adoption of the protocol, which is already not so high, given the burden and the effort required to implement those changes.

Moreover, DKIM was originally designed to achieve an immediate verification of the signature for antispam purposes and was not intended to guarantee or ensure its long-term validity. This limitation suggests that the real solution might need to come from something external. From here started the idea of developing an external tool, that does not interfere with the protocol itself but that could be complementary and built to solve the problem highlighted by this thesis.

As discussed earlier, verifying a DKIM signature requires only two elements:

- The email to be verified.
- The public key associated with the private key used to sign the email, which is published as a TXT record in the DNS.

However, a significant problem arises when these public keys are sometimes rotated or removed, so they may not be available anymore in the DNS server when needed. How can a user retrieve this key if it's no longer available?

4.3.1 Log generation and preservation

The initial solution tried to solve this problem in a simple and straightforward way: logging the DKIM records of specific domains. During the normal verification of the DKIM signature, the recipient's SMTP server performs a DNS query to retrieve the public key. In a similar way, the first operation done by the tool was logging the DKIM record of a list of couples (selector, domains), provided by the user.

The command used to perform those queries is dig, which stands for Domain Information Groper. Dig is a versatile tool used for verifying and troubleshooting DNS issues but also to perform DNS lookups. This is an example of queries performed by the tool:

dig selector._domainkey.domain txt +noall +answer

In this command:

- *selector* and *domains* are placeholders that change with each iteration reading all the list of couples provided by the user
- The txt flag specifies that the query is aiming at a TXT record type, as DKIM public keys are stored in TXT records
- The +noall option tells dig not to display the default output, which includes lots of additional and non-relevant information, but only shows the specific details.
- The +answer option specifies dig to display only the answer section of the DNS response, containing the interesting information.

The output of the command was designed to be printed inside a txt file, with each entry preceded by the timestamp of the execution date. By following this format, each (selector, domain) pair has its own txt file that tracks the daily evolution of the corresponding DKIM record. This approach allows for the creation of a big database of DKIM records, organized by domain and selector, which can be referenced later to recover the public key used at a specific time.

However, as discussed in the context of long-term verification, it is also important that such data is not only logged but also validated, ensuring its existence can be certified at a specific point in time. To achieve this status a Timestamp authority was introduced into the process. While there are various types of Timestamp Authority and many companies that offer timestamping services, which will be better described in a later section, the initial TSA chosen for this tool was a free service provided by freeTSA.org.

Thanks to this service a user is able to timestamps electronic document or websites content, by providing a URL, via a TCP-based connection or, alternatively, by using an online form to upload files and download both the timestamp request and the timestamp response.

For the purpose of our tool, a simple timestamp applied to the txt file containing the DKIM records was sufficient. The commands used to generate the timestamp request and to send it to the TSA (*"openssl ts"* and *"curl"*) were previously described in the "Timestamp Request" paragraph of the Background chapter. The output files generated by this process are saved in the same directory as the txt file, following a similar naming convention, to permit an easier association between them. Moreover, the timestamp recorded in txt file before the DKIM record allows for a double check against the timestamp token issued by the TSA, as the two actions are executed sequentially.

Furthermore, since the timestamp request includes the hash of the file at the time of request, this hash is performed and added to the txt file, before writing the new record. This provides an additional way to check the integrity of the whole system. Here is an example of what the TXT file might look like on the second day of the tool's execution:

```
1  New execution at: Jul 17 09:18:47 2024 GMT
2
3  ; <<>> DiG 9.10.6 <<>> google._domainkey.spotify.com txt +noall +answer
4  ;; global options: +cmd
5  google._domainkey.spotify.com. 300 IN TXT "v=DKIM1; k=rsa;
       p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCSiZTnXPysNqgPjX0hSeHC8wVY62
6  ooFHCcC/Jo8TtWRNMoO75qdRUT2qnuF3+gc4/NI2qWMy0yMYqliLHviLMNeKzWjkOaR3VaVH
7  rSuDf+8WSgaWk0HJS5cDStqjknBpd8tmdQDXbSUE8WRUxTzQuWVUmMqU83kMrSygzqmunNiw
8  IDAQAB"
9
10 Previous file SHA512 Hash:
       f4138d3eda05b3c361ac1d56ae4f1cf9470b4428a4f18b575c7a4efd3c92180ef7d4
11 323b75ddf29ccf2643964386b2b297952393f9d7adf89efef2ca78d9a80d
12
13 New execution at: Jul 18 08:20:50 2024 GMT
14
15 ; <<>> DiG 9.10.6 <<>> google._domainkey.spotify.com txt +noall +answer
16 ;; global options: +cmd
17 google._domainkey.spotify.com. 300 IN TXT "v=DKIM1; k=rsa;
       p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCSiZTnXPysNqgPjX0hSeHC8wVY62
18 ooFHCcC/Jo8TtWRNMoO75qdRUT2qnuF3+gc4/NI2qWMy0yMYqliLHviLMNeKzWjkOaR3VaVH
19 rSuDf+8WSgaWk0HJS5cDStqjknBpd8tmdQDXbSUE8WRUxTzQuWVUmMqU83kMrSygzqmunNiw
20 IDAQAB"
```

Figure 4.1. Example of tool output.

Of course, during the first execution, since the file does not yet exist, there is no file hash to include. The format of the timestamp, however, remains consistent with that used in the timestamp request/response, to ensure better comparability.

4.3.2 Traffic capture

To add a forensic perspective to this tool, permitting to add an additional source of authenticity to this log, the traffic of the DNS queries performed by the dig command and the traffic with the timestamp server was captured separately thanks to another command, tcpdump.

Tcpdump in fact is a command line utility that allows to capture and analyze any kind of network traffic allowing also to choose some filtering options, not having to capture undesired traffic. Here is the command used for the capture:

sudo tcpdump -i en0 (port 53 or tcp port 443) -w pcap_file

In this command:

- sudo runs the command with administrator privileges, which is necessary for capturing network traffic on a network interface. Because of this command even the full script needs to be called as a superuser otherwise, it won't be executed.
- -i en0 specifies the network interface to listen on, in this case, en0 is the name of the interface
- (port 53 or tcp port 443) is the filter expression that tells tcpdump what kind of traffic to capture. In this case, port 53 is used to capture DNS queries while tcp port 443 is used to capture HTTPS traffic, which is the protocol used for the communication with the Timestamp server.
- -w pcap_file tells tcpdump to write the captured packets to a file, specified by the path of the pcap_file variable, rather than displaying them on screen. This file created has the pcap extension, which can be analyzed using various network tools.

This capture process ensures that each (selector, domain) pair has its own proof that the communication has been executed correctly. Additionally, the information exchanged can be verified inside the packets captured. Since DNS traffic is not encrypted, by utilizing a tool such as Wireshark, that helps users inspect packets contained inside a pcap file, it is possible to clearly see the query sent to the DNS server and the corresponding reply, where the DKIM record is contained.

However, the situation is different for communication with the timestamp server. Depending on the TSA chosen, the server might be accessible via HTTP or HTTPS. The TSA provided by freeTSA.org, which was initially selected, only accepts encrypted traffic. The challenge here was that by normally inspecting the packets no information about the timestamps could be retrieved, as all the contents was encrypted. In order to solve this problem, two solutions were considered:

1. The first solution required using a proxy. In this case, the proxy would have intercepted and decrypted the HTTPS traffic before re-encrypting it to send it to the destination. There are various tools that can help in achieving this scenario, tools like Burp Suite or mitmproxy can generate their own SSL/TLS certificates and present them to the client, while they will establish a separate connection with the real server. Once the proxy's certificate is installed on the client, all the HTTPS traffic passing through the proxy can be decrypted and inspected. Even if theoretically this solution would have worked, its implementation inside the scenario of the tool was perceived as a little bit too complex. Moreover, the proxy methods involve a modification of the traffic which may not work in all scenarios, as some clients might detect and block the proxy.
2. The second solution, definitely much lighter, involved the usage of the SSL/TLS session keys. By simply setting an environment variable on the host system, a log file can record

all the session keys used during HTTPS communications. After having set the environment variable and captured the traffic, by specifying the path to the ssl_keys.log file inside the preferences of Wireshark, the tool itself is able to decrypt the traffic and display it in plaintext.

This is the command used inside the script to set the environment variable:

os.environ["SSLKEYLOGFILE"] = ssl_key_log_file

Where the "ssl_key_log_file" contains the path to a file created with the name "ssl_key.log"

Here instead is a screenshot of the traffic captured during an iteration of the tool. As shown, both the DNS query request and response are visible, and the HTTPS traffic has been decrypted to highlight the content of the timestamp response, which in this example contains a granted token.

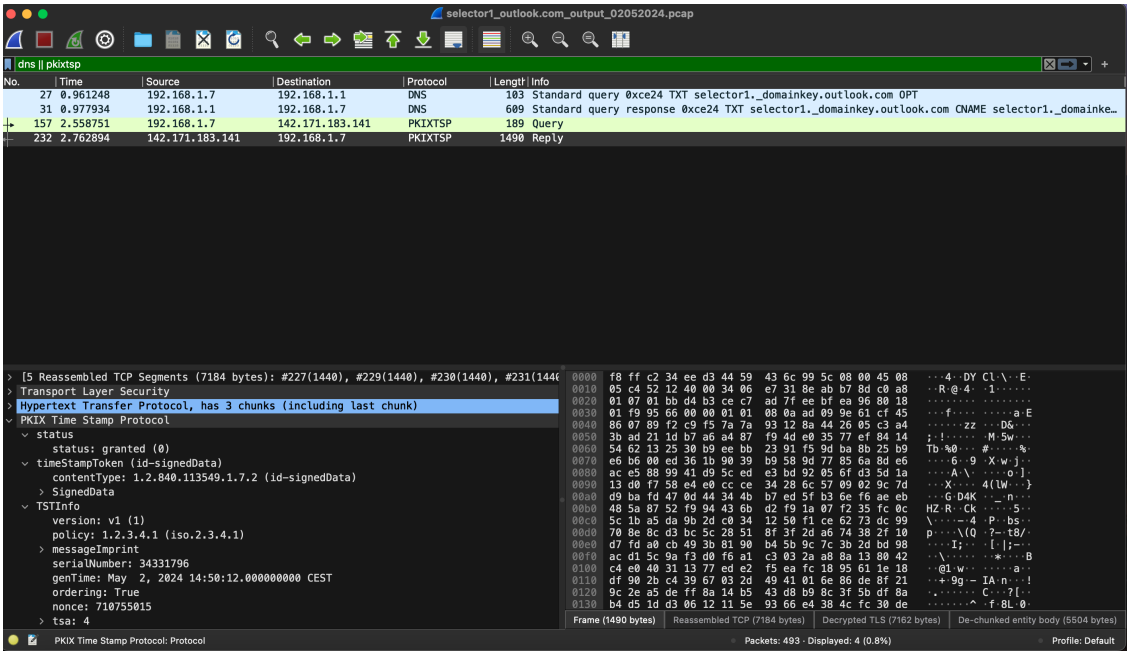

Figure 4.2. Wireshark capture of an iteration of the tool

4.3.3 Output files

To summarize the information covered so far, this section is utilized as a recap of all the files generated by the tool. As already written, each (selector, domain) pair, has one txt file in which the DKIM record is written and updated daily. This file is named:

selector_domain_output.txt

In addition, three new files are created daily for each pair, stored in the same output folder as the txt file:

1. *selector_domain_output_date.pcap*: This file contains the packets captured by tcpdump.
2. *selector_domain_output_date.tsq*: This is the timestamp request generated by the tool.
3. *selector_domain_output_date.tsr*: This is the timestamp response received from the TSA.

As shown, each file follows the same naming convention, which helps to clearly distinguish between different (selector, domain) pairs. The date in the file name instead is used to differentiate between the output files generated for the same pair on different days. Although these files may seem numerous, they occupy very little disk space. The heaviest files here are the timestamp response and the packet capture which are approximately 5KB each. In contrast, the txt file and the timestamp request are typically only a few hundred bytes.

4.3.4 eIDAS and EU rules

At this point, given the usage of a Timestamping authority in the tool and the different possibilities offered by such services, it is important to consider EU regulations regarding Electronic Signatures. This is particularly relevant because most of the service providers offering a certified electronic signature often provide a service of certified timestamps, additionally, those timestamps are even signed by the respective TSA.

The eIDAS regulation, which stands for electronic identification and trust services for electronic transactions in the internal market, became effective on 1 July 2016 and established a unified legal framework across the EU for electronic signatures and other trust services. It applies uniformly across all the EU member states, without the necessity for national implementation, as a member state may choose to adopt a standard implemented by another member while still retaining legal validity.

Regarding electronic signatures, eIDAS distinguishes between:

- Electronic Signature: This is the most basic type, broadly defined as electronic data that is attached to other electronic data, used by the signatory-the individual creating the electronic signature-to sign. This could include actions like typing the name into a document or an email using an online platform. While this type of signature is relatively easy to create, it offers minimal security and legal assurance.
- Advanced Electronic Signature (AdES): This electronic signature fulfills additional requirements in respect of the electronic signatures. It must have a unique link to the signatory, the possibility of identifying the signatory, be created using a private key under the signatory's sole control, and finally ensures that any changes in the data must be detectable.
- Qualified Electronic Signature (QES): This is an extension of the AdES, and represents the highest level of security and legal validity among electronic signatures. It requires the signature to be generated by a qualified electronic signature creation device and based on a qualified certificate for electronic signatures issued by a qualified trust service provider (QTSP). This signature is particularly valuable for signing contracts, legal agreements or others documents with a high level of trust needed.

The importance of these standards is highlighted by their legal impact, as outlined by the Article 25 of eIDAS regulation, which states:

"An electronic signature shall not be denied legal effect and admissibility as evidence in legal proceedings solely on the grounds that it is in an electronic form or that it does not meet the requirements for QES.

A QES shall have the equivalent legal effect of a handwritten signature."

The creation, validation and preservation of electronic signatures or electronic timestamps are activities provided by a Trusted Service Provider (TSP). These entities can operate as either qualified or non-qualified trust service providers, with the key difference being that the latter cannot issue qualified certificates required to create QES. Qualified Trust Service Providers (QTSPs) must adhere to strict regulations and security standards governed by eIDAS and are subject to ongoing oversight to ensure they maintain high levels of security and reliability. The eIDAS certification process involves a comprehensive evaluation of the timestamping authority's systems and procedures to verify their compliance with the eIDAS Regulation. This evaluation examines various aspects, such as security measures, data protection, and auditability, among others. If

the authority successfully meets these criteria, it is issued with a certificate that confirms the conformity of its systems and procedures with the eIDAS Regulation.

To ensure transparency and trust, each EU member state is responsible for publishing and maintaining a trusted list of QTSPs. This list, which includes details about the services offered, is monitored by the respective country. The European Commission makes these lists accessible through a Trusted List Browser, allowing businesses, governments and customers to verify the authenticity and status of the TSPs they intend to use.

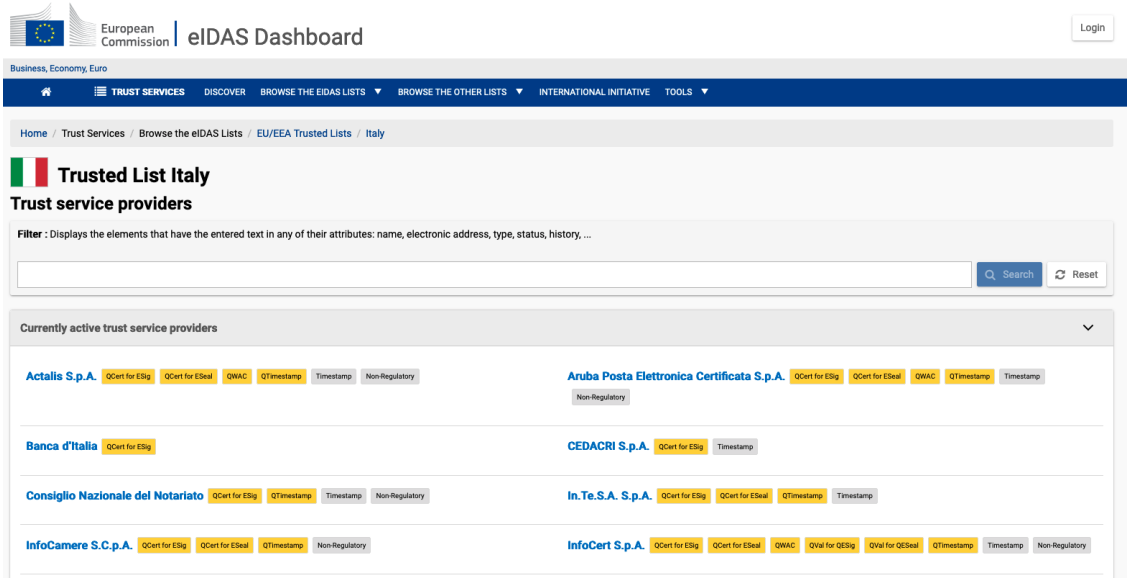

Figure 4.3. Part of the Italy's QTSPs list (Source: eIDAS Dashboard).

In addition to highlighting the QTSPs available in Italy issued on 30 July 2024, the image above is useful to introduce the difference between a simple timestamp and a Qualified timestamp, as these are two different services offered that appear below the name of some providers.

The qualified electronic timestamp is a specialized type of timestamp that benefits from a presumption of accuracy regarding date and time, ensuring that the data protected by the timestamp has remained unaltered since it was marked. Two key features of these timestamps are their recognition across all EU member states and their admissibility as evidence in court. Typically, they are secured with an electronic seal guaranteed by the qualified trust service provider and are based on a precise time source associated with Coordinated Universal Time (UTC), a global reference time expressed using a 24-hour format that specifies the relevant time zone.

These services and functionalities do not come for free, as the primary Qualified Trusted Service Providers charge a fee for timestamping, typically based on the number of timestamps requested. Given the forensic perspective associated with this tool, this aspect was crucial to consider. As highlighted in the previous subsection, a timestamp is requested daily for each pair (selector, domain). As the tool stores more pairs, the number of daily timestamps requested increases, leading to a significant economic burden for the user.

To reduce the daily number of timestamps required, a new solution was implemented. Instead of timestamping each individual txt file, a zip folder containing all the files produced and updated each day is created and timestamped. This approach significantly reduces the number of files that need to be timestamped daily, thereby lowering the tool's cost and the traffic generated. As a result, the tool's output and behavior are now slightly modified:

1. For each pair (selector, domain) a DNS query is sent to the DNS server, with the response saved in the respective txt file.

2. The traffic generated with the DNS server is captured separately for each pair, restricting the tcpdump command only to port 53.

3. After all the iterations, a zip file is created within a folder named "zip". This zip file contains all the updated txt files, the list of pairs, the general log file, and is named with the current date.

4. A timestamp request is created for this zip and then sent to the Timestamping Authority (TSA).

5. The traffic generated with the TSA is captured and saved in the zip folder.

While this new solution addresses some of the previous issues, it introduces some disadvantages with respect to the old one. For instance, if a single file within the zip folder has been manipulated the entire timestamp would result invalid. Additionally, detecting such manipulation would become a more time-consuming process, requiring manual inspection of all the files within the zip.

Given these pros and cons, the user has the flexibility to choose between timestamping the zip of all the files or timestamping each file singularly, by simply adding a flag when launching the tool.

4.3.5 Architecture of the tool

All the work described so far is part of the main or core module of the solution, which users can interact with through a command line or a web interface. In order to address some existing issues and enrich the tool's functionalities, additional modules were added throughout the development process. Below is a schema that summarizes the architecture of the tool:

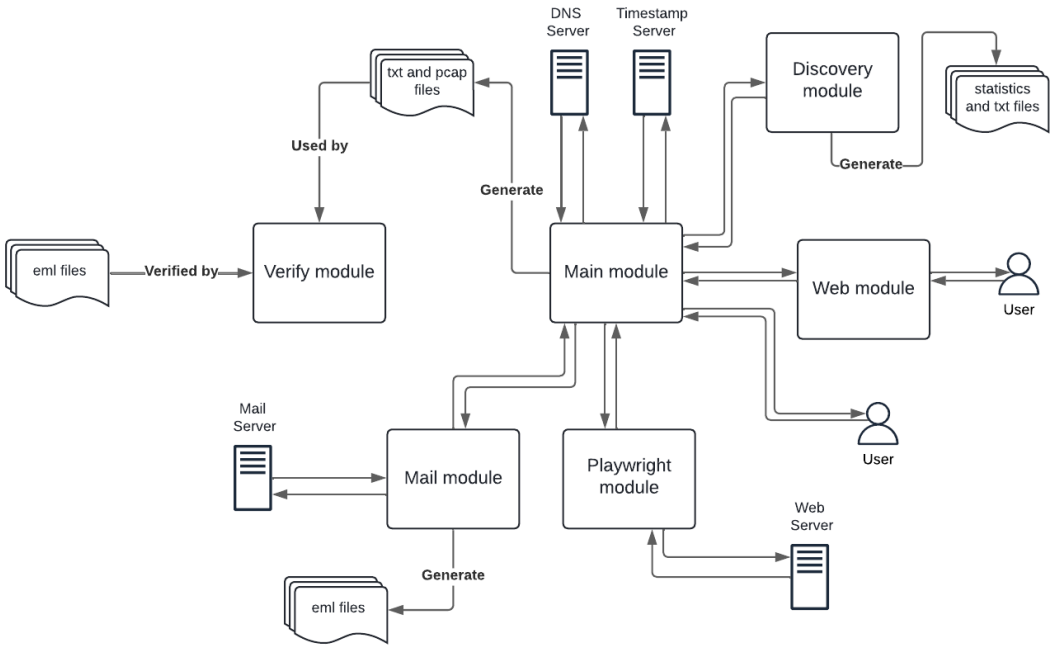

Figure 4.4. Software Architecture

While these modules will be discussed in detail in the following subsections, here is a brief overview of their functionalities:

- Mail: This module interacts with mail servers to both receive and send emails.
- Playwright: This module interacts with web servers, to generate emails.
- Discovery: This module identifies (domain, selector) pairs for a given domain. It can be directly accessed by the user via the command line or web interface.
- Verify: This module verifies DKIM signatures using the files generated by the main module. It is also accessible through the command line or web interface.
- Web: This module creates the web interface that allows users to interact with all the tool's functionalities.

4.3.6 Mail module

With the information provided so far, the tool is ready to execute, allowing users to monitor the pairs they manually insert.

Even if standardization constraint users to follow a specific methodology of work, sometimes for the interoperability with other system, a protocol well standardized could lead to a better integration. However, DKIM does not standardize in any way the usage of the selector for publishing the public key in the DNS txt record nor does it dictate the process for selector substitution. This lack of standardization complicates the task of finding and monitoring domains, as each domain is free to use its own naming conventions. In addition, the principal command line tool used for querying DNS server does not come with built-in features for checking multiple records simultaneously.

Through research involving the examination of various emails from different inboxes, some observations were made regarding the selectors used by three different domains:

Domain	**Selector**	**Start Time of Usage**
`google.com`	`20230601`	September 2023
	`20221208`	April 2023
	`20210112`	January 2023
`mp1.tripadvisor.com`	`scph0523`	September 2023
	`scph0417`	March 2022
`twitch.tv`	`dh6r6vaod6penm5x6ufsnymrqcq2rt3a`	February 2023
	`ihcl5uso4lqpgom7nwqnxs2in6mo4xoc`	December 2023
	`wcdtjomfpuxxefyhk6a7dbwy5fy6ofyc`	April 2023
	`uder6jfl2ivm6ra4d56gtvzkfaq2gr2v`	August 2022

Table 4.1. Selector domain information.

It appears that there is no consistent logic behind the naming of selectors. For instance, while Google uses selectors that resemble dates, their usage periods do not align with those dates. When it comes to handling replaced selectors, three different approaches were observed:

- Google keeps the old record online but empties the "p" field, leaving it as "p="
- TripAdvisor keeps the old record online even after switching to a new selector
- Twitch employs a hybrid solution; some old selectors are still online, while the older one are emptied, like Google does.

This raises the following question:

1. How can the tool know if the selector for a specific domain has changed?
2. How can the tool reach new pairs?

Since the DKIM protocol is designed for email authentication, the initial solution proposed was about the possibility of integrating a mailbox within the tool. Besides DNS, the principal source of DKIM information, resides in the email headers. Accessing a mailbox and reading the inbox could provide valuable insights for tracking domains and selectors, as well as discovering new ones.

After setting up a fresh new Gmail mail account, the first task for this new module was to configure access to the inbox, enabling the reading and downloading of received emails. Gmail, along with other mail providers, allows inbox access via a third-party solution only using a so-called "app password". This password must be generated through the provider's web interface only after enabling the two-factor authentication. Once generated, this will replace the real password for SMTP/IMAP authentication. The tool reads the email address and app password from a JSON file, which contains the login credentials and the name of the IMAP server to connect to. Here is an example of a configuration file:

```
1 {
2   "server_address": "imap.xxx.com",
3   "username": "xxxx@gmail.com",
4   "password": "xxxxxx"
5 }
```

Figure 4.5. Example of config file.

Since the tool performs daily checks, it requires a new email from each monitored domain every day to verify whether the selector used for email authentication has changed. However, this raises several challenges:

1. How can a mailbox be populated with new mails from scratch?
2. How can these emails be received on a daily basis?

Currently, there is no standard method to achieve this, and before proceeding with the proposed solution, it's important to distinguish between two types of domains that can be monitored:

- Mail provider domains: These are domains that allow users to send emails and access an inbox.
- Non-mailing domains: These are domains that do not provide email-sending services but allow users to sign up or create an account to receive newsletters or notification.

For mail provider domains, the research focused on finding a method to automatically send emails to a specific email address. Typically, the automation capabilities provided by these email services are limited to scheduling the invoice of a single email at a certain time, without allowing any deeper configurations. One exception is Outlook, which, when combined with the Windows Task Scheduler and VBA (Visual Basic for Applications), a programming language developed by Microsoft, can be used to create scripts that send emails automatically at a specific time each day. Although this solution is theoretically viable, one major drawback is its dependency on the operating system, as it requires a specific version of Microsoft Outlook developed for Windows.

Fortunately, using Python's "smtplib" library, it is relatively simple and straightforward to compose an email and connect to the SMTP server to send it. To leverage this, seven different accounts of email providers were created:

1. Gmail
2. Outlook

3. Tim
4. Virgilio
5. Libero
6. GMX
7. iCloud

As with the Gmail account setup, most of these providers required an app password to be configured. However, Tim does not offer this option and instead requires the usage of the standard password for SMTP authentication. Additionally, the iCloud mail account could only be created from an Apple device, as the creation via a web browser was not allowed.

Similar to the Json file described in figure 4.5, configuration files were created for these accounts, replacing the IMAP server with the SMTP server and specifying the port used for sending email. All this information was retrieved online directly from the service providers' websites.

This is the piece of code responsible for the creation of the email:

```
1     msg = MIMEMultipart()
2     msg['From'] = sender_email
3     msg['To'] = "dkimmarco@gmail.com"
4     msg['Subject'] = "DKIM Update mail"
5     msg['Date'] = formatdate(localtime=True)
6
7     # Set Message-ID manually
8     msg['Message-ID'] = make_msgid()
9
10    # To avoid sending the same email forever
11    number = random.randint(1, 1000)
12    body = "This is a test email to update DKIM selector. Random number: " +
          str(number)
```

Figure 4.6. Creation of the email.

As the script iterates through all the configuration files, it selects a new sender for each iteration, while the recipient, subject, and most of the email body will always remain the same. To introduce a degree of randomness and avoid triggering anti-spam mechanisms, a random number is added to the email body. Additionally, the Message-ID field is manually set, since one of the servers in the list of domains was not generating it correctly.

This mail module was integrated early in the tool's workflow, so before querying the DNS servers the tool performs the following steps:

1. Sends an email from each domain in the list.
2. Downloads the received emails in the inbox into the "email" folder, naming the file after the "message-ID".
3. Extracts DKIM-related information from the downloaded email.
4. Updates or inserts any new domain-selector pairs found.

Finding a fully functional solution for non-mailing domains presented several challenges. The email address monitored by the tool, was used to subscribe to various social networks such as Facebook, Instagram and TikTok with all notifications set to be received via email. Despite

enabling this setting, Instagram and TikTok did not generate any email to be received, at least not with the daily cadence needed by the tool. Facebook, instead, managed to send multiple emails per day by joining public groups and enabling notifications for new posts, almost to the point of overwhelming the inbox with spam. After adjusting group subscription, the frequency of emails started to slow down, stabilizing at a couple per day. From the tool's perspective, downloading multiple emails from the same sender could lead to storage issues. This was resolved by downloading only the latest email when multiple emails from the same domain and selector pair were detected.

The subscription to service providers like ASOS, easyJet and Ryanair instead, generated a less consistent flow of emails in the inbox. The first two domains sent emails approximately every three to four days, which, while not ideal for the tool's daily needs, was still sufficient for periodic review.

With the introduction of the mail module, it is also important to discuss a file which is maintained and updated throughout the lifetime of the tool called "log.txt". As the name suggests, this file logs, all the potential updates and modifications to the monitored pairs. For instance, whenever a DNS query response is received, the tool checks if it differs from the previously logged response, and if so, the modification is reported in the log file. Other possible usages comprehend the detection of a selector change by a specific domain or a new pair discovery, both of which are verified using emails as described in the previous paragraph. Each update in the log file is preceded by a timestamp indicating the current time and a message ID that references the email responsible for the update, which can be found in the email folder. If the update does not originate from an email, a special placeholder, "AUTO-UPDATE," is inserted instead. This typically occurs when the update is triggered by a DNS query for a domain that did not receive an email that day.

This is an example of the log.txt file content:

```
1  Jun 03 08:05:22 2024 GMT - [Message-ID:
       <6655a146.170a0220.4db6c.0c73@mx.google.com>] - Selector for domain
       'gmail.com' has changed from '20230602' to '20230601'.
2  Jun 03 08:05:22 2024 GMT - [Message-ID:
       <6657422b.170a0220.ac989.477bSMTPIN_ADDED_MISSING@mx.google.com>] - New
       domain 'iniziative.intesasanpaolo.com' found with selector 'clab1',
       adding it to the list.
3  Jun 03 09:54:39 2024 GMT - [Message-ID:
       <AS8PR02MB750576EDD4DAABDBE750F@AS8PR02MB7505.eurprd02.prod.outlook.com>]
       - TXT record for domain 'OUTLOOK.IT' has changed:
4  Previous:
       v=DKIM1;k=rsa;p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvWyktrIL8DO/+
5  UGvMbv7cPdXogpbs7pgVw8y9ldO6AAMmg8+ijENlc7Fb1MfKM7uG3LMwAr0dVVKyM+mbkoX2k5L
6  7lsROQr0Z9gGSpu7xrnZOa58+pIhd2XkDFPpa5+TKbWodbsSZPRN8z0RY5x59jdzSclXlEyN9mE
7  ZdmOiKTsOP6A7vQxfSya9jg55RCQIDAQAB;n=2048,1452627113,1468351914"
8  New:
9  v=DKIM1;k=rsa;p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvWyktrIL8DO/+
10 UGvMbv7cPdXogpbs7pgVw8y9ldO6AAMmg8+ijENlc7Fb1MfKM7uG3LMwAr0dVVKyM+mbkoX2k5L
11 7lsROQr0Z9gGSpu7xrnZOa58+pIhd2XkDFPpa5+TKbWodbsSZPRN8z0RY5x59jdzSclXlEyN9mE
12 ZdmOiKTsOP6A7vKzT6sCCrg55RCQIDAQAB;n=2048,1452627113,1468351913"
```

Figure 4.7. Example of log.txt.

4.3.7 Playwright module

One of the possible solutions thought for non-mailing domains, in order to receive emails, was the password recovery process. Typically, during a password reset, a user subscribed to a particular

service must enter his email address into a form to receive a special code or a link via email. The user then will input the code back in the form or follow the link to proceed with resetting the password.

The main challenge is that this kind of procedure is usually only available through a web interface and cannot be easily executed with standard command line connections. However, by controlling a web browser it becomes straightforward to achieve this behavior. Among the various libraries available for Python, Playwright [26] was created ad hoc for end-to-end testing. It supports all modern rendering engines, such as Chronium, WebKit, and Firefox, and can run on different operative systems like Linux, Windows, and macOS, but also locally or on command line. Creating a script that interacts with a web browser is relatively simple, as demonstrated in the following example:

```
1  from playwright.sync_api import sync_playwright
2
3  def reset_password(email: str):
4
5      with sync_playwright() as p:
6
7          print("Initializing browser for password reset")
8
9          browser = p.chromium.launch(headless=False)
10         page = browser.new_page()
11         page.goto("https://www.asos.com/it/uomo/")
12
13         # Wait for the button to be visible and then click it
14         page.wait_for_selector('button[data-testid="myAccountIcon"]')
15         page.click('button[data-testid="myAccountIcon"]')
16
17         # Wait for the "Il mio account" link to be visible and click it
18         page.wait_for_selector('a[data-testid="myaccount-link"]')
19         page.click('a[data-testid="myaccount-link"]')
20
21         # Wait for the "Password dimenticata" button to be visible and click
               it
22         page.wait_for_selector('a[id="forgot-password-link"]')
23         page.click('a[id="forgot-password-link"]')
24
25         # Wait for the email input field to be visible and fill it
26         page.wait_for_selector('input[name="Email"]')
27         page.fill('input[name="Email"]', email)
28
29         # Wait for the "Reimposta password" button to be visible and click it
30         page.wait_for_selector('input#send')
31         page.click('input#send')
32
33         browser.close()
34
35         print("Password reset done!")
```

Figure 4.8. Playwright code.

As shown in the code, a browser instance is first created, and by setting "headless" to "false" the whole procedure can be observed on screen. After navigating to the desired page, the subsequent commands perform always two paired operations. Since this library is based on the interaction with HTML elements, the first instruction of the pair waits for the HTML element to

be rendered and displayed on the page. Once the element is present, an action can be performed, such as clicking on it, or filling it with text.

In this particular case, the procedure involved clicking on different elements to reach the mail form, filling in the email address, and clicking a button to receive the password reset link via email.

It's evident that this is an extremely vertical solution, designed to work for a single domain. Since it relies on HTML elements rather than graphical elements, it is difficult to generalize it to work with different websites, as naming conventions typically vary by the programmer. Additionally, another potential issue could arise from captchas, which are often used to distinguish between computers and humans.

By summarizing, this could be a valuable solution for monitoring specific domains, particularly those for which receiving emails is infrequent, without requiring significant effort.

4.3.8 Discovery module

The discovery process of new pairs selector domains can be difficult, and until now all the couples controlled by the tool were updated via the mail module or manually by the user. Online tools exist that offer DKIM record discovery services, by simply inserting the domain name they would display the DKIM record found However, testing these tools revealed that they could only resolve specific domains. By aggregating the results and comparing the selector names of those domains for which a record was found, it was clear that all of them were using "standard" selector names such as "dkim" or "selector1" Regardless of the domain's popularity, the key factor here was the selector name, as a widely used domain like Gmail, failed to be discovered by most of these tools since the selector it uses is a sequence of numbers, which again is not "standard". These results suggested that these tools were probably using a brute force approach, querying the DNS server of the specified domain with a list of common selectors.

However, as previously highlighted in the introduction of the mail module, there is no default way of using a DNS query command to execute multiple queries at once. On the other hand, when the command is used within a script, multiple queries can be sent sequentially, speeding up the discovery process. This was the starting point for the development of the discovery module, which then needed to be integrated with all the tool's functionalities.

The first challenge addressed was selecting the list to be used for the brute force requests, where two solutions emerged:

1. The obvious solution was to start with the top ten popular selectors mentioned in the previously cited paper on DKIM deployment. This list could then be expanded with the selectors already discovered by the tool for the monitored domains as well as those identified by the mail module. Naturally, the more selectors included in the list the higher the chances of finding a used selector. On the other hand, an excessively long list would significantly increase the discovery time.

2. Another approach to this challenge involved the usage of regular expressions (regex). These are specific patterns of characters used to describe a group of words, without having to list them all. This is particularly useful when dealing with fixed words that may be completed by a number, such as "selector1", "selector2" and so on. While checking if a word matches a particular regex is straightforward, the process of generating all the possible words that match a specific regex is not trivial, and in some cases, a regex can generate an infinite number of solutions. Exrex is a Python library that permits the generation of these solutions, with limitations to prevent infinite outputs.

When using the discovery module, the user is free to choose whether to use the standard list of selectors described in the previous paragraph or provide a txt file containing multiple regexes, one per line, to generate selectors.

Speaking of additional inputs that can be given to the module, users can decide whether to perform the research for a single domain, which can be specified via the command line or directly input in the web interface, or to query multiple domains. For this purpose, users can provide a txt file containing a list of domains, each on a separate line.

With the introduction of this new module, it became crucial to think about how to organize and possibly save the additional information generated. Until now, all the couples (selector, domain) were saved and monitored thanks to a txt file named "selectors_domains.txt". With this new scenario, three potential solutions were analyzed:

- The first solution involved adding the newly discovered pairs to the existing list of monitored domains. This would result in a significantly larger list of pairs coming from different sources. However, the main issue with this approach was the loss of the list's original purpose. Previously all the pairs contained within this list were representing the "active" selectors currently used by these domains. The pairs discovered by this new module in fact, are just found in the DNS server of the specific domain, and they may be no more in use or just present as a test record. Moreover, including these new pairs in the existing list would require monitoring all of them, as this text file is the source used by the tool to run its entire process.

- The second solution aimed to address the issues of the first by slightly modifying the approach. The main list would remain unchanged, while the found couples would have been saved in a different and collective list, where all the couples discovered and monitored are mixed just to be saved as a sort of "historical archive". This new solution would have preserved the meaning of the main list while saving all the discovered couples. However, the downside was that the discovered couples would have not been monitored, losing a possible source of useful information.

- The third solution came as a mix of the previous two. In addition to the existing "main list", a second list was created, called the "discovered list" (named selectors_domains_discovered.txt). By keeping them separated, each pair in its respective list would assume a different meaning, allowing users to distinguish between selectors that are currently in use and those that are not. Additionally, by having two lists it would have been possible to choose whether to monitor or not the discovered list, or maybe just use a different timestamping server, by simply modifying the source code.

The introduction of this module necessitated managing the pairs in both lists, including operations like removal updating, and exchange. These operations could be performed automatically by the tool using its built-in functionalities.

To clarify, here is the sequence of operations performed to update the lists when using the main module of the tool. After downloading all the emails via the mail module, the tool checks if any monitored domain has changed its active selector. If so, the new (domain, selector) pair is added to the main list, if it was already in the discovered list, it is removed from there. The old pair, meanwhile, is instead moved to the discovered list, to maintain a log and keep a record of it.

When using the discovery tool instead, all the valuable outputs found (i.e., properly formatted DKIM records) are evaluated for inclusion in the discovery list. Only the pairs which are not already present in either list are added.

The tool's default behavior is to monitor the discovered list and save the output files generated by the process in the same output folder as the main list. This approach allows for a single file to log all DNS records of a pair, even if the pair is repeatedly activated and deactivated.

To provide useful insights about the usage of DKIM, the research performed by this tool is used to generate some statistics. A counter is added next to each selector in the brute-force list, and these counters are updated at the end of each tool run, creating a ranking of the most commonly used selectors. In order to avoid the increment of the same selector using the same domain repeatedly, an additional check is performed. This check uses the "history" file, where each line records a domain and a list of selectors. Before updating the counters, the tool checks

whether the selector has already been discovered for that domain, and if so, the update is canceled. Since these statistics are generated through the use of the tool, the more domains the tool checks, the more reliable the statistics will be.

The combination of the history file and the selectors counters is used to generate the following:

- A Top 5 list of the most used selectors.
- The percentage of usage of each selector relative to the others.
- The number of domains using a specific number of selectors.

After running the discovery tool over more than sixty domains from the Ahrefs most visited websites list [27], these were the results obtained:

```
Number of Selectors Used by Domains:

1 selector: 29 domains
2 selectors: 17 domains
3 selectors: 2 domains
5 selectors: 1 domain

Top 5 Selectors:

google: 23
k1: 13
selector1: 10
selector2: 7
20230601: 4

Percentage Usage:

google: 31.08%
k1: 17.57%
selector1: 13.51%
selector2: 9.46%
20230601: 5.41%
default: 4.05%
mail: 2.70%
dkim: 2.70%
GS2: 2.70%
scph0523: 1.35%
clab1: 1.35%
20211207: 1.35%
s2017: 1.35%
s2021: 1.35%
s31663417: 1.35%
1a1hai: 1.35%
s1024-2013-q3: 1.35%
```

Figure 4.9. Statistics output

4.3.9 Verify module

The principal purpose behind the development of this tool was to enable the verification of signatures on older emails. After creating the structure and gathering the necessary data for such

verification, a dedicated module was created to handle the task of signature validation.

This module is built upon the Dkimpy library, which was introduced in the "Verification tools" section of the Background chapter. While the library served as a robust foundation, it has been slightly modified in order to adapt to the tool's needs. The general workflow is as follows:

- The user provides the email and optionally a record to the module for verification.
- The module extracts the selector, domain, and the signature timestamp. Since it is not a mandatory field, if the signature timestamp is not present, the email's date of receipt is taken.
- If a record is not specifically provided, using the extracted selector and domain, the tool reconstructs the filename to locate the corresponding public key within the output folder. The date is then used to identify the appropriate key, specifically, the first available key after the given timestamp.
- In case the public key is not available within the tool's files, a DNS query is performed by the module to retrieve it.
- The signature is then verified, and the result is displayed. The tool can discriminate between a valid signature and an invalid one, particularly if the body or the header has been modified.

As underlined earlier in this section, the Dkimpy library needed to be modified. By default, there was no function of the library that permitted to use a fixed public key to verify the signature, so this was the first addition to perform. Secondly, the library's default signature verification function initially checks for expiration, which is useful when assessing the validity of an email after its receipt date. However, the tool's purpose is to verify the authenticity of the email as it was at the time of receipt, so instead of comparing the expiry date with the actual date, the latter has been substituted with the receipt date.

During testing, a notable issue was encountered when using this module with emails from Outlook. The emails downloaded from both the web and desktop clients, in fact, failed the verification due to a body mismatch, even when the mail content appeared unmodified. However, by reading the headers of the email, the DKIM signature appeared to be verified, as the Authentication-results was successful. By performing a further investigation, emerged that this is a known issue, after Outlook verifies the DKIM signature, it appears to apply some modifications to the email content in order to adjust its visual formatting. As a result, the final user has no longer access to the original message sent as it was sent.

4.3.10 Web module

To provide end users with a different and more user-friendly way to access and use the tool's functionalities, a web interface has been developed. Since the entire tool was written in Python, this part of the application also uses Python, thanks to the Flask framework. Flask in fact is a lightweight web framework, written in Python and designed to build simple and smaller projects or applications. Through this web page, users can upload emails to be DKIM verified, discover DKIM records of domains, or initiate the log process with just the click of a button. A comprehensive guide on how to use this interface is provided in the user manual included in the appendix.

4.3.11 Final usage

Given all the functionalities provided by this tool, its usage can be categorized into two types: active and passive. The active functionalities, such as those offered by the discovery and the verify module, can be accessed via the command line or web interface and are classified active as they require the human collaboration to be executed.

The passive functionality, on the other hand, is the tool's core feature, which resides in the monitoring phase. As for the active, even this functionality is available both via the command line and web interface. However, additional work has been done to enhance this aspect. Due to its passive nature, this operation does not require the intervention of the user making it suitable for automated execution.

On an Ubuntu 24.04 virtual machine, after installing the necessary Python libraries, the tool script was scheduled to run daily using crontab. Crontab is a command line utility in Linux and macOS used to schedule automated tasks, called "cron jobs" to run at specific time. By configuring a logger associated to this cron job it is also possible to capture the tool's output and make it available for future monitoring and troubleshooting. This setup represents the ideal scenario for the passive functionality, allowing the tool to be installed on a running server and operate automatically.

Additionally, by starting the web interface on the same server and making it available at a specific port, all users within the network can freely use the tool.

4.3.12 Comparison with Metaspike tools

As written in the introduction, temporarily at the end of the development process it was possible to compare the solution with two tools developed by Metaspike, a software company that operates in the digital forensic field with a focus on email forensics. These are the two programs analyzed:

The Forensic Email Collector (FEC)[28], has the main goal of preserving email evidence by collecting the content of a mailbox without breaking the environment. This tool allows even for the acquisition of Google calendars and appointments, contacts, or notes, retrieved from the Exchange servers but also Google Drive files. The authentication to the mailbox can be performed in three different ways; via OAuth 2.0, using a Remote authenticator, or using a single set of admin-level credentials for enterprise auth. Depending then on the mailbox providers it has a specific way of accessing the inbox.

These instead are some of the features offered by the tool:

- In-place and Inline search: Experts often require looking at specific emails, which contain some possible keywords, with FEC It is possible to perform in-place searches on the mailbox server before acquiring the evidence, reducing the number of emails to be downloaded but still preserving the target mailbox. This is particularly useful for privacy concerns, as with other tools the whole mailbox is required to be downloaded to be able to perform research. The inline search instead is executed after the acquisition but before being written to the disk. It is optimal when you can have email in memory to work on it, but you cannot save it all on the disk. The query can be built by a query builder and can be tested in a sandbox.
- Google Drive Attachments: During email inspection is fundamental to have the possibility of looking at the attachments contained in it but when those attachments are in the form of hyperlinks to a cloud storage system, as for Google Drive attachments, it is not trivial to get it after the initial acquisition, because of authentication issues. FEC automatically acquires those during Gmail and Google Workspace acquisitions.
- Server Metadata: The server's metadata as IMAP UIDs and internal dates are captured by FEC during the acquisition, to be decoded and presented in a more user-friendly way to facilitate the investigation process.
- MX Lookup: If the target email address is not one of the pre-configured profiles, FEC performs a lookup for the MX record to determine the target server settings.

This is an example of the process required to collect a forensic copy of a Gmail mailbox:

Given the address inserted in the user interface, the tool will select a proper profile and give the user the possibility to choose which kind of data needs to be collected. For a Gmail account users can choose between emails, calendars, files in Google Drive, and other options. Then the user can choose to assign labels to the capture, such as Case name, ID, Client Name, and other info. At this point, it is possible to choose:

- The output path
- The output format (EML/MSG/PST)
- To containerize or not the MIME output into a VHDX container, a useful format that can be injected into other forensics tools to perform analysis.
- To timestamp logs folder by creating a zip package.
- To decrypt S/MIME encrypted items.
- To perform differential acquisition, to complete an already started work.

After the authentication, the user can see the Gmail labels and the folder of the inbox and choose which one to download, eventually performing in place or inline search. The credentials are removed by default at the end of the process, but this behavior can be modified in the settings. During the acquisition, it is possible to look at the items remaining, and the items excluded by the search filter. Finally, some post-acquisition actions are displayed, such as opening the output folder, containerize the MIME output, or creating an output PST.

Three log files are generated at the end of the process:

1. Acquisition log: Txt file in which is displayed a summary of the acquisition, reporting the items analyzed, settings, and the time elapsed.
2. Exception log: Txt file in which are displayed info about possible errors encountered in the process.
3. Downloaded Items: Tsv file with information about the downloaded items, such as Message ID, Date, labels, etc.

One functionality that is in common with the developed solution, resides in the possibility of timestamping the log folder. While the target files are different, the timestamping process is similar, as even in this tool a zip package is created before sending the file to the TSA. If enabled, the zip file, together with the timestamp request and timestamp response will appear in the acquisition folder. FEC allows even for verification of these timestamps, thanks to a built-in functionality. The URL and the credential related to the Timestamp server instead, can be modified in the settings.

The Forensic Email Intelligence (FEI)[29], is used by experts to investigate all the mail-related violations, providing various possibilities of integration with already existing services like MaxMind, SecurityTrails, and other specialists in forensic solutions. It is strictly related to FEC, as the acquisition of the mailbox is always the starting point needed before investigating. Moreover, the output produced by FEC can be used as an input for FEI.

These are some of the features offered by FEI:

- In-place Decoding: When looking at the email source code, FEI can automatically highlight and identify MIME headers, providing additional descriptions about them.
- DKIM & ARC Verification: FEI is able to complete the verification process of DKIM and ARC, automatically computing the hashes and retrieving the public keys needed.
- Automated Timelining: FEI is able to extract from the email content all the timestamps encountered and then reconstruct a timeline.
- Insight and scoring: This tool can derive insights from the email content, such as inconsistencies among data or which service providers or software were involved in the email invoice.

Upon creating a new project from a FEC file by the respective tool, a table containing information about the ingested email is presented. This interface is interactable, and the user can choose to filter or order by the different attributes presented, from the Sent Date to the Email ID or Gmail label. By navigating the lateral bar, it is possible to extract information about the usage and the number of times that appeared certain email addresses, domains, and IPs.

Since the verification of DKIM signatures is the key feature of this thesis work, let's concentrate on how FEI handles this process. By looking at the previous release of this tool, two of them marked significant improvements on this topic:

- July 2022 - FEI v 1.8: With this release, FEI had a new option to save into the file system the public keys needed for DKIM and ARC verification. This is particularly useful since this kind of data can only be retrieved by querying the DNS server and could become unavailable at some time. Specifically, this is achieved when creating a new FEI project, the tool will save the keys as files with no particular extension in a specific folder named Cache. The file content is itself the public key value, while in the DKIM logger developed by this thesis, the whole record is reported and additionally timestamped.
- February 2023 - FEI v 2.1.7: This release signs the creation of the so-called "DKIM Supercache", a feature that enables the verification of all Gmail and Google Workspace DKIM signatures since back in 2004.

This is what Arman Gungor, the founder of Metaspike, said about the development process of it:

"We mostly relied on our own archives (having been involved with DKIM for some time helped) and supplemented that data set with online research."

By inspecting the network traffic, the DKIM Supercache might be present in the files of the application, as even with no network connection the verification process can be completed. In the same way as the DKIM logger, when the record is not present in the cache the application opens a connection to the DNS server to retrieve it, and in case of no response, the action is repeated two times with a five-second delay.

Other similarities come from the ability to discriminate between a body hash mismatch and a change in the headers of an email and the possibility of validating a DKIM signature even if expired. Finally, the problem encountered with Outlook emails that was described in the "Verify module" section of this chapter is confirmed, as also using this tool the DKIM signatures could not be verified.

Screenshots of the two applications can be found in Appendix D.

Chapter 5

Conclusion and future development

The primary objective of this thesis was to create a tool that permitted DKIM signatures of old emails to be re-verified. Since no previous work directly addressed this issue, it was initially crucial to look at similar studies in related fields to understand which methodologies and technologies were needed. For instance, the long-term verification standard highlighted the importance of timestamps for establishing a precise point in time for data existence and for adding digital signatures to such data. After thinking and reasoning about some of the proposed solutions, only one of them emerged as complete enough to fulfill the initial problem with fewer overall drawbacks, the DKIM logger solution.

Since DKIM does not require its users to adopt a standard naming convention for selectors, the initial task was oriented to collect as many pairs (domain, selector) as possible, by analyzing email headers. These pairs were inserted into a list and monitored daily thanks to a Python script that executed DNS queries, logged the output, and timestamped the resulting file through a Timestamping Authority. This whole process, if traffic recorded, could guarantee an additional source of validity for the data archive. This is one of the strong points of this tool, as in the forensic field, it is fundamental the ability of verifying the authenticity of data that can be used in legal processes. By creating an archive of DKIM records, it would have been easy to recover the public keys, contained in such records, needed to begin the verification process.

Once a base list of domains was established, it was necessary to keep it up to date, and the best way of doing that was to get these updates from an email inbox, the primary source of information. The challenge here lay in ensuring daily emails from the monitored domains, which was straightforward to implement for mailing domains, that can be used to send emails, and less intuitive for non-mailing domains. For the latter category, ad hoc solutions were implemented, such as using the Playwright library to control a web browser and automate the password recovery process (which generates an email) or enabling email notifications for social network domains. This may be one of the limitations of this tool, as non-mailing domains are really difficult to keep track of and needs a specific solution.

To facilitate the extension of the monitored domains, an additional module was developed, capable of discovering the selector used by a specific domain through a brute force approach. The data collected by this module were also recycled to generate some statistical insights about selector usage, which is really helpful but at the same time requests an intensive usage over a large number of domains in order to obtain reliable results.

Finally, a web interface was developed, to provide a more user-friendly interaction with all the tool's functionalities. Since the entire tool was developed in Python, the Flask framework was chosen for the web interface due to its lightweight nature and ease of use.

It is important to notice that, given the ability of the tool to send multiple emails to an email address, its adoption must be carried out with strict policies, leaving no gap that would permit malicious use.

Given the variety of modules implemented for this tool and the underlying mechanics, there is still room for future enhancements within the existing architecture:

Since the tool can only start verifying emails from the first day of logging, transforming it into an online service could be beneficial. This would allow to have a shared archive of records, without the need of creating a private archive from scratch. By adopting this solution, also the discovery module could take advantage, by enriching the pool of monitored domains and providing more reliable statistics. To achieve this, the web interface would also need to be updated, with a focus on enhancing its backend security to prevent the execution of malicious commands on the host machine.

Another area of improvement concerns the automation of the verification process for the archive, as it is currently a manual operation that requires two steps:

1. Comparing the hash value of the zip archive or the log file, with the hash contained in the timestamp token and in the timestamp request. This hash can also be found in the packet capture related to the exchange with the Timestamping Authority.
2. Comparing the DKIM record found in the log file with the one from the packet capture related to the DNS query.

Data retention could be further improved by moving the logs, currently contained in txt files into a more secure and structured format, such as a database. In order to do so a consistent structure for the database must be established, along with a method for signing the database files.

Lastly, the weakness in the daily reception of emails could be addressed by integrating an external system. For instance, an initial idea that was not developed further due to material limitations and privacy concerns was the idea of creating forwarding rules on existing email accounts to increase the possibility of receiving mail from different domains throughout the day.

To conclude, here are two final considerations about the DKIM protocol itself, and how it could be improved besides the adoption of additional tools.

Since DKIM relies heavily on the DNS architecture to retrieve public keys, it could benefit from the adoption of DNSSEC (Domain Name System Security Extension). Improving the functionality of the basic DNS system, this newer version permits the DNS response to be signed and verifiable, mitigating the risk of DNS spoofing. However, the adoption of DNSSEC is slowed down due to its technical complexity, cost, and necessity of backward compatibility.

Finally, being DKIM based on asymmetric cryptography for its signature, it is essential to recall that these encryption schemes are under threat due to the advent of quantum computers. The math behind these algorithms is based on the difficulty of factoring the product of two large prime numbers. Turns out that quantum computers could theoretically break existing schemes like RSA and Diffie-Hellman in hours, the same work that would require a modern computer trillion of years. Even if this problem may be many years away, this is the reason why in 2016, NIST, asked cryptography experts to start developing quantum-resistant algorithms in order to define a new post-quantum cryptographic standard.

Bibliography

[1] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman, "Neither snow nor rain nor mitm...: An empirical analysis of email delivery security", Proceedings of the 2015 Internet Measurement Conference, New York, NY, USA, 2015, p. 27â39, DOI 10.1145/2815675.2815695

[2] C. Wang, K. Shen, M. Guo, Y. Zhao, M. Zhang, J. Chen, B. Liu, X. Zheng, H. Duan, Y. Lin, and Q. Pan, "A large-scale and longitudinal measurement study of DKIM deployment", 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, August 2022, pp. 1185–1201

[3] European Telecommunications Standards Institute, "Etsi ts 102 778-4 v1.1.2 (2009-12): Electronic signatures and infrastructures (esi); pdf advanced electronic signature profiles; part 4: Long term validation", Tech. Rep. TS 102 778-4 V1.1.2, European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France, December 2009

[4] European Telecommunications Standards Institute, "ETSI TS 101 733 V2.2.1: Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)", Technical Specification V2.2.1, ETSI, April 2013. Available online: https://www.etsi.org/deliver/etsi_ts/101700_101799/101733/02.02.01_60/ts_101733v020201p.pdf

[5] D. Crocker, "Internet Mail Architecture", RFC 5598, July 2009, DOI 10.17487/RFC5598

[6] M. Kucherawy, "Message Header Field for Indicating Message Authentication Status", RFC 7001, September 2013, DOI 10.17487/RFC7001

[7] E. P. Allman, J. Callas, J. Fenton, M. Libbey, M. Thomas, and M. Delany, "DomainKeys Identified Mail (DKIM) Signatures", RFC 4871, May 2007, DOI 10.17487/RFC4871

[8] M. Kucherawy, D. Crocker, and T. Hansen, "DomainKeys Identified Mail (DKIM) Signatures", RFC 6376, September 2011, DOI 10.17487/RFC6376

[9] S. Kitterman, "Cryptographic Algorithm and Key Usage Update to DomainKeys Identified Mail (DKIM)", RFC 8301, January 2018, DOI 10.17487/RFC8301

[10] J. R. Levine, "A New Cryptographic Signature Method for DomainKeys Identified Mail (DKIM)", RFC 8463, September 2018, DOI 10.17487/RFC8463

[11] M. Kucherawy, "Message Header Field for Indicating Message Authentication Status", RFC 8601, May 2019, DOI 10.17487/RFC8601

[12] D. Crocker, P. Hallam-Baker, and T. Hansen, "DomainKeys Identified Mail (DKIM) Service Overview", RFC 5585, July 2009, DOI 10.17487/RFC5585

[13] D. Srivastava, "Understanding dkim and mandating 1024 bit key encryption", NATIONAL CONFERENCE ON RECENT TRENDS IN OPERATIONS RESEARCH (NCRTOR-2013), 2013, p. 189

[14] H. Hu, P. Peng, and G. Wang, "Towards understanding the adoption of anti-spoofing protocols in email systems", 2018 IEEE Cybersecurity Development (SecDev), 2018, pp. 94–101, DOI 10.1109/SecDev.2018.00020

[15] S. Kitterman, "dkimpy - dkim (domainkeys identified mail)", 2024, Accessed on June 4, 2024

[16] P. Resnick, "Internet Message Format", RFC 5322, October 2008, DOI 10.17487/RFC5322

[17] S. Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", RFC 7208, April 2014, DOI 10.17487/RFC7208

[18] M. Kucherawy and E. Zwicky, "Domain-based Message Authentication, Reporting, and Conformance (DMARC)", RFC 7489, March 2015, DOI 10.17487/RFC7489

[19] A. Gungor, "metaspike - trusted timestamping (rfc 3161) in digital forensics", 2021, Accessed on June 4, 2024

[20] R. Zuccherato, P. Cain, D. C. Adams, and D. Pinkas, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, August 2001, DOI 10.17487/RFC3161

[21] "Openssl ts time stamping authority command", Copyright 2006-2024 The OpenSSL Project Authors. All Rights Reserved., Accessed on June 5, 2024

[22] R. Housley, "Cryptographic Message Syntax (CMS)", RFC 5652, September 2009, DOI 10.17487/RFC5652

[23] A. Ansper, A. Buldas, M. Roos, and J. Willemson, "Efficient long-term validation of digital signatures", Proceedings of the 12th International Conference on Financial Cryptography and Data Security, Tenerife, Spain, 2012, pp. 24–39, DOI 10.1007/978-3-642-33027-8_2

[24] S. Arseni, E. Bureaca, and M. Togan, "Long-term preservation of digital signatures: a need-to-have or a nice-to-have?", Journal of Military Technology, vol. 5, 06 2022, pp. 41–48, DOI 10.32754/JMT.2022.1.06

[25] Radicati Group, "Email statistics report 2021-2025 - executive summary", 2020, Accessed: 2024-08-05

[26] Microsoft, "Playwright for python", https://playwright.dev/python/, 2024, Accessed: 2024-08-16

[27] Ahrefs, "Top 100 most visited websites (2024)", 2024, Accessed: 2024-08-26

[28] Metaspike, "Forensic email collector", 2024, Version 4.0.167.1390, Utilized under a temporary license

[29] Metaspike, "Forensic email intelligence", 2024, Version 2.2.158.1118, Utilized under a temporary license

Appendices

Appendix A

Testing Environment documentation

This chapter will provide a full guide on the installation of the test environment. As the procedure for the second VM is similar to the first, the difference will be directly highlighted. As underlined in the second chapter, the two VMs are using Windows Server 2019 as the operating system.

A.1 Network Configuration

On the VM1 it has been set up a static IP address and configured a DNS server with its own address. The VM2 instead has a different IP address but the same DNS server IP address.

Here is the path to change the address:

Control Panel - Network and Internet - Network and Sharing Center - Change adapter settings - Right-click on the adapter - Properties - Internet Protocol Version 4 (TCP/IPv4) - Properties

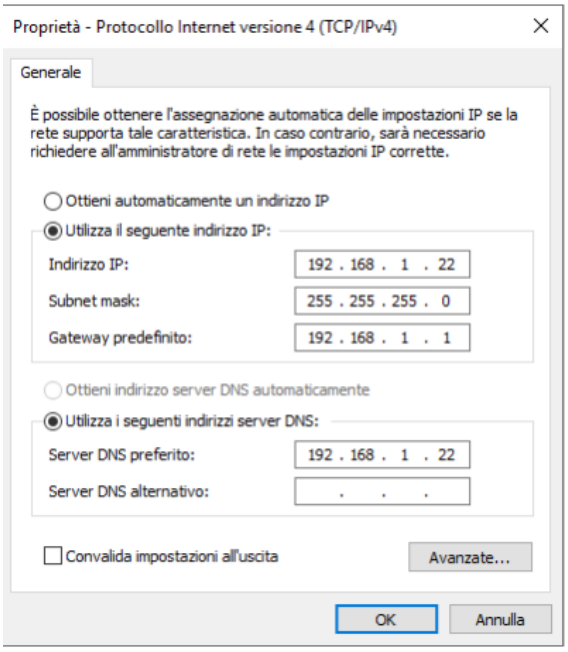

Figure A.1. IP address

To enable sending and receiving emails, a rule in the Windows firewall needs to be set up:

Firewall - Advanced settings - Inbound Rules - New Rule - Port - TCP - Enter local ports 25, 110, 143, 587 - Allow the connection - Next - After entering the name, click Finish.

The rule should appear as follows:

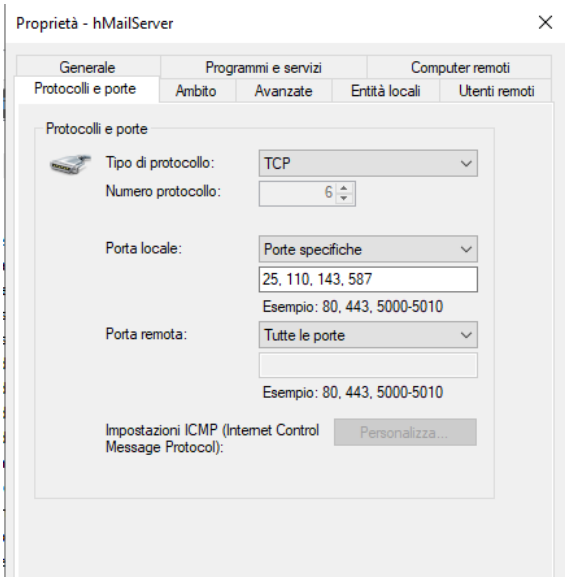

Figure A.2. Firewall Rule

A.2 Key Pair and Record creation

For the creation of the key pair, the *dknewkey* script from the Python library DKIMPY was used.

After installing Python on the VM by downloading it from the official site, DKIMPY can be installed using pip with the command:

pip install dkimpy

With the command *dknewkey testdkim.com*, it runs the script that creates the keys associated with the domain provided in two separate files:

- *testdkim.com.dns* will contain the value of the TXT record that must be entered in the DNS server.
- *testdkim.com.key* will contain the private key that must be installed in HMailServer.

If the DKIM signature is desired on both domains then this command must be run even for the second domain, creating a new key pair and a new record.

A.3 DNS Setup

Let's proceed now to the creation of the necessary DNS records:

For each domain that will be created a forward lookup zone and three records need to be inserted. This is an example of the records created for the first domain:

- Host (A) Record with the name "mail" and the value being the IP address of VM1.

- Mail Exchanger (MX) Record that will refer to the host record from the previous point
- Text (TXT) Record with the value being the content of the file "testdkim.com.dns" generated in the previous step and with the name "key1._domainkey". In this notation, key1 corresponds to the DKIM public key selector, and ._domainkey is a fixed string to be included for this type of record.

This operation needs to be repeated for the second domain that will be created on VM2, this time setting the IP address of the A record to the address of VM2 and coherently changing the content of the txt record.

This is how the DNS manager should look like:

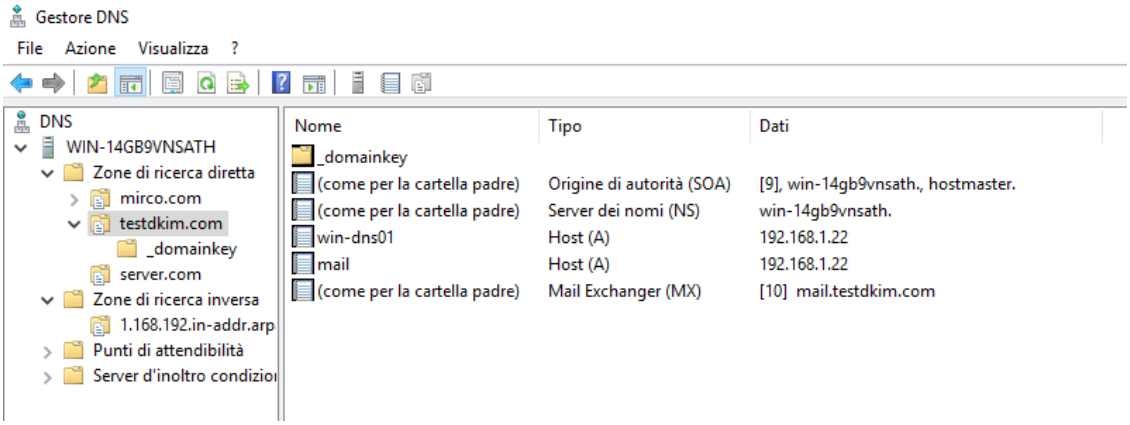

Figure A.3. DNS Records

A.4 HMailServer Installation

Let's proceed with the complete installation of HMailServer. It's important to have already installed MySQL as a database. After having downloaded and executed the installation wizard, MySQL must be selected as the "external database". During the setup of the HMailServer database, enter "localhost" as the Database server address, and use the credentials created during the MySQL installation for user and password authentication.

If during step 6, an error regarding libmysql.dll is encountered, this DLL file must be inserted inside the bin directory of the HMailServer installation folder, as specified in the error message. This file must be the 32-bit version.

The same procedure must be performed on the second VM.

A.4.1 Domain and user creation

To create a domain within HMailServer is sufficient to click on "Add domain..." from its user interface. Once the domain name has been entered it can be saved by clicking "Save".

To create the accounts, click on the Accounts folder - Add.. - and then choose the address and password for the user to be added.

Once the accounts are created, they will appear under the Accounts folder as follows:

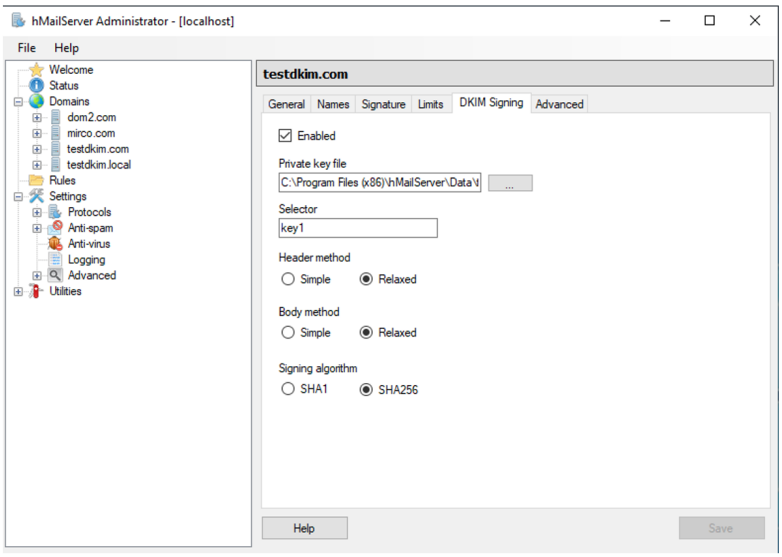

Figure A.4. Account and Domain creation

A.4.2 Enabling DKIM

To enable signing on HMailServer, click on the domain name - DKIM Signing - Check "Enabled".

Then, enter the path to the file containing the private key in the "Private key file" space, in this case, testdkim.com.key which was generated in one of the previous steps. Fill in the selector name for the public key generated in the previous step, and entered in the DNS record, and finally choose the types of canonicalization and the hash algorithm.

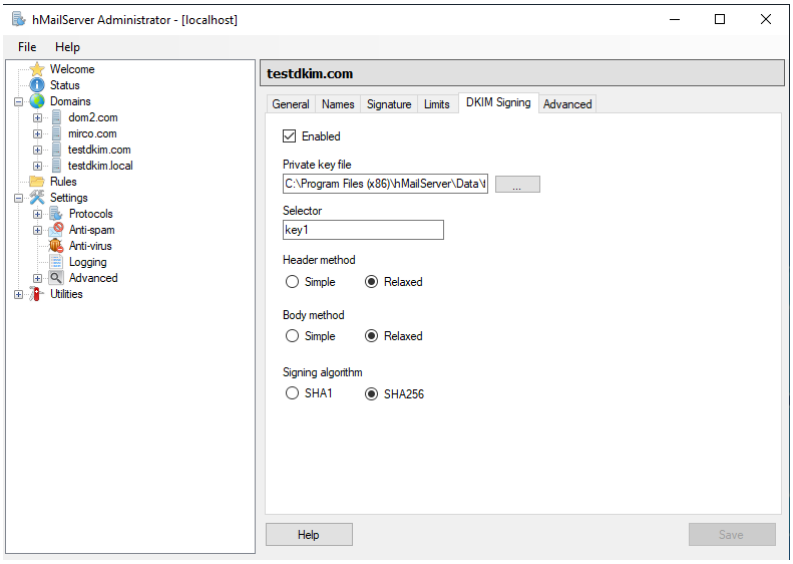

Figure A.5. Enabling DKIM

A.5 Thunderbird Configuration

It is now possible to install Thunderbird as the email client and then set up the accounts created in HMailServer with the following sequence of operations:

Settings - Account Settings - Account Actions - Add Mail Account - After filling in the email name and password - Configure manually - Enter "localhost" under both the Incoming and Outgoing server fields and finally click "Re-test" at the bottom of the page. If a confirmation message is received, it is possible to conclude the procedure by clicking "Done".

The mailbox will then appear in Thunderbird.

Appendix B

User Manual

This chapter will provide a comprehensive guide on how to install and use the tool object of this thesis. Given the presence of multiple modules, the guide will be divided for the use of each part. The user is free to choose whether to use the command line or the web interface to interact with the tool, as both the modalities will be explained.

B.1 Prerequisites

Since this tool is written in Python, the first prerequisite is to have it installed on the subject machine. Specifically, all the project was developed using Python Version 3.12. Together with Python, pip is required as the package installer.

Thanks to the requirements.txt file, all the various dependencies are listed together and can be easily installed with the following command:

pip install -r requirements.txt

These are the additional libraries used in this project:

- requests: Used in the main module for the interaction with the Timestamp Server.
- exrex: Used in the discovery module for the creation of the list of selectors.
- dkimpy: With some modifications, is used in the verify module for the verification of DKIM signatures.
- Flask: Used in the web module as the web framework for the creation of the interface.
- playwright: Used in the homonym module for the password reset process.

This last library is the one imposing some stricter requirements about the operating system in use, that will be listed below:

- Python 3.8 or higher.
- Windows 10+, Windows Server 2016+ or Windows Subsystem for Linux (WSL).
- macOS 13 Ventura, or macOS 14 Sonoma.
- Debian 11, Debian 12, Ubuntu 20.04 or Ubuntu 22.04, Ubuntu 24.04, on x86-64 and arm64 architecture.

Since the tool utilizes different email accounts, it is essential to have written down the credentials of these accounts, making sure to use app password if possible. One account is designated as the receiving mail account, while the others are used to send emails to this receiving account. The credentials for the recipient account should be saved in a file named config.json, which must be placed in the same directory as the other Python files. The credentials for the sending accounts should be stored in individual files within a folder named configs, which should also be located in the same directory as the Python files. Below are examples of the configuration files for both sending and receiving accounts:

```
1 {
2   "server_address": "smtp.xxx.com",
3   "smtp_port": "xxx",
4   "username": "xxxx@gmail.com",
5   "password": "xxxxxx"
6 }
```

Figure B.1. Sending config

```
1 {
2   "server_address": "imap.xxx.com",
3   "username": "xxxx@gmail.com",
4   "password": "xxxxxx"
5 }
```

Figure B.2. Receiving config

There are also some modifications that need to be executed on the source code, in order to have the tool fully functional:

- Changing the name of the interface used by tcpdump to capture the output (lines 138-140-142 of main.py)
- Creating an ssl_key.log file and changing the path at line 14 of main.py to make sure keys are saved in it.
- Changing the email address with the recipient account created at line 405 of main.py and 144 of mail.py

Finally, to run the web interface just use the following command and visit the link created by Flask:

sudo python3 web.py

B.2 Main module

This module, responsible for logging DKIM records, can be run in two different modes. The user can choose to either timestamp only a zip folder containing all the produced files or timestamp each individual log file associated with each domain. The choice depends on the availability of timestamps for the user.

For help, run:

python3 main.py -help

To execute this module in its basic mode, run the following command:

sudo python3 main.py

To switch modality and timestamp all the log files individually just append the "-all" option:

sudo python3 main.py -all

It's important to note that only this module requires root privileges to be executed because of the network capture operated by tcpdump.

The same actions can be performed through the web interface by simply pressing the "Run" button at the top of the page. Make sure to tick the checkbox in case the timestamping of each log file is required. The output produced by the command will be visible at the bottom of the page under the "Output" section.

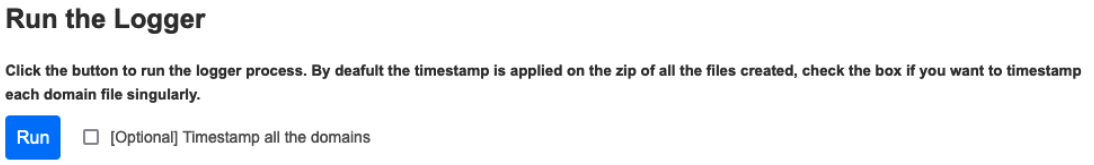

Figure B.3. Run button on the Web Interface

If the autonomous run is preferred, these are the steps to follow on an Ubuntu machine:

First, open the crontab file by using the command sudo crontab -l

Then, write into it the following entry:

*mm hh * * * cd <path project folder>&& /usr/bin/python3 main.py 2>&1 — logger -t cronjob*

Here, "mm hh * * *" represents the minute, hour, day of the month, month, and day of the week respectively. By using an asterisk ("*"), any time is valid. For instance, "30 14 * * *" schedules the command to execute daily at 14:30. Then, we have a command to move to the project folder and execute the main.py script, redirecting the standard error (2) to the standard output (1), so that regular output and error messages are treated the same. By piping (|) the output to the logger command, it will be recorded in the system log and will be available by using different commands such as journalctl.

This module will generate and update various files and folders:

- log.txt: In which all the information about changes and updates will appear.
- selector_domains.txt: In which are present the pairs monitored.
- Output folder: In which all the files related to the logging process will be saved, log files, timestamp requests and responses, and eventually packet captures.
- Zip folder: In which will be created the zip folder to be timestamped together with the timestamping files and packet capture.
- Emails folder: Where all the emails downloaded will be saved.

B.3 Discovery module

This is the module used to discover new DKIM records and offers various execution options. As for the previous module, it is possible to get general help on the modes available for running thanks to the command:

python3 discover.py --help

By default, when the script is executed, the program will prompt the user to enter a domain name. After that, the results will be displayed on the same command prompt, along with additional statistics about the selector usage. This can be done using the following command:

python3 discover.py

Additionally, the user can choose to query multiple domains at once, by providing a text file, where each domain name is listed on a separate line. The file must be passed to the script during the run phase, by postponing the "–domain-file" flag followed by the file path.

python3 discover.py --domain-file <filepath>

Furthermore, users have the option to specify which selectors to query for the given domains by providing a text file with one regex per line. The tool will generate all the possible names that match the provided regex patterns and then use this list to query the selected domains. This is achieved by postponing the "–regex-file" flag followed by the file path, during the script execution. This mode can be used independently or in combination with the domain-file mode.

python3 discover.py --regex-file <filepath>

These actions can also be performed through the web interface. First, the user selects whether to input a single domain through a text box or upload a file containing multiple domains. Then it can optionally upload a second file with one regex per line for the selector generation. By pressing the "Discover records" button, the discovery process will begin, and the results will be displayed at the bottom of the page under the "Output" section.

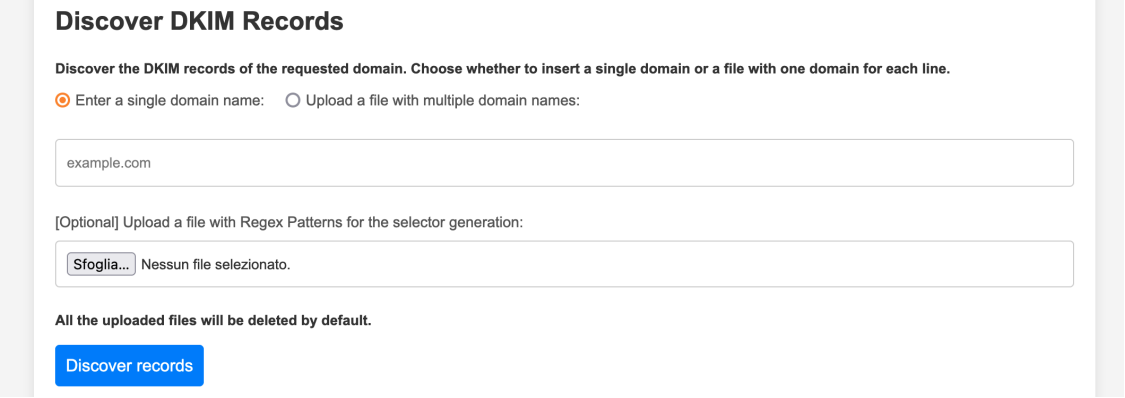

Figure B.4. Discover a single domain.

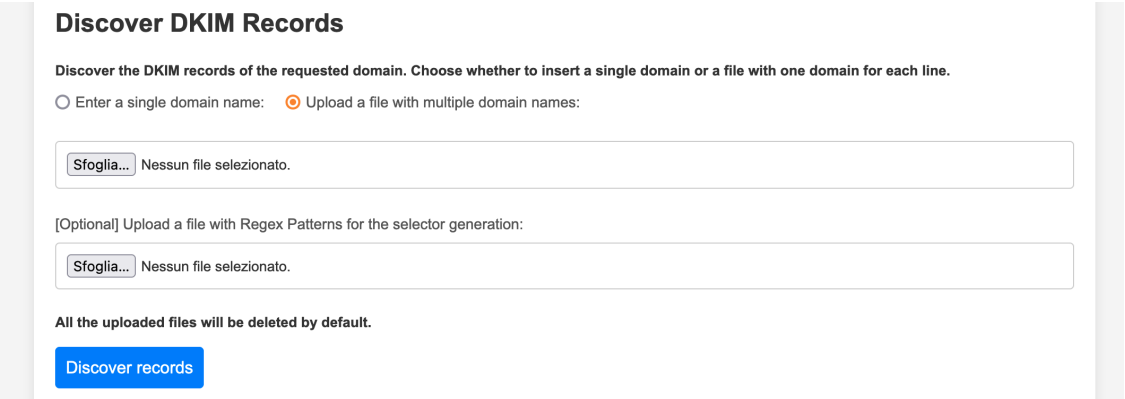

Figure B.5. Discover multiple domains.

This module will generate and update various files:

- history.txt: In which will be written all the domains and selectors discovered through the tool. This is useful to obtain reliable statistics.
- selectors_domains_discovered.txt: In which are present the pairs discovered that will be monitored.
- selectors_list_counters.txt: In which each selector has assigned a counter incremented after each utilization. This is useful for generating statistics.

B.4 Verify module

This module has the duty of verifying the DKIM signatures of a given email. Similar to the other two modules, additional help on its usage settings can be obtained by running:

python3 verify.py --help

The command line execution requires the user to append the file path to the eml file, representing the email, to the basic Python command. The results, performed by considering the time of receipt as the time of verification, will be displayed in the same command line.

python3 verify.py <filepath>

The user can optionally provide the tool with a specific record for the verification process by adding the "--record" flag and pasting the record as a string:

python3 verify.py <filepath>--record "<record>"

The web interface instead requires the user to upload the eml file through the dedicated form, optionally choose to insert inside a textbox the record to use, and then click the "Upload and verify" button to begin the verification process. Since the uploaded file may contain personal information, the tool will delete it by default at the end of the verification process. However, the user can choose to keep the file saved in the system by selecting the appropriate checkbox. As for the other modules, the results will be displayed at the bottom of the page under the "Output" section.

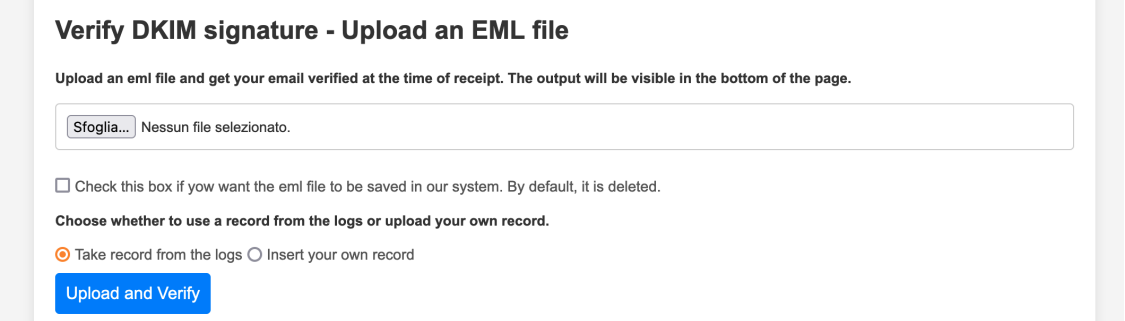

Figure B.6. Verify module standard.

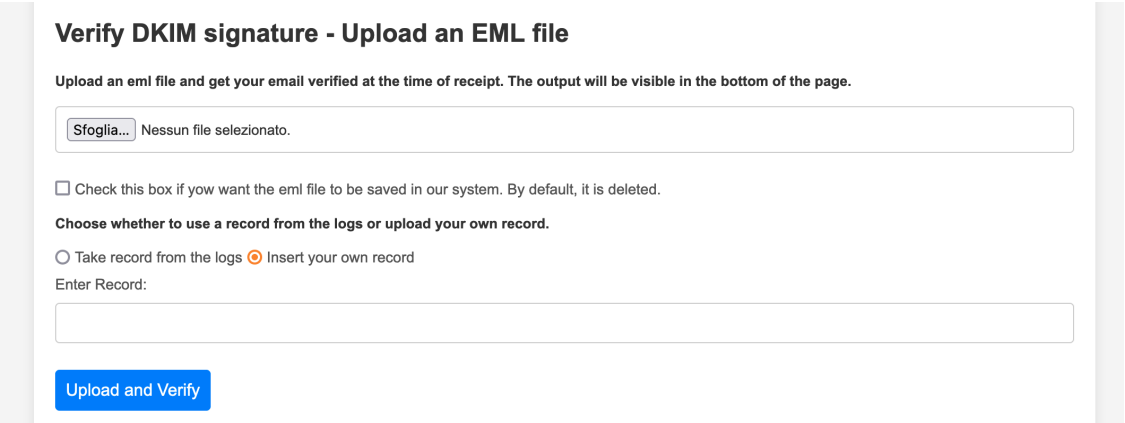

Figure B.7. Verify module with record insertion.

Appendix C

Programmer's Manual

This appendix will explain the main functions of the tool's modules in order to help programmers to further extended and maintain the source code. To recap and have a better view of the general architecture of the tool, this is the whole schema, already cited in the "Architecture of the tool" subsection of the Solution chapter.

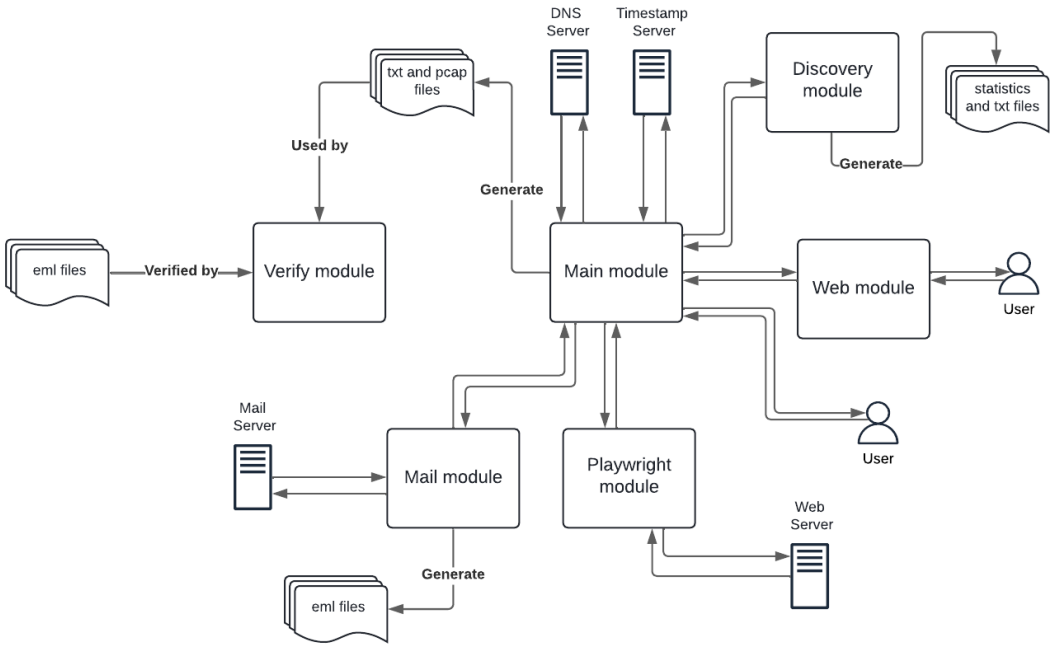

Figure C.1. Software Architecture

Each module will be separately described in order to achieve a clearer understanding of the data workflow and the tool's general usage.

C.1 Main.py

To fully comprehend this module's behavior, it's necessary to have a look at its main function. Each call will be generally analyzed, in the same order as they appear in the main.

- read_selectors_domains: This is the function used to retrieve from the txt file the lists of pairs (selector, domain), for both the main and the discovered file.
- reset_password: This is a function imported from the Playwright module to start the password recovery process needed to receive the email. Further description about this function will be provided in the dedicated section.
- get_domain_selector_pairs: This is a function imported from the Mail module, used to retrieve the updated pairs (selector, domain) from the mail inbox. Further description about this function will be provided in the dedicated section.
- update_selectors_list_with_counts: This function is utilized to update the list of the selectors in case of new selector name are found.
- update_selector_domains: This is one of the most important functions of this module. By taking as input the list of pairs previously retrieved, it has the duty of updating the main list of the domains monitored, eventually replacing some pairs with the updated one and moving some pairs from the main list to the discovered list. All these movements are logged into the "log.txt" file, with a timestamp of the time and optionally the email ID of the message that helped identifying the update.
- process_selectors_domains: This is another important function, which is called twice over the two different lists. Its role is to perform the DNS queries for each pair contained in the list and capture the traffic related to this operation. When receiving the reply of the DNS query, it also performs an additional check looking for possible record changes. If those changes are present, a message will be written in the "log.txt" file. The received record is written in the correspondent output file thanks to another function, named "execute_command" which will prepend to the record a timestamp and the previous SHA512 file hash, useful for verification. Finally, if the module is called with the "-all" flag, this function will have the duty of capturing and executing the exchange with the Timestamp server to get the Timestamp token. All the files produced by this function will appear in the "output" folder.
- zip_and_timestamp: The default execution of this module will involve the call of this function, which has the duty of creating the zip archive that will be timestamped. The whole process's traffic is captured, and together with the zip archive and the timestamp files will appear in the "zip" folder. At the top of the function, two lists represent the files and folders to be zipped.
- capture_traffic: As the name says this function has the duty of generating a pcap files with the recorder traffic. By passing as a parameter the traffic type, a different filter will be inserted in the tcpdump command. This function is called by "zip_and_timestamp", and "process_selectors_domains".

C.2 Mail.py

This is a support module for main.py which has the duty of interacting with mail servers and handling email content. It includes five functions:

- read_config: This function is used to load the configuration details, such as server addresses and credentials, written in a json file.
- extract_dkim_info: Given an eml file, this function leverages the email Python library to extract the domain and the selector from a DKIM signature.
- send_email: Given a config file read by the "read_config" function, this function will create and send an email to a fixed email address. The email is created as a Mime object with the necessary options. Once connected to the server specified in the config file, the email is sent, and the connection is closed. This function is repeatedly called by the "send_multiple_mails" functions, once for each configuration file in the "config" folder.

- get_domain_selector_pairs: This is the function invoked by the main module described in the previous section. To extract the updated pairs, it first needs to send new mails to the inbox. This is achieved by calling the "send_multiple_mails" function. Once the sending process is finished, this function will connect to the IMAP server to download the received emails saving one email per domain-selector pair in the "email" folder.

C.3 Discover.py

To fully understand this module's behavior, it's necessary to have a look at its main function. Before entering the discovery phase, the main function uses different functions to read and prepare the required information:

By default, the domain name is asked from the user via command line. If a txt file containing multiple domain names is provided using the "--domain-file" flag, the "read_domains_from_file" function returns a list of these domains.

By default, a fixed list of selectors will be used for the discovery, which is read from a file using the "read_selectors_with_counters" function. If a txt file with regex is provided using the "--regex-file" flag, the list of domains will be generated by the "generate_selector_regex" function, which leverages the exrex Python library.

To complete the information needed by the script, the "read_existing_couples" function reads the existing domain-selector pairs from both the main and the discovered list, while the "read_history" function reads the history file, which is needed to obtain reliable statistics.

At this point, for each provided domain, the list of default selectors or the regex generated list, is used to check for existing records. This is performed thanks to the "execute_dig" function, which will execute the command and return the reply. This response will be analyzed by the "record_exist" function that will check if it contains a DKIM record. If the check is affirmative, the selector name is saved in a list, and the selector list counters are updated. Before updating, the tool verifies that the selector has not already been discovered for that domain by referencing the history file. This operation is also handled by the "update_selectors" function, called when using the regex generated selector list.

Finally, the discovered domain-selector pairs are printed on the command line and added to the discovered list if they are not already present in either list. The history and selector list files are then updated using the "write_history" and "write_selectors_with_counters" functions, respectively.

There are three statistics generated by three different functions:

- stats_count_selectors: This function reads the history file and prints the number of domains that have a certain number of available selectors.
- stats_top_n_selectors: This function sorts the selector list with counters and displays the top n selectors.
- stats_percentage_usage: This function calculates the percentage of usage for each selector in the list with counters, displaying the utilization of each selector relative to the others.

C.4 Verify.py and custom_dkim.py

The verify.py module is essentially an extension of the "dkimverify.py" script provided inside the dkimpy library. This section in fact, comprehend the custom_dkim.py module, which has been used to extend the previously cited library to meet the tool's specific need. These are the operations performed by the main function:

First, the eml file that needs to be verified is used to create a DKIM object and three key information are extracted using the "extract_dkim_fields" function: the timestamp of the signature, the selector and the domain. Since the timestamp is not mandatory the receipt time is used

if it is missing. If no record is provided by the user, the extracted selector and domain are then used to reconstruct the filename where the record is stored, and the "find_correct_dkim_record" function retrieves the appropriate record using the provided timestamp. At this point, there are two possible execution flows:

1. If the record is successfully retrieved, the verification process can begin through the "verify_fixed" function. This is where the standard library was modified, as it previously did not accept a static record passed as a parameter.
2. If no record is found for the extracted domain selector, the tool proceeds by querying the DNS server for the appropriate record. This is the standard behavior of the library, executed by the "verify" function, which was modified for the previous scenario.

In both cases, the "verify_headerprep" function called by both "verify" and "verify_fixed" has been modified. Instead of calling the standard "validate_signature_fields" function, it calls the "custom_validate_signature_fields" function, altered to use the receipt time as the time of verification, rather than the current time.

C.5 Playw.py

This is a support module for main.py which has the duty of generating a password reset mail. Since this is a specific solution, it can be changed to work with other specific websites. In its current form, the code will navigate to the ASOS website and follow the necessary steps to generate a password reset link. It is important to note that an account must exist for the provided email address.

The script begins by creating a browser object and a page within it, then uses the "goto" function to navigate to the target website.

From this point on, the script primarily repeats two key instructions:

- wait_for_selector: Which waits for the HTML element to be rendered and displayed on the page.
- click/fill: Which performs the actions on the HTML element.

The whole procedure is made accessible to the main.py module through the "reset_password" function, which exports these instructions.

C.6 Web.py

This module is responsible for the web interface of the tool. Specifically, the "web.py" file handles the backend, while the frontend is managed by the "index.html" and "style.css" files, located in the "templates" and "static" folders respectively.

The application is built thanks to the Flask framework, with the backend designed to manage the various functionalities offered by the tool. Different requests are handled based on the tags in the HTML form or the files uploaded by the user. All uploaded files are temporarily stored in a directory within the project, called "uploads" and deleted at the end of each execution. Each functionality then executes a corresponding Python script, captures the output, and passes it to the HTML page, where it is displayed under the output section. Additionally, the web interface can be made accessible on a specific port by modifying the "app.run" function parameters.

Appendix D

Metaspike's tools

These screenshots are related to the Forensic Email Collector v4.0.167.1390 and Forensic Email Intelligence v2.2.158.1118, both utilized under a temporary license. The email content shown in the screenshots is the property of the thesis author.

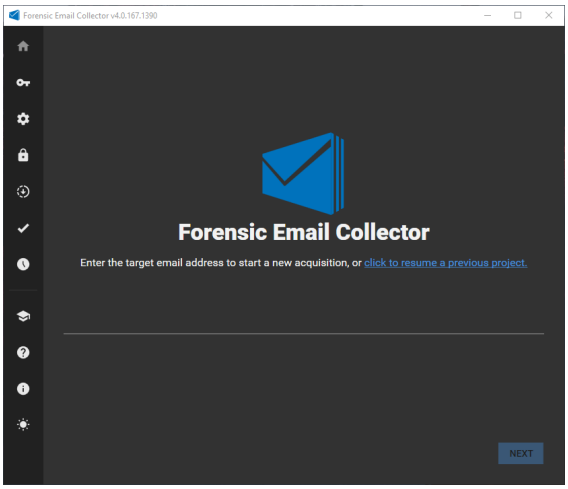

Figure D.1. FEC: Homepage

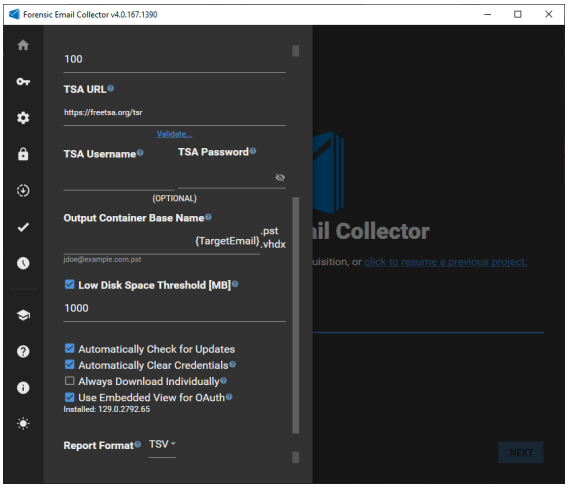

Figure D.2. FEC: Timestamping authority settings

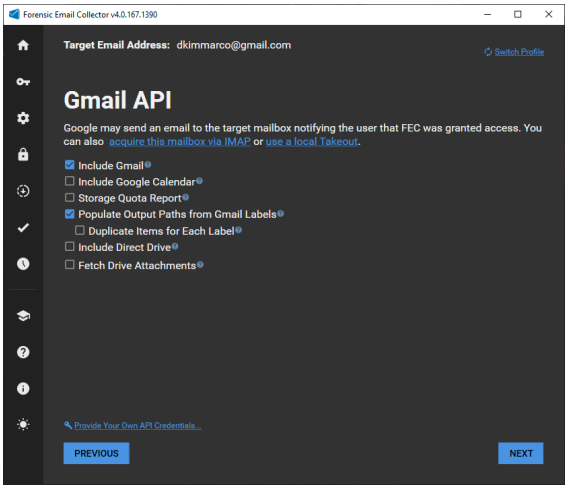

Figure D.3. FEC: Acquisition settings

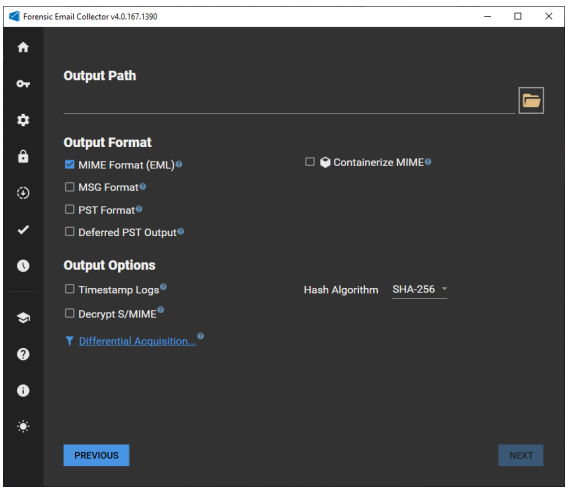

Figure D.4. FEC: Output settings

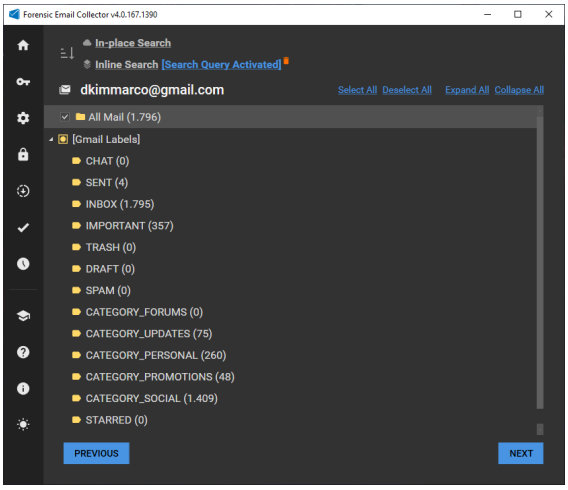

Figure D.5. FEC: Email address labels

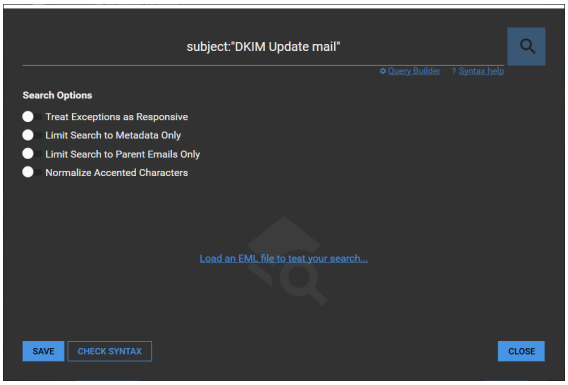

Figure D.6. FEC: Inline-search

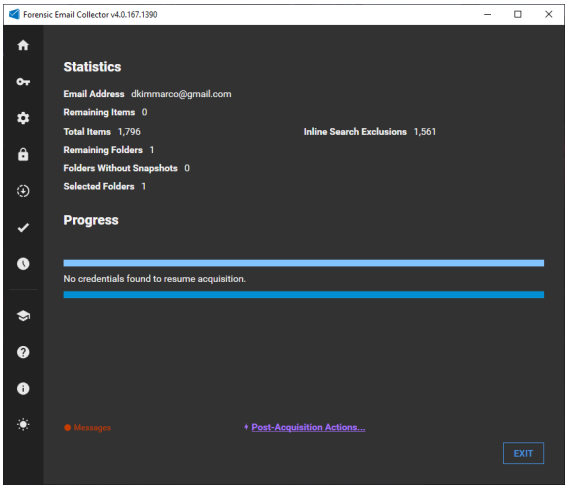

Figure D.7. FEC: Acquisition completed

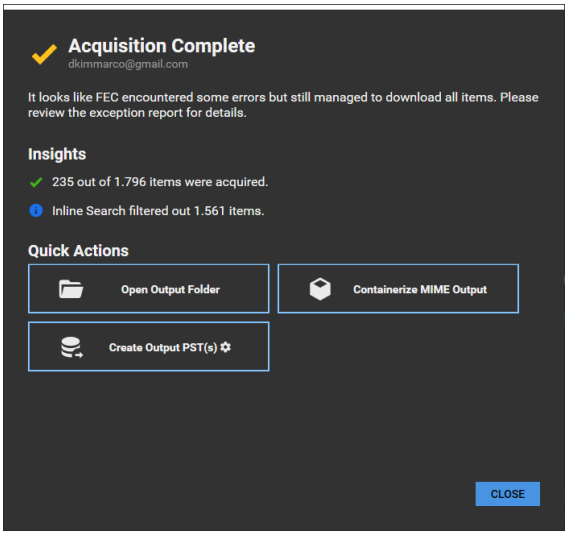

Figure D.8. FEC: Post-acquisition action

This instead is an example of the content of the Acquisition Log generated by the Forensic Email Collector:

```
Log file initialized at 03/10/2024 10:38:55 +02:00
Software version: Forensic Email Collector v4.0.167.1390

2024/10/03 10:38:55.553 - Getting total message count for Gmail account.
2024/10/03 10:39:06.335 - The Gmail mailbox dkimmarco@gmail.com contains
    1.794 messages and 1.257 threads.
2024/10/03 10:39:06.707 - Collecting detailed information for the 14 labels.
2024/10/03 10:39:06.826 - Gmail label information has been acquired.
2024/10/03 10:43:04.116 - Starting acquisition.

Current Output Path: D:\FORENSIC\prova

------ Settings Used ------

Target Email Address: dkimmarco@gmail.com
Examiner Name: Marco Vitale
Agency: PoliTO

MIME Output: True
MSG Output: False
PST Output: False
Deferred PST Output: False
Decrypt S/MIME: False
Timestamp Logs: True
Differential Acquisition: False
Hash Algorithm: Sha256
Split Output: False
Output Pst Base Name: {TargetEmail}

Server Settings
   Server Address: https://www.googleapis.com/gmail/v1/
   Protocol: GoogleApi
   Authentication Mode: Oauth2
   Include Gmail: True
   Include Google Calendar: False
   Populate Output Paths from Gmail Labels: True
   Duplicate Items for Each Gmail Label: False
   Use Domain-wide Delegation: False
   Include Google Drive Directly: False
   Fetch Drive Attachments: False

Inline Search Query: subject:"DKIM Update mail"
Treat Inline Search Exceptions as Responsive: False
Include Attachments during Inline Search: True
Limit Inline Search to Metadata Only: False
Normalize Accented Characters: False

2024/10/03 10:43:04.137 - Acquiring mailbox rules.
2024/10/03 10:43:04.446 - Acquired 0 mailbox rules.
```

Figure D.9. FEC: Acquisition log (Part 1)

```
------ Selected Folder Structure ------

All Mail - 1.794

------ Gmail Labels ------

CHAT (0) [ID: CHAT]
SENT (4) [ID: SENT]
INBOX (1.793) [ID: INBOX]
IMPORTANT (357) [ID: IMPORTANT]
TRASH (0) [ID: TRASH]
DRAFT (0) [ID: DRAFT]
SPAM (0) [ID: SPAM]
CATEGORY_FORUMS (0) [ID: CATEGORY_FORUMS]
CATEGORY_UPDATES (74) [ID: CATEGORY_UPDATES]
CATEGORY_PERSONAL (260) [ID: CATEGORY_PERSONAL]
CATEGORY_PROMOTIONS (48) [ID: CATEGORY_PROMOTIONS]
CATEGORY_SOCIAL (1.408) [ID: CATEGORY_SOCIAL]
STARRED (0) [ID: STARRED]
UNREAD (126) [ID: UNREAD]

2024/10/03 10:43:04.653 - Starting to capture snapshots from folders.
2024/10/03 10:43:04.659 - Capturing snapshot for folder All Mail.
2024/10/03 10:43:06.354 - Finished capturing snapshot for folder All Mail.
2024/10/03 10:43:06.356 - Finished capturing snapshots from folders.
2024/10/03 10:43:06.360 - Starting to download items from selected folders.
2024/10/03 10:43:06.370 - Downloading items from folder All Mail.
2024/10/03 10:43:06.637 - Collecting detailed information for the 14 labels.
2024/10/03 10:43:06.736 - Gmail label information has been acquired.
2024/10/03 10:43:07.427 - Item with ID:'2' Service ID:'1924f2bc8530ebe1' not
    responsive to Inline Search. Excluding item.
2024/10/03 10:43:07.437 - Item with ID:'3' Service ID:'1924ebf16ac6210c' not
    responsive to Inline Search. Excluding item.

...

2024/10/03 10:45:07.827 - Item with ID:'1698' Service ID:'18fee2975a7bb595'
    not responsive to Inline Search. Excluding item.
2024/10/03 10:45:08.277 - Finished downloading items from folder All Mail.
2024/10/03 10:45:08.279 - Finished downloading items from selected folders.
2024/10/03 10:46:14.933 - Starting retry cycle 2 of 10.
2024/10/03 10:46:14.933 - Starting to download items from selected folders.
2024/10/03 10:46:14.933 - Downloading items from folder All Mail.
2024/10/03 10:46:15.247 - Item with ID:'239' Service ID:'19169ac8a68a5f67'
    not responsive to Inline Search. Excluding item.
2024/10/03 10:46:16.540 - Finished downloading items from folder All Mail.
2024/10/03 10:46:16.541 - Finished downloading items from selected folders.
2024/10/03 10:46:16.542 - Acquisition complete. All items were downloaded.
```

Figure D.10. FEC: Acquisition log (Part 2)

```
------ Acquisition Summary ------

Selected Folders: 1
Total Folders: 1
Selected Folders without Snapshots: 0
Total Items in Selected Folders Reported by Provider: 1.794 (Estimated)*
Total Items in FEC's Snapshot: 1.795
Items Excluded via Inline Search: 1.560
Downloaded Items: 235
Remaining Items: 0

* Count does not include calendar events, spam, and trash.

2024/10/03 10:46:16.755 - All items were downloaded. Clearing credentials.

Time elapsed during this acquisition session: 0:03:13,540694
```

Figure D.11. FEC: Acquisition log (Part 3)

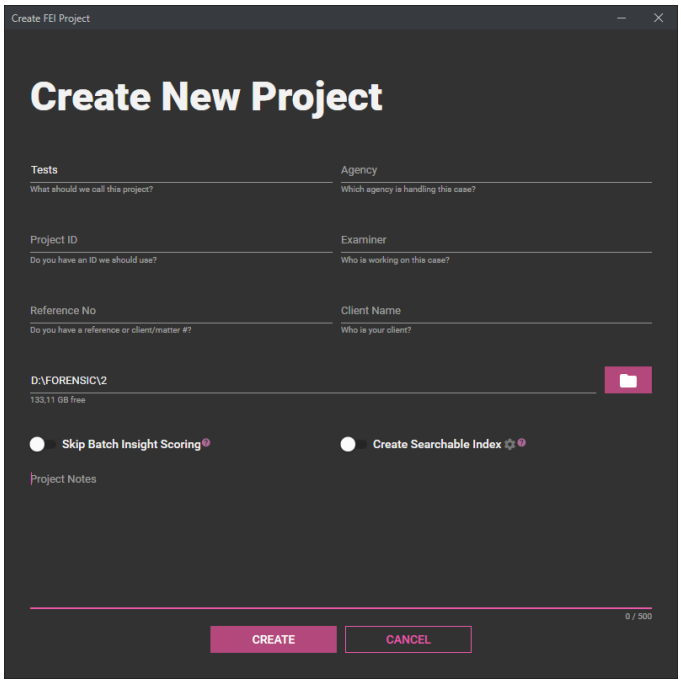

Figure D.12. FEI: Project creation

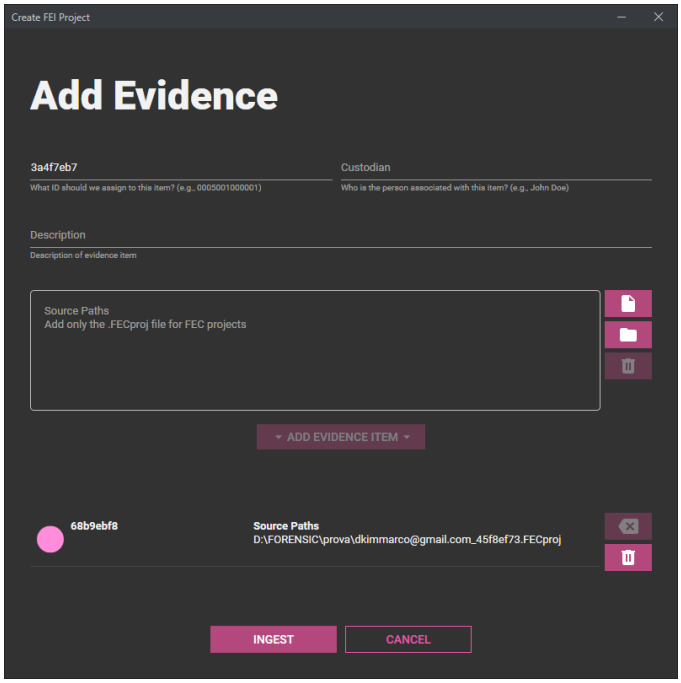

Figure D.13. FEI: Evidence insertion

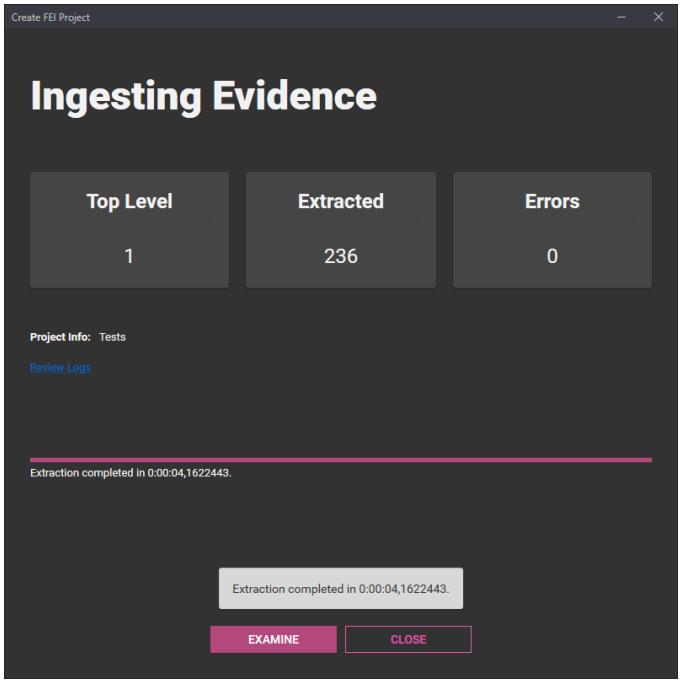

Figure D.14. FEI: Evidence extraction

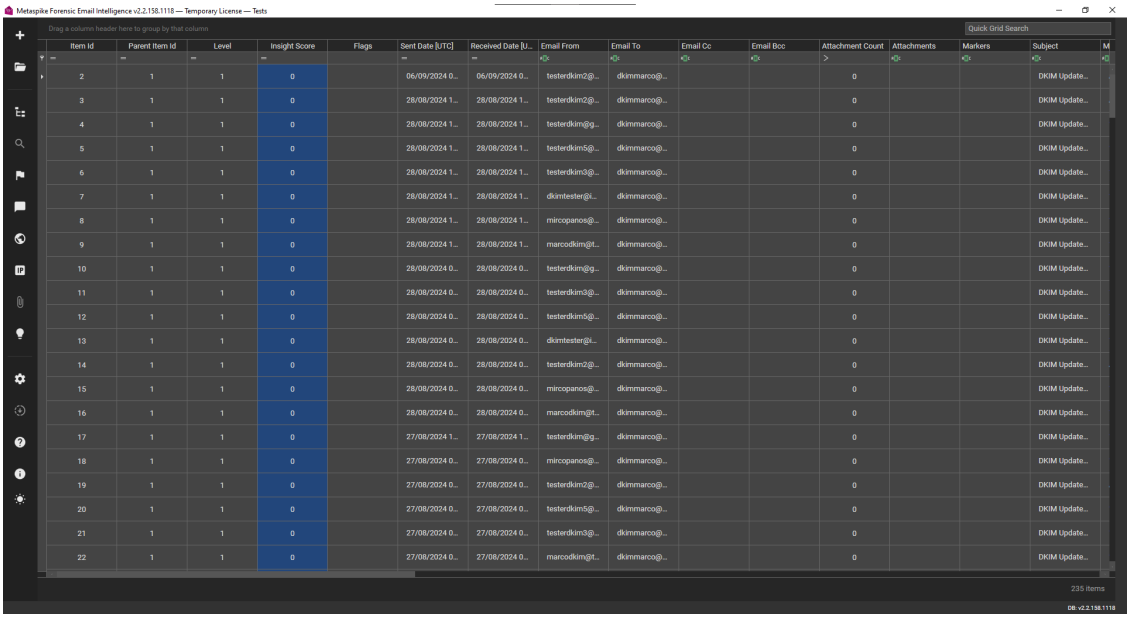

Figure D.15. FEI: Evidence inspection

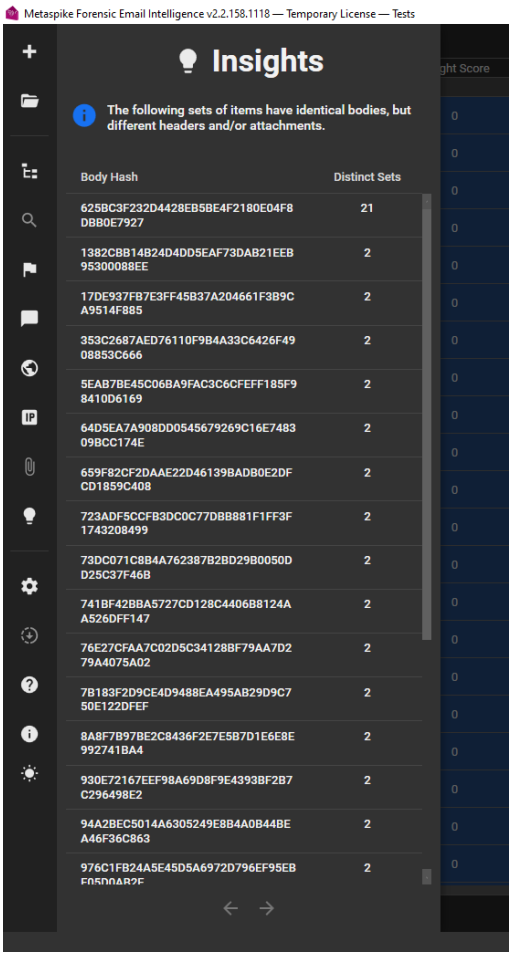

Figure D.16. FEI: Project insights

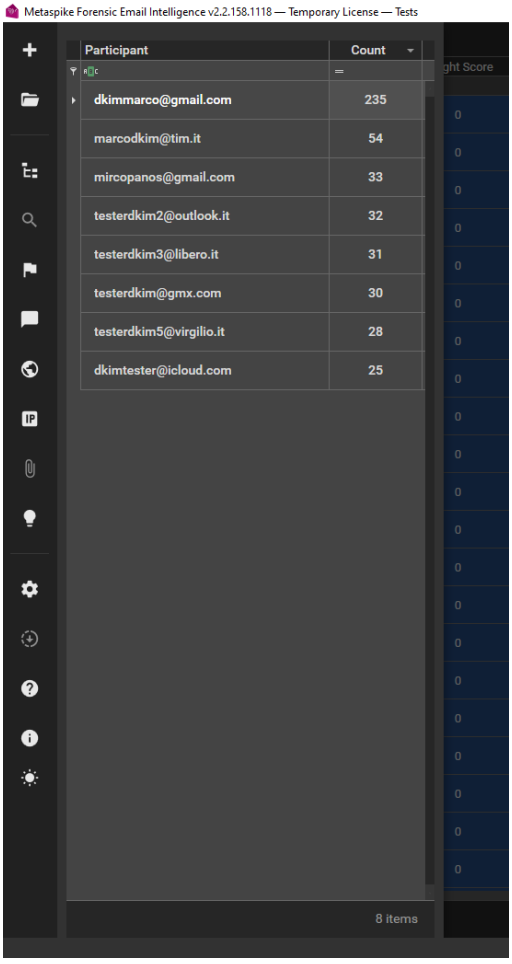

Figure D.17. FEI: Project participants

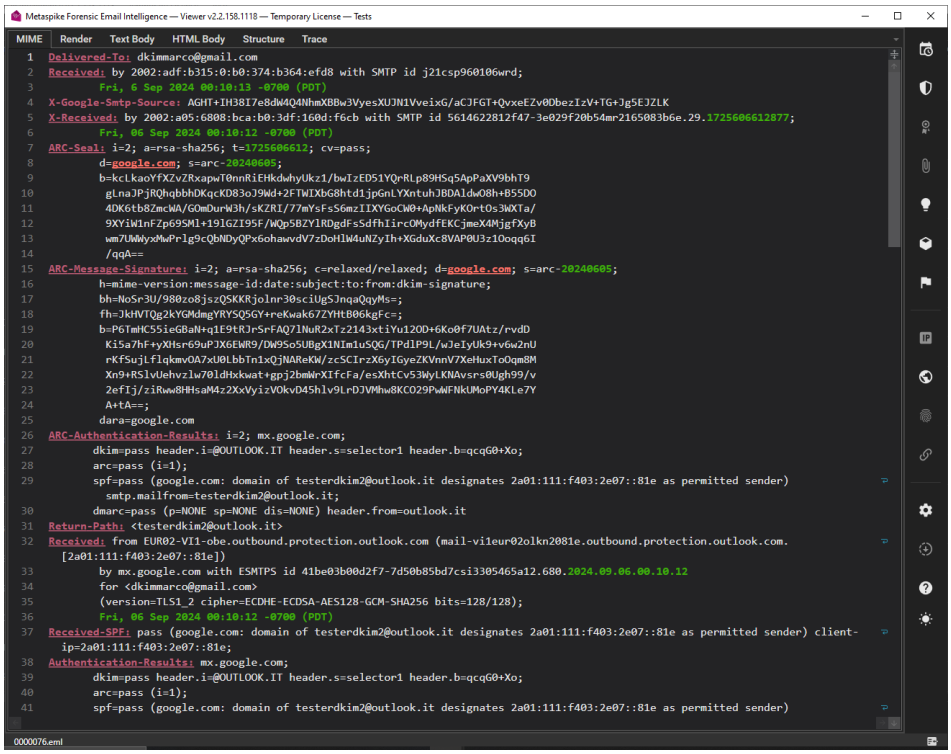

Figure D.18. FEI: Email examination

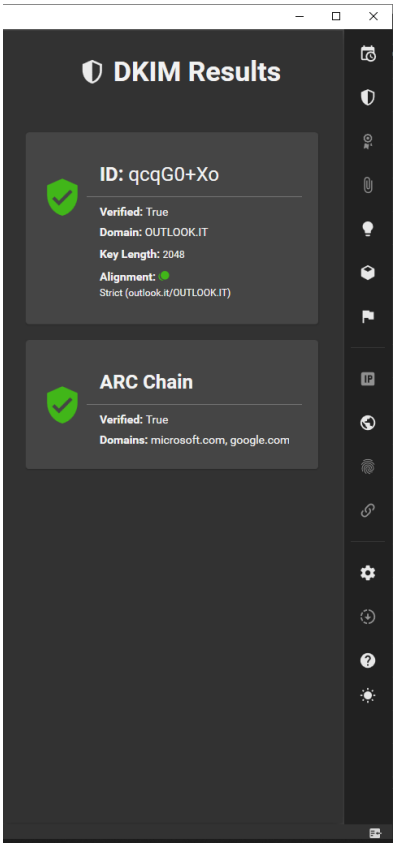

Figure D.19. FEI: DKIM and ARC Verification

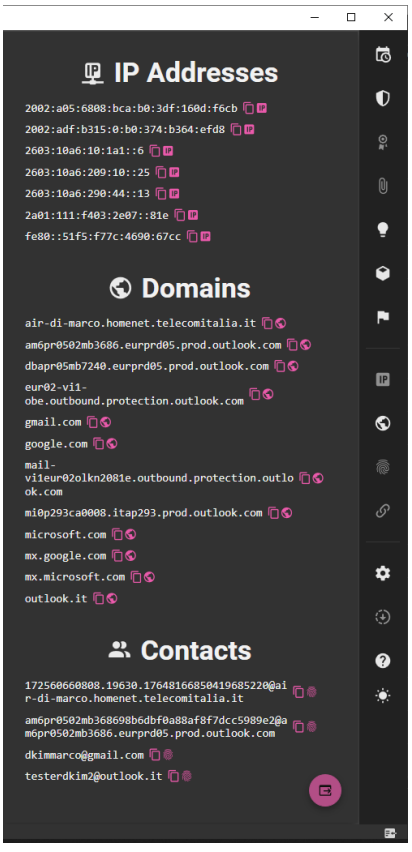

Figure D.20. FEI: Email entities

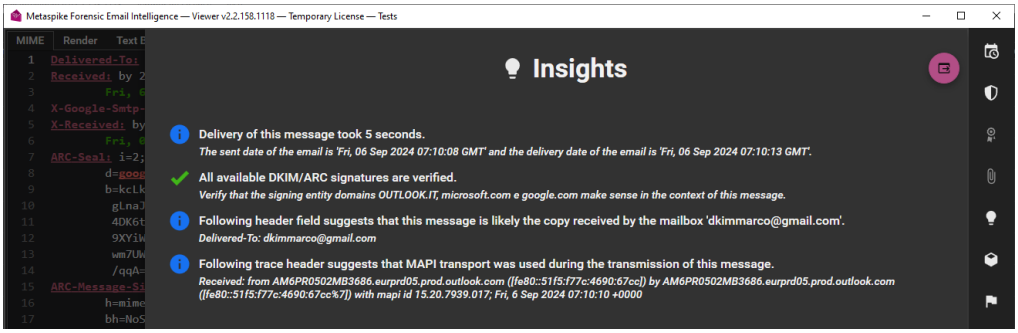

Figure D.21. FEI: Email insights

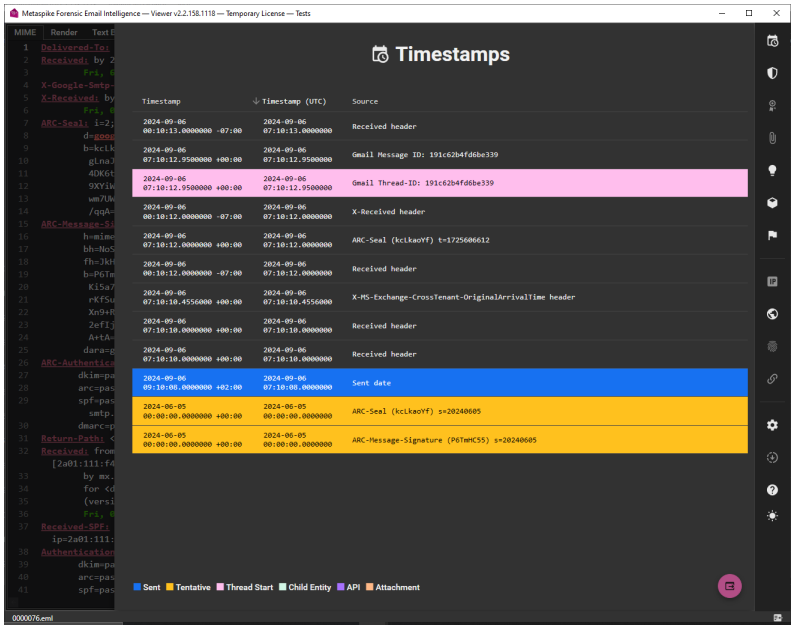

Figure D.22. FEI: Email timestamps

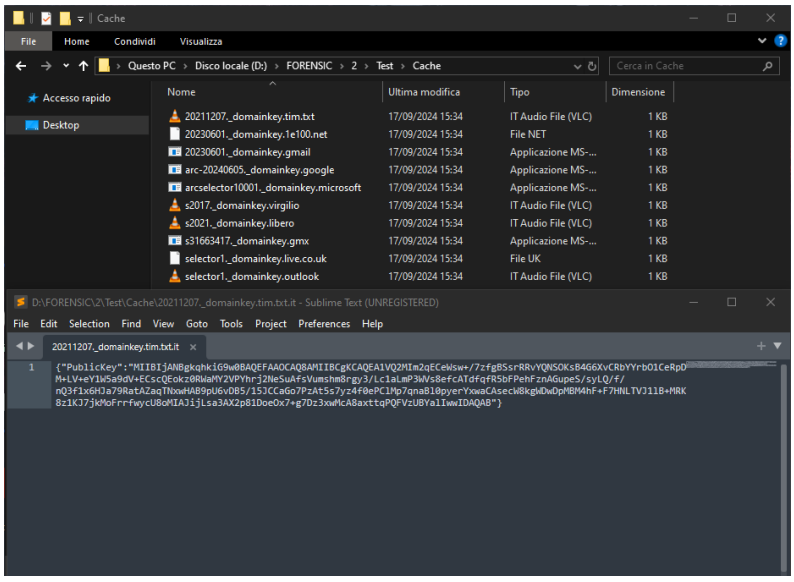

Figure D.23. FEI: DKIM keys saving